

Vertica® Analytic Database 4.1, Revision 1

Administrator's Guide

Copyright© 2006-2011 Vertica Systems, Inc.

Date of Publication: January 7, 2011



CONFIDENTIAL

Contents

Technical Support	1
--------------------------	----------

About the Documentation	2
--------------------------------	----------

Where to Find the Vertica Documentation	2
Reading the Online Documentation	2
Printing Full Books	4
Suggested Reading Paths	4
Where to Find Additional Information	6
Typographical Conventions	7

Preface	9
----------------	----------

Administration Overview	10
--------------------------------	-----------

Configuring the Database	11
---------------------------------	-----------

Configuration Procedure	11
Prepare Disk Storage Locations	12
Prepare the Logical Schema Script	14
Prepare Data Files	14
Prepare Load Scripts	15
Create an Optional Sample Query Script	15
Create an Empty Database	16
Create the Logical Schema	17
Perform a Partial Data Load	17
Test the Database	18
Optimize Query Performance	18
Complete the Data Load	18
Test the Optimized Database	18
Set Up Incremental (Trickle) Loads	19
Implement Security	19
Implement Locales for International Data Sets	19
Change Transaction Isolation Levels	25
Configuration Parameters	25
General Parameters	25
Tuple Mover Parameters	27
Internationalization Parameters	29
Epoch Management Parameters	30
Monitoring Parameters	32
Profiling Parameters	33
Query Repository Parameters	33
Security Parameters	35
Kerberos Authentication Parameters	35
Designing a Logical Schema	37
Data Warehouse Schema Types	37

Using Multiple Schemas.....	40
Creating Tables.....	45
Adding Constraints.....	49
Removing Constraints.....	54
Enforcing Constraints.....	55
Analyzing Constraints (Detecting Constraint Violations).....	55
Using Sequences.....	60
Implementing Views.....	74
Altering Tables.....	76
Creating a Physical Design.....	77
Using the Database Designer.....	77
Creating Custom Designs.....	91
Implementing Security.....	112
Implementing Client Authentication.....	114
Implementing SSL.....	130
Implementing Database Authorization.....	136

Operating the Database **140**

Managing Your License Key.....	140
Starting the Database.....	142
Stopping the Database.....	142
Loading and Modifying Data.....	144
Bulk Loading.....	144
Trickle Loading.....	148
Using the COPY Command.....	148
Tuple Mover.....	165
Collecting Statistics.....	172
Bulk Deleting and Purging Data.....	176
Best Practices for DELETE and UPDATE.....	177
Purging Deleted Data.....	180
Partitioning Tables.....	183
Defining Partitions.....	184
Dropping Partitions.....	186
Partitioning and Segmenting Data.....	188
Partitioning and Data Storage.....	189
Managing Partitions.....	190
Auto Partitioning.....	192
Partition Elimination.....	194
Monitoring the Database.....	197
Monitoring the Log Files.....	197
Rotating the Log Files.....	197
Using the SQL Monitoring API.....	199
Monitoring Processes.....	205
Monitoring Events.....	205
Monitoring Linux Resource Usage.....	215
Monitoring Vertica Using Ganglia.....	218
Recovering the Database.....	233
Failure Recovery.....	235
Restarting Vertica on a Host.....	247
Restarting the Database.....	248
Recovering the Cluster from a Backup.....	251

Monitoring Recovery.....	251
Exporting a Catalog.....	251
Backup and Restore.....	252
When to Back Up the Database.....	252
Backing Up the Database.....	253
Restoring the Database from a Backup.....	255
Restoring to the Same Cluster.....	255
Restoring the Database.....	256
Copying a Database to Another Cluster.....	258
Best Practices for Disaster Recovery.....	261
Managing Nodes.....	263
Adding Nodes.....	264
Rebalancing Data Across Nodes.....	268
Removing Nodes.....	271
Replacing Nodes.....	275
Modifying Database Designs for Updated Nodes.....	279
Testing Modified Database Designs.....	282
Managing Disk Space.....	283
Adding Disk Space to a Node.....	283
Adding Disk Space Across the Cluster.....	284
Replacing Failed Disks.....	285
Creating and Configuring Storage Locations.....	286
Reclaiming Disk Space.....	290
Managing Workloads.....	292
The Resource Manager.....	293
Resource Manager Impact on Query Execution.....	293
Resource Pool Architecture.....	294
Monitoring Resource Pools and Resource Usage by Queries.....	297
User Profiles.....	299
Best Practices for Workload Management.....	301
Managing System Resource Usage.....	312
Load Balancing.....	316
Configuring Vertica Nodes.....	317
Configuring the Directors.....	320
Connecting to the Virtual IP (VIP).....	323
Monitoring Which Nodes Are Connected.....	323
Determining Where Connections Are Going.....	324
Virtual IP Connection Problems.....	326
Troubleshooting Keepalived Issues.....	327

Using the Administration Tools 329

Using the Graphical User Interface.....	330
K-Safety Support in Administration Tools.....	332
Notes for Remote Terminal Users.....	333
Using the Online Help.....	333
Password Authentication.....	334
Distributing Changes Made to the Administration Tools Metadata.....	334
Administration Tools Reference.....	336
Viewing Database Cluster State.....	336
Connecting to the Database.....	337
Starting a Database.....	337
Stopping a Database.....	338

Restarting Vertica on Host	341
Configuration Menu Item	342
Advanced Menu Options	346
Writing Administration Tools Scripts.....	351

Using vsql **358**

Connecting From the Administration Tools	359
Connecting from the Command Line.....	360
Command Line Options.....	360
Connecting From a Non-Cluster Host	365
Meta-Commands.....	366
! [COMMAND]	366
?	366
a	368
b	368
c (or \connect) [dbname [username]]	368
C [STRING]	368
cd [DIR].....	368
The \d [PATTERN] meta-commands	368
e \edit [FILE]	375
echo [STRING]	376
f [string].....	376
g.....	376
H.....	376
h \help [command].....	376
i FILE	377
l.....	377
locale	377
o.....	378
p.....	378
password [USER]	378
pset NAME [VALUE].....	379
q.....	380
qecho [STRING]	380
r.....	380
s [FILE]	381
set [NAME [VALUE [...]]].....	381
t.....	381
T [STRING].....	381
timing.....	382
unset [NAME]	382
w [FILE]	382
x.....	382
z	382
Variables.....	382
AUTOCOMMIT.....	383
DBNAME.....	384
ECHO	384
ECHO_HIDDEN	384
ENCODING	385
HISTCONTROL	385

HISTSIZE.....	385
HOST.....	385
IGNOREEOF	385
ON_ERROR_STOP	385
PORT.....	385
PROMPT1 PROMPT2 PROMPT3	385
QUIET	386
SINGLELINE.....	386
SINGLESTEP.....	386
USER.....	386
VERBOSITY.....	386
VSQL_HOME.....	386
Prompting	387
Command Line Editing.....	388
Environment	389
Locales.....	389
Files	390
Exporting Data Using vsql.....	390
Copying Data Using vsql.....	392
Notes for Windows Users	393
Output Formatting Examples.....	393

Appendix: Locales **395**

Locale Specification	397
Long Form.....	397
Short Form.....	403
Supported Locales	404
Locale Restrictions and Workarounds	416

Index **419**

Copyright Notice **429**

Technical Support

To submit problem reports, questions, comments, and suggestions, use the Technical Support page on the Vertica Systems, Inc., Web site.

Note: You must be a registered user in order to access the support page.

- 1 Go to <http://www.vertica.com/support> (*http://www.vertica.com/support*).
- 2 Click **My Support**.

You can also email verticahelp@vertica.com.

Before you report a problem, run the Diagnostics Utility described in the Troubleshooting Guide and attach the resulting `.zip` file to your ticket.

About the Documentation

This section describes how to access and print Vertica documentation. It also includes *suggested reading paths* (page 4).

Where to Find the Vertica Documentation

You can read or download the Vertica documentation for the current release of Vertica® Analytic Database from the *Product Documentation Page* http://www.vertica.com/v-zone/product_documentation. You must be a registered user to access this page.

The documentation is available as a compressed tarball (.tar) or a zip archive (.zip) file. When you extract the file on the database server system or locally on the client, contents are placed in a /vertica41_doc/ directory.

Note: The documentation on the Vertica Systems, Inc., Web site is updated each time a new release is issued. If you are using an older version of the software, refer to the documentation on your database server or client systems.

See Installing Vertica Documentation in the Installation Guide.

Reading the Online Documentation

Reading the HTML documentation files

The Vertica documentation files are provided in HTML browser format for platform independence. The HTML files require only a browser that displays frames properly with JavaScript enabled. The HTML files do not require a Web (HTTP) server.

The Vertica documentation is supported on the following browsers:

- Mozilla FireFox
- Internet Explorer
- Apple Safari
- Opera
- Google Chrome (server-side installations only)

The instructions that follow assume you have installed the documentation on a client or server machine.

Mozilla Firefox

- 1 Open a browser window.
- 2 Choose one of the following methods to access the documentation:
 - Select **File > Open File**, navigate to `..\HTML-WEBHELP\index.htm`, and click **Open**.
 - OR drag and drop `index.htm` into a browser window.

- OR press **CTRL+O**, navigate to `index.htm`, and click **Open**.

Internet Explorer

Use one of the following methods:

- 1 Open a browser window.
- 2 Choose one of the following methods to access the documentation:
 - Select **File > Open > Browse**, navigate to `..\HTML-WEBHELP\index.htm`, click **Open**, and click **OK**.
 - OR drag and drop `index.htm` into the browser window.
 - OR press **CTRL+O**, Browse to the file, click **Open**, and click **OK**.

Note: If a message warns you that Internet Explorer has restricted the web page from running scripts or ActiveX controls, right-click anywhere within the message and select **Allow Blocked Content**.

Apple Safari

- 1 Open a browser window.
- 2 Choose one of the following methods to access the documentation:
 - Select **File > Open File**, navigate to `..\HTML-WEBHELP\index.htm`, and click **Open**.
 - OR drag and drop `index.htm` into the browser window.
 - OR press **CTRL+O**, navigate to `index.htm`, and click **Open**.

Opera

- 1 Open a browser window.
- 2 Position your cursor in the title bar and right click > **Customize > Appearance**, click the **Toolbar** tab and select **Main Bar**.
- 3 Choose one of the following methods to access the documentation:
 - Open a browser window and click **Open**, navigate to `..\HTML-WEBHELP\index.htm`, and click **Open**.
 - OR drag and drop `index.htm` into the browser window.
 - OR press **CTRL+O**, navigate to `index.htm`, and click **Open**.

Google Chrome

Google does not support access to client-side installations of the documentation. You'll have to point to the documentation installed on a server system.

- 1 Open a browser window.
- 2 Choose one of the following methods to access the documentation:
 - In the address bar, type the location of the `index.htm` file on the server. For example: file:///<servername>//vertica41_doc//HTML/Master/index.htm
 - OR drag and drop `index.htm` into the browser window.
 - OR press **CTRL+O**, navigate to `index.htm`, and click **Open**.

Notes

The `.tar` or `.zip` file you download contains a complete documentation set.

The documentation page of the **Downloads Web site** http://www.vertica.com/v-zone/download_vertica is updated as new versions of Vertica are released. When the version you download is no longer the most recent release, refer only to the documentation included in your RPM.

The Vertica documentation contains links to Web sites of other companies or organizations that Vertica does not own or control. If you find broken links, please let us know.

Report any script, image rendering, or text formatting problems to **Technical Support** (on page 1).

Printing Full Books

Vertica also publishes books as Adobe Acrobat™ PDF. The books are designed to be printed on standard 8½ x 11 paper using full duplex (two-sided) printing.

Note: Vertica manuals are topic driven and not meant to be read in a linear fashion. Therefore, the PDFs do not resemble the format of typical books. Each topic starts a new page, so some of the pages are very short, and there are blank pages between each topic.

Open and print the PDF documents using Acrobat Acrobat Reader. You can download the latest version of the free Reader from the **Adobe Web site** (<http://www.adobe.com/products/acrobat/readstep2.html>).

The following list provides links to the PDFs.

- Release Notes
- Concepts Guide
- Installation Guide
- Getting Started Guide
- Administrator's Guide
- Programmer's Guide
- SQL Reference Manual
- Troubleshooting Guide

Suggested Reading Paths

This section provides a suggested reading path for various users. Vertica recommends that you read the manuals listed under All Users first.

All Users

- Release Notes — Release-specific information, including new features and behavior changes to the product and documentation
- Concepts Guide — Basic concepts critical to understanding Vertica

- Getting Started Guide — A tutorial that takes you through the process of configuring a Vertica database and running example queries
- Troubleshooting Guide — General troubleshooting information

System Administrators

- Installation Guide — Platform configuration and software installation
- Release Notes — Release-specific information, including new features and behavior changes to the product and documentation

Database Administrators

- Installation Guide — Platform configuration and software installation
- Administrator's Guide — Database configuration, loading, security, and maintenance

Application Developers

- Programmer's Guide — Connecting to a database, queries, transactions, and so on
- SQL Reference Manual — SQL and Vertica-specific language information

Where to Find Additional Information

Visit the *Vertica Systems, Inc. Web site* (<http://www.vertica.com>) to keep up to date with:

- Downloads
- Frequently Asked Questions (FAQs)
- Discussion forums
- News, tips, and techniques
- Training

Typographical Conventions

The following are the typographical and syntax conventions used in the Vertica documentation.

Typographical Convention	Description
Bold	Indicates areas of emphasis, such as a special menu command.
Button	Indicates the word is a button on the window or screen.
Code	SQL and program code displays in a monospaced (fixed-width) font.
Database objects	Names of database objects, such as tables, are shown in san-serif type.
<i>Emphasis</i>	Indicates emphasis and the titles of other documents or system files.
monospace	Indicates literal interactive or programmatic input/output.
<i>monospace italics</i>	Indicates user-supplied information in interactive or programmatic input/output.
UPPERCASE	Indicates the name of a SQL command or keyword. SQL keywords are case insensitive; <code>SELECT</code> is the same as <code>Select</code> , which is the same as <code>select</code> .
User input	Text entered by the user is shown in bold san serif type.
↵	indicates the Return/Enter key; implicit on all user input that includes text
Right-angle bracket >	Indicates a flow of events, usually from a drop-down menu.
Click	Indicates that the reader clicks options, such as menu command buttons, radio buttons, and mouse selections; for example, "Click OK to proceed."
Press	Indicates that the reader perform some action on the keyboard; for example, "Press Enter."
Syntax Convention	Description
Text without brackets/braces	Indicates content you type as shown.
< <i>Text inside angle brackets</i> >	Placeholder for which you must supply a value. The variable is usually shown in italics. See Placeholders below.
[<i>Text inside brackets</i>]	Indicates optional items; for example, <code>CREATE TABLE [schema_name.]table_name</code> The brackets indicate that the <code>schema_name</code> is optional. Do not type the square brackets.
{ <i>Text inside braces</i> }	Indicates a set of options from which you choose one; for example: <code>QUOTES { ON OFF }</code> indicates that exactly one of ON or OFF must

	be provided. You do not type the braces: QUOTES ON
Backslash \	Continuation character used to indicate text that is too long to fit on a single line.
Ellipses . . .	Indicate a repetition of the previous parameter. For example, <code>option[, . . .]</code> means that you can enter multiple, comma-separated options. Note: Showing an ellipses in code examples might also mean that part of the text has been omitted for readability, such as in multi-row result sets.
Indentation	Is an attempt to maximize readability; SQL is a free-form language.
<i>Placeholders</i>	Items that must be replaced with appropriate identifiers or expressions are shown in italics.
Vertical bar	Is a separator for mutually exclusive items. For example: <code>[ASC DESC]</code> Choose one or neither. You do not type the square brackets.

Preface

This document describes how to set up and maintain a Vertica database. You might find that these tasks are extremely simple in Vertica compared to other database management systems.

Prerequisites

This document assumes that you have already:

- Become familiar with the concepts discussed in the Concepts Guide.
- Performed the procedures described in the Installation Guide.
 - Constructed a hardware platform.
 - Installed Linux.
 - Installed Vertica (configured a cluster of hosts).
- Followed the Tutorial in the Getting Started Guide to experiment with setting up an example database.

Audience

This document is intended for anyone with responsibility for configuring, loading, securing, and maintaining a Vertica database.

For More Information

There is a great deal of published literature available about dimensional modeling and data warehouse design in general.

Administration Overview

This document describes the functions performed by a Vertica database administrator (DBA). Perform these tasks using only the dedicated database administrator account that was created when you installed Vertica. The examples in this documentation set assume that the administrative account name is dbadmin.

- To perform certain cluster configuration and administration tasks, the DBA (users of the administrative account) must be able to supply the root password for those hosts. If this requirement conflicts with your organization's security policies, these functions must be performed by your IT staff.
- If you perform administrative functions using a different account than the one provided during installation, Vertica encounters file ownership problems.
- If you share the administrative account password, make sure that only one user runs the Administration Tools at any time. Otherwise, automatic configuration propagation does not work correctly.
- The Administration Tools require that the calling user's shell be `/bin/bash`. Other shells give unexpected results and are not supported.

Configuring the Database

This section provides information about:

- The Vertica® Analytic Database **Configuration procedure** (page 11)
- **Configuration parameters** (page 25)
- Designing a **logical schema** (page 37)
- Creating the **physical schema** (page 77)
- Implementing **security** (page 19)
- Implementing **locales** (page 19) for international data sets

Note: Before you begin this section, Vertica strongly recommends that you follow the Tutorial in the Getting Started Guide to quickly familiarize yourself with creating and configuring a fully-functioning n example database.

Configuration Procedure

This section describes the tasks required to set up a Vertica database. It assumes that you have obtained a valid license key file, installed the Vertica rpm package, and run the installation script as described in the Installation Guide.

You'll complete the configuration procedure using the:

- Administration Tools

Note: If you are unfamiliar with Dialog-based user interfaces, read **Using the Graphical User Interface** (page 330) before you begin. See also the **Administration Tools Reference** (page 336) for details.

- vsql interactive interface
- The Database Designer, described fully in **Creating a Physical Design** (page 77)

IMPORTANT NOTES

Follow the configuration procedure in the order presented in this book.

Vertica strongly recommends that you follow the Tutorial in the Getting Started Guide to experiment with creating and configuring a database before you begin this section.

The generic configuration procedure described here can be used several times during the development process and modified each time to fit changing goals. You can omit steps such as preparing actual data files and sample queries, and run the Database Designer without optimizing for queries. For example, you can create, load, and query a database several times for development and testing purposes, then one final time to create and load the production database.

Prepare Disk Storage Locations

Preparing the disk storage locations for Vertica involves choosing the disk directory paths that contain the catalog and data files (physical schema) for each host in the cluster. These are referred to as catalog path and data path respectively.

You can use a single directory to contain both the catalog and data files or you can use separate directories. If you use separate directories, they can be on different drives. The directories can be either on drives local to the host or can be on a shared storage, such as an external disk enclosure or a SAN.

Notes

- The topics in this section are intentionally included in both the Vertica Installation Guide and Administrator's Guide because the choice of disk storage locations for a database can be made at installation time, database configuration time, or later during the operation of the database
- The catalog and data directory pathnames must be identical on each host in the cluster, and the directories must be owned by the Database Administrator.
- The choice of disk storage locations for a database can be made at installation time, database configuration time, or later during the operation of the database.

Disk Space Requirements for Vertica

In addition to actual data stored in the database, disk space is required by a number of data reorganization operations in Vertica, such as mergeout and **managing nodes** (page 263) in the cluster. For best results, Vertica recommends that disk utilization per node be no more than sixty percent (60%) for a K-Safe=1 database to allow such operations to proceed.

In addition, disk space is temporarily required by certain query execution operators, such as hash joins and sorts, in the case when they have to spill to disk. Such operators might be encountered during queries, recovery, refreshing projections, and so on. The amount of disk space needed in this manner (known as temp space) depends on the nature of the queries, amount of data on the node and number of concurrent users on the system. By default, any unused space on the data disk can be used as temp space, however, it is possible and recommended to provision temp space separate from data disk space. See **Configuring Disk Usage to Optimize Performance** (page 13).

Specifying Disk Storage at Installation Time

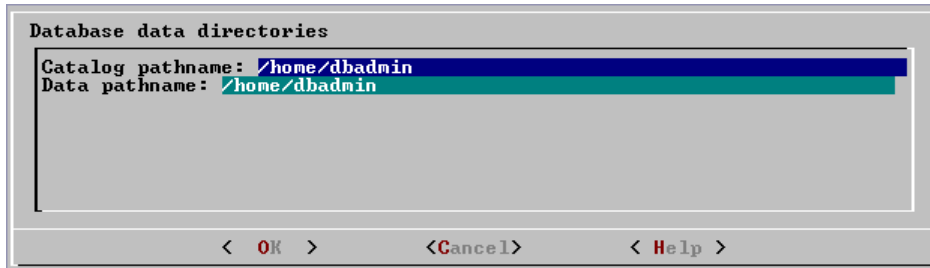
When you install Vertica, the `data_directory` parameter in the `install_vertica` script lets you specify a directory to contain database data and catalog files. The default is the Database Administrator's default home directory:

```
/home/dbadmin
```

There is no requirement that you use this directory. It is created for your convenience. However, if you choose a different location, make sure that the location exists on each host in the cluster and is owned by the Database Administrator before you create a database.

Specifying Disk Storage at Database Creation Time

When you invoke the **Create Database** (page 342) command in the Administration Tools, the following dialog allows you to specify the catalog and data locations. These locations must exist on each host in the cluster and must be owned by the Database Administrator.



When you click **OK**, Vertica automatically creates the following subdirectories:

```
catalog-pathname/database-name/node-name_catalog/
data-pathname/database-name/node-name_data/
```

For example, if you use the default value (the Database Administrator's home directory) of `/home/dbadmin` for the Stock Exchange example database, the catalog and data directories would be as follows, on each node in the cluster:

```
/home/dbadmin/Stock_Schema/stock_schema_node1_host01_catalog
/home/dbadmin/Stock_Schema/stock_schema_node1_host01_data
```

Notes

- Catalog and data path names must contain only alphanumeric characters and cannot have leading space characters. Failure to comply with these restrictions will result in database creation failure.
- Vertica refuses to overwrite a directory if it appears to be in used by another database. Therefore, if you created a database for evaluation purposes, dropped the database, and want to reuse the database name, make sure that the disk storage location previously used has been completely cleaned up.

Configuring Disk Usage to Optimize Performance

Once you have created your initial storage location, you can add additional storage locations to the database later. Not only does this provide additional space, it lets you control disk usage and increase I/O performance by isolating files that have different I/O or access patterns. For example, consider:

- Isolating execution engine temporary files from data files by creating a separate storage location for temp space.
- Creating a tiered disk architecture in which projections are stored on different disks based on predicted or measured access patterns.

See **Creating and Configuring Storage Locations** (page 286) for details.

Using Shared Storage with Vertica

If using shared SAN storage, you will need to extra care to ensure that there is no contention among the nodes for disk space or bandwidth.

- Each host must have its own catalog and data locations. In other words, hosts cannot share catalog or data locations.
- Configure the storage so that there is enough I/O bandwidth for each nodes to access the storage independently.
- For latest information on SAN configuration and recommended hardware configurations, visit the **Online Training** <http://www.vertica.com/v-zone/training> section of the Vertica Systems, Inc. Web site.

Contact **Technical Support** (on page 1) if you need a consultation.

Prepare the Logical Schema Script

Designing a logical schema for a Vertica database is no different from designing one for any other SQL database. See **Designing a Logical Schema** (page 37) for details.

To create your logical schema, you must prepare a SQL script (plain text file, typically with an extension of .sql) that:

- 1 Creates additional schemas (as necessary). See **Using Multiple Schemas** (page 40).
- 2 Creates the tables and column constraints in your database using the CREATE TABLE command.
- 3 Defines the necessary table constraints using the ALTER TABLE command.
- 4 Defines any views on the table using the CREATE VIEW command.

You can generate a script file using:

- A schema designer application.
- A schema extracted from an existing database.
- A text editor.
- One of the example database `example-name_define_schema.sql` scripts as a template. (See the example database directories in `/opt/vertica/examples`.)

In your script file, make sure that:

- Each statement ends with a semicolon.
- You use only data types supported by Vertica. See the SQL Reference Manual for details.

Once you have created a database, you can test your schema script by executing it as described in **Create the Logical Schema** (page 17). If you encounter errors, drop all tables, correct the errors, and run the script again.

Prepare Data Files

Prepare two sets of data files:

- Test data files. Use test files to test the database after the partial data load. If possible, use part of the actual data files to prepare the test data files.

- Actual data files. Once the database has been tested and optimized, use your data files for your initial **bulk load** (page 144).

How to Name Data Files

Name each data file to match the corresponding table in the logical schema. Case does not matter.

Use the extension `.tbl` or whatever you prefer. For example, if a table is named `Stock_Dimension`, name the corresponding data file `stock_dimension.tbl`. When using multiple data files, append `_nnn` (where `nnn` is a positive integer in the range 001 to 999) to the file name. For example, `stock_dimension.tbl_001`, `stock_dimension.tbl_002`, and so on.

Prepare Load Scripts

Note: You can postpone this step if your goal is to test a logical schema design for validity.

Prepare SQL scripts that use the `COPY...DIRECT` statement via `vsq` or the `LCOPY...DIRECT` statement through ODBC to load data directly into physical storage.

You need scripts that load the:

- Large tables
- Small tables

Vertica Systems, Inc. recommends that you load large tables using multiple files. To test the load process, use files of 10GB to 50GB in size. This size provides several advantages:

- You can use one of the data files as a sample data file for the Database Designer.
- You can load just enough data to **perform a partial data load** (page 17) before you load the remainder.
- If a single load fails and rolls back, you do not lose an excessive amount of time.
- Once the load process is tested, for multi-terabyte tables, break up the full load in file sizes of 250-500GB.

See the **Loading and Modifying Data** (page 144) and the following additional topics for details:

- **Bulk Loading** (page 144)
- **Using Load Scripts** (page 148)
- **Using Parallel Load Streams** (page 149)
- **Loading Data into Pre-join Projections** (page 153)
- **Enforcing Constraints** (page 55)
- **About Load Errors** (page 157)

Tip: You can use the load scripts included in the example databases in the Getting Started Guide as templates.

Create an Optional Sample Query Script

The purpose of a sample query script is to test your schema and load scripts for errors.

Include a sample of queries your users are likely to run against the database. If you don't have any real queries, just write simple SQL that collects counts on each of your tables. Alternatively, you can skip this step.

Create an Empty Database

- 1 Run the Administration Tools from your Administration Host as follows:

```
$ /opt/vertica/bin/admintools
```

If you are using a remote terminal application such as PuTTY or a Cygwin bash shell, see **Notes for Remote Terminal Users** (page 333).

- 2 If this is the first time you have run the Administration Tools, accept the license agreement and specify the location of your license file. See **Managing Your License Key** (page 140) for more information.
- 3 On the Main Menu, click **Configuration Menu**, and click **OK**.
- 4 On the Configuration Menu, click **Create Database**, and click **OK**.
- 5 Enter the name of the database and an optional comment, and click **OK**.
- 6 Establish the superuser password for your database.
 - To provide a password enter the password and click **OK**. Confirm the password by entering it again, and then click **OK**.
 - If you don't want to provide the password, leave it blank and click **OK**. If you don't set a password, Vertica prompts you to verify that you truly do not want to establish a superuser password for this database. Click **Yes** to create the database without a password or **No** to establish the password.

Caution: If you do not enter a password at this point, the superuser password is set to empty. Unless the database is for evaluation or academic purposes, Vertica Systems, Inc. strongly recommends that you enter a superuser password.

- 7 Select the hosts to include in the database from the list of hosts specified when Vertica was installed (`install_vertica -s`), and click **OK**.
- 8 Specify the directories in which to store the data and catalog files, and click **OK**.
Catalog and data pathnames must contain only alphanumeric characters and cannot have leading spaces. Failure to comply with these restrictions could result in database creation failure.

For example:

Catalog pathname: /home/dbadmin

Data Pathname: /home/dbadmin

- 9 Review the **Current Database Definition** screen to verify that it represents the database you want to create, and then click **Yes** to proceed or **No** to modify the database definition.

If you clicked **Yes**, Vertica creates the database you defined and then displays a message to indicate that the database was successfully created.

Note: For databases created with 3 or more nodes, Vertica automatically sets K-safety to 1 to ensure that the database is fault tolerant in case a node fails. For more information, see the **Failure Recovery** (page 235) topic in Administrator's Guide and `SELECT MARK_DESIGN_KSAFE` in the SQL Reference Manual.

- 10 Click **OK** to acknowledge the message.

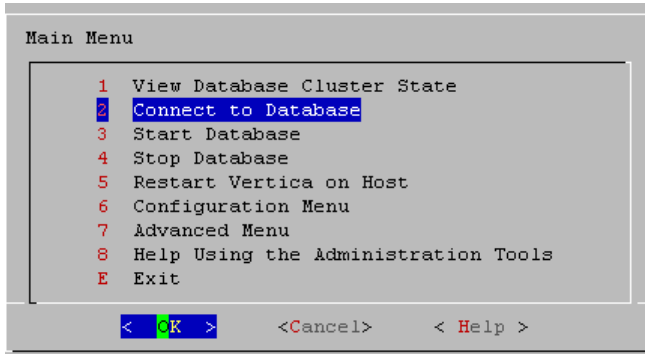
If you receive an error message, see **Startup Problems** (page 241).

Create the Logical Schema

1 Connect to the database.

In the Administration Tools Main Menu, click **Connect to Database** and click **OK**.

See Connecting to the Database for details.



The vsql welcome script appears:

```
Welcome to the vsql, Vertica_Database v4.1.x interactive terminal.
Type:  \h for help with SQL commands
        \? for help with vsql commands
        \g or terminate with semicolon to execute query
        \q to quit
vmartdb=>
```

2 Run the logical schema script

Using the ***v* meta-command** (see "***i* FILE**" on page 377) in vsql to run the SQL ***logical schema script*** (page 14) that you prepared earlier.

3 Disconnect from the database

Use the ***lq* meta-command** (see "***q***" on page 380) in vsql to return to the Administration Tools.

Perform a Partial Data Load

Vertica recommends that for large tables, you perform a partial data load and then test your database before completing a full data load. This load should load a representative amount of data.

1 Load the small tables.

Load the small table data files using the SQL ***load scripts*** (page 15) and ***data files*** (page 14) you prepared earlier.

2 Partially load the large tables.

Load 10GB to 50GB of table data for each table using the SQL ***load scripts*** (page 15) and ***data files*** (page 14) that you prepared earlier.

Vertica automatically creates a superprojection for each table that is loaded. This ensures that all SQL queries can be answered, and it is suitable for testing the database. Once you've tested the database, you can use Database Designer to optimize it further as needed. (For more information about projections, see Physical Schema in the Concepts Guide.)

Test the Database

Test the database to verify that it is running as expected.

Check queries for syntax errors and execution times.

- 1 Use the vsql ***timing meta-command*** (see "***timing***" on page 382) to enable the display of query execution time in milliseconds.
- 2 Execute the SQL sample query script that you prepared earlier.
- 3 Execute several ad hoc queries.

Optimize Query Performance

Optimizing the database consists of optimizing for compression and tuning for queries. (See ***Creating a Physical Design*** (page 77).)

To optimize the database, use the Database Designer to create and deploy a design for optimizing the database.

See the Tutorial for an example of using the Database Designer.

Complete the Data Load

1 Monitor system resource usage

Continue to run the top, free, and df utilities and watch them while your load scripts are running (as described in ***Monitoring Linux Resource Usage*** (page 215)). You can do this on any or all nodes in the cluster. Make sure that the system is not swapping excessively (watch kswapd in top) or running out of swap space (watch for a large amount of used swap space in free).

2 Complete the large table loads

Run the remainder of the large table load scripts.

Test the Optimized Database

In order to test your optimized design, you can check query execution times:

- 1 Use the vsql `\timing` (see "***timing***" on page 382) meta-command to enable the display of query execution time in milliseconds.

Execute a SQL sample query script to test your schema and load scripts for errors.

Note: Include a sample of queries your users are likely to run against the database. If you don't have any real queries, just write simple SQL that collects counts on each of your tables. Alternatively, you can skip this step.

2 Execute several ad hoc queries

1. Run Administration Tools and select Connect to Database.
2. Use the ***v*** meta-command (see "***i FILE***" on page 377) to execute the query script; for example:

```
vmartdb=> \i vmart_query_01.sql
```


Once the database has been optimized, it should run queries efficiently. However, you might discover additional queries that you want to optimize. If this is the case, modify and update the design. See *Modifying Designs* and ***Creating a Query-specific Design Using the Database Designer*** (page 87) in the Administrator's Guide.

Set Up Incremental (Trickle) Loads

Once you have a working database, you can use trickle loading to load new data while concurrent queries are running.

Trickle load is accomplished by using the COPY command (without the DIRECT keyword) to load 10k to 100k rows per transaction into the WOS. This allows Vertica to batch multiple loads when it writes data to disk. The COPY command loads data into the WOS and automatically overflow to the ROS if the WOS is full.

See the following sections in this guide for details:

- ***Trickle Loading*** (page 148)
- Loading data through ODBC
- Loading data through JDBC

See Also

COPY in the SQL Reference Manual

Implement Security

Once you have created the database, implement security before allowing users access to it. See ***Implementing Security*** (page 112).

Implement Locales for International Data Sets

The locale is a parameter that defines the user's language, country, and any special variant preferences, such as collation. Vertica uses the locale to determine the behavior of various string functions as well for collation for various SQL commands that require ordering and comparison; for example, GROUP BY, ORDER BY, joins, the analytic ORDER BY clause, and so forth.

By default, the locale for the database is `en_US@collation=binary` (English US). You can establish a new default locale that is used for all sessions on the database, as well as override individual sessions with different locales. Additionally the locale can be set through ODBC, JDBC, and ADO.net.

The locale used by the database session is not derived from the operating system; for instance LANG variable. Vertica uses the ICU library for locale support; thus, you must specify the locale using the ICU Locale syntax.

Notes

- Even though ICU locales can normally specify collation, currency, and calendar preferences, Vertica supports only the collation component. The `SET DATESTYLE TO ...` command provides some aspects of the calendar; only dollars are supported for currency. Any keywords not relating to collation are rejected.

- Projections are always collated using the `en_US@collation=binary` collation regardless of the session collation. Any locale-specific collation is applied at query time.
- The maximum length parameter for VARCHAR and CHAR data type refers to the number of octets (bytes) that can be stored in that field and not number of characters. When using multi-byte UTF-8 characters, size fields to accommodate from 1 to 4 bytes per character, depending on the data.

See Also

Supported Locales (page 404) in the **Appendix** (page 395)

SET in the SQL Reference Manual

ICU User Guide <http://userguide.icu-project.org/locale> (external link)

Specify the Default Locale for the Database

The default locale configuration parameter sets the initial locale for every database session once the database has been restarted. Sessions may override this value.

To set the local for the database, use the configuration parameter as follows:

```
SELECT SET_CONFIG_PARAMETER('DefaultSessionLocale' ,
'<ICU-locale-identifier>');
```

For example:

```
mydb=> SELECT SET_CONFIG_PARAMETER('DefaultSessionLocale', 'en_GB');
        SET_CONFIG_PARAMETER
-----
Parameter set successfully
(1 row)
```

Override the Default Locale for a Session

To override the default locale for a specific session, use one of the following commands:

- The vsql command **Vocale** (see "**locale**" on page 377) `<ICU-locale-identifier>`.

For example:

```
\locale en_GB
INFO:  Locale: 'en_GB'
INFO:    English (United Kingdom)
INFO:  Short form: 'LEN'
```

- The statement `SET LOCALE TO <ICU-locale-identifier>`.

```
SET LOCALE TO en_GB;
SET LOCALE TO en_GB;
INFO:  Locale: 'en_GB'
INFO:    English (United Kingdom)
INFO:  Short form: 'LEN'
```

You can also use the **short form** (page 403) of a locale in either of these commands:

```
SET LOCALE TO LEN;
INFO:  Locale: 'en'
INFO:    English
```

```
INFO: Short form: 'LEN'
```

```
\locale LEN
```

```
INFO: Locale: 'en'
```

```
INFO: English
```

```
INFO: Short form: 'LEN'
```

You can use these commands to override the locale as many times as needed within a session. The session locale setting applies to any subsequent commands issued in the session.

See Also

SET in the SQL Reference Manual

Best Practices for Working with Locales

It is important to understand the distinction between the locale settings on the database server and locale settings at the client application level. The server locale settings impact only the collation behavior for server-side query processing. The client application is responsible for ensuring that the correct locale is set in order to display the characters correctly. Below are the best practices recommended by Vertica to ensure predictable results:

Server locale

Server session locale should be set using the set as described in ***Specify the Default Locale for the Database*** (page 20). If using different locales in different session, set the server locale at the start of each session from your client.

vsq client

- If there is no default session locale at database level, the server locale for the session should be set to the desired locale, as described in ***Override the Default Locale for a Session*** (page 20).
- The locale setting in the terminal emulator where vsq client is run should be set to be equivalent to session locale setting on server side (ICU locale) so data is collated correctly on the server and displayed correctly on the client.
- All input data for vsq should be in UTF-8 and all output data is encoded in UTF-8
- Non UTF-8 encodings and associated locale values should not be used because they are not supported.
- Refer to the documentation of your terminal emulator for instructions on setting locale and encoding.

ODBC clients

- ODBC applications can be either in ANSI or Unicode mode. If Unicode, the encoding used by ODBC is UCS-2. If the user application is ANSI, the data must be in single-byte ASCII, which is compatible with UTF-8 used on the database server. The ODBC driver converts UCS-2 to UTF-8 when passing to the Vertica server and converts data sent by the Vertica server from UTF-8 to UCS-2.
- If the user application is not already in UCS-2, the application is responsible for converting the input data to UCS-2, or unexpected results could occur. For example:

- On non-UCS-2 data passed to ODBC APIs, when it is interpreted as UCS-2, it could result in an invalid UCS-2 symbol being passed to the APIs, resulting in errors.
- The symbol provided in the alternate encoding could be a valid UCS-2 symbol; in this case, incorrect data is inserted into the database.
- If there is no default session locale at database level, ODBC applications should set the desired server session locale using `SQLSetConnectAttr` (if different from database wide setting) in order to get expected collation and string functions behavior on the server.

JDBC and ADO.NET clients

- JDBC and ADO.NET applications use a UTF-16 character set encoding and are responsible for converting any non-UTF-16 encoded data to UTF-16. The same cautions apply as for ODBC if this encoding is violated.
- The JDBC and ADO.NET drivers convert UTF-16 data to UTF-8 when passing to the Vertica server and convert data sent by Vertica server from UTF-8 to UTF-16.
- If there is no default session locale at the database level, JDBC and ADO.NET applications should set the correct server session locale by executing the `SET LOCALE TO` command in order to get expected collation and string functions behavior on the server. See the `SET` command in the SQL Reference Manual.

Notes and Restrictions

Session related:

- The locale setting is session scoped and applies to queries only (no DML/DDDL) run in that session. You cannot specify a locale for an individual query.
- The default locale for new sessions can be set using a configuration parameter

Query related:

The following restrictions apply when queries are run with locale other than the default `en_US@collation=binary`:

- Multicolumn NOT IN subqueries are not supported when one or more of the left-side NOT IN columns is of CHAR or VARCHAR data type. For example:

```
=> CREATE TABLE test (x VARCHAR(10), y INT);
=> SELECT ... FROM test WHERE (x,y) NOT IN (SELECT ...);
ERROR: Multi-expression NOT IN subquery is not supported because a left hand expression could be NULL
```

Note: An error is reported even if columns `test.x` and `test.y` have a "NOT NULL" constraint.

- Correlated HAVING clause subqueries are not supported if the outer query contains a GROUP BY on a CHAR or a VARCHAR column. In the following example, the GROUP BY x in the outer query causes the error:

```
=> DROP TABLE test CASCADE;
=> CREATE TABLE test (x VARCHAR(10));
=> SELECT COUNT(*) FROM test t GROUP BY x HAVING x
      IN (SELECT x FROM test WHERE t.x||'a' = test.x||'a' );
ERROR: subquery uses ungrouped column "t.x" from outer query
```
- Subqueries that use analytic functions in the HAVING clause are not supported. For example:

```
=> DROP TABLE test CASCADE;
=> CREATE TABLE test (x VARCHAR(10));
=> SELECT MAX(x)OVER(PARTITION BY 1 ORDER BY 1)
      FROM test GROUP BY x HAVING x IN (
      SELECT MAX(x) FROM test);
ERROR: Analytics query with having clause expression that involves
aggregates
and subquery is not supported
```
- The operators LIKE/ILIKE do not currently respect UTF-8 character boundaries. Therefore, expressions such as 'SS' LIKE 'ß' and 'SS' ILIKE 'ß' always return false even in locales where 'SS' = 'ß' return true.

DML/DDDL related:

- SQL identifiers (such as table names, column names, and so on) are restricted to ASCII characters. For example, the following CREATE TABLE statement fails because it uses the non-ASCII ß in the table name:

```
=> CREATE TABLE straÙe(x int, y int);
ERROR: Non-ASCII characters are not supported in names
```

- Projection sort orders are made according to the default `en_US@collation=binary` collation. Thus, regardless of the session setting, issuing the following command creates a projection sorted by `coll` according to the binary collation:

```
=> CREATE PROJECTION p1 AS SELECT * FROM table1 ORDER BY coll;
```

Note that in such cases, `straße` and `strasse` would not be near each other on disk.

Sorting by binary collation also means that sort optimizations do not work in locales other than binary. Vertica returns the following warning if you create tables or projections in a non-binary locale:

```
WARNING: Projections are always created and persisted in the default
Vertica locale. The current locale is de_DE
```
- When creating pre-join projections, the projection definition query does not respect the locale or collation setting. This means that when you insert data into the fact table of a pre-join projection, referential integrity checks are not locale or collation aware.

For example:

```
\locale LDE_S1      -- German
=> CREATE TABLE dim (coll varchar(20) primary key);
=> CREATE TABLE fact (coll varchar(20) references dim(coll));
=> CREATE PROJECTION pj AS SELECT * FROM fact JOIN dim
    ON fact.coll = dim.coll UNSEGMENTED ALL NODES;
=> INSERT INTO dim VALUES('ß');
=> COMMIT;
```

The following INSERT statement fails with a "nonexistent FK" error even though 'ß' is in the dim table, and in the German locale 'SS' and 'ß' refer to the same character.

```
=> INSERT INTO fact VALUES('SS');
    ERROR: Nonexistent foreign key value detected in FK-PK join (fact x
    dim)
    using subquery and dim_node0001; value SS
=> => ROLLBACK;
=> DROP TABLE dim, fact CASCADE;
```
- When the locale is non-binary, the collation function is used to transform the input to a binary string which sorts in the proper order.

This transformation increases the number of bytes required for the input according to this formula:

$$\text{result_column_width} = \text{input_octet_width} * \text{CollationExpansion} + 4$$

CollationExpansion defaults to 5 and should be changed only under the supervision of Vertica **Technical Support** (on page 1).
- CHAR fields are displayed as fixed length, including any trailing spaces. When CHAR fields are processed internally, they are first stripped of trailing spaces. For VARCHAR fields, trailing spaces are usually treated as significant characters; however, trailing spaces are ignored when sorting or comparing either type of character string field using a non-BINARY locale.

Change Transaction Isolation Levels

By default, Vertica uses the `READ COMMITTED` isolation level for every session. If you prefer, you can change the default isolation level for the database or for a specific session.

To change the isolation level for a specific session, use the `SET SESSION CHARACTERISTICS` command.

To change the isolation level for the database, use the `TransactionIsolationLevel` configuration parameter. Once modified, Vertica uses the new transaction level for every new session.

The following examples set the default isolation for the database to `SERIALIZABLE` and then back to `READ COMMITTED`:

```
=> SELECT SET_CONFIG_PARAMETER('TransactionIsolationLevel','SERIALIZABLE');
=> SELECT SET_CONFIG_PARAMETER('TransactionIsolationLevel','READ COMMITTED');
```

Notes

- The default isolation value for databases created prior to 4.0 was `SERIALIZABLE`. See [Read Committed and Simplification of Epoch Management in Release Notes for 4.0](#)
- The change to isolation level only applies to future sessions. Existing sessions and their transactions continue to use the original isolation level.
- A transaction retains its isolation level until it completes, even if the session's transaction isolation level has changed mid-transaction. Vertica internal processes (such as the Tuple Mover and Refresh operations) and DDL operations are run at `SERIALIZABLE` isolation to ensure consistency.

See Also

Transactions in the Concepts Guide for an overview of how Vertica uses session-scoped isolation levels

Configuration Parameters (page 25) in the Administrator's Guide

Configuration Parameters

The following tables describe parameters for configuring Vertica.

IMPORTANT: Before you modify a parameter, review the entire documentation for the parameter to determine the context under which you can modify it.

See Also

`CONFIGURATION_PARAMETERS` in the SQL Reference Manual

`SET_CONFIG_PARAMETER` in the SQL Reference Manual

General Parameters

The following table describes the general parameters for configuring Vertica.

Parameters	Description	Default	Example
AnalyzeRowCountInterval	Automatically runs every 60 seconds to collect the number of rows in the projection and aggregates row counts calculated during loads. See Collecting Statistics (page 172).	60 seconds	<pre>SELECT SET_CONFIG_PARAMETER ('AnalyzeRowCountInterval', 3600);</pre>
CompressNetworkData	When enabled (set to value 1), Vertica will compress all of the data it sends over the network. This speeds up network traffic at the expense of added CPU load. You can enable this if you find that the network is throttling your database performance.	0	<pre>SELECT SET_CONFIG_PARAMETER ('CompressNetworkData',1);</pre>
MaxClientSessions	Determines the maximum number of client sessions that can be run on the database. Tip: Setting this parameter to 0 is useful for preventing new client sessions from being opened while you are shutting down the database. Be sure to restore the parameter to its original setting once you've restarted the database. See the section "Interrupting and Closing Sessions" in Managing Sessions (page 313).	50 (with 5 additional administrative logins)	<pre>SELECT SET_CONFIG_PARAMETER ('MaxClientSessions', 100);</pre>
TopKHeapMaxMem	Controls how much memory can be used for <code>TopK(Heap)</code> . If <code>K</code> tuples can fit into the space allocated by this parameter (default 80MB), the optimizer uses <code>TopK(Heap)</code> ; otherwise no <code>TopK</code> is used (the query is sorted). Note: Once the optimizer chooses <code>TopK(Heap)</code> , the Resource Manager can reject the plan if the <code>TopK</code> operator requires too much	80MB	<pre>SELECT SET_CONFIG_PARAMETER ('TopKHeapMaxMem', '70');</pre>

	memory. To prevent the query from being rejected, you can lower the parameter <code>TopKHeapMaxMem</code> , but be careful in changing the setting. Too low and no TopK used (you lose the optimization); too high and the query could get rejected. In most cases, the default setting of 80MB should work.		
TransactionIsolationLevel	Changes the isolation level for the database. Once modified, Vertica uses the new transaction level for every new session. Existing sessions and their transactions continue to use the original isolation level. See <i>Change Transaction Isolation Levels</i> (page 25).	READ COMMITTED	<pre>SELECT SET_CONFIG_PARAMETER ('TransactionIsolationLevel', 'SERIALIZABLE');</pre>
TransactionMode	Controls whether transactions are read/write or read-only. Read/write is the default. Existing sessions and their transactions continue to use the original isolation level.	READ WRITE	<pre>SELECT SET_CONFIG_PARAMETER ('TransactionMode', 'READ ONLY');</pre>

Tuple Mover Parameters

These parameters control how the Tuple Mover operates.

Parameters	Description	Default	Example
ActivePartitionCount	Sets the number of partitions that are to be loaded at the same time. By default, the Tuple Mover assumes that data is only inserted into the most recent partition of a partitioned table. If this is not the case, then set this parameter to the number of partitions that are receiving data at the same time. Note: this parameter's value is ignored if <code>EnableStrataBasedMrgOutPo</code>	1	<pre>SELECT SET_CONFIG_PARAMETER ('ActivePartitionCount', 2);</pre>

	<p>lity is disabled. See Table Partitioning.</p>		
EnableStrataBasedMrgOutPolicy	<p>When set to 1 (the default) enables Vertica 4.0 mergeout behavior. Set this parameter to disabled (value 0) if you want to revert back to 3.5 mergeout behavior.</p> <p>Note: This parameter is deprecated and will be removed in a future release. Avoid using it except under the guidance of Technical Support (on page 1).</p>	1	<pre>SELECT SET_CONFIG_PARAMETER ('EnableStrataBasedMrgOutPolicy', 0);</pre>
MaxMrgOutROSSizeMB	<p>Sets the largest size in MB that a mergeout job can make on a non-partitioned ROS. By setting this to a small value, you can prevent the Tuple Mover from trying to merge large ROS containers, which requires more time to process. Raise this value during periods of lower activity, so the Tuple Mover can consolidate the larger ROS containers.</p>	100	<pre>SELECT SET_CONFIG_PARAMETER ('MaxMrgOutROSSizeMB', 150);</pre>
MergeOutInterval	<p>The number of seconds the Tuple Mover waits between checks for new ROS files to merge out. If ROS containers are added frequently, this value might need to be decreased.</p>	600	<pre>SELECT SET_CONFIG_PARAMETER ('MergeOutInterval', 1200);</pre>
MergeOutPolicySizeList	<p>A list of the file sizes, in bytes, that a ROS file must reach before the Tuple Mover selects it for mergeout. This parameter has an effect only if <code>EnableStrataBasedMrgOutPolicy</code> is disabled.</p> <p>Note: This parameter is deprecated and will be removed in a future release. Avoid using it except under the guidance of Technical Support.</p>	10, 100, 1000, 10000, 100000, 1000000, 10000000, 100000000, 1000000000, 10000000000, 100000000000, 0, 50000000000, 0, 100000000000	
MoveOutInterval	<p>The number of seconds the Tuple mover waits between checks for new data in the WOS to move to ROS.</p>	300	<pre>SELECT SET_CONFIG_PARAMETER ('MoveOutInterval', 600);</pre>
MoveOutMaxAgeTime	<p>Forces the WOS to be written to</p>	30	<pre>SELECT SET_CONFIG_PARAMETER ('MoveOutMaxAgeTime', 20);</pre>

	<p>disk at the specified interval (in seconds). The default is 30.</p> <p>Tip: If you have been running the <code>force_moveout.sh</code> script, you no longer need to run it.</p>		
MoveOutSizePct	<p>The percentage of the WOS that can be filled with data before the Tuple Mover performs a moveout operation.</p>	0	<pre>SELECT SET_CONFIG_PARAMETER ('MoveOutSizePct', 50);</pre>

Internationalization Parameters

The following table describes the internationalization parameters for configuring Vertica.

Parameters	Description	Default	Example
DefaultSessionLocale	<p>Sets the default session startup locale for the database. This parameter does not take effect until the database is restarted.</p>	en_US@collation= binary	<pre>SELECT SET_CONFIG_PARAMETER ('DefaultSessionLocale', 'en_GB');</pre>
EscapeStringWarning	<p>Issues a warning when back slashes are used in a string literal. This is provided to help locate back slashes that are being treated as escape characters so they can be fixed to follow the Standard conforming string syntax instead.</p>	1	<pre>SELECT SET_CONFIG_PARAMETER ('EscapeStringWarning', '1');</pre>
StandardConformingStrings	<p>In Vertica 4.0, determines whether ordinary string literals ('...') treat backslashes (\) as string literals or escape characters. When set to '1', backslashes are treated as string literals, when set to '0', back slashes are treated as escape characters.</p> <p>Tip: To treat backslashes as escape characters, use the Extended string syntax: (E'...');</p> <p>See String Literals (Character) in the SQL Reference Manual.</p>	1	<pre>SELECT SET_CONFIG_PARAMETER ('StandardConformingStrings', '0');</pre>

Epoch Management Parameters

The following table describes the epoch management parameters for configuring Vertica.

Parameters	Description	Default	Example
AdvanceAHMInterval	<p>Determines how frequently (in seconds) Vertica checks the history retention status. By default the AHM interval is set to 180 seconds (3 minutes).</p> <p>Note: AdvanceAHMInterval cannot be set to a value less than the EpochMapInterval.</p>	180	<pre>SELECT SET_CONFIG_PARAMETER ('AdvanceAHMInterval', '3600');</pre>
AdvanceEpochInterval	<p>When EpochAdvancementMode is set to AdvanceEpochInterval, this parameter determines the length of the epoch interval in seconds. When this interval has passed, the epoch is advanced.</p> <p>This parameter is ignored by default.</p> <p>Tip: Decreasing this interval increases the number of epochs saved on disk. Therefore, you might want to reduce the HistoryRetentionTime parameter to limit the number of history epochs that Vertica retains.</p>	180	<pre>SELECT SET_CONFIG_PARAMETER ('AdvanceEpochInterval', '60');</pre>
DefaultIntervalStyle	<p>Sets the default style of interval to use. If set to 1, the interval is in UNITS. If the parameter is set to 0, the interval is in PLAIN style (the SQL standard).</p>	1	<pre>SELECT SET_CONFIG_PARAMETER ('DefaultIntervalStyle', '0');</pre>
EpochAdvancementMode	<p>Determines when epochs are advanced. Setting this parameter requires that you restart the database.</p> <p>Epochs can be advanced based on the following settings:</p> <ul style="list-style-type: none"> ▪ DML— When data is inserted, updated, or deleted 	DML	<pre>SELECT SET_CONFIG_PARAMETER ('EpochAdvancementMode', 'ADVANCEEPOCHINTERVAL');</pre>

	<ul style="list-style-type: none"> ADVANCEEPOCHINTERVAL — When a prescribed period of time has passed. See <code>AdvanceEpochInterval</code> for information about setting the epoch interval. This setting is provided for compatibility with releases of Vertica prior to 4.0. 		
EpochMapInterval	<p>Determines the granularity of mapping between epochs and time available to historical queries. When a historical queries <code>AT TIME T</code> is issued, Vertica maps it to an epoch within a granularity of <code>EpochMapInterval</code> seconds. It similarly affects the time reported for Last Good Epoch during manual recovery (page 235). Note that it does not affect internal precision of epochs themselves.</p> <p>By default, <code>EpochMapInterval</code> is set to 180 seconds (3 minutes).</p> <p>Tip: Decreasing this interval increases the number of epochs saved on disk. Therefore, you might want to reduce the <code>HistoryRetentionTime</code> parameter to limit the number of history epochs that Vertica retains.</p>	180	<pre>SELECT SET_CONFIG_PARAMETER ('EpochMapInterval', '300');</pre>
HistoryRetentionTime	<p>Determines how long deleted data is saved (in seconds) as a historical reference. The default is 0, which means that Vertica saves historical data only when nodes are down. Once the specified time has passed since the delete, the data is eligible to be purged. Use the -1 setting if you prefer to use <code>HistoryRetentionEpochs</code> for determining which deleted data can be purged.</p> <p>Note: The default setting of 0 effectively prevents the use of the Administration Tools 'Roll Back Database to Last Good Epoch' option because the AHM remains close to the current epoch and a rollback is not permitted to an epoch prior to the AHM.</p>	0	<pre>SELECT SET_CONFIG_PARAMETER ('HistoryRetentionTime', '240');</pre>

	<p>Tip: If you rely on the Roll Back option to remove recently loaded data, consider setting a day-wide window for removing loaded data; for example:</p> <pre>SELECT SET_CONFIG_PARAMETER ('HistoryRetentionTime', '86400');</pre>		
HistoryRetentionEpochs	<p>Specifies the number of historical epochs to save, and therefore, the amount of deleted data.</p> <p>Unless you have a reason to limit the number of epochs, Vertica recommends that you specify the time over which delete data is saved. The -1 setting disables this configuration parameter.</p> <p>If both <code>History</code> parameters are specified, <code>HistoryRetentionTime</code> takes precedence, and if both parameters are set to -1, all historical data is preserved.</p> <p>See Setting a Purge Policy (page 181).</p>	-1	<pre>SELECT SET_CONFIG_PARAMETER ('HistoryRetentionEpochs', '40');</pre>

Monitoring Parameters

The following table describes the monitoring parameters for configuring Vertica.

Parameters	Description	Default	Example
SnmpTrapDestinationsList	Defines where Vertica send traps for SNMP. See Configuring Reporting for SNMP (page 212).	none	<pre>SELECT SET_CONFIG_PARAMETER ('SnmpTrapDestinationsList' , 'localhost 162 public');</pre>
SnmpTrapsEnabled	Enables event trapping for SNMP. See Configuring Reporting for SNMP (page 212).	0	<pre>SELECT SET_CONFIG_PARAMETER ('SnmpTrapsEnabled', 1);</pre>
SnmpTrapEvents	Define which events Vertica traps through SNMP. See Configuring Reporting for SNMP (page 212).	Low Disk Space, Read Only File System, Loss of K Safety, Current Fault Tolerance at Critical Level, Too	<pre>SELECT SET_CONFIG_PARAMETER ('SnmpTrapEvents', 'Low Disk Space, Recovery Failure');</pre>

		Many ROS Containers, WOS Over Flow, Node State Change, Recovery Failure, and Stale Checkpoint	
SyslogEnabled	Enables event trapping for syslog. See Configuring Reporting for Syslog (page 210).	0	<code>SELECT SET_CONFIG_PARAMETER ('SyslogEnabled', 1);</code>
SyslogEvents	Defines events that generate a syslog entry. See Configuring Reporting for Syslog (page 210).	none	<code>SELECT SET_CONFIG_PARAMETER ('SyslogEvents', 'Low Disk Space, Recovery Failure');</code>
SyslogFacility	Defines which SyslogFacility Vertica uses. See Configuring Reporting for Syslog (page 210).	user	<code>SELECT SET_CONFIG_PARAMETER ('SyslogFacility' , 'ftp');</code>

Profiling Parameters

The following table describes the profiling parameters for configuring Vertica. See Profiling Database Performance for more information on profiling queries.

Parameters	Description	Default	Example
GlobalEEProfiling	Enables profiling for query execution runs in all sessions, on all nodes.	0	<code>SELECT SET_CONFIG_PARAMETER ('GlobalEEProfiling',1);</code>
GlobalQueryProfiling	Enables query profiling for all sessions on all nodes.	0	<code>SELECT SET_CONFIG_PARAMETER ('GlobalQueryProfiling',1);</code>
GlobalSessionProfiling	Enables session profiling for all sessions on all nodes.	0	<code>SELECT SET_CONFIG_PARAMETER ('GlobalSessionProfiling',1);</code>

See Also

Profiling Database Performance in the Troubleshooting Guide

Query Repository Parameters

Vertica provides a query repository to collect information about all the SQL queries Vertica processes. The following table describes the parameters for configuring the query repository. For more information see Collecting Query Information in the Troubleshooting Guide.

Parameters	Description	Default Setting	Example
CleanQueryRepoInterval	Determines how frequently, in seconds, query data is cleared from the query repository. The default is 1 day (86,400 seconds). How much query data is retained is governed by QueryRepoRetentionTime parameter (see below).	86400	<pre>SELECT SET_CONFIG_PARAMETER ('CleanQueryRepoInterval', '43200');</pre>
QueryRepoMemoryLimit	Determines the maximum memory available for the query repository and for storing database profiling data. The default is 67108864 bytes (64MB). When the cache becomes full, query repository data is dumped to a persistent database table. Profiling data is cleared from the cache. See Configuring Query Repository and Profiling Database Performance.	67108864	<pre>SELECT SET_CONFIG_PARAMETER ('QueryRepoMemoryLimit', '32108854');</pre>
QueryRepoRetentionTime	Determines the maximum number of days worth of query data to save in the query repository. The data is purged according to this parameter ever ClearQueryRepoInterval seconds. See Configuring Query Repository and Managing and Viewing Query Repository.	7	<pre>SELECT SET_CONFIG_PARAMETER ('QueryRepoRetentionTime', '0');</pre>
QueryRepositoryEnabled	Enables the query repository to collect and save query data for all sessions. It also enables query profiling on a global level if it is not already enabled. By default this parameter is off (0). Disable the query repository by setting this parameter to 0. Warning: Disabling this parameter does not disable query profiling. If you are concerned about the amount of memory used by query profiling, either disable it globally or occasionally clear the query profiling data. See Profiling Database	0	<pre>SELECT SET_CONFIG_PARAMETER ('QueryRepositoryEnabled', '1');</pre>

	Performance.		
SaveQueryRepoInterval	Determines how frequently, in seconds, query data is saved to the query repository. The default is five minutes (300 seconds). Note: This parameter cannot be set to less than 300.	300	SELECT SET_CONFIG_PARAMETER ('SaveQueryRepoInterval', '600');

Security Parameters

The following table describes the parameters for configuring the client authentication method and enabling SSL for Vertica.

Parameters	Description	Default Setting	Example
ClientAuthentication	Configures client authentication. By default, Vertica uses user name and password (if supplied) to grant access to the database. The preferred method for establishing client authentication is to use the Administration Tools. See Implementing Client Authentication (page 114) and Creating Records (page 118).	local all trust	SELECT SET_CONFIG_PARAMETER ('ClientAuthentication', 'hostnossl dbadmin 0.0.0.0/0 trust');
EnableSSL	Configures SSL for the server. See Implementing SSL (page 130).	0	SELECT SET_CONFIG_PARAMETER ('EnableSSL', '1');

Kerberos Authentication Parameters

The following parameters set up authentication using Kerberos. See **Configuring Authentication Through Kerberos and GSS** (page 123) for details.

Parameter	Description
KerberosRealm	A string that provides the Kerberos domain name. Usually it consists of all uppercase letters. Example: KerberosRealm=VERTICA.COM
KerberosHostname	A string that provides the Kerberos host name for the KDC server where Kerberos is running. This parameter is optional. If not specified, Vertica uses the return value from the function

	<p>gethostname().</p> <p>Example: KerberosHostname=Host1</p>
KerberosKeytabFile	<p>A string that provides the location of the keytab file. By default, the keytab file is located in /etc/krb5.keytab.</p> <p>The keytab file requires read and write permission only for the file owner who is running the process. Under Linux, for example, file permissions would be: 0600.</p> <p>Example: KerberosKeytabFile=/etc/krb5.keytab</p>
KerberosServiceName	<p>A string that provides the Kerberos service name. By default, the service name is 'vertica'. To construct the principal, follow the format:</p> <p>KerberosServiceName/Kerberos Hostname@Kerberos Realm</p> <p>Example: KerberosServiceName='vertica'</p>

Designing a Logical Schema

Designing a logical schema for a Vertica database is no different than designing for any other SQL database. A logical schema consists of objects such as Schemas, Tables, Views and Referential Integrity constraints that are visible to SQL users.

Vertica supports any relational schema design of your choice. A common practice in data warehouses is to use a Star or a Snowflake schema, hence this document explains the basic concepts of designing Star or Snowflake schema. If you would like assistance with logical schema design, contact **Technical Support** (page 1) for a consultation or refer to the published literature on Data modeling.

To implement your logical schema design, write a SQL script (plain text file) that creates the tables in your database and defines the required referential integrity constraints. You use vsql to run this script to create the actual logical schema of your database.

Data Warehouse Schema Types

Typical schema designs used in Data Warehousing, such as a **star schema** (page 37) or snowflake schema design produce excellent performance on Vertica. These designs are discussed in a separate topic below. Note that it is not required that your schema be a star or snowflake as Vertica supports any arbitrary relational schema design.

Porting an Existing Schema

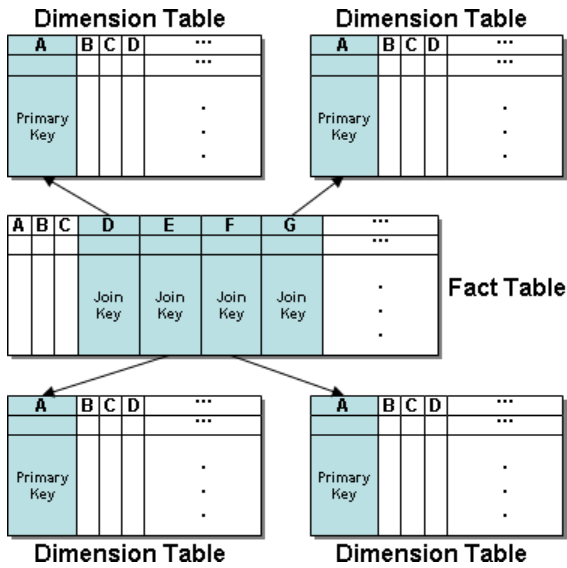
You might need to reconstruct the CREATE TABLE and CREATE VIEW statements, etc., that define an existing schema. An ETL tool can do this for you or your source database system might have a way to generate DDL. For example:

- DB2 has the db2look tool.
- Oracle has the DBMS_METADATA package.
- In case you are migrating from one Vertica database to another, use the EXPORT_CATALOG function.

Keep in mind that these tools (especially from Oracle) tend to use proprietary data types and additional storage clauses, so some edits are needed before they can be used with Vertica.

Star Schema

Sometimes called a star join schema, a star schema is the simplest data warehouse schema. In a star schema design there is a central fact table with a large number of records, optionally surrounded by a collection of dimension tables, each with a lesser number of records. Every dimension table participates in a 1::n join with the fact table.



The table that represents quantitative or factual data. For example, in a business, a fact table might represent orders.

A fact table is often located at the center of a star schema or snowflake schema and might also be referred to as the anchor table. It typically has a large number of records and is surrounded by a collection of dimension tables, each with fewer records. The fact table participates in a join with every dimension table. It can contain data but generally contains many join columns (with optional FOREIGN KEY constraints), each of which corresponds to the primary key column of a dimension table.

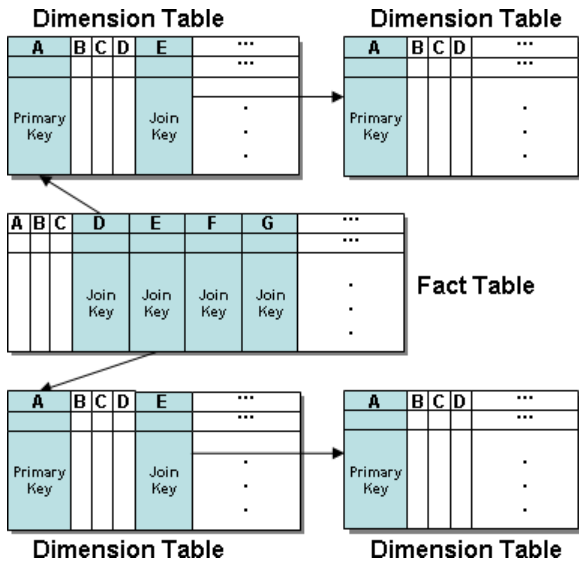
Sometimes called a lookup or reference table, a dimension table is one of a set of companion tables to a large (fact/anchor) table in a star schema. It contains the PRIMARY KEY column corresponding to the join columns in fact tables. For example, a business might use a dimension table to contain item codes and descriptions.

Dimension tables can be connected to other dimension tables to form a hierarchy of dimensions in a snowflake schema.

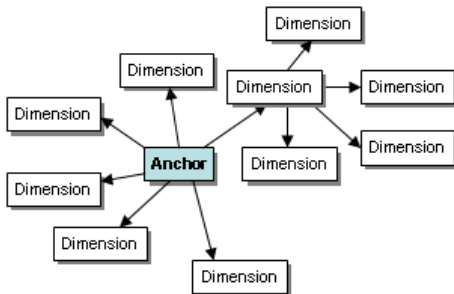
The Retail Sales Example Database in the Getting Started Guide is an example of a star schema.

Snowflake Schema

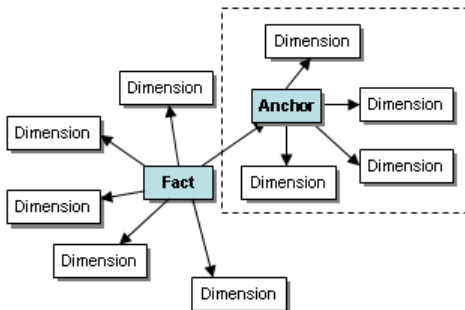
A snowflake schema is the same as a star schema except that a dimension table can be normalized (hierarchically decomposed) into additional dimension tables. Every dimension table participates in a 1::n join with the fact table or another dimension table.



In a snowflake schema, the fact table used in a query or projection is called the anchor table. A query or projection based on that anchor table can include columns from any table in the schema, as shown below.



In a snowflake schema, if a query only includes joins between snowflake dimensions, then the anchor table is the top-most dimension in the hierarchy. In the diagram below, imagine a query on the tables within the box and note its anchor table.



Using Multiple Schemas

Using a single schema is effective if there is only one database user or if a few users cooperate in sharing the database. In many cases, however, it makes sense to use additional schemas to allow users and their applications to create and access tables in separate namespaces. For example, using additional schemas allows:

- Many users to access the database without interfering with one another. Individual schemas can be configured to grant specific users access to the schema and its tables while restricting others.
- Third-party applications to create tables that have the same name in different schemas, preventing table collisions.

Unlike other RDBMS, a schema in a Vertica database is not a collection of objects bound to one user.

Multiple Schema Examples

This section provides examples of when and how you might want to use multiple schemas to separate database users. These examples fall into two categories: using multiple private schemas and using a combination of private schemas (i.e. schemas limited to a single user) and shared schemas (i.e. schemas shared across multiple users).

Using Multiple Private Schemas

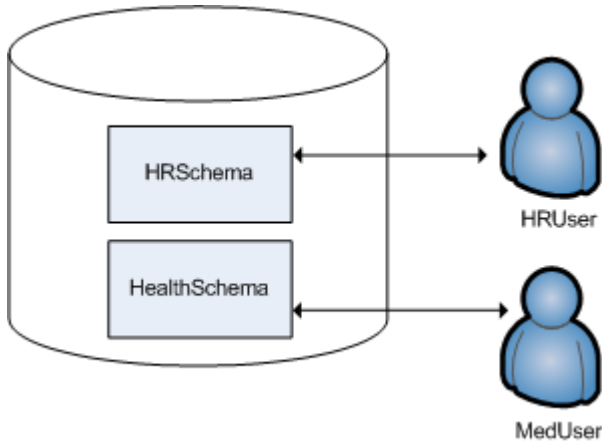
Using multiple private schemas is an effective way of separating database users from one another when sensitive information is involved. Typically a user is granted access to only one schema and its contents, thus providing database security at the schema level. Database users can be running different applications, multiple copies of the same application, or even multiple instances of the same application. This enables you to consolidate applications on one database to reduce management overhead and use resources more effectively. The following examples highlight using multiple private schemas.

- **Using Multiple Schemas to Separate Users and Their Unique Applications**

In this example, both database users work for the same company. One user (HRUser) uses a Human Resource (HR) application with access to sensitive personal data, such as salaries, while another user (MedUser) accesses information regarding company healthcare costs through a healthcare management application. HRUser should not be able to access company healthcare cost information and MedUser should not be able to view personal employee data.

To grant these users access to data they need while restricting them from data they should not see, two schemas are created with appropriate user access, as follows:

- HRSchema—A schema owned by HRUser that is accessed by the HR application.
- HealthSchema—A schema owned by MedUser that is accessed by the healthcare management application.



- **Using Multiple Schemas to Support Multitenancy**

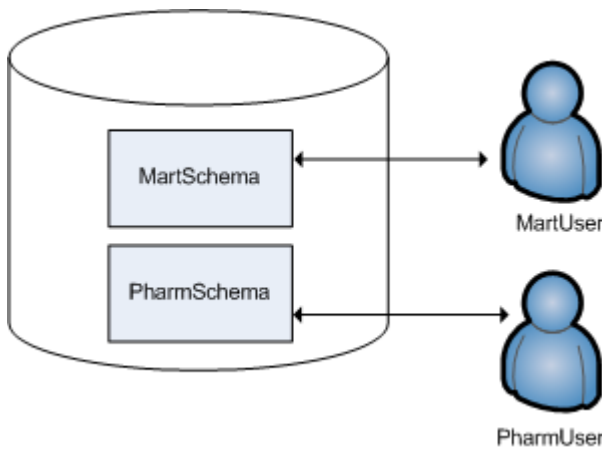
This example is similar to the last example in that access to sensitive data is limited by separating users into different schemas. In this case, however, each user is using a virtual instance of the same application.

An example of this is a retail marketing analytics company that provides data and software as a service (SaaS) to large retailers to help them determine which promotional methods they use are most effective at driving customer sales.

In this example, each database user equates to a retailer, and each user only has access to its own schema. The retail marketing analytics company provides a virtual instance of the same application to each retail customer, and each instance points to the user's specific schema in which to create and update tables. The tables in these schemas use the same names because they are created by instances of the same application, but they do not conflict because they are in separate schemas.

Example of schemas in this database could be:

- MartSchema—A schema owned by MartUser, a large department store chain.
- PharmSchema—A schema owned by PharmUser, a large drug store chain.

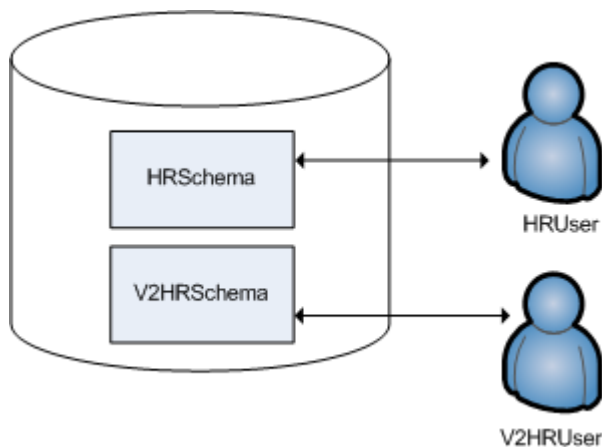


- **Using Multiple Schemas to Migrate to a Newer Version of an Application**

Using multiple schemas is an effective way of migrating to a new version of a software application. In this case, a new schema is created to support the new version of the software, and the old schema is kept as long as necessary to support the original version of the software. This is called a “rolling application upgrade.”

For example, a company might use a HR application to store employee data. The following schemas could be used for the original and updated versions of the software:

- HRSchema—A schema owned by HRUser, the schema user for the original HR application.
- V2HRSchema—A schema owned by V2HRUser, the schema user for the new version of the HR application.

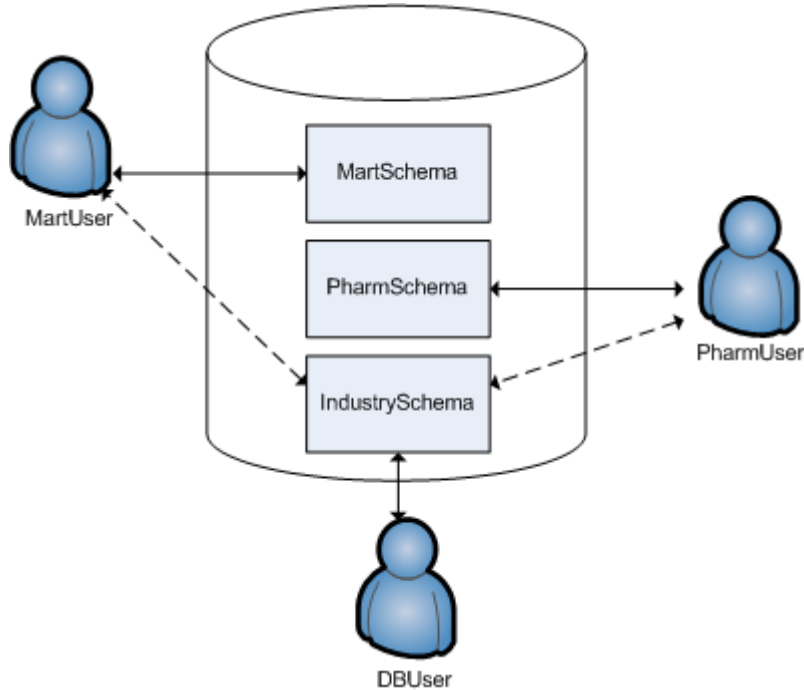


Using Combinations of Private and Shared Schemas

The previous examples illustrate cases in which all schemas in the database are private and no information is shared between users. However, users might want to share common data. In the retail case, for example, MartUser and PharmUser might want to compare their per store sales of a particular product against the industry per store sales average. Since this information is an industry average and is not specific to any retail chain, it can be placed in a schema on which both users are granted USAGE privileges. (For more information about schema privileges, see **Schema Privileges** (page 136).)

Example of schemas in this database could be:

- MartSchema—A schema owned by MartUser, a large department store chain.
- PharmSchema—A schema owned by PharmUser, a large drug store chain.
- IndustrySchema—A schema owned by DBUser (from the retail marketing analytics company) on which both MartUser and PharmUser have USAGE privileges. It is unlikely that retailers would be given any privileges beyond USAGE on the schema and SELECT on one or more of its tables.



Creating Schemas

You can create as many schemas as necessary for your database. For example, you could create a schema for each database user. However, schemas and users are not synonymous as they are in Oracle.

By default, only the superuser can create a schema or give a user the right to create a schema. (See *GRANT (Database)* in the SQL Reference Manual.)

To create a schema use the `CREATE SCHEMA` statement, as described in the SQL Reference Manual.

Referencing Objects When Multiple Schemas are Used

Once two or more schemas have been created, a reference to an object such as a table or a view within a SQL statement or a function must identify the schema in which the object resides. Users can refer to an object by:

- Writing a qualified name that consists of the schema name and object name separated by a dot. For example: `Schema1.MyTable`.
- Using a search path that includes the desired schemas when an object reference is unqualified. Vertica will automatically search the specified schemas to find the object. See *Setting Schema Search Paths* (page 44).

Setting Schema Search Paths

If a user provides an unqualified reference to a table within a SQL statement or a function, the schema search path determines which schemas Vertica searches to locate the table and in which order it searches these schemas. Schemas are searched in the order determined by the SET search_path statement. If, for example, the search path is set to Schema1, Schema2, Vertica searches for the table in Schema1 first. If it can't locate the table, it then searches Schema2. If it can't locate the table in any of the schemas in the search path, it halts the search and report an error even if the table exists in another schema.

Note: The user must be granted access to the schemas in the search path for Vertica to be able to search for tables within them. If the user doesn't have access to a schema n the search path, Vertica skips to the next schema in the search path, if one exists.

The first schema in the search path is called the current schema. The current schema is the location where tables are created if the CREATE TABLE name statement does not specify a schema name.

By default, the search path for all users is "\$user", public, v_catalog, v_monitor, v_internal.

```
mydb=> SHOW SEARCH_PATH;
  name      | setting
-----+-----
search_path | "$user", public, v_catalog, v_monitor, v_internal
(1 row)
```

\$user is a placeholder that resolves to the user name, and public references the public schema. v_catalog and v_monitor refer to Vertica system tables. v_internal is for internal use.

The search path means that Vertica looks in the user's schema first, assuming that each user has a separate schema that uses the same name as their user name. If the schema doesn't exist or it cannot find the table, Vertica then looks in the public schema, as well as the v_catalog and v_monitor built-in schemas.

Tip: The SET search_path statement is equivalent in function to the CURRENT_SCHEMA statement found in some other databases.

To see the current search path, use the SHOW SEARCH_PATH statement. To view the current schema, use SELECT CURRENT_SCHEMA(). The function SELECT CURRENT_SCHEMA() also shows the resolved name of \$User.

Examples:

```
USER1 uses the following statement to set the search path to COMMON,
  $USER, PUBLIC: mydb=> SET SEARCH_PATH TO COMMON, $USER, PUBLIC;
```

Vertica returns the following output:

Statement/Function	Output
SHOW SEARCH_PATH;	COMMON, \$USER, PUBLIC
SELECT CURRENT_SCHEMA ();	COMMON

USER1 uses the following statement to set the search path to \$USER, COMMON, PUBLIC: `mydb=> SET SEARCH_PATH TO $USER, COMMON, PUBLIC;`

Vertica returns the following output for `SELECT CURRENT_SCHEMA()` if schema USER1 exists:

```
mydb=> USER1
```

If schema USER1 does not exist, the output for `SELECT CURRENT_SCHEMA()` is:

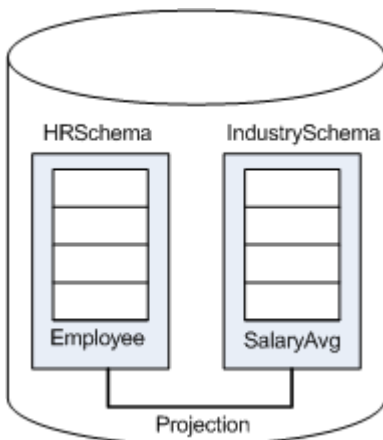
```
mydb=> COMMON
```

See Also

SQL System Tables in the SQL Reference Manual

Creating Objects that Span Multiple Schemas

Vertica supports views or pre-join projections that reference tables across multiple schemas. For example, a user might need to compare employee salaries to industry averages. In this case, the application would query a shared schema (IndustrySchema) for salary averages in addition to its own private schema (HRSchema) for company-specific salary information.



Best Practice: When creating objects that span schemas, use qualified table names. This naming convention avoids confusion if the query path or table structure within the schemas changes at a later date.

Creating Tables

In Vertica you can create both base tables and temporary tables, depending on what you are trying to accomplish. For example, base tables are created in the Vertica logical schema while temporary tables are useful for dividing complex query processing into multiple steps.

Creating Base Tables

The CREATE TABLE statement creates a table in the Vertica logical schema. The example databases described in the Getting Started Guide include sample SQL scripts that demonstrate this procedure. For example:

```
CREATE TABLE vendor_dimension (  
  vendor_key          INTEGER          NOT NULL PRIMARY KEY,  
  vendor_name         VARCHAR(64),  
  vendor_address      VARCHAR(64),  
  vendor_city         VARCHAR(64),  
  vendor_state        CHAR(2),  
  vendor_region       VARCHAR(32),  
  deal_size           INTEGER,  
  last_deal_update    DATE  
);
```

Automatic Projection Creation

To get your database up and running quickly, Vertica automatically creates a default projection for each table created through the CREATE TABLE and CREATE TEMPORARY TABLE statements. The timing of when the projection is created depends on how you use the CREATE TABLE statement:

- If you create a table without providing the projection-related clauses, a superprojection is automatically created for the table when an INSERT, COPY, or LCOPY command is issued to load data into the table for the first time. The projection is created in the same schema as the table. Once Vertica has created the projection, it loads the data.
- If you use CREATE TABLE AS SELECT to create a table from the results of a query, the table is created first and a projection is created immediately after, using some of the properties of the underlying SELECT query.
- (Advanced users only) If you use any of the following parameters, the default projection is created immediately upon table creation using the specified properties:
 - column-definition (ENCODING encoding-type and ACCESSRANK integer)
 - ORDER BY table-column
 - hash-segmentation-clause
 - range-segmentation-clause
 - UNSEGMENTED { NODE *node* | ALL NODES }
 - KSAFE

Note: Before you define a superprojection in the above manner, read *Creating Custom Designs* (page 91) in the Administrator's Guide.

Characteristics of Default Automatic Projections

A default projection has the following characteristics:

- It is a superprojection.
- It uses the default encoding-type AUTO.

- If the table has one or more primary keys defined, the projection is sorted by these columns. Otherwise, the projection is sorted in the same order as defined in the table column-definition list.
- If the K-safety for the database is zero (K-Safety=0), the projection is unsegmented on the initiator node. If K-Safety is greater than zero (K-Safety>0), the superprojection is replicated (unsegmented) on all nodes. See Segmentation in the Concepts Guide.
- If the projection was created through the CREATE TABLE AS SELECT statement, the projection uses the sort order, segmentation, and encoding specified for the columns in the query table.

Default automatic projections let you get your database up and running quickly; however, they might not necessarily provide the best performance. Vertica recommends that you start with these projections and then use the Database Designer to optimize your database. The Database Designer creates projections that optimize your database based on the characteristics of the data and, optionally, the queries you use.

See Also

Projections in the Concepts Guide

CREATE TABLE in the SQL Reference Manual

Creating Temporary Tables

You create temporary tables using the CREATE TEMPORARY TABLE statement. A common use case for a temporary table is to divide complex query processing into multiple steps. Typically, a reporting tool holds intermediate results while reports are generated (for example, first get a result set, then query the result set, and so on). You can also write subqueries.

Note: The default is ON COMMIT DELETE ROWS, where data is discarded at the end of the transaction or session.

Global Temporary Tables

Global temporary tables are created in the public schema, and they are visible to all users and sessions. However, the contents (data) of a global table are private to the transaction or session in which the data was inserted. Data is automatically removed when the transaction commits, rolls back, or the session ends. This allows two users to use the same temporary table, concurrently, but see only data specific to his or her own transactions for the duration of those transactions or sessions.

The definition of a global temporary table persists in the database catalogs until explicitly removed by using the DROP TABLE statement.

Local Temporary Tables

A local temporary table is created in the `v_TEMP_SCHEMA` namespace and is transparently inserted into the user's search path. It is visible only to the user who creates the table for the duration of the session in which it is created. When the session ends, the table definition is automatically dropped from the database catalogs.

Note that You cannot add projections to non-empty, session-scoped temporary tables with ON COMMIT PRESERVE ROWS specified. Be sure that projections exist before you load data. See the "Automatic Projection Creation" in the CREATE TABLE statement. Also, although adding projections is allowed for tables with ON COMMIT DELETE ROWS specified, be aware that you could lose all the data.

Automatic Projection Creation and Characteristics

When you use the CREATE TEMPORARY TABLE command, the table is created first and the default superprojection is created immediately after unless you specify NO PROJECTION.

A default projection has the following characteristics:

- It uses the default encoding-type AUTO.
- It is automatically unsegmented on the initiator node and pinned if you do not specify a segmentation clause (hash-segmentation-clause, range-segmentation-clause, or UNSEGMENTED).
- If the table has one or more primary keys defined, the projection is sorted by these columns. Otherwise, the the projection is sorted in the same order as defined in the table column-definition list.
- Temp tables are not recoverable, so the superprojection is not K-Safe (K-SAFE=0), and you cannot make the table K-safe.

Advanced users can modify the default projection created through the CREATE TEMPORARY TABLE statement by defining any or all of the following parameters:

- column-definition (ENCODING encoding-type and ACCESSRANK integer)
- ORDER BY table-column
- hash-segmentation-clause
- range-segmentation-clause
- UNSEGMENTED { NODE node | ALL NODES }
- NO PROJECTION

Note: Before you define the superprojection in this manner, read *Creating Custom Designs* (page 91) in the Administrator's Guide.

See Also

Projections in the Concepts Guide

CREATE TEMPORARY TABLE in the SQL Reference Manual

Adding Constraints

Constraints specify rules on data that can go into a table. Constraints are specified on columns and tables using the following SQL commands:

- `CREATE TABLE` lets you define a column-constraint on a single column or multiple columns
- `ALTER TABLE` lets you define (or drop) a table-constraint on single columns or on multiple columns (also called a compound key)

The examples that follow illustrate several ways of defining constraints on single and multiple columns. For additional details, see:

- ***PRIMARY KEY constraints*** (page 51)
- ***FOREIGN KEY constraints*** (page 52)
- ***UNIQUE constraints*** (page 53)
- ***NOT NULL constraints*** (page 54)

Defining single-column constraints

The following example defines a `PRIMARY KEY` constraint on a single column in a dimension table, assigning a constraint name of `dim1PK`:

```
CREATE TABLE dim1 (
  c1 INTEGER CONSTRAINT dim1PK PRIMARY KEY,
  c2 INTEGER
);
```

Constraint names are optional, so you could write the same statement as follows:

```
CREATE TABLE dim1 (
  c1 INTEGER NOT NULL PRIMARY KEY,
  c2 INTEGER
);
```

Note: Vertica recommends naming constraints so they are easier to identify and/or remove, if necessary.

A column can have more than one constraint. Write the constraints one after another. For example, the following SQL statement enforces both `NOT NULL` and `PK` constraints on the `customer_key` column, which means the column cannot accept `NULL` values, and the column is also the primary key.

```
CREATE TABLE customer_dimension (
  customer_key INTEGER NOT NULL PRIMARY KEY,
  customer_name VARCHAR(256),
  customer_address VARCHAR(256),
  ...
);
```

The following syntax defines a `FOREIGN KEY` constraint on a single column in a fact table, assigning a constraint name of `fact1dim1PK`:

```
CREATE TABLE fact1 (
  c1 INTEGER CONSTRAINT fact1dim1FK REFERENCES dim1,
  c2 INTEGER
```

```
);
```

Note that a FOREIGN KEY constraint can be specified solely by referencing the table that contains the primary key. The columns in the referenced table do not need to be explicitly specified.

Defining multicolumn constraints

The following example defines a PRIMARY KEY constraint on multiple columns:

```
CREATE TABLE dim (  
    c1 INTEGER NOT NULL,  
    c2 INTEGER NOT NULL,  
    PRIMARY KEY (c1, c2)  
);
```

To specify multi-column (compound) keys, you can use an ALTER TABLE statement *in addition to* a CREATE TABLE statement.

```
CREATE TABLE dim2 (  
    c1 INTEGER NOT NULL,  
    c2 INTEGER NOT NULL,  
    c3 INTEGER NOT NULL,  
    c4 INTEGER UNIQUE  
);  
ALTER TABLE dim2  
    ADD CONSTRAINT dim2PK PRIMARY KEY (c1, c2);
```

You could also define a compound PRIMARY KEY using just the CREATE TABLE statement:

```
CREATE TABLE dim2 (  
    c1 INTEGER NOT NULL,  
    c2 INTEGER NOT NULL,  
    c3 INTEGER NOT NULL,  
    c4 INTEGER UNIQUE,  
    PRIMARY KEY (c1, c2)  
);
```

The matching FOREIGN KEY constraint to table dim2 is specified as follows:

```
CREATE TABLE fact2 (  
    c1 INTEGER,  
    c2 INTEGER,  
    c3 INTEGER NOT NULL,  
    c4 INTEGER UNIQUE  
);  
ALTER TABLE fact2  
    ADD CONSTRAINT fact2FK FOREIGN KEY (c1, c2) REFERENCES dim2(c1, c2);
```

A FOREIGN KEY constraint can be specified solely by a reference to the table that contains the PRIMARY KEY. In the ADD CONSTRAINT command above, the REFERENCES column names are optional; you could just enter REFERENCES dim2.

Enforcing Constraints

In order to maximize query performance, Vertica checks for constraint violations when queries are run, not when data is loaded. One exception is that PRIMARY and FOREIGN KEY violations are detected when loading into the fact table of a pre-join projection. For more details see **Enforcing Constraints** (page 55).

To enforce constraints, you can load data without committing it using the COPY with NO COMMIT option and then perform a post-load check using the ANALYZE_CONSTRAINTS function. If constraint violations are found, you can roll back the load because you have not committed it. For more details see **Analyzing Constraints (Detecting Constraint Violations)** (page 55).

See Also

ALTER TABLE, CREATE TABLE, and COPY NO COMMIT, statements in the SQL Reference Manual

ANALYZE_CONSTRAINTS function in the SQL Reference Manual

PRIMARY KEY Constraints

A primary key (PK) is a single column or combination of columns (called a compound key) that uniquely identifies each row in a table. A PRIMARY KEY constraint contains unique, non-null values.

The following example specifies a single column, `customer_key`, as the PRIMARY KEY.

```
CREATE TABLE customer_dimension (
  customer_key INTEGER NOT NULL PRIMARY KEY,
  customer_name VARCHAR(256),
  customer_address VARCHAR(256),
  customer_city VARCHAR(64),
  customer_state CHAR(2),
  household_id INTEGER UNIQUE
);
```

Note: If you specify a PK constraint using ALTER TABLE, the system returns the following informational, which is for information only: WARNING: Column customer_key definition changed to NOT NULL

Primary keys can also constrain more than one column:

```
CREATE TABLE customer_dimension (
  customer_key INTEGER NOT NULL,
  customer_name VARCHAR(256),
  customer_address VARCHAR(256),
  customer_city VARCHAR(64),
  customer_state CHAR(2),
  household_id INTEGER UNIQUE
  PRIMARY KEY (customer_key, household_id)
);
```

FOREIGN KEY Constraints

A foreign key (FK) is a column that is used to join a table to other tables to ensure referential integrity of the data. A FOREIGN KEY constraint is a rule that states that a column cannot be null and can contain only values from the PRIMARY KEY column on a specific dimension table.

Notes

In Vertica, the fact table's join columns are required to have FOREIGN KEY constraints in order to participate in pre-join projections.

If the fact table join column has a FOREIGN KEY constraint, outer join queries produce the same result set as inner join queries.

FOREIGN KEY constraint can be specified solely by referencing the table that contains the primary key. The columns in the referenced table do not need to be explicitly specified.

Examples

Assume you have created a table that stores inventory data:

```
CREATE TABLE inventory_fact (
  date_key INTEGER NOT NULL,
  product_key integer not null,
  product_version INTEGER NOT NULL,
  warehouse_key INTEGER NOT NULL,
  qty_in_stock INTEGER
);
```

Now you have created a table that stores information about warehouses. To ensure referential integrity, you define a FOREIGN KEY constraint on the `inventory_fact` table that references the `warehouse_dimension` table:

```
CREATE TABLE warehouse_dimension (
  warehouse_key INTEGER NOT NULL PRIMARY KEY,
  warehouse_name VARCHAR(20),
  warehouse_address VARCHAR(256),
  warehouse_city VARCHAR(60),
  warehouse_state CHAR(2),
);
ALTER TABLE inventory_fact
  ADD CONSTRAINT fk_inventory_warehouse FOREIGN KEY (warehouse_key)
  REFERENCES warehouse_dimension (warehouse_key);
```

In this example, the `inventory_fact` table is the *referencing* table and the `warehouse_dimension` table is the *referenced* table.

You can also shorten the second CREATE TABLE statement and eliminate the ALTER TABLE statement. In the absence of a column list, the PRIMARY KEY of the referenced table is used as the referenced column or columns.

```
CREATE TABLE warehouse_dimension (
  warehouse_key INTEGER NOT NULL PRIMARY KEY REFERENCES warehouse_dimension,
  warehouse_name VARCHAR(20),
  warehouse_address VARCHAR(256),
  warehouse_city VARCHAR(60),
);
```

```
warehouse_state CHAR(2),
);
```

A FOREIGN KEY can also constrain and reference a group of columns:

```
CREATE TABLE t1 (
  c1 INTEGER PRIMARY KEY,
  c2 INTEGER,
  c3 INTEGER,
  FOREIGN KEY (c2, c3) REFERENCES other_table (c1, c2)
);
```

Note: The number and data type of the constrained columns must match the number and type of the referenced columns.

UNIQUE Constraints

UNIQUE constraints ensure that the data contained in a column or a group of columns is unique with respect to all the rows in the table.

When written as a column-constraint, the syntax is:

```
CREATE TABLE product_dimension (
  product_key INTEGER NOT NULL CONSTRAINT product_key_UK UNIQUE,
  sku_number CHAR(32),
  product_cost INTEGER
  ...
);
```

Notice that the above syntax names the UNIQUE constraint `product_key_UK`.

When written as a table-constraint, the syntax is:

```
CREATE TABLE product_dimension (
  product_key INTEGER NOT NULL
  sku_number CHAR(32),
  product_cost INTEGER,
  UNIQUE (product_key)
);
```

You can also use the ALTER TABLE statement to specify a UNIQUE constraint. In this example, name the constraint `product_key_UK`:

```
ALTER TABLE product_dimension
  ADD CONSTRAINT product_key_UK UNIQUE (product_key);
```

You can also use the ALTER TABLE statement to specify multiple columns:

```
ALTER TABLE dim1
  ADD CONSTRAINT constraint_name_unique UNIQUE (c1, c2);
```

If a UNIQUE constraint refers to a group of columns, separate the columns list using commas:

```
CREATE TABLE dim1 (
  c1 INTEGER,
  c2 INTEGER,
  c3 INTEGER,
  UNIQUE (c1, c2)
);
```

The column listing specifies that the combination of values in the indicated columns is unique across the whole table, though any one of the columns need not be (and ordinarily isn't) unique.

NOT NULL Constraints

A NOT NULL constraint specifies that a column must NOT contain a null value. This means that new rows cannot be inserted or updated unless you specify a value for this column.

The following SQL statement enforces a NOT NULL constraint on the `customer_key` column, which means the column cannot accept NULL values.

```
CREATE TABLE customer_dimension (
    customer_key INTEGER NOT NULL PRIMARY KEY,
    customer_name VARCHAR(256),
    customer_address VARCHAR(256),
    ...
);
```

Notes:

- A NOT NULL constraint is always written as a column-constraint
- NOT NULL constraints are not named.

Removing Constraints

To drop named constraints, use the ALTER TABLE command, such as in the following example, which drops constraint `factfk2`:

```
=> ALTER TABLE fact2 DROP CONSTRAINT fact2fk;
```

To drop unnamed constraints, query the system table `TABLE_CONSTRAINTS`, which returns both system-generated and user-named constraint names:

```
=> SELECT * FROM TABLE_CONSTRAINTS;
```

In the following output, notice the system-generated constraint name of `C_PRIMARY` versus the user-defined constraint name of `fk_inventory_date`:

```
-[ RECORD 1 ]-----+-----
constraint_id      | 45035996273707984
constraint_name    | C_PRIMARY
constraint_schema_id | 45035996273704966
constraint_key_count | 1
foreign_key_count  | 0
table_id          | 45035996273707982
foreign_table_id   | 0
constraint_type    | p
-[ ... ]-----+-----
-[ RECORD 9 ]-----+-----
constraint_id      | 45035996273708016
constraint_name    | fk_inventory_date
constraint_schema_id | 0
constraint_key_count | 1
foreign_key_count  | 1
table_id          | 45035996273708014
foreign_table_id   | 45035996273707994
```

```
constraint_type | f
```

Once you know the name of the constraint, you can then drop it using the ALTER TABLE command.

Notes

- Non-null constraints do not have names and cannot be dropped.
- Constraints cannot be dropped on tables that have projections.

See Also

ANALYZE_CONSTRAINTS, ALTER TABLE, CREATE TABLE, and COPY with NO COMMIT in the SQL Reference Manual

Enforcing Constraints

Enforcing Primary Key Constraints

Vertica does not enforce uniqueness of primary keys when they are loaded into a table. However, when data is loaded into a table with a pre-joined dimension, or when the table is joined to a dimension table during a query, a key enforcement error could result if there is not exactly one dimension row that matches each foreign key value.

Note: Consider using sequences or auto-incrementing columns for primary key columns, which guarantees uniqueness and avoids the constraint enforcement problem and associated overhead. For more information see *Using Sequences* (page 60).

Enforcing Foreign Key Constraints

A table's foreign key constraints are enforced during data load only if there is a pre-join projection that has that table as its anchor table. If there is no such pre-join projection then it is possible to load data that causes a constraint violation. Subsequently a constraint violation error can happen when:

- An inner join query is processed
- An outer join is treated as an inner join due to the presence of foreign key
- A new pre-join projection anchored on the table with the foreign key constraint is refreshed

To detect constraint violations you can load data without committing it using the COPY NO COMMIT option and then perform a post-load check using the ANALYZE_CONSTRAINTS function. If constraint violations are found, you can roll back the load because you have not committed it. For more details see *Analyzing Constraints (Detecting Constraint Violations)* (page 55).

Analyzing Constraints (Detecting Constraint Violations)

The ANALYZE_CONSTRAINTS() function analyzes and reports on constraint violations within the current schema search path. You can check for constraint violations by passing an empty argument (which returns violations on all tables within the current schema), by passing a single table argument, or by passing two arguments containing a table name and a column or list of columns.

Given the following inputs, Vertica returns one row, indicating one violation, because the same primary key value (10) was inserted into table t1 twice:

```
CREATE TABLE t1(c1 INT);
ALTER TABLE t1 ADD CONSTRAINT pk_t1 PRIMARY KEY (c1);
CREATE PROJECTION t1_p (c1) AS SELECT * FROM t1 UNSEGMENTED ALL NODES;
INSERT INTO t1 values (10);
INSERT INTO t1 values (10); --Duplicate primary key value
SELECT ANALYZE_CONSTRAINTS('t1');
 Schema Name | Table Name | Column Names | Constraint Name | Constraint Type | Column Values
-----+-----+-----+-----+-----+-----
 public      | t1         | c1           | pk_t1           | PRIMARY         | ('10')
(1 row)
```

If the second INSERT statement above had contained any different value, the result would have been 0 rows (no violations).

In this example, create a table that contains 3 integer columns, one a unique key and one a primary key:

```
CREATE TABLE fact_1(
  f INTEGER,
  f_UK INTEGER UNIQUE,
  f_PK INTEGER PRIMARY KEY
);
```

Try issuing a command that refers to a nonexistent column:

```
SELECT ANALYZE_CONSTRAINTS('f_BB', 'f2');
ERROR: 'f_BB' is not a table name in the current search path
```

Insert some values into table fact_1 and commit the changes:

```
INSERT INTO fact_1 values (1, 1, 1);
COMMIT;
```

Now issue the ANALYZE_CONSTRAINTS command on table fact_1. No constraint violations are expected and none are found:

```
SELECT ANALYZE_CONSTRAINTS('fact_1');
 Schema Name | Table Name | Column Names | Constraint Name | Constraint Type | Column Values
-----+-----+-----+-----+-----+-----
(0 rows)
```

Now insert duplicate unique and primary key values and run ANALYZE_CONSTRAINTS on table fact_1 again. The system shows two violations: one against the primary key and one against the unique key:

```
INSERT INTO fact_1 VALUES (1, 1, 1);
COMMIT;
SELECT ANALYZE_CONSTRAINTS('fact_1');
 Schema Name | Table Name | Column Names | Constraint Name | Constraint Type | Column Values
-----+-----+-----+-----+-----+-----
 public      | fact_1     | f_pk         | -               | PRIMARY         | ('1')
 public      | fact_1     | f_uk         | -               | UNIQUE          | ('1')
(2 rows)
```

The following command looks for constraint validation on only the unique key in table fact_1:

```
SELECT ANALYZE_CONSTRAINTS('fact_1', 'f_UK');
 Schema Name | Table Name | Column Names | Constraint Name | Constraint Type | Column Values
-----+-----+-----+-----+-----+-----
 public      | fact_1     | f_uk         | C_UNIQUE        | UNIQUE          | ('1')
(1 row)
```

The following example shows that you can specify the same column more than once; the function, however, returns the violation once only:

```
SELECT ANALYZE_CONSTRAINTS('fact_1', 'f_PK, F_PK');
 Schema Name | Table Name | Column Names | Constraint Name | Constraint Type | Column Values
-----+-----+-----+-----+-----+-----
 public      | fact_1     | f_pk         | C_PRIMARY      | PRIMARY        | ('1')
(1 row)
```

The following example creates a new dimension table, `dim_1`, and inserts a foreign key and different (character) data types:

```
CREATE TABLE dim_1 (b VARCHAR(3), b_PK VARCHAR(4), b_FK INTEGER REFERENCES fact_1(f_PK));
```

Alter the table to create a multicolumn unique key and multicolumn foreign key and create superprojections:

```
ALTER TABLE dim_1 ADD CONSTRAINT dim_1_multiuk PRIMARY KEY (b, b_PK);
```

The following command inserts a missing foreign key (0) in table `dim_1` and commits the changes:

```
INSERT INTO dim_1 VALUES ('r1', 'Xpk1', 0);
COMMIT;
```

Checking for constraints on table `dim_1` detects a foreign key violation:

```
SELECT ANALYZE_CONSTRAINTS('dim_1');
 Schema Name | Table Name | Column Names | Constraint Name | Constraint Type | Column Values
-----+-----+-----+-----+-----+-----
 public      | dim_1     | b_fk         | C_FOREIGN      | FOREIGN        | ('0')
(1 row)
```

Now add a duplicate value into the unique key and commit the changes:

```
INSERT INTO dim_1 values ('r2', 'Xpk1', 1);
INSERT INTO dim_1 values ('r1', 'Xpk1', 1);
COMMIT;
```

Checking for constraint violations on table `dim_1` detects the duplicate unique key error:

```
SELECT ANALYZE_CONSTRAINTS('dim_1');
 Schema Name | Table Name | Column Names | Constraint Name | Constraint Type | Column Values
-----+-----+-----+-----+-----+-----
 public      | dim_1     | b, b_pk     | dim_1_multiuk  | PRIMARY        | ('r1', 'Xpk1')
 public      | dim_1     | b_fk         | C_FOREIGN      | FOREIGN        | ('0')
(2 rows)
```

Now create a table with multicolumn foreign key and create the superprojections:

```
CREATE TABLE dim_2(z_fk1 VARCHAR(3), z_fk2 VARCHAR(4));
ALTER TABLE dim_2 ADD CONSTRAINT dim_2_multifk FOREIGN KEY (z_fk1, z_fk2) REFERENCES dim_1(b, b_PK);
```

Now insert a foreign key that matches a foreign key in table `dim_1` and commit the changes:

```
INSERT INTO dim_2 VALUES ('r1', 'Xpk1');
COMMIT;
```

Checking for constraints on table `dim_2` detects no violations:

```
SELECT ANALYZE_CONSTRAINTS('dim_2');
 Schema Name | Table Name | Column Names | Constraint Name | Constraint Type | Column Values
-----+-----+-----+-----+-----+-----
(0 rows)
```

Add a value that does not match and commit the change:

```
INSERT INTO dim_2 values ('r1', 'NONE');
```

```
COMMIT;
```

Checking for constraints on table `dim_2` detects a foreign key violation:

```
SELECT ANALYZE_CONSTRAINTS('dim_2');
 Schema Name | Table Name | Column Names | Constraint Name | Constraint Type | Column Values
-----+-----+-----+-----+-----+-----
 public      | dim_2      | z_fk1, z_fk2 | dim_2_multifk  | FOREIGN         | ('r1', 'NONE')
(1 row)
```

Now analyze all constraints on all tables:

```
SELECT ANALYZE_CONSTRAINTS('');
 Schema Name | Table Name | Column Names | Constraint Name | Constraint Type | Column Values
-----+-----+-----+-----+-----+-----
 public      | dim_1      | b, b_pk      | dim_1_multiuk  | PRIMARY        | ('r1', 'Xpk1')
 public      | dim_1      | b_fk         | C_FOREIGN      | FOREIGN        | ('0')
 public      | dim_2      | z_fk1, z_fk2 | dim_2_multifk  | FOREIGN        | ('r1', 'NONE')
 public      | fact_1     | f_pk         | C_PRIMARY      | PRIMARY        | ('1')
 public      | fact_1     | f_uk         | C_UNIQUE       | UNIQUE         | ('1')
(5 rows)
```

To quickly clean up your database, issue the following command:

```
DROP TABLE fact_1 cascade;
DROP TABLE dim_1 cascade;
DROP TABLE dim_2 cascade;
```

Fixing Constraint Violations

Use the function `DISABLE_DUPLICATE_KEY_ERROR` to suppress error messaging when Vertica finds duplicate PRIMARY KEY/UNIQUE KEY values at run time. Queries execute as though no constraints are defined on the schema, and the effects are session scoped.

CAUTION: When called, `DISABLE_DUPLICATE_KEY_ERROR()` suppresses data integrity checking and can lead to incorrect query results. Use this function only after you insert duplicate primary keys into a dimension table in the presence of a pre-join projection. Then correct the violations and turn integrity checking back on with `REENABLE_DUPLICATE_KEY_ERROR()`.

The following series of commands create a table named `dim` and the corresponding projection:

```
CREATE TABLE dim (pk INTEGER PRIMARY KEY, x INTEGER);
CREATE PROJECTION dim_p (pk, x) AS SELECT * FROM dim ORDER BY x UNSEGMENTED ALL
NODES;
```

The next two statements create a table named `fact` and the pre-join projection that joins `fact` to `dim`.

```
CREATE TABLE fact(fk INTEGER REFERENCES dim(pk));
CREATE PROJECTION prejoin_p (fk, pk, x) AS SELECT * FROM fact, dim WHERE pk=fk ORDER
BY x;
```

The following statements load values into table `dim`. Notice the last statement inserts a duplicate primary key value of 1:

```
INSERT INTO dim values (1,1);
INSERT INTO dim values (2,2);
INSERT INTO dim values (1,2); --Constraint violation
COMMIT;
```


Table `dim` now contains duplicate primary key values, but you cannot delete the violating row because of the presence of the pre-join projection. Any attempt to delete the record results in the following error message:

```
ROLLBACK: Duplicate primary key detected in FK-PK join Hash-Join (x dim_p), value
1
```

In order to remove the constraint violation (`pk=1`), use the following sequence of commands, which puts the database back into the state just before the duplicate primary key was added.

To remove the violation:

- 1 First save the original `dim` rows that match the duplicated primary key.

```
CREATE TEMP TABLE dim_temp(pk integer, x integer);
INSERT INTO dim_temp SELECT * FROM dim WHERE pk=1 AND x=1; -- original
dim row
```

- 2 Temporarily disable error messaging on duplicate constraint values:

```
SELECT DISABLE_DUPLICATE_KEY_ERROR();
```

Caution: Remember that issuing this command suppresses the enforcement of data integrity checking.

- 3 Remove the the original row that contains duplicate values:

```
DELETE FROM dim WHERE pk=1;
```

- 4 Allow the database to resume data integrity checking:

```
SELECT REENABLE_DUPLICATE_KEY_ERROR();
```

- 5 Reinsert the original values back into the dimension table:

```
INSERT INTO dim SELECT * from dim_temp;
COMMIT;
```

- 6 Validate your dimension and fact tables.

If you receive the following error message, it means that the duplicate records you want to delete are not identical. That is, the records contain values that differ in at least one column that is not a primary key; for example, (1,1) and (1,2).

```
ROLLBACK: Delete: could not find a data row to delete (data integrity violation?)
```

The difference between this message and the rollback message in the previous example is that a fact row contains a foreign key that matches the duplicated primary key, which has been inserted. Thus, a row with values from the fact and dimension table is now in the prejoin projection. In order for the DELETE statement (Step 3 in the following example) to complete successfully, extra predicates are required to identify the original dimension table values (the values that are in the prejoin).

This example is nearly identical to the previous example, except that an additional INSERT statement joins the fact table to the dimension table by a primary key value of 1:

```
INSERT INTO dim values (1,1);
INSERT INTO dim values (2,2);
INSERT INTO fact values (1); -- New insert statement joins fact with dim on
primary key value=1
INSERT INTO dim values (1,2); -- Duplicate primary key value=1
COMMIT;
```

To remove the violation:

- 1 First save the original `dim` and `fact` rows that match the duplicated primary key:

```
CREATE TEMP TABLE dim_temp(pk integer, x integer);
CREATE TEMP TABLE fact_temp(fk integer);
INSERT INTO dim_temp SELECT * FROM dim WHERE pk=1 AND x=1; -- original
dim row
INSERT INTO fact_temp SELECT * FROM fact WHERE fk=1;
```

- 2 Temporarily suppresses the enforcement of data integrity checking:

```
SELECT DISABLE_DUPLICATE_KEY_ERROR();
```

- 3 Remove the duplicate primary keys. These steps implicitly remove all fact rows with the matching foreign key, as well.

- a) Remove the the original row that contains duplicate values:

```
DELETE FROM dim WHERE pk=1 AND x=1;
```

Note: The extra predicate (`x=1`) specifies removal of the original (1, 1) row, rather than the newly inserted (1, 2) values that caused the violation.

- b) Remove all remaining rows:

```
DELETE FROM dim WHERE pk=1;
```

- 4 Turn on integrity checking:

```
SELECT REENABLE_DUPLICATE_KEY_ERROR();
```

- 5 Reinsert the original values back into the fact and dimension table:

```
INSERT INTO dim SELECT * from dim_temp;
INSERT INTO fact SELECT * from fact_temp;
COMMIT;
```

- 6 Validate your dimension and fact tables.

Reenabling error reporting

Use the `REENABLE_DUPLICATE_KEY_ERROR()` function to restore the default behavior of error reporting and reverse the effects of `DISABLE_DUPLICATE_KEY_ERROR()`.

Effects are session scoped.

Using Sequences

Sequences are database objects that generate unique numbers in sequential order. They are most often used when an application requires a unique identifier in a table; for example you can use sequences as primary/unique keys because sequences are always unique. Once a sequence returns a value, it never returns that same value again.

When a sequence number is generated, the sequence is incremented or decremented irrespective of any committed or rolled back transactions. The increment or decrement value is specified when the sequence is created. The default is 1.

Sequence number values are accessed by calling the **NEXTVAL** (<http://www.postgresql.org/docs/8.0/static/app-vacuumdb.html>) function, which increments/decrements the current sequence and returns the *next* value, and CURRVAL function, which returns the *current* value. These functions can also be used in INSERT and COPY expressions.

Auto increment is a setting available for numeric column types to automatically assign the next incremental sequence value for that column when a new row is added to the table.

Sequences can be dropped and altered to change the parameters of the sequence. It is possible to rename a sequence within the same schema and move a sequence between schemas.

Sequence DDL commands and functions

For details, see the following related statements and functions in the SQL Reference Manual:

- To create a sequence, run the CREATE SEQUENCE statement.
- To alter a sequence, run the ALTER SEQUENCE statement to define a new sequence number generator.
- To drop a sequence, run the DROP SEQUENCE statement. This is useful when a sequence is no longer needed.
- To grants privileges on a sequence generator to a user, run the GRANT SEQUENCE statement. See also **Sequence Privileges** (page 138).

Sequences are used via two functions:

- CURRVAL — For a sequence generator, returns the LAST value across all nodes returned by a previous invocation of NEXTVAL in the same session. If there were no calls to NEXTVAL, an error is returned; for example:

```
ERROR: Sequence seq2 has not been accessed in the session
```
- NEXTVAL — Advances the sequence and returns the new value from the sequence. This value is incremented for ascending sequences and decremented for descending sequences.

Identity and auto-increment columns

Identity and auto-increment columns are defined through column constraints in the CREATE TABLE statement and are incremented each time a row is added to the table. The identity value is never rolled back even if the transaction that tries to insert a value into the table is not committed.

The LAST_INSERT_ID function returns the last value generated for an auto-increment or identity column.

Notes

- While sequences are guaranteed to be unique, the sequence values are not guaranteed to be in ascending order. For example, if you use numbers on one node that are lower than the numbers used by other nodes, the newly assigned sequence values are not necessarily higher than the numbers assigned on other nodes.
- Sequence values are not guaranteed to be contiguous, which means there could be gaps in the values.

Creating Sequences

Create a sequence using the CREATE SEQUENCE statement. All of the parameters (besides a sequence name) are optional.

The following example creates an ascending sequence called `sequential`, starting at 101:

```
=> CREATE SEQUENCE my_seq START 101;
```

After a sequence is created, use the sequence functions NEXTVAL and CURRVAL to operate on the sequence. These functions provide simple, multiuser-safe methods for obtaining successive sequence values from sequence objects.

Note:

CURRVAL returns a sequence's most recent value, so if you run CURRVAL before NEXTVAL, the system returns an error:

```
ERROR: Sequence my_seq has not been accessed in the session
```

NEXTVAL must be called at least one time in a session to provide a value for CURRVAL. A cache is created when NEXTVAL is called.

The following command generates the first number for this sequence:

```
=> SELECT NEXTVAL('my_seq');
nextval
-----
      101
(1 row)
```

The following command returns the current value of this sequence. Since no other operations have been performed on the newly-created sequence, the function returns the expected value of 101:

```
=> SELECT CURRVAL('my_seq');
currval
-----
      101
(1 row)
```

The following command increments the value for this sequence by one (1):

```
=> SELECT NEXTVAL('my_seq');
nextval
-----
      102
(1 row)
```

Calling the CURRVAL again function returns only the current value:

```
=> SELECT CURRVAL('my_seq');
currval
-----
      102
```

(1 row)

The following example shows how to use the `my_sequence` sequence in an INSERT statement.

```
=> CREATE TABLE customer (
    lname VARCHAR(25),
    fname VARCHAR(25),
    membership_card INTEGER,
    ID INTEGER
);
=> INSERT INTO customer VALUES ('Hawkins', 'John', 072753, NEXTVAL('my_seq'));
```

Now query the table you just created. Notice that the ID column has been incremented 1 value to 103:

```
=> SELECT * FROM customer;
  lname | fname | membership_card | ID
-----+-----+-----+-----
Hawkins | John  |           72753 | 103
(1 row)
```

The following example shows how to use a sequence as the default value for an INSERT command:

```
=> CREATE TABLE customer2(
    ID INTEGER DEFAULT NEXTVAL('my_seq'),
    lname VARCHAR(25),
    fname VARCHAR(25),
    membership_card INTEGER
);
=> INSERT INTO customer2 VALUES (default, 'Carr', 'Mary', 87432);
```

Now query the table you just created. The ID column has been incremented by (1) again to 104:

```
=> SELECT * FROM customer2;
  ID | lname | fname | membership_card
-----+-----+-----+-----
 104 | Carr  | Mary  |           87432
(1 row)
```

The following example shows how to use NEXTVAL in a SELECT statement:

```
=> SELECT NEXTVAL('my_seq'), lname FROM customer2;
NEXTVAL | lname
-----+-----
    105 | Carr
(1 row)
```

As you can see, each time NEXTVAL is called, the value increments by 1.

The following example shows how to use CURRVAL in a SELECT statement:

```
=> SELECT CURRVAL('my_seq'), lname FROM customer2;
CURRVAL | lname
-----+-----
    105 | Carr
(1 row)
```

The value doesn't change above because the CURRVAL function returns only the current value.

Altering Sequences

The ALTER SEQUENCE statement lets you change the attributes of a previously-defined sequence. Changes take effect in the next session. Any parameters not specifically set in the ALTER SEQUENCE command retain their previous settings.

Note: Using ALTER SEQUENCE to set a START value below the CURRVAL can result in duplicate keys.

Examples

The following example modifies an ascending sequence called sequential to start at 105:

```
ALTER SEQUENCE sequential RESTART WITH 105;
```

The following example moves a sequence from one schema to another:

```
ALTER SEQUENCE [public.]sequence SET SCHEMA vmart;
```

The following example renames a sequence in the Vmart schema:

```
ALTER SEQUENCE [vmart.]sequence RENAME TO serial;
```

Remember that changes occur only after you start a new session. For example, if you create a sequence named `my_sequence` and start the value at 10, each time you call the NEXTVAL function, you increment by 1, as in the following series of commands:

```
CREATE SEQUENCE my_sequence START 10;
SELECT NEXTVAL('my_sequence');
  nextval
-----
         10
(1 row)
SELECT NEXTVAL('my_sequence');
  nextval
-----
         11
(1 row)
```

Now issue the ALTER SEQUENCE statement to assign a new value starting at 50:

```
ALTER SEQUENCE my_sequence START 50;
```

When you call the NEXTVAL function, the sequence is incremented again by 1 value:

```
NEXTVAL
-----
         12
(1 row)
```

The sequence starts at 50 only after restarting the session:

```
SELECT NEXTVAL('my_sequence');
  NEXTVAL
-----
         50
(1 row)
```

Distributed Sequences

The CACHE parameter is used to control the efficiency of a sequence. In other databases, when NEXTVAL is called, the state of the sequence is cached within a session. The state is available across statements and transactions. In Vertica, a session is distributed across all nodes. When the NEXTVAL() function is called on two different nodes when executing a SQL statement, each node creates and maintains its own cache of values per session.

The current value of a sequence is calculated as follows:

- At the end of every statement, the state of all sequences used in the session is sent back to the initiator node.
- The initiator node calculates the maximum CURRVAL of each sequence across all states on all nodes.
- This maximum value is used as CURRVAL in subsequent statements until another NEXTVAL is invoked.

The behavior of sequences across Vertica nodes is explained in the following examples.

Note: IDENTITY and AUTO_INCREMENT columns behave in a similar manner.

Example 1: The following example, which illustrates sequence distribution, assumes a 3-node cluster with node01 as the initiator node.

First create a simple table called dist:

```
CREATE TABLE dist (i INT, j VARCHAR);
```

Create a projection called oneNode and segment by column i on node01:

```
CREATE PROJECTION oneNode AS SELECT * FROM dist
SEGMENTED BY i NODES node01;
```

Create a second projection called twoNodes and segment column x by modularhash on node02 and node03:

```
CREATE PROJECTION twoNodes AS SELECT * FROM dist
SEGMENTED BY MODULARHASH(i) NODES node02, node03;
```

Create a third projection called threeNodes and segment column i by modularhash on all nodes (1-3):

```
CREATE PROJECTION threeNodes as SELECT * FROM dist
SEGMENTED BY MODULARHASH(i) ALL NODES;
```

Insert some values:

```
COPY dist FROM STDIN;
1|ONE
2|TWO
3|THREE
4|FOUR
5|FIVE
6|SIX
\.
```

Query the STORAGE_CONTAINERS table to return the projections on each node:

```
SELECT node_name, projection_name, total_row_count FROM storage_containers;
node_name | projection_name | total_row_count
-----+-----+-----
node0001  | oneNode         |                6  --Contains rows with i=(1,2,3,4,5,6)
node0001  | threeNodes      |                2  --Contains rows with i=(3,6)
node0002  | twoNodes        |                3  --Contains rows with i=(2,4,6)
node0002  | threeNodes      |                2  --Contains rows with i=(1,4)
node0003  | twoNodes        |                3  --Contains rows with i=(1,3,5)
node0003  | threeNodes      |                2  --Contains rows with i=(2,5)
(6 rows)
```

The following table shows the segmentation of rows for projection `oneNode`:

1	ONE	Node01
2	TWO	Node01
3	THREE	Node01
4	FOUR	Node01
5	FIVE	Node01
6	SIX	Node01

The following table shows the segmentation of rows for projection `twoNodes`:

1	ONE	Node03
2	TWO	Node02
3	THREE	Node03
4	FOUR	Node02
5	FIVE	Node03
6	SIX	Node02

The following table shows the segmentation of rows for projection `threeNodes`:

1	ONE	Node02
2	TWO	Node03
3	THREE	Node01
4	FOUR	Node02
5	FIVE	Node03
6	SIX	Node01

Create a sequence and specify a cache of 10. The sequence will cache up to 10 values in memory for performance. As per the CREATE SEQUENCE statement, the minimum value is 1 (only one value can be generated at a time, for example, no cache).

Cache operations:

- In each session, every node maintains its own cache of the sequence state and once those values are consumed, a catalog lock is taken in order to obtain a new set of cached values.
- It is possible for one session to allocate a cache and use it slowly while another statement requests and loads many values. Therefore, the values returned from NEXTVAL in one statement could be distant from the values returned in another statement.
- Regardless of the number of calls to NEXTVAL and CURRVAL, sequences are incremented only once per row. This means multiple calls to NEXTVAL within the same row return the same value. If joins are used, a sequence is incremented one time for each composite row output by the join.
- If a statement fails after NEXTVAL is called (thereby consuming a sequence value from the cache), the value is lost.

- If a disconnect occurs (for example, dropped session), any remaining values in the cache that have not been returned through NEXTVAL (unused) are lost.
- To recover the lost sequence values, you could run an ALTER SEQUENCE command to define a new sequence number generator, which resets the counter to the correct value.

Example 2: Create a sequence named s1 and specify a cache of 10:

```
CREATE SEQUENCE s1 cache 10;
SELECT s1.nextval, s1.currval, s1.nextval, s1.currval, j FROM oneNode;
  nextval | currval | nextval | currval |   j
-----+-----+-----+-----+---
         1 |         1 |         1 |         1 | ONE
         2 |         2 |         2 |         2 | TWO
         3 |         3 |         3 |         3 | THREE
         4 |         4 |         4 |         4 | FOUR
         5 |         5 |         5 |         5 | FIVE
         6 |         6 |         6 |         6 | SIX
(6 rows)
```

The following table illustrates the current state of the sequence for that session. It holds the current value, values remaining (the difference between the current value (6) and the cache (10)), and cache activity. There is no cache activity on node02 or node03.

Sequence Cache State	Node01	Node02	Node03
Current value	6	NO CACHE	NO CACHE
Remainder	4	NO CACHE	NO CACHE

Example 3: Return the current values from twoNodes:

```
SELECT s1.currval, j FROM twoNodes;
  currval |   j
-----+---
         6 | ONE
         6 | THREE
         6 | FIVE
         6 | TWO
         6 | FOUR
         6 | SIX
(6 rows)
```

Example 4: Now call NEXTVAL from threeNodes. The assumption is that node02 holds the cache before node03:

```
SELECT s1.nextval, j from threeNodes;
  nextval |   j
-----+---
       101 | ONE
       201 | TWO
         7 | THREE
       102 | FOUR
       202 | FIVE
```

```

      8 | SIX
(6 rows)

```

The following table illustrates the sequence cache state with values on node01, node02, and node03:

Sequence Cache State	Node01	Node02	Node03
Current value	8	102	202
Left	2	8	8

Example 5: Insert values from twoNodes into the destination table:

```

SELECT s1.currval, j FROM twoNodes;
 nextval |    j
-----+-----
      202 | ONE
      202 | TWO
      202 | THREE
      202 | FOUR
      202 | FIVE
      202 | SIX
(6 rows)

```

The following table illustrates the sequence cache state:

Sequence Cache State	Node01	Node02	Node03
Current value	6	102	202
Left	4	8	8

Example 6: The following command runs on node02 only:

```

SELECT s1.nextval, j FROM twoNodes WHERE i = 2;
 nextval |    j
-----+-----
      103 | TWO
(1 row)

```

The following table illustrates the sequence cache state:

Sequence Cache State	Node01	Node02	Node03
Current value	6	103	202
Left	4	7	8

Example 7: The following command calls the current value from twoNodes:

```

SELECT s1.currval, j FROM twoNodes;
 currval |    j
-----+-----

```

```

-----+-----
      103 | ONE
      103 | TWO
      103 | THREE
      103 | FOUR
      103 | FIVE
      103 | SIX
(6 rows)

```

Example 8: This example assume that node02 holds the cache before node03:

```

SELECT s1.nextval, j FROM twoNodes;
nextval |  j
-----+-----
      203 | ONE
      104 | TWO
      204 | THREE
      105 | FOUR
      205 | FIVE
      106 | SIX
(6 rows)

```

The following table illustrates the sequence cache state:

Sequence Cache State	Node01	Node02	Node03
Current value	6	106	205
Left	4	6	5

Example 9: The following command calls the current value from oneNode:

```

SELECT s1.currval, j FROM twoNodes;
currval |  j
-----+-----
      205 | ONE
      205 | TWO
      205 | THREE
      205 | FOUR
      205 | FIVE
      205 | SIX
(6 rows)

```

Example 10: This example calls the NEXTVAL function on oneNode:

```

SELECT s1.nextval, j FROM oneNode;
nextval |  j
-----+-----
        7 | ONE
        8 | TWO
        9 | THREE
       10 | FOUR
      301 | FIVE
      302 | SIX
(6 rows)

```

The following table illustrates the sequence cache state:

Sequence Cache State	Node01	Node02	Node03
Current value	302	106	205
Left	8	4	5

Example 11: In this example, twoNodes is the outer table and threeNodes is the inner table to a merge join. threeNodes is resegmented as per twoNodes.

```
SELECT s1.nextval, j FROM twoNodes JOIN threeNodes ON twoNodes.i = threeNodes.i;
SELECT s1.nextval, j FROM oneNode;
  nextval |    j
-----+-----
      206 | ONE
      107 | TWO
      207 | THREE
      108 | FOUR
      208 | FIVE
      109 | SIX
(6 rows)
```

The following table illustrates the sequence cache state:

Sequence Cache State	Node01	Node02	Node03
Current value	302	109	208
Left	8	1	2

Example 12: This next example shows how sequences work with buddy projections.

```
--Same session
DROP TABLE t CASCADE;
CREATE TABLE t (i INT, j varchar(20));
CREATE PROJECTION threeNodes AS SELECT * FROM t
SEGMENTED BY MODULARHASH(i) ALL NODES KSAFE 1;
COPY t FROM STDIN;
1|ONE
2|TWO
3|THREE
4|FOUR
5|FIVE
6|SIX
\.
SELECT node_name, projection_name, total_row_count FROM storage_containers;
  node_name | projection_name | total_row_count
-----+-----+-----
 node01    | threeNodes_b0  |                2
 node03    | threeNodes_b0  |                2
 node02    | threeNodes_b0  |                2
 node02    | threeNodes_b1  |                2
 node01    | threeNodes_b1  |                2
 node03    | threeNodes_b1  |                2
```

(6 rows)

The following function call assumes that node02 is down. It is the same session. Node03 takes up the work of node02:

```
SELECT s1.nextval, j FROM t;
  nextval |    j
-----+-----
         401 | ONE
         402 | TWO
         305 | THREE
         403 | FOUR
         404 | FIVE
         306 | SIX
```

(6 rows)

The following table illustrates the sequence cache state:

Sequence Cache State	Node01	Node02	Node03
Current value	306	110	404
Left	4	0	6

Example 13: This example starts a new session.

```
DROP TABLE t CASCADE;
CREATE TABLE t (i INT, j VARCHAR);
CREATE PROJECTION oneNode AS SELECT * FROM t SEGMENTED BY i NODES node01;
CREATE PROJECTION twoNodes AS SELECT * FROM t SEGMENTED BY MODULARHASH(i) NODES
node02, node03;
CREATE PROJECTION threeNodes AS SELECT * FROM t SEGMENTED BY MODULARHASH(i) ALL
NODES;
INSERT INTO t values (nextval('s1'), 'ONE');
SELECT * FROM t;
  i | j
-----+-----
  501 | ONE
(1 rows)
```

The following table illustrates the sequence cache state:

Sequence Cache State	Node01	Node02	Node03
Current value	501	NO CACHE	NO CACHE
Left	9	0	0

Example 14:

```
INSERT INTO t SELECT s1.nextval, 'TWO' FROM twoNodes;
SELECT * FROM t;
  i | j
-----+-----
```

```

501 | ONE  --stored in node01 for oneNode, node02 for twoNodes, node02 for
threeNodes
601 | TWO  --stored in node01 for oneNode, node03 for twoNodes, node01 for
threeNodes
(2 rows)

```

The following table illustrates the sequence cache state:

Sequence Cache State	Node01	Node02	Node03
Current value	501	601	NO CACHE
Left	9	9	0

Example 15:

```

INSERT INTO t select s1.nextval, 'TRE' from threeNodes;
SELECT * FROM t;
  i      | j
-----+-----
501 | ONE  --stored in node01 for oneNode, node02 for twoNodes, node02 for
threeNodes
601 | TWO  --stored in node01 for oneNode, node03 for twoNodes, node01 for
threeNodes
502 | TRE  --stored in node01 for oneNode, node03 for twoNodes, node03 for
threeNodes
602 | TRE  --stored in node01 for oneNode, node02 for twoNodes, node02 for
threeNodes
(4 rows)

```

The following table illustrates the sequence cache state:

Sequence Cache State	Node01	Node02	Node03
Current value	502	602	NO CACHE
Left	9	9	0

Example 16:

```

INSERT INTO t SELECT s1.currval, j FROM threeNodes WHERE i != 502;
select * from t;
  i      | j
-----+-----
501 | ONE  --stored in node01 for oneNode, node02 for twoNodes, node02 for
threeNodes
601 | TWO  --stored in node01 for oneNode, node03 for twoNodes, node01 for
threeNodes
502 | TRE  --stored in node01 for oneNode, node03 for twoNodes, node03 for
threeNodes
602 | TRE  --stored in node01 for oneNode, node02 for twoNodes, node02 for
threeNodes
602 | ONE  --stored in node01 for oneNode, node02 for twoNodes, node02 for
threeNodes
502 | TWO  --stored in node01 for oneNode, node03 for twoNodes, node03 for
threeNodes

```

```

602 | TRE --stored in node01 for oneNode, node02 for twoNodes, node02 for
threeNodes
(7 rows)

```

The following table illustrates the sequence cache state:

Sequence Cache State	Node01	Node02	Node03
Current value	502	602	NO CACHE
Left	9	9	0

Example 17:

```

INSERT INTO t VALUES (s1.currval + 1, 'QUA');
SELECT * FROM t;
  i      | j
-----+-----
501 | ONE --stored in node01 for oneNode, node02 for twoNodes, node02 for
threeNodes
601 | TWO --stored in node01 for oneNode, node03 for twoNodes, node01 for
threeNodes
502 | TRE --stored in node01 for oneNode, node03 for twoNodes, node03 for
threeNodes
602 | TRE --stored in node01 for oneNode, node02 for twoNodes, node02 for
threeNodes
602 | ONE --stored in node01 for oneNode, node02 for twoNodes, node02 for
threeNodes
502 | TWO --stored in node01 for oneNode, node03 for twoNodes, node03 for
threeNodes
602 | TRE --stored in node01 for oneNode, node02 for twoNodes, node02 for
threeNodes
603 | QUA
(8 rows)

```

The following table illustrates the sequence cache state:

Sequence Cache State	Node01	Node02	Node03
Current value	502	602	NO CACHE
Left	9	9	0

Loading Sequences

The following example shows how to use a sequence as the default value for an INSERT command:

```

CREATE TABLE customer2(
  ID INTEGER DEFAULT NEXTVAL('my_seq'),
  lname VARCHAR(25),
  fname VARCHAR(25),
  membership_card INTEGER

```

```
);  
INSERT INTO customer2 VALUES (default,'Carr', 'Mary', 87432);
```

Now query the table you just created. The ID column has been incremented by (1) again to 104:

```
SELECT * FROM customer2;  
  ID | lname | fname | membership_card  
-----+-----+-----+-----  
 104 | Carr  | Mary  |             87432  
(1 row)
```

Dropping Sequences

Use `DROP SEQUENCE` to remove a sequence.

Notes:

- You cannot drop a sequence upon which another objects depends unless `CASCADE` is specified.
- The `CASCADE` keyword is not supported. Sequences used in a default expression of a column cannot be dropped until all references to the sequence are removed from the default expression.

Example

The following command drops the sequence named `my_sequence`:

```
=> DROP SEQUENCE my_sequence;
```

Implementing Views

A view is a stored query that dynamically accesses and computes data from the database at execution. It differs from a projection in that it is not materialized: it does not store data on disk. This means that it doesn't need to be refreshed whenever the data in the underlying tables change, but it does require additional time to access and compute data.

Views are read-only and they support references to tables, temp tables, and other views. They do not support inserts, deletes, or updates. You can use a view as an abstraction mechanism to:

- Hide the complexity of `SELECT` statements from users for support or security purposes. For example, you could create a view that selects specific columns from specific tables to ensure that users have easy access to the information they need while restricting them from confidential information.
- Encapsulate the details of the structure of your tables, which could change as your application evolves, behind a consistent user interface.

See Also

Flattening FROM Clause Subqueries and Views in the Programmer's Guide

Creating Views

A view contains one or more `SELECT` statements that reference any combination of one or more tables, temp tables, or views. Additionally, views can specify the column names used to display results.

The user who creates the view must be a superuser or have the following privileges:

- CREATE on the schema in which the view is created.
- SELECT on all the tables and views referenced within the view's defining query.
- USAGE on all the schemas that contain the tables and views referenced within the view's defining query.

To create a view:

- 1 Use the CREATE VIEW statement to create the view.
- 2 Use the GRANT (View) statement to grant users the privilege to use the view.

Note: Once created, a view cannot be actively altered. It can only be deleted and recreated.

Using Views

Views can be used in the FROM clause of any SQL query or subquery. At execution, Vertica internally substitutes the name of the view used in the query with the actual contents of the view. The following example defines a view (`ship`) and illustrates how a query that refers to the view is transformed internally at execution.

- **New view**

```
=> CREATE VIEW ship AS SELECT * FROM public.shipping_dimension;
```

- **Original query**

```
=> SELECT * FROM ship;
```

- **Transformed query**

```
=> SELECT * FROM (SELECT * FROM public.shipping_dimension) AS ship;
```

Tip: To use a view, a user must be granted SELECT permissions on the view. See GRANT (View).

The following example creates a view named `myview` that sums all individual incomes of customers listed in the `store.store_sales_fact` table by state. The results are grouped in ascending order by state.

```
=> CREATE VIEW myview AS
SELECT SUM(annual_income), customer_state
FROM public.customer_dimension
WHERE customer_key IN
  (SELECT customer_key
   FROM store.store_sales_fact)
GROUP BY customer_state
ORDER BY customer_state ASC;
```

The following example uses the `myview` view with a WHERE clause that limits the results to combined salaries of greater than 2,000,000,000.

```
=> SELECT * FROM myview where sum > 2000000000;
```

```

SUM      | customer_state
-----+-----
2723441590 | AZ
```

```
29253817091 | CA
4907216137 | CO
3769455689 | CT
3330524215 | FL
4581840709 | IL
3310667307 | IN
2793284639 | MA
5225333668 | MI
2128169759 | NV
2806150503 | PA
2832710696 | TN
14215397659 | TX
2642551509 | UT
(14 rows)
```

Notes

If Vertica does not have to evaluate an expression that would generate a runtime error in order to answer a query, the run-time error might not occur. See the following sequence of commands for an example of this scenario.

If you run a query like the following, Vertica returns an error:

```
=> SELECT TO_DATE('F','dd mm yyyy') FROM customer_dimension;
      ERROR: Invalid input for DD: "F"
```

Now create a view using the same query. Note that the view gets created when you would expect it to return the same error:

```
=> CREATE VIEW temp AS SELECT TO_DATE('F','dd mm yyyy') FROM customer_dimension;
CREATE VIEW
```

The view, however, cannot be used in all queries without generating the same error message. For example, the following query returns the same error, which is what you would expect:

```
=> SELECT * FROM temp;
      ERROR: Invalid input for DD: "F"
```

When you then issue a COUNT command, the returned rowcount is correct:

```
=> SELECT COUNT(*) FROM temp;
      count
-----
         100
(1 row)
```

This behavior works as intended. You might want to create views that contain subqueries, where not every row is intended to pass the predicate.

See Also

Flattening Subqueries and Views in the FROM Clause in the Programmer's Guide

Altering Tables

You can use the ALTER TABLE statement to:

- Add new columns to tables

Add table constraints Adding a new column to a table:

- Automatically adds the new column with a unique column name to all superprojections of the table
- Populates the column according to the column-constraint (DEFAULT for example).
- Does not affect the K-safety of the physical schema design.

For more information about ALTER TABLE, see the SQL Reference Manual.

Creating a Physical Design

Data in Vertica is physically stored in projections. When you initially load data into a table using INSERT, COPY, or LCOPY, Vertica creates a default superprojection for the table. This superprojection ensures that all of the data is available for queries. However, these default superprojections might not optimize database performance, resulting in slow query performance and low data compression.

To improve performance, you should create a physical design for your database that will optimize both query performance and data compression. You can create this design by hand or by using the Database Designer.

Vertica recommends that you load sample data and then use the Database Designer to optimize your database. Database Designer recommends new projections that optimize your database based on its data statistics and the queries you provide.

Using the Database Designer

The Database Designer analyzes a logical schema definition, sample queries, and sample data, and creates a physical schema (a set of projections) in the form of a SQL script that can be deployed automatically or manually. The script creates a minimal set of projections to ensure K-Safety.

The Database Designer can create a comprehensive design, which replaces the existing design for your database, or a query-specific design, which adds projections to your design to optimize a single query (see **Design Types** (page 78) for details).

The Database Designer needs representative data to analyze in order to create the most efficient projections for your database. You should load a moderate amount of data for each table into the database before running the Database Designer. Loading too much data (over 10GB or so per table) significantly slows the design process and is unlikely to result in a better design.

If you already have queries that you will want to run on your data, you can also supply them to the Database Designer so it can attempt to optimize the projections for them.

In most cases, the sophisticated algorithms used by the Database Designer result in a design that provides excellent query performance within physical constraints while using disk space efficiently.

Vertica recommends that you first create a design using the Database Designer. If you find that the performance of this design is not adequate, you can design **custom projections** (page 91) with assistance from Vertica.

Design Types

The Database Designer provides two design types: comprehensive and query specific. The design you choose depends on what you are trying to accomplish.

Comprehensive Design

A comprehensive design creates an initial or replacement design for all the tables in the specified schemas. You should create a comprehensive design when you are creating a new database. You can also create a new comprehensive design for an existing database if you want to optimize its performance.

To help the Database Designer create an efficient design, load representative data into the database before you begin the comprehensive design process. You can also supply the Database Designer with queries you will be performing on your data so that the Database Designer can optimize the design for them. The Database Designer considers only the first 100 queries you supply to it; if you have more than 100 queries, ensure that the sample you supply is representative of all the types of queries you plan to run on your database.

The comprehensive design flow lets you select several options that control how the Database Designer generates the design and what it does with it:

- Optimize with queries: Lets you supply queries for which the Database Designer should optimize the design.
- Update statistics: Collects or refreshes statistics about the data in the database. Accurate statistics help the Database Designer optimize the compression and query performance of the database. By selecting this option, database statistics are updated to maximize design quality.
Note: Updating statistics takes time and resources. If the current statistics are up to date, this step is unnecessary. When in doubt, update the statistics.
- Deploy design: Deploys the new database design to your database. During deployment, new projections are added, some existing projections might be retained, and any unnecessary existing projections are removed. Any new projections are refreshed so they are populated with data. If you decide not to deploy the design, Database Designer saves the SQL script for the new design so you can review it and deploy it manually later.

The Database Designer also lets you choose how you want your database optimized:

- Optimized for query performance, so that the queries run faster. This could result in a larger database storage footprint because additional projections might be created.
- Optimized for load performance, so the size of the database is minimized. This could result in slower query performance.
- Balanced optimization, which balances between database size and query performance.

For details, see [Deploying Designs Using the Database Designer](#) in this guide.

Query-specific Design

A query-specific design creates an enhanced design with additional projections that are optimized specifically for the query you provide. Create a query-specific design when you have a query that you want to optimize.

The query-specific design process lets you specify the following options:

- Update statistics: Collects or refreshes statistics about the data in the database. Accurate statistics help the Database Designer optimize the compression and query performance of the database. By selecting this option, database statistics are updated to maximize design quality.

Note: Updating statistics takes time and resources, so if the current statistics are up to date, this is unnecessary. When in doubt, update the statistics.

- Deploy design: Deploys the new database design. New projections are added to the database and refreshed so they are populated with data. No existing projections are affected by the deployment.

For details, see [Creating a Query-specific Design Using the Database Designer](#) in this guide.

Creating a Comprehensive Design Using the Database Designer

You'll want to create a comprehensive design for a new database after you have loaded representative data into it. You can also use the comprehensive design process to redesign a database whenever you need (for example, after you have made significant changes to the database's schemas). The Database Designer creates a complete initial or replacement physical schema design based on data statistics and queries. It can create segmented superprojections for large tables when deploying to multiple node clusters, and replicated superprojections for smaller tables.

Note: If you just have a query for which you want to optimize your existing database design, you should use the Database Designer to create a query-specific design. See [Creating a Query-specific Design Using the Database Designer](#) (page 87) for details.

This procedure guides you through creating a comprehensive design and assumes you have already performed the following prerequisite steps:

- Set up the example environment
- Created the example database
- Defined the database schema
- Loaded the data

If you have not performed the above steps, refer to the Tutorial in the Getting Started Guide.

- 1 Type `\q` to exit the vsql session and return to the Main Menu in the Administration Tools.

Alternatively, restart the Administrative Tools:

```
$ /opt/vertica/bin/admintools
```

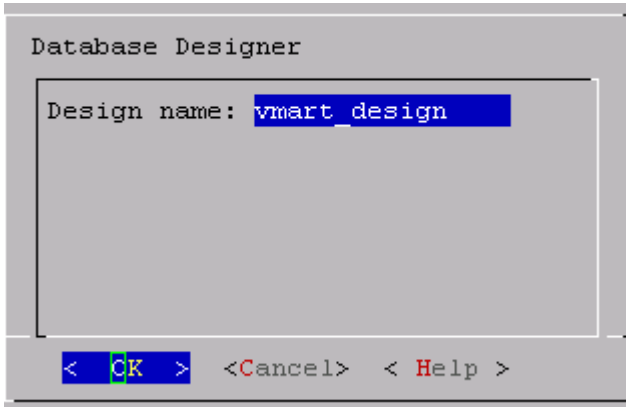
- 2 From the **Main Menu**, click **Configuration Menu** and click **OK**.
- 3 From the **Configuration Menu**, click **Run Database Designer**, and click **OK**.
- 4 Select **vmartdb** as the database and click **OK**.

If you are asked to enter the password for the database, click **OK** to bypass. No password was assigned in Step 2: Create the Example Database, so you do not need to enter one now.

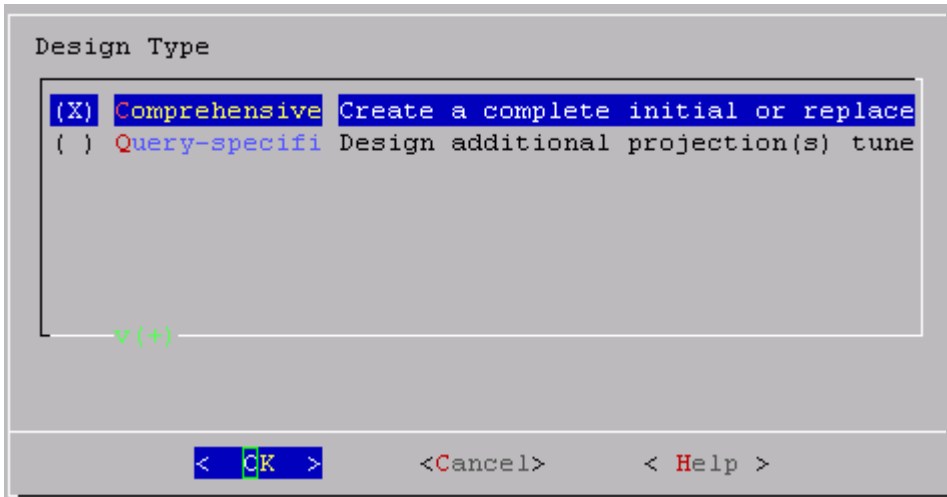
- 5 Click **OK** to accept the default directory for storing Database Designer output and log files. **Note this location.**

Note: If you choose to not deploy your design now, the Database Designer saves the SQL script to implement the design in this directory where you can review and manually deploy it later.

- 6 In the **Database Designer** window, enter a name for the design (this example uses **vmart_design**) and click **OK**.



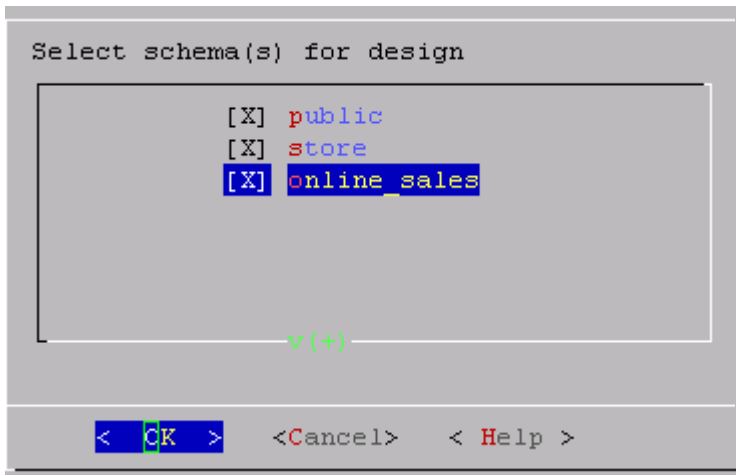
- 7 In the **Design Type** window, click **Comprehensive** to create a complete initial design, and click **OK**.



- 8 Select the schemas for your design, and click **OK**.

If you include a schema that contains tables without data, the Administration Tools returns a message notifying you that designing for tables without data could be suboptimal. You can choose to continue, but Vertica recommends that you click Cancel and deselect the schemas that contain empty tables before you proceed.

Note: In this example, the Vmart design is a multi-schema database, so be sure to select all three options: public, store, and online_sales

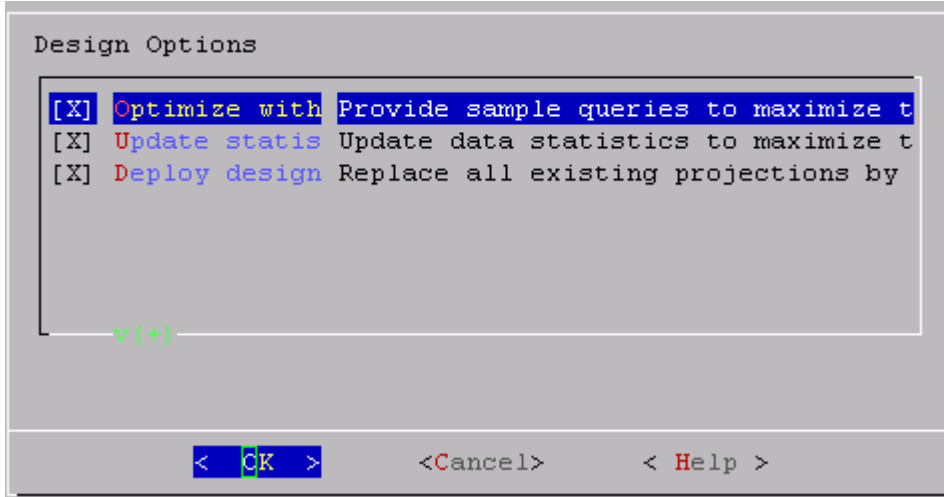


- 9 In the **Design Options** window, accept the default of all three options described below and click **OK**.

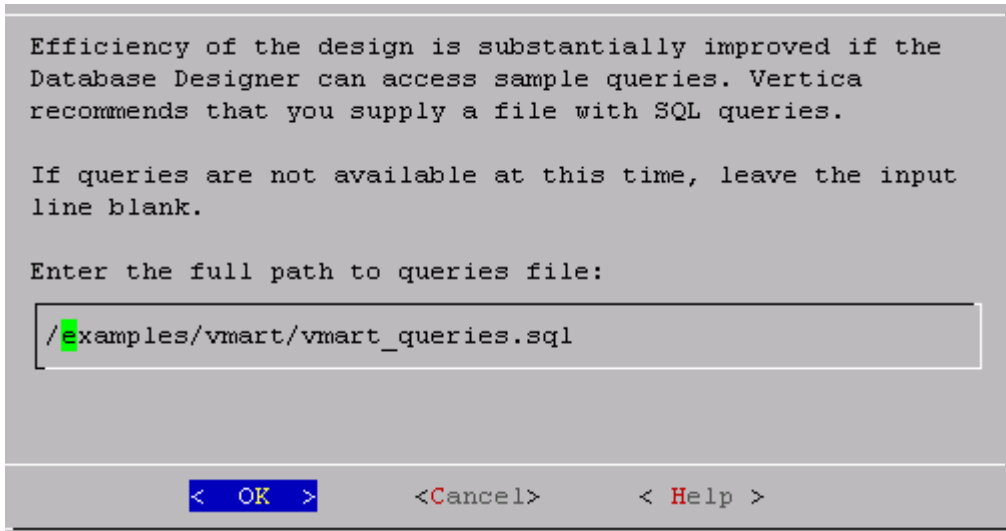
Generally, you want to accept the default of enabling all three because the Database Designer is best positioned to generate a new comprehensive design and create a complete set of projections for the tables in the selected schema. The three options are:

- *Optimize with queries:* Efficiency of the design is substantially improved if the Database Designer can access sample queries.
Supplying the Database Designer with queries is especially important if you want to optimize the database design for query performance.
- *Update statistics:* Accurate statistics help the Database Designer choose the best strategy for data compression. If you select this option, the database statistics are updated to maximize design quality.
Note that updating statistics takes time and resources, so if the current statistics are up to date, this step is unnecessary. When in doubt, update statistics.
- *Deploy design:* The new design will be automatically deployed, which means that during deployment, new projections are added, some existing projections might be retained, and any unnecessary existing projections are removed. Any new projections are refreshed so that they are populated with data.

Note: For large databases, a full design session could take a long time, yet it is best to allow this process to complete uninterrupted. If the session must be canceled, use CTRL+C.

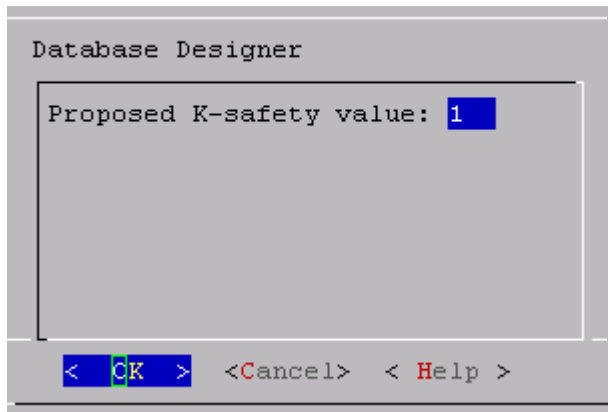


- 10 If you selected the Optimize with queries option, you are prompted for the query file. Type the full path to the file containing the queries that will be run on your database. In this example it is:
/examples/vMart_Schema/vmart_queries.sql



- 11 Choose the **K-safety value** you want. In this example, it is 1. Click **OK**.

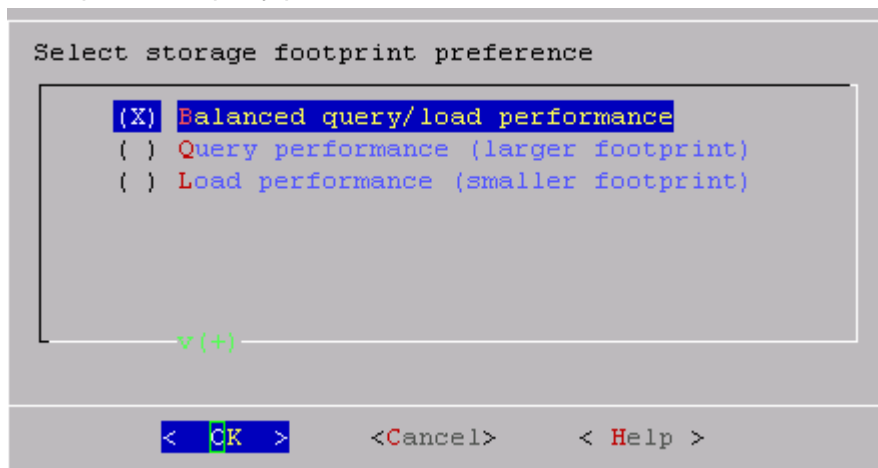
Note: There will be no K-safe form if you are creating a comprehensive design on a single node. In that case, you can skip this step.



- 12** Choose the Database Designer's priority for the design (in this procedure choose **Balanced**) and click **OK**.

The options are:

- *Balanced* query/load performance tells the Database Designer to create a design that is balanced between database size and query performance.
- *Query* load performance creates a design focused on faster query performance, which might recommend additional projections. These projections could result in a larger database storage size.
- *Load* performance is optimized for loads, minimizing size of the database, potentially at the expense of query performance.



- 13** When the informational message displays, click **Proceed**.

The Database Designer:

- Sets up the design session
- Examines table data
- Loads queries from the query file you provided
- Creates the design

- Deploys the design or saves a SQL file containing the design, depending on what you selected for the Deploy design option in step 9.

You can watch the progress on the terminal window. The following image is just an example and might not match exactly what you see:

```
Creating design...
[ 6%] Analyzing data statistics... Completed 1 of 15 tables. Analyzing p
[ 13%] Analyzing data statistics... Completed 2 of 15 tables. Analyzing p
[ 13%] Analyzing data statistics... Completed 2 of 15 tables. Analyzing p
[ 26%] Analyzing data statistics... Completed 4 of 15 tables. Analyzing p
[ 40%] Analyzing data statistics... Completed 6 of 15 tables. Analyzing p
[ 66%] Analyzing data statistics... Completed 10 of 15 tables. Analyzing
[ 73%] Analyzing data statistics... Completed 11 of 15 tables. Analyzing
[ 93%] Analyzing data statistics... Completed 14 of 15 tables. Analyzing
[100%] Analyzing data statistics... Completed 15 of 15 tables. Analyzing
[100%] Analyzing data statistics... Completed 15 of 15 tables.

[ 0%] Optimizing for query performance... Completed 0 of 9 queries. Sett
[ 0%] Optimizing for query performance... Completed 0 of 9 queries. Sett
[ 0%] Optimizing for query performance... Completed 0 of 9 queries. Choo
[ 0%] Optimizing for query performance... Completed 0 of 9 queries. Choo
[ 66%] Optimizing for query performance... Completed 6 of 9 queries. Choo
[100%] Optimizing for query performance... Completed 9 of 9 queries.

[ 0%] Optimizing storage footprint... Completed 0 of 15 tables. Optimizi
[ 6%] Optimizing storage footprint... Completed 1 of 15 tables. Optimizi
[ 6%] Optimizing storage footprint... Completed 1 of 15 tables. Optimizi
[ 26%] Optimizing storage footprint... Completed 4 of 15 tables. Optimizi
[ 46%] Optimizing storage footprint... Completed 7 of 15 tables. Optimizi
[ 60%] Optimizing storage footprint... Completed 9 of 15 tables. Optimizi
[ 66%] Optimizing storage footprint... Completed 10 of 15 tables. Optimiz
[ 66%] Optimizing storage footprint... Completed 10 of 15 tables. Optimiz
[ 66%] Optimizing storage footprint... Completed 10 of 15 tables. Optimiz
[ 73%] Optimizing storage footprint... Completed 11 of 15 tables. Optimiz
[ 73%] Optimizing storage footprint... Completed 11 of 15 tables. Optimiz
[ 93%] Optimizing storage footprint... Completed 14 of 15 tables. Optimiz
[ 93%] Optimizing storage footprint... Completed 14 of 15 tables. Optimiz
[ 93%] Optimizing storage footprint... Completed 14 of 15 tables. Optimiz
[ 93%] Optimizing storage footprint... Completed 14 of 15 tables. Optimiz
[ 93%] Optimizing storage footprint... Completed 14 of 15 tables. Optimiz
[ 93%] Optimizing storage footprint... Completed 14 of 15 tables. Optimiz
[100%] Optimizing storage footprint... Completed 15 of 15 tables.

[100%] All done...

Query optimization results...
9 queries FULLY OPTIMIZED BY NEW PROJECTIONS

Deploying design...
Adding 32 new projections
Dropping 56 unnecessary existing projections

[100%] Deploying/Dropping projections... Completed 88 of 88 projections.

Completed 88 of 88 projections.

Database Designer finished.
```

- 14 When the Database Designer finishes, press **Enter** to return to the Administration Tools menu.

Note: The Database Designer creates a backup of the current design of your database before deploying the new design. This backup is stored in the output directory you entered in step 5, and is named `design_name_projection_backup_nnnnnnnnnn.sql`

See Also

Connect to the Database and Run a Simple Query in the Getting Started Guide

Creating a Query-specific Design Using the Database Designer

If you used the Tutorial in the Getting Started Guide, you have already created a comprehensive design.

If you have a new query file that you want to optimize, you can create an enhanced design with additional projections that are tuned it. The query-specific design that you create in this procedure will be optimized to balance query performance and compression for the provided query.

- 1 Log in to a terminal using the database administrator account that was created during product installation.

The default account name is `dbadmin`.

- 2 Start the Administrative Tools:

```
$ /opt/vertica/bin/admintools
```

- 3 If the database is not already running, on the Main Menu select **Start Database** and click **OK**.

- 4 Click **Configuration Menu** and click **OK**.

- 5 From the Configuration Menu, click **Run Database Designer**, and then and click **OK**.

- 6 Select your database and click **OK**.

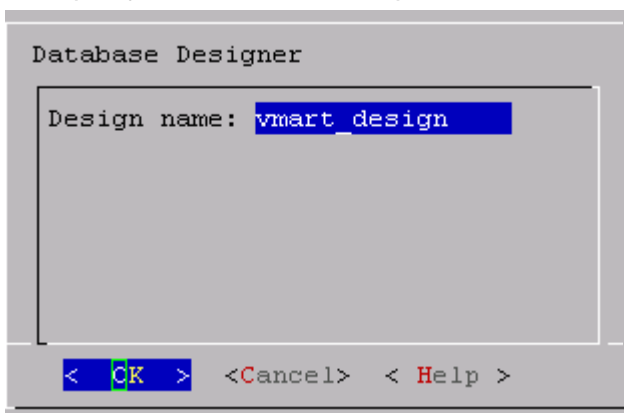
Note: This procedure assumes you are optimizing the `vmartdb` database you created in the Tutorial.

If you are asked to enter the password for the database, enter it and click **OK**. In the case of the `vmart` database, no password was assigned in Step 2: Create the Example Database, so you should not be prompted for one now.

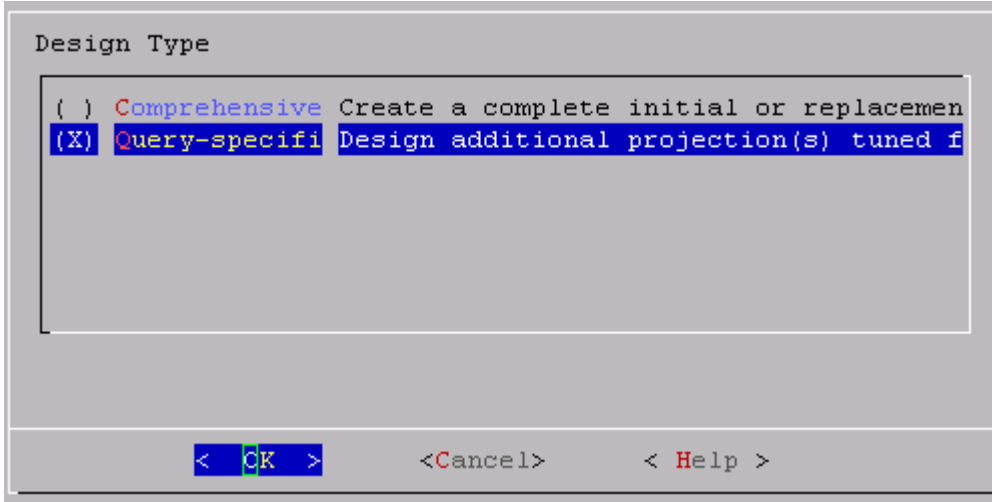
- 7 Click **OK** to accept the default directory for storing Database Designer output and log files.

Note this location.

- 8 In the **Database Designer** window, enter a name for the design and click **OK**. For this example, just click **OK** to accept the default `vmart_design` name.

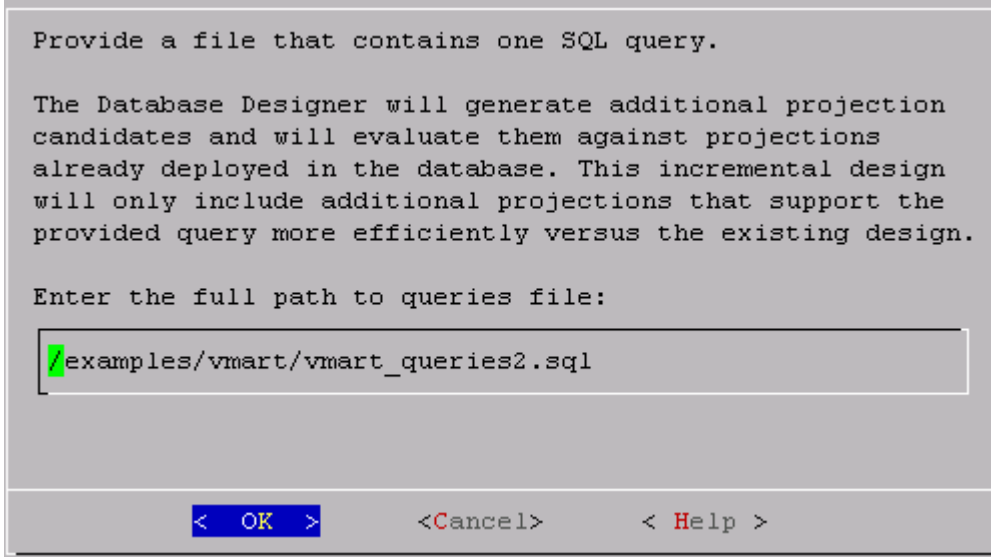


- 9 In the **Design Type** window, click **Query-specific** and click **OK**.



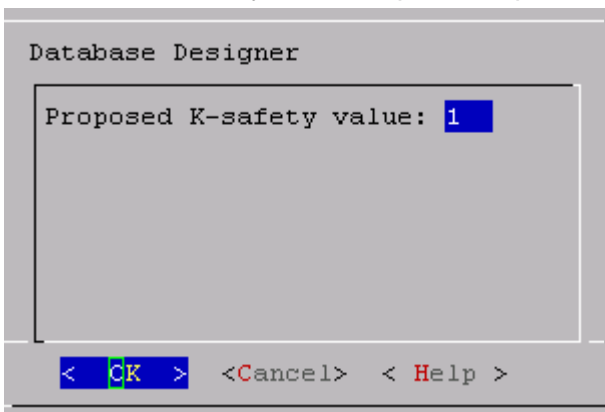
- 10 In the **Design Options** window, select the options you want and click **OK**.
- *Update statistics*: Accurate statistics help the Database Designer choose the best strategy for data compression. If you select this option, the database statistics are updated to maximize design quality.
Note that updating statistics takes time and resources, so if the current statistics are up to date, this step is unnecessary. When in doubt, update statistics.
 - *Deploy design*: The new design will be automatically deployed, which means that during deployment, new projections are added, some existing projections might be retained, and any unnecessary existing projections are removed. Any new projections are refreshed so that they are populated with data.
- Note:** For large databases a full design session could take a long time, yet it is best to allow this process to complete uninterrupted. If the session must be canceled, use CTRL+C.
- 11 You are prompted for the query file. Type the full path to the file containing the queries that will be run on your database. In this example it is:

```
/examples/vmart/vmart_queries2.sql
```



- 12 Accept the default or enter a new value for the **K-safety value** (in this case 1) and click **OK**.

Note: There will be no K-safe form if you are creating a comprehensive design on a single node. In that case, you can skip this step.



- 13 When the informational message displays, click **Proceed**.

The Database Designer:

- Sets up the design session.
- Examines table data.
- Loads the query file that you provided.
- Creates the design.
- Creates and refreshes any new projections called for by the design if you selected to deploy the design in step 10. Otherwise, it saves a SQL script containing the design.

Note: A message that the Database Designer did not optimize projections means that the auto projections created in the initial design were already optimized, so the Database Designer makes no new suggestions.

- 14 When the Database Designer finishes, press **Enter** to return to the Administration Tools menu.

Deploying Designs

Vertica recommends that you test your design on a non-production server before you deploy it to your production server.

The method you use to deploy a design depends on the method you used to create the design:

- Letting the Database Designer deploy your design at design time
- **Manually deploying** (page 90) your design at a later time

Deploying Designs Manually

If you chose *not* to deploy your design through the Database Designer at design time, you can deploy it later manually. You can either run a deployment script or you can follow a series of steps.

Deploying a design using the deployment script:

- 1 Be sure that the environment consists of a database loaded with a logical schema.
To deploy the projections to a test or production environment, use the ***v meta-command*** (see "***i FILE***" on page 377) in vsql to run the SQL script.
- 2 Run the Database Designer deployment script:

```
<design name>_deploy.sql
```


Where <design_name> is the name of the database design.

Deploying a design manually:

- 1 Use the START_REFRESH function to update the newly-created projections to the same level as the existing projections. See ***Refreshing Projections*** (page 280).
You can also use the REFRESH function which invokes refresh synchronously, rather than as a background process.
- 2 Use the MAKE_AHM_NOW function to set the Ancient History Mark (AHM) to the greatest allowable epoch (now).
- 3 Optionally use the DROP PROJECTION function to drop the temporary projections that were created for the temporary design. See ***Dropping Projections*** (page 281).

Note: You can keep the temporary projections, but they could reduce query processing speed if they remain in the database.

- 4 Run the ANALYZE_STATISTICS function on all projections in the database. This function collects and aggregates data samples and storage information from all nodes on which a projection is stored, then writes statistics into the catalog. For example:

```
vmartdb=> SELECT ANALYZE_STATISTICS ('');
```


Creating Custom Designs

Vertica Systems, Inc. strongly recommends that you use the physical schema design produced by Database Designer, which provides K-Safety, excellent query performance, and efficient use of storage space. If you find one of your queries is not running as efficiently as you would like, you can use the Database Designer query-specific design process to optimize the database design for the query.

If the projections created by Database Designer still do not meet your needs, you can write custom projections from scratch or based on projection designs created by the Database Designer.

If you are unfamiliar with writing custom projections, Vertica Systems, Inc. suggests that you start by modifying an existing design generated by Database Designer.

The Design Process

The process for customizing an existing design or creating a new one entails:

- 1 Planning the design or design modification.
As with most successful projects, a good design requires some up-front planning. See ***Planning Your Design*** (page 91).
- 2 Creating or modifying projections.
See ***Design Fundamentals*** (page 95) for an overview of the CREATE PROJECTION statement and guidelines for creating common projections. Also refer to the CREATE PROJECTION statement in the SQL Reference Manual.
- 3 Deploying the projections to a test environment. See ***Writing and Deploying Custom Projections*** (page 95).
- 4 Testing the projections.
- 5 Modifying the projections as necessary.
- 6 Once you have finalized the design, deploying the projections to the production environment.

Planning Your Design

The syntax for creating a design is easy for anyone who is familiar with SQL. As with any successful project, however, a successful design requires some initial planning. Before you create your first design, be sure you:

- Are familiar with standard design requirements and that you plan your design to include them. See ***Design Requirements*** (page 91).
- Determine how many projections you need to include in the design. See ***Determining the Number of Projections to Use*** (page 92).
- Determine the type of compression and encoding to use for columns. See **Maximizing Performance Through Compression and Encoding**.
- Determine whether or not you want the database to be K-Safe. Vertica recommends that all production databases have a minimum K-Safety of one (K=1). Up to K=2 is possible. See ***Designing for K-Safety*** (page 92).

Design Requirements

A physical schema design is a script that contains CREATE PROJECTION statements. These statements determine which columns are included in projections and how they are optimized.

If you use the Database Designer as a starting point, it automatically creates designs that meet all fundamental design requirements. If you intend to create or modify designs manually, be aware that all designs must meet the following requirements:

- Every design must create at least one superprojection for every table in the database that are used by the client application. These projections provide complete coverage that enables users to perform ad-hoc queries as needed. They can contain joins and they are usually configured to maximize performance through sort order, compression, and encoding.
- Query-specific projections are optional. If you are satisfied with the performance provided through superprojections, you do not need to create additional projections. However you can maximize performance by tuning for specific query work loads.
- Vertica recommends that all production databases have a minimum K-Safety of one (K=1) to support high availability and recovery. (Up to K=2 is possible.) See High Availability Through Projections in the Concepts Guide and *Designing for K-Safety* (page 92).

Determining the Number of Projections to Use

In many cases, a design that consists of a set of superprojections (and their buddies) provides satisfactory performance through compression and encoding. This is especially true if the sort orders for the projections have been used to maximize performance for one or more query predicates (WHERE clauses).

However, you might want to add additional query-specific projections to increase the performance of one or more specific queries because they run slowly, are used frequently, or are run as part of business critical reporting. The number of additional projections (and their buddies) that you create should be determined by:

- Your organization's needs.
- The amount of disk space you have available on each node in the cluster.
- The amount of time available for loading data into the database.

As the number of projections that are tuned for specific queries increases, the performance of these queries increases as well. However, the amount of disk space used and the amount of time required to load data increases as well. Therefore, you should create and test designs to determine the optimum number of projections for your database configuration. On average, organizations that choose to implement query-specific projections achieved optimal performance through the addition of a few query-specific projections.

Designing for K-Safety

Before creating custom physical schema designs, determine whether you want the database to be K-safe and adhere to the appropriate design requirements for K-Safe databases or databases with no K-Safety. Vertica Systems, Inc. recommends that all production databases have a minimum K-Safety of one (K=1). Up to K=2 is possible. Non-production databases do not have to be K-Safe. You can start by creating a physical schema design with no K-Safety, and then modify it to be K-Safe at a later point in time. See High Availability and Recovery and High Availability Through Projections in the Concepts Guide for an explanation of how Vertica implements high availability and recovery through replication and segmentation.

Requirements for a K-Safe Physical Schema Design

Database Designer automatically generates designs with a K-Safety of one ($K=1$) for clusters that contain at least three nodes. (If your cluster has one or two nodes, it generates designs with a K-Safety of zero ($K=0$).) Therefore, you can modify a design created for a three-node (or greater) cluster and the K-Safe requirements will already have been completed for you.

If you choose to write custom projections, be aware that your physical schema design must meet the following requirements to be able to successfully recover the database in the event of a failure:

- Projections that are based on a fact table or a pre-join of fact and dimension tables should be segmented across all nodes. Refer to *Designing for Segmentation* (page 93) and *Designing Segmented Projections for K-Safety* (page 101).
- In general, dimension table projections must be replicated on all nodes. For a snowflake, all dimension tables within the snowflake must be replicated. If, however, a projection for a large dimension table is similar in size to a fact table, it can be segmented. See *Designing Replicated Projections for K-Safety* (page 100).
- Segmented projections must have K buddy projections (projections that have identical columns and segmentation criteria, except that corresponding segments are placed on different nodes).

Requirements for a Physical Schema Design with No K-Safety

If you choose to use Database Designer to generate an comprehensive design that you can modify and you do not want the design to be K-Safe, set K-safety level to 0 (zero).

If you want to start from scratch, do the following to establish minimal projection requirements for a functioning database with no K-Safety ($K=0$).

- 1 Define at least one superprojection for each table in the logical schema.
- 2 Replicate (define an exact copy of) each dimension table superprojection on each node.

Designing for Segmentation

If you are creating segmented nodes to ensure high availability and recovery, you need to determine:

- Which segmentation method to use.
- Which columns to use to segment the projection.
- Which nodes on which to distribute segments.

Determining the segmentation method

Vertica provides two methods for segmenting projections: hash and range.

- Hash segmentation is the preferred method of segmentation. It allows you to segment a projection based on a built-in hash function that provides even distribution of data across multiple nodes, resulting in optimal query execution. In a projection, the data to be hashed consists of one or more column values, each having a large number of unique values and an acceptable amount of skew in the value distribution. Primary key columns that meet the criteria could be an excellent choice for hash segmentation. Refer to the CREATE PROJECTION command in the SQL Reference Manual for detailed information about using hash segmentation in a projection.
- Range segmentation allows you to specify a list of nodes, each of which stores a specific range of data values, except for the MAXVALUE node, which has no upper limit. Refer to the CREATE PROJECTION command in the SQL Reference Manual for detailed information about using range segmentation in a projection.

Unlike hash segmentation, range does not automatically distribute data evenly across some or all nodes in a cluster. Therefore, range segmentation is not typically used. Use range segmentation only when your projection includes a column that is known to contain data suitable for use as a segmentation expression. In other words, avoid using columns that distribute data in a way that causes skewed distribution and execution. This causes some nodes to work harder than others because they are consistently storing more data. This includes data that does not yet exist but can cause skew if loaded in the future.

In particular, avoid using a date/time column for range segmentation because it causes temporal skew. For example, consider a fact table in which each row contains a timestamp representing that point in time at which the fact was established. In that case, all new fact table rows would be stored on the MAXVALUE node, causing skew that would increase over time and, thus, would have a negative effect on query performance.

Determining the columns to use for segmentation

The columns to use vary depending upon the type of segmentation method you intend to use.

- **Columns for Hash Segmentation**

If you decide to use hash segmentation, choose one or more columns that have a large number of unique data values and acceptable skew in their data distribution. Primary key columns that meet the criteria could be an excellent choice for hash segmentation. The columns must be unique across all the tables being used in a query.

- **Columns for Range Segmentation**

If you decide to use range segmentation, choose one column to use for segmentation that:

- Uses an INTEGER or FLOAT data type.
- Has a known range of data values.
- Has an even distribution of data values.
- Has a large number of unique data values.
- Is unique across all columns in a query.

Avoid columns that:

- Are foreign keys.
- Are used in query predicates.
- Have a date/time data type.
- Have correlations with other columns due to functional dependencies.

Note: Segmenting on DATE/TIME data types is valid, but guaranteed to produce temporal skew in the data distribution and is not recommended. If you choose this option, do not use TIME or TIMETZ because their range is only 24 hours.

Determining the nodes on which to distribute projection segments

Typically, segments are distributed across all nodes in the database to ensure high availability and recovery. However, you can choose specific nodes on which to store projections segments. Be sure to store segments on enough nodes to ensure a K-Safety value of one (1) or two (2).

Design Fundamentals

Writing and Deploying Custom Projections

Although you can write custom projections from scratch, Vertica Systems, Inc. recommends that you use Database Designer to create a design to use as a starting point. This ensures that you have projections that meet basic requirements.

Before you write custom projections, be sure to review the topics in *Planning Your Design* (page 91) carefully. Failure to follow these considerations can result in non-functional projections. Vertica Systems, Inc. strongly recommends that you contact **Technical Support** (on page 1) to inspect your customized physical schema design to ensure its integrity before you run it.

To manually modify or create a projection:

- 1 Create a script to contain the projection.
- 2 Use the CREATE PROJECTION command described in the SQL Reference Manual.
- 3 Use the ***v* meta-command** (see "*i FILE*" on page 377) in vsql to run the script.

Note: The environment must consist of a database loaded with a logical schema.

- 4 For a K-Safe database, use the function `SELECT get_projections('table_name')` to verify that the projections were properly created. Good projections are noted as being "safe." This means that the projection has enough buddies to be K-Safe.
- 5 If you added the new projection to a database that already has projections that contain data, you need to update the newly-created projection to work with the existing projections. By default, the new projection is out-of-date (not available for query processing) until you refresh it.

Vertica internal operations (mergeout, refresh, and recovery) maintain partition separation except in certain cases:

- Recovery of a projection when the buddy projection from which the partition is recovering is identically sorted. If the projection is undergoing a full rebuild, it is recovered one ROS container at a time. The projection ends up with a storage layout identical to its buddy and is, therefore, properly segmented.

Note: In the case of a partial rebuild, all recovered data goes into a single ROS container and must be partitioned manually.

- Manual tuple mover operations often output a single storage container, combining any existing partitions; for example, after executing any of the `PURGE ()` operations.

See *Refreshing Projections* (page 280).

- 1 Use the `MAKE_AHM_NOW` function to set the Ancient History Mark (AHM) to the greatest allowable epoch (now).

- 2 Use the DROP_PROJECTION function to drop the any previous projections which are no longer needed. See **Dropping Projections** (page 281).

These projections can waste disk space and reduce load speed if they remain in the database.

- 3 Run the ANALYZE_STATISTICS function on all projections in the database. This collects and aggregates data samples and storage information from all nodes on which a projection is stored, then writes statistics into the catalog. For example:

```
=>SELECT ANALYZE_STATISTICS ('');
```

Anatomy of a Projection

The CREATE PROJECTION statement specifies individual projections. The following sample depicts the elements of a projection.

```
CREATE PROJECTION retail_sales_fact_P1(
  store_sales_fact_store_key          ENCODING RLE ,
  store_sales_fact_pos_transaction_number ENCODING RLE
  store_sales_fact_sales_dollar_amount
  store_sales_fact_cost_dollar_amount
) AS
```

Column list and encoding

```
SELECT
  T_store_sales_fact.store_key,
  T_store_sales_fact.pos_transaction_number,
  T_store_sales_fact.sales_dollar_amount,
  T_store_sales_fact.cost_dollar_amount
FROM store_sales_fact T_store_sales_fact
```

Base query
(Only FK/PK joins are allowed.)

```
ORDER BY T_store_sales_fact.store_key
```

Sort order

```
SEGMENTED BY HASH(T_store_sales_fact.pos_transaction_number) ALL NODES
OFFSET 1;
```

Segmentation
SEGMENTED or
UNSEGMENTED (replicate)

The previous example contains the following significant elements:

Column list and encoding

Lists every column within the projection and defines the encoding for each columns. Unlike traditional database architectures, Vertica operates on encoded data representations. Therefore, Vertica encourages you to use data encoding because it results in less disk I/O.

Base query

Identifies all the columns to incorporate in the projection through column name and table name references. The base query for fact table projections can contain PK/FK joins to dimension tables.

Sort order

The ORDER BY clause specifies a projection's sort order, which localizes logically-grouped values so that a disk read can pick up many results at once. The sort order optimizes for a specific query or commonalities in a class of queries based on the query predicate. The best sort orders are determined by the WHERE clauses. For example, if a projection's sort order is (x, y) , and the query's WHERE clause specifies $(x=1 \text{ AND } y=2)$, all of the needed data is found together in the sort order, so the query runs almost instantaneously.

You can also optimize a query by matching the projection's sort order to the query's GROUP BY clause. If you do not specify a sort order, Vertica uses the order in which columns are specified in the column definition as the projection's sort order.

Segmentation

The segmentation clause determines whether a projection is segmented or replicated across nodes within the database. Replication stores identical copies of projections for dimension tables across all nodes in the cluster. This ensures high availability and recovery. Segmentation distributes contiguous pieces of projections, called segments, for fact and large dimension tables across database nodes. This maximizes database performance by distributing the load.

Designing Superprojections

Superprojections have the following requirements:

- They must contain every column within the table.
- For a K-Safe design, super projections must either be replicated on all nodes within the database cluster (for dimension tables) or paired with buddies and segmented across all nodes (for fact tables and large dimension tables). See [Physical Schema and High Availability Through Projections](#) in the Concepts Guide for an overview of projections and how they are stored and [Designing for K-Safety](#) (page 92) for design specifics.

To provide maximum usability, superprojections need to minimize storage requirements while maximizing query performance. To achieve this, the sort order for columns in superprojections are based on storage requirements and commonly-used queries.

Minimizing Storage Requirements

Minimizing storage not only saves on physical resources, it increases performance by requiring the database to perform less disk I/O. To minimize storage space for a projection:

- Analyze the type of data stored in each projection column and choose the most effective encoding method. See the CREATE PROJECTION statement and encoding-type within the SQL Reference Manual.

The Vertica optimizer gives Run Length Encoding (RLE) preference, so be sure to use it whenever appropriate. Run Length Encoding (RLE) replaces sequences (runs) of identical values with a single pair that contains the value and number of occurrences. Therefore, use it only when the run length is large, such as when low-cardinality columns are sorted.

- Prioritize low cardinality columns in the column sort order. This minimizes the number of rows that Vertica stores and accesses to retrieve query results.

For more information about minimizing storage requirements, see ***Choosing Sort-orders for Low Cardinality Predicates*** (page 102) and ***Choosing Sort-orders for High Cardinality Predicates*** (page 103).

Maximizing Query Performance

In addition to minimizing storage requirements, the column sort order facilitates the most commonly-used queries for the table. This means that the column sort order prioritizes the lowest-cardinality columns that are actually used in queries, not the lowest cardinality columns. See ***Choosing Sort-orders for Low Cardinality Predicates*** (page 102) for examples that take into account both storage and query requirements.

Projections within a buddy set can all have different sort orders. This enables you to maximize query performance for groups of queries with common WHERE clauses, but different sort orders. If, for example, you have a three node cluster, your buddy set would contain three interrelated projections, each of which could have its own sort order.

In a database with a K-Safety of one (1) or two (2), buddy projections are used for data recovery. If a node fails, it queries the other nodes to recover data through buddy projections. (See How Result Sets are Stored in the Concepts Guide.) If a projection's buddies use different sort orders, it takes longer to recover the projection because the data has to be resorted during recovery to match the sort order of the projection. Therefore, consider using identical sort orders for tables that are rarely queried or that are repeatedly accessed by the same query, and use multiple sort orders for tables that are accessed by queries with common WHERE clauses, but different sort orders.

If you have queries that access multiple tables or you want to maintain the same sort order for projections within buddy sets, create query-specific projections. Designs that contain projections for specific queries are called optimized designs.

Designing for Group By Queries

Database Designer does not optimize for Group By queries. Thus, projections created through Database Designer do not take advantage of the Group By Pipeline execution technique.

In cases in which a large amount of data is being grouped, possibly due to non-selective predicates or the absence of predicates altogether, this technique may be preferable because it requires significantly less RAM than Group By Hash and it reduces disk I/O. This is due to the fact that Group By Hash requires Vertica to create a hash table to process the aggregates and group by expressions.

To apply this optimization, use the same projection column sort order as is used by the GROUP BY clause in the query statement.

Note: Using the Group By Pipeline optimization might defeat other optimizations based on the predicate, especially if the predicate is very selective. Therefore, use it with care and only when required.

Example Group By Query

The following query sums all the deals made for each customer in the Deal table and then groups them by the Customer column.

```
SELECT Customer, SUM(DealPrice)
FROM Deal
```



```
GROUP BY Customer;
```

Column Descriptions

The columns from the Deal table contain the following values:

Deal_ID	DealDate	DealPrice	Customer
1	2005-01-01	500	Carr
2	2004-01-01	300	Gupta
3	2003-02-05	1000	Carr
4	2007-03-28	1200	Lee
5	2007-03-30	400	Frank
6	2003-02-16	1800	Gupta

Result Set

The result set for the query is:

Customer	Sum(DealPrice)
Carr	1500
Frank	400
Gupta	2100
Lee	1200

Projection Column Sort Order

To optimize for this query, set the sort order for the projection to the Customer Column:

```
ORDER BY Customer
```

Projection Column Sort Order for Multiple Columns

To optimize for a query that uses a Group By statement on more than one column, sort the projection using the same columns and in the same order. If you want to sort the projection using additional columns that are not present in the Group By statement, place them at the end of the sort order.

The following query uses the Group By statement to group the result sets by both the Customer and DealDate columns:

```
SELECT Customer, DealDate, SUM(DealPrice)
FROM Deal
GROUP BY Customer, DealDate;
```

To set the projection column sort order for this query:

```
ORDER BY Customer, DealDate
```

Designing Replicated Projections for K-Safety

If you are creating or modifying a design for a K-Safe database, you need to ensure that projections for dimension tables are replicated on each node in the database. (For an overview of replicated projections, see Projection Replication in the Concepts Guide.)

You can accomplish this using a single CREATE PROJECTION command for each dimension table. The UNSEGMENTED ALL NODES syntax within the segmentation clause automatically creates an unsegmented projection on each node in the database, as follows:

```
UNSEGMENTED ALL NODES;
```

When you run your design script, Vertica generates a list of nodes based on the number of nodes in the database and replicates the projection accordingly. Replicated projections have the name:

projection-name_node-name

If, for example, the nodes are named NODE01, NODE02, and NODE03, the projections are named ABC_NODE01, ABC_NODE02, and ABC_NODE03.

Note: This naming convention could affect functions that provide information about projections, for example, GET_PROJECTIONS or GET_PROJECTION_STATUS, where you must provide the name ABC_NODE01 instead of just ABC. To view a list of the nodes in a database, use the **View Database** (page 344) command in the Administration Tools.

The following script uses the UNSEGMENTED ALL SITES syntax to create one unsegmented superprojection for the store_dimension table on each node.

```
CREATE PROJECTION store_dimension(  
  C0_store_dimension_floor_plan_type ENCODING RLE ,  
  C1_store_dimension_photo_processing_type ENCODING RLE ,  
  C2_store_dimension_store_key ,  
  C3_store_dimension_store_name ,  
  C4_store_dimension_store_number ,  
  C5_store_dimension_store_street_address ,  
  C6_store_dimension_store_city ,  
  C7_store_dimension_store_state ,  
  C8_store_dimension_store_region ,  
  C9_store_dimension_financial_service_type ,  
  C10_store_dimension_selling_square_footage ,  
  C11_store_dimension_total_square_footage ,  
  C12_store_dimension_first_open_date ,  
  C13_store_dimension_last_remodel_date )  
AS SELECT T_store_dimension.floor_plan_type,  
  T_store_dimension.photo_processing_type,  
  T_store_dimension.store_key,  
  T_store_dimension.store_name,  
  T_store_dimension.store_number,  
  T_store_dimension.store_street_address,  
  T_store_dimension.store_city,  
  T_store_dimension.store_state,  
  T_store_dimension.store_region,  
  T_store_dimension.financial_service_type,  
  T_store_dimension.selling_square_footage,  
  T_store_dimension.total_square_footage,  
  T_store_dimension.first_open_date,
```

```
T_store_dimension.last_remodel_date
FROM store_dimension T_store_dimension
ORDER BY T_store_dimension.floor_plan_type,
T_store_dimension.photo_processing_type
UNSEGMENTED ALL NODES;
```

Note: Large dimension tables can be segmented. A dimension table is considered to be large when it is approximately the same size as a fact table.

Designing Segmented Projections for K-Safety

If you are creating or modifying a design for a K-Safe database, you need to create K-Safe projections for fact tables and large dimension tables. (A dimension table is considered to be large if it is similar in size to a fact table.) This entails:

- Creating a segmented projection for each fact and large dimension table.
- Creating segmented buddy projections for each of these projections. The total number of projections in a buddy set must be two (2) for a K=1 database or three (3) for a K=2 database.

For an overview of segmented projections and their buddies, see Projection Segmentation in the Concepts Guide. For information about designing for K-Safety, see *Designing for K-Safety* (page 92) and *Designing for Segmentation* (page 93).

Segmenting Projections

To segment a projection, use the segmentation clause to specify the:

- Segmentation method to use.
- Column to use to segment the projection.
- Nodes on which to segment the projection. You can segment projections across all the nodes, or just the number of nodes necessary to maintain K-Safety, either three (3) for a K=1 database or five (5) for a K=2 database.

See the CREATE PROJECTION statement in the SQL Reference Manual.

The following segmentation clause uses hash segmentation to segment the projection across all nodes based on the T_retail_sales_fact.pos_transaction_number column:

```
CREATE PROJECTION retail_sales_fact_P1...
SEGMENTED BY HASH(T_retail_sales_fact.pos_transaction_number) ALL NODES;
```

Creating Buddy Projections

To create a buddy projection, copy the original projection and modify it as follows:

- Rename it to something similar to the name of the original projection. For example, a projection named *retail_sales_fact_P1* could have buddies named *retail_sales_fact_P1_B1* and *retail_sales_fact_P1_B2*.
- Modify the sort order as needed.
- Create an offset to store the segments for the buddy on different nodes. For example the first buddy in a projection set would have an offset of one (OFFSET1;) the second buddy in a projection set would have an offset of two (OFFSET2;), and so on.

For example, to create a buddy for the projection created in the previous example:

```
CREATE PROJECTION retail_sales_fact_P1_B1...
SEGMENTED BY HASH(T_retail_sales_fact.pos_transaction_number) ALL NODES OFFSET 1;
```

Maximizing Projection Performance

Choosing Sort-orders for Low Cardinality Predicates

When dealing with predicates on low-cardinality columns, you can use the combination of RLE and Sorting to reduce disk I/O. This is achieved by bucketing data such that all rows that correspond to the same value are clustered together on disk. The following example shows how RLE is combined with the column sort order to minimize storage requirements and maximize query performance.

Example Query

```
SELECT name FROM students
WHERE gender = 'M' AND pass_fail = 'P' AND class = 'senior';
```

Column Descriptions

The columns from the students table contain the following values and encoding types:

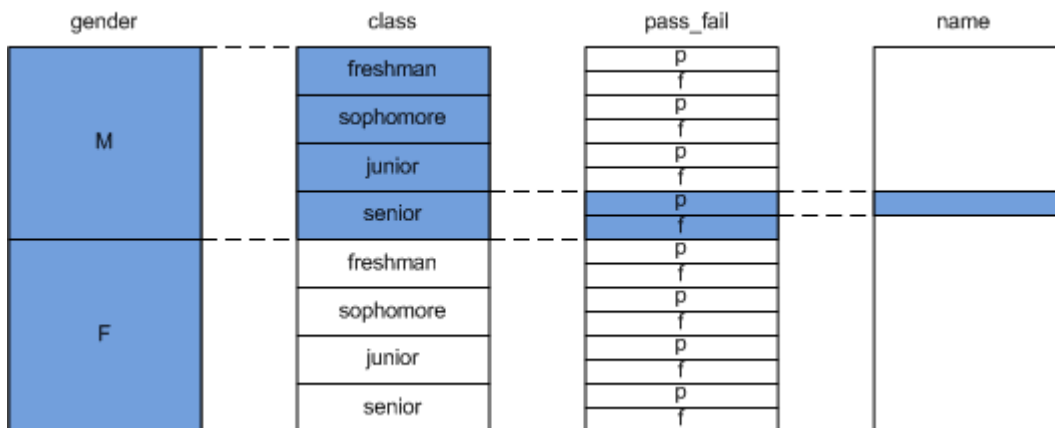
Column	# of Values	Encoded With
gender	2 (M or F)	RLE
pass_fail	2 (P or F)	RLE
class	4 (freshman, sophomore, junior, or senior)	RLE
name	10000 (too many to list)	Auto

Optimal Sort Order

The fastest way to access the names of students who are male, have passed their grade level, and are seniors is to work through the low cardinality columns with the smallest number of values before the high cardinality columns. The following example illustrates a column sort order that minimizes storage and maximizes query performance for the example query.

```
ORDER BY = student.gender, student.class, student.pass_fail, student.name
```

This example creates the following buckets:



This query operates efficiently because only a subset of buckets (highlighted in blue) is evaluated for each condition in the where clause.

Sub-optimal Sort Order

The following example shows a sort order that starts with the name column. This sort order maximizes the number of rows that are stored and minimizes query performance because the students' gender, pass/fail status, and class must be evaluated for every name.

```
ORDER BY = student.name, student.gender, student.pass_fail, student.class,
```

Choosing Sort-orders for High Cardinality Predicates

In some cases, your query predicate might require you to prioritize a high cardinality column in the projection's sort order. For example, you might have predicates based on phone numbers or timestamps. To avoid establishing a sub-optimal projection, you can insert a new column into the table and the projection. This pseudo column artificially creates a low cardinality bucket that you can then prioritize in the projection's sort order.

To be effective, the number of unique values in the column you insert should be almost equal to the square root of the number of values in the original high cardinality column. Use `SELECT DISTINCT` to determine the number of unique values in the high cardinality column. See the `SELECT` statement in the SQL Reference Manual.

The following example illustrates this concept.

Query Without Bucketing

The following query requires a full column scan on the high cardinality column (Number) because the sort order is prioritized on the Number column:

```
SELECT Address
FROM cdr_table WHERE Number='9788876542';
```

Number	Address
5089761234	47 Elm St.
5087654321	29 School St.
.	.
617325467	57 Maple Ave.
617234256	22 Post Rd.
.	.
9784748859	73 Post Rd.
9788876542	9 Main St.
.	.

Query With Bucketing

Inserting the low cardinality column `Area_Code` and prioritizing it in the projection's sort order enables a partial scan of the `Number` column.

```
SELECT Address
```

```
FROM cdr_table WHERE Area_Code='978' AND Number='9788876542';
```

Area_Code	Number	Address
508	5089761234 5087654321 . .	47 Elm St. 29 School St. . .
617	617325467 617234256 . .	57 Maple Ave. 22 Post Rd. . .
978	9784748859 9788876542 . .	73 Post Rd. 9 Main St. . .

Prioritizing Column Access Speed

If you measure and set the performance of storage locations within your cluster, Vertica uses this information to determine where to store columns based on their rank. See **Setting Location Performance** (page 288).

How Columns are Ranked

Vertica stores columns that are included in the sort order of projections on the fastest disks and columns that are not included within the sort order of projections on slower disks. It accomplishes this by ranking columns for each projection as follows:

- Columns in the sort order are given the highest priority (numbers > 1000).
- The last column in the sort order is given the rank number 1001.
- The next to the last column in the sort order is given the rank number 1002, and so on until the first column in the sort order is given 1000 + # of sort columns.
- The remaining columns are given numbers from 1000 - 1, starting with 1000 and working down from there.

Then it stores these columns on disk as follows:

Columns are stored on disk from the highest ranking to the lowest ranking in which the highest ranking columns are placed on the fastest disks and the lowest ranking columns are placed on the slowest disks.

Overriding Default Column Ranking

You can modify which columns are stored on fast disks by manually overriding the default ranks for these columns. To accomplish this, set the ACCESSRANK keyword in the column list. Be sure to use an integer that is not already being used for another column. For example, if you want to give a column the fastest access rank, be sure to use a number that is significantly higher than 1000 + the number of sort columns. This allows you to enter more columns over time without bumping into the access rank you set.

The following example sets the access rank for the C1_retail_sales_fact_store_key column to 1500.

```
CREATE PROJECTION retail_sales_fact_P1 (  
  C1_retail_sales_fact_store_key ENCODING RLE ACCESSRANK 1500,  
  C2_retail_sales_fact_pos_transaction_number ,  
  C3_retail_sales_fact_sales_dollar_amount ,  
  C4_retail_sales_fact_cost_dollar_amount )
```

Projection Examples

This sections provides examples that show how to create projections:

- ***For a new database where K-Safety=2*** (page 107)
- ***When adding a node to an existing database*** (page 110)

New K-Safe=2 Database

In this example, projections are created for a new five node database with a K-Safety of two (2). To simplify the example, this database contains only two tables: retail_sale_fact and store_dimension. Creating projections for this database consists of creating the following segmented and unsegmented (replicated) superprojections:

- **Segmented projections**

To support K-Safety=2, the database requires three segmented projections (one projection and two buddy projections) for each fact table. In this case, it requires three segmented projections for the retail_sale_fact table:

Projection	Description
P1	The primary projection for the retail_sale_fact table.
P1_B1	The first buddy projection for P1. This buddy is required to provide K-Safety=1.
P1_B2	The second buddy projection for P1. This buddy is required to provide K-Safety=2.

- **Unsegmented Projections**

To support the database, one unsegmented superprojection must be created for each dimension table on each node. In this case, one unsegmented superprojection must be created on each node for the store_dimension table:

Node	Unsegmented Projection
Node01	store_dimension_Node01
Node02	store_dimension_Node02
Node03	store_dimension_Node03
Node04	store_dimension_Node04
Node05	store_dimension_Node05

Creating Segmented Projections Example

The following SQL script creates the P1 projection and its buddies, P1_B1 and P1_B2, for the retail_sales_fact table. The following syntax is significant:

- CREATE PROJECTION creates the named projection (retail_sales_fact_P1, retail_sales_fact_P1_B1, or retail_sales_fact_P1_B2).
- ALL NODES automatically segments the projections across all five nodes in the cluster without specifically referring to each node.
- HASH evenly distributes the data across these nodes.

- OFFSET ensures that the same data is not stored on the same nodes for each of the buddies. The first buddy uses OFFSET 1 to shift the storage locations by one and the second buddy uses OFFSET 2 to shift the storage locations by two. This is critical to ensure K-safety.

```

CREATE PROJECTION retail_sales_fact_P1 (
    C1_retail_sales_fact_store_key ENCODING RLE ,
    C2_retail_sales_fact_pos_transaction_number ,
    C3_retail_sales_fact_sales_dollar_amount ,
    C4_retail_sales_fact_cost_dollar_amount )
AS SELECT T_retail_sales_fact.store_key,
    T_retail_sales_fact.pos_transaction_number,
    T_retail_sales_fact.sales_dollar_amount,
    T_retail_sales_fact.cost_dollar_amount
FROM retail_sales_fact T_retail_sales_fact
ORDER BY T_retail_sales_fact.store_key
SEGMENTED BY HASH(T_retail_sales_fact.pos_transaction_number) ALL NODES;

```

```

-----
-- Projection #                : 6
-- Projection storage (KBytes) : 4.8e+06
-- Note: This is a super projection for table: retail_sales_fact

```

```

CREATE PROJECTION retail_sales_fact_P1_B1 (
    C1_retail_sales_fact_store_key ENCODING RLE ,
    C2_retail_sales_fact_pos_transaction_number ,
    C3_retail_sales_fact_sales_dollar_amount ,
    C4_retail_sales_fact_cost_dollar_amount )
AS SELECT T_retail_sales_fact.store_key,
    T_retail_sales_fact.pos_transaction_number,
    T_retail_sales_fact.sales_dollar_amount,
    T_retail_sales_fact.cost_dollar_amount
FROM retail_sales_fact T_retail_sales_fact
ORDER BY T_retail_sales_fact.store_key
SEGMENTED BY HASH(T_retail_sales_fact.pos_transaction_number) ALL NODES
OFFSET 1;

```

```

-----
-- Projection #                : 6
-- Projection storage (KBytes) : 4.8e+06
-- Note: This is a super projection for table: retail_sales_fact

```

```

CREATE PROJECTION retail_sales_fact_P1_B2 (
    C1_retail_sales_fact_store_key ENCODING RLE ,
    C2_retail_sales_fact_pos_transaction_number ,
    C3_retail_sales_fact_sales_dollar_amount ,
    C4_retail_sales_fact_cost_dollar_amount )
AS SELECT T_retail_sales_fact.store_key,
    T_retail_sales_fact.pos_transaction_number,
    T_retail_sales_fact.sales_dollar_amount,
    T_retail_sales_fact.cost_dollar_amount
FROM retail_sales_fact T_retail_sales_fact
ORDER BY T_retail_sales_fact.store_key
SEGMENTED BY HASH(T_retail_sales_fact.pos_transaction_number) ALL NODES
OFFSET 2;

```

Creating Unsegmented Projections Example

The following script uses the UNSEGMENTED ALL NODES syntax to create one unsegmented superprojection for the store_dimension table on each node.

```

CREATE PROJECTION store_dimension (
  C0_store_dimension_floor_plan_type ENCODING RLE ,
  C1_store_dimension_photo_processing_type ENCODING RLE ,
  C2_store_dimension_store_key ,
  C3_store_dimension_store_name ,
  C4_store_dimension_store_number ,
  C5_store_dimension_store_street_address ,
  C6_store_dimension_store_city ,
  C7_store_dimension_store_state ,
  C8_store_dimension_store_region ,
  C9_store_dimension_financial_service_type ,
  C10_store_dimension_selling_square_footage ,
  C11_store_dimension_total_square_footage ,
  C12_store_dimension_first_open_date ,
  C13_store_dimension_last_remodel_date )
AS SELECT T_store_dimension.floor_plan_type,
  T_store_dimension.photo_processing_type,
  T_store_dimension.store_key,
  T_store_dimension.store_name,
  T_store_dimension.store_number,
  T_store_dimension.store_street_address,
  T_store_dimension.store_city,
  T_store_dimension.store_state,
  T_store_dimension.store_region,
  T_store_dimension.financial_service_type,
  T_store_dimension.selling_square_footage,
  T_store_dimension.total_square_footage,
  T_store_dimension.first_open_date,
  T_store_dimension.last_remodel_date
FROM store_dimension T_store_dimension
ORDER BY T_store_dimension.floor_plan_type,
  T_store_dimension.photo_processing_type
UNSEGMENTED ALL NODES;

```

Add Node to a Database

In this example, a fourth node (Node04) is being added to a three-node database. The database contains two tables: retail_sale_fact and store_dimension. It also contains the following segmented and unsegmented (replicated) superprojections:

- **Segmented projections**
P1 and its buddy, B1, are projections for the retail_sale_fact table. They were created using the ALL NODES syntax, so Vertica automatically segmented the projections across all three nodes.
- **Unsegmented Projections**
Currently three unsegmented superprojections exist for the store_dimension table, one for each node, as follows:

Node	Unsegmented Projection
Node01	store_dimension_Node01
Node02	store_dimension_Node02
Node03	store_dimension_Node03

To support an additional node, replacement projections need to be created for the segmented projections, P1 and B1. The new projections could be called P2 and B2, respectively. Additionally, an unsegmented superprojection (store_dimension_Node04) needs to be created for the dimension table on the new node (Node04).

Creating Segmented Projections Example

The following SQL script created the original P1 projection and its buddy, B1, for the retail_sales_fact table. Since the script uses the ALL NODES syntax, creating a new projection that includes the fourth node is as easy as copying the script and changing the names of the projection and its buddy to unique names (for example, P2 for the projection and P2_B2 for its buddy). The names that need to be changed are highlighted within the example.

```
CREATE PROJECTION retail_sales_fact_P1 (
  C1_retail_sales_fact_store_key ENCODING RLE ,
  C2_retail_sales_fact_pos_transaction_number ,
  C3_retail_sales_fact_sales_dollar_amount ,
  C4_retail_sales_fact_cost_dollar_amount )
AS SELECT T_retail_sales_fact.store_key,
  T_retail_sales_fact.pos_transaction_number,
  T_retail_sales_fact.sales_dollar_amount,
  T_retail_sales_fact.cost_dollar_amount
FROM retail_sales_fact T_retail_sales_fact
ORDER BY T_retail_sales_fact.store_key
SEGMENTED BY HASH(T_retail_sales_fact.pos_transaction_number) ALL NODES;
-----
-- Projection #                : 6
-- Projection storage (KBytes) : 4.8e+06
-- Note: This is a super projection for table: retail_sales_fact
CREATE PROJECTION retail_sales_fact_P1_B1 (
```

```

C1_retail_sales_fact_store_key ENCODING RLE ,
C2_retail_sales_fact_pos_transaction_number ,
C3_retail_sales_fact_sales_dollar_amount ,
C4_retail_sales_fact_cost_dollar_amount )
AS SELECT T_retail_sales_fact.store_key,
T_retail_sales_fact.pos_transaction_number,
T_retail_sales_fact.sales_dollar_amount,
T_retail_sales_fact.cost_dollar_amount
FROM retail_sales_fact T_retail_sales_fact
ORDER BY T_retail_sales_fact.store_key
SEGMENTED BY HASH(T_retail_sales_fact.pos_transaction_number) ALL NODES
OFFSET 1;
-----

```

Creating Unsegmented Projections Example

The following script used the ALL NODES syntax to create the original three unsegmented superprojections for the store_dimension table, one per node.

The following syntax is significant:

- CREATE PROJECTION creates a superprojection called store_dimension.
- ALL NODES automatically places a complete copy of the superprojection on each of the three original nodes.

```

CREATE PROJECTION store_dimension (
C0_store_dimension_floor_plan_type ENCODING RLE ,
C1_store_dimension_photo_processing_type ENCODING RLE ,
C2_store_dimension_store_key ,
C3_store_dimension_store_name ,
C4_store_dimension_store_number ,
C5_store_dimension_store_street_address ,
C6_store_dimension_store_city ,
C7_store_dimension_store_state ,
C8_store_dimension_store_region ,
C9_store_dimension_financial_service_type ,
C10_store_dimension_selling_square_footage ,
C11_store_dimension_total_square_footage ,
C12_store_dimension_first_open_date ,
C13_store_dimension_last_remodel_date )
AS SELECT T_store_dimension.floor_plan_type,
T_store_dimension.photo_processing_type,
T_store_dimension.store_key,
T_store_dimension.store_name,
T_store_dimension.store_number,
T_store_dimension.store_street_address,
T_store_dimension.store_city,
T_store_dimension.store_state,
T_store_dimension.store_region,
T_store_dimension.financial_service_type,
T_store_dimension.selling_square_footage,
T_store_dimension.total_square_footage,
T_store_dimension.first_open_date,
T_store_dimension.last_remodel_date
FROM store_dimension T_store_dimension

```

```
ORDER BY T_store_dimension.floor_plan_type,  
T_store_dimension.photo_processing_type  
UNSEGMENTED ALL NODES;
```

To create another copy of the superprojection on the fourth node (Node04), the best approach is to create a copy of that projection on Node04 only. This means avoiding the ALL NODES syntax. The following script shows how to create the fourth superprojection.

The following syntax is significant:

- **CREATE PROJECTION** creates a superprojection called `store_dimension_Node04`.
- **UNSEGMENTED SITE Node04** creates the projection on just Node04.

```
CREATE PROJECTION store_dimension_Node04 (  
  C0_store_dimension_floor_plan_type ENCODING RLE ,  
  C1_store_dimension_photo_processing_type ENCODING RLE ,  
  C2_store_dimension_store_key ,  
  C3_store_dimension_store_name ,  
  C4_store_dimension_store_number ,  
  C5_store_dimension_store_street_address ,  
  C6_store_dimension_store_city ,  
  C7_store_dimension_store_state ,  
  C8_store_dimension_store_region ,  
  C9_store_dimension_financial_service_type ,  
  C10_store_dimension_selling_square_footage ,  
  C11_store_dimension_total_square_footage ,  
  C12_store_dimension_first_open_date ,  
  C13_store_dimension_last_remodel_date )  
AS SELECT T_store_dimension.floor_plan_type,  
  T_store_dimension.photo_processing_type,  
  T_store_dimension.store_key,  
  T_store_dimension.store_name,  
  T_store_dimension.store_number,  
  T_store_dimension.store_street_address,  
  T_store_dimension.store_city,  
  T_store_dimension.store_state,  
  T_store_dimension.store_region,  
  T_store_dimension.financial_service_type,  
  T_store_dimension.selling_square_footage,  
  T_store_dimension.total_square_footage,  
  T_store_dimension.first_open_date,  
  T_store_dimension.last_remodel_date  
FROM store_dimension T_store_dimension  
ORDER BY T_store_dimension.floor_plan_type,  
T_store_dimension.photo_processing_type  
UNSEGMENTED NODE Node04;
```

Implementing Security

In Vertica, there are three primary security concerns:

- Client authentication that prevents unauthorized access to the database.
- Connection encryption that prevents the interception of data, as well as authenticating the the identity of the server and the client.

- Client authorization that controls what users can access and change in the database.

Database Authentication

To gain access to Vertica, a user or client application must supply the name of a valid user account. While you can configure Vertica to just require a user name, you will usually require an additional means of authentication, such as a password. There are several ways to implement this added authentication:

- **Password authentication** (page 115) using passwords stored in the database.
- **Authentication using outside means** (page 118), such as LDAP or Kerberos.

Different authentication methods can be used based on the based on the connection type, client IP address range, and user name for the client that is attempting to access the server.

Connection Encryption

To secure the connection between the client and the server, you can configure Vertica and database clients to use Secure Socket Layer (SSL) to communicate. Vertica uses SSL to:

- Authenticate the server, so the client can confirm the server's identity. Vertica supports mutual authentication in which the server can also confirm the identity of the client. This authentication helps prevent "man-in-the-middle" attacks.
- Encrypt data sent between the client and database server to significantly reduce the likelihood that the data can be read if the connection between the client and server is compromised.
- Verify that data sent between the client and server has not been altered during transmission.

See **Implementing SSL** (page 130).

Controlling Database Authorization

Database users should have access to just the database resources they need to perform their tasks. For example, some users may only need to perform queries on certain sets of data. To prevent unauthorized access to additional data, their access should be limited to just the data that they need to perform their queries. Also, they should only be able to read the data, not modify or insert new data. Other users may need less restrictive access, such as the right to create and modify schemas, tables, and views. Users may also need the ability to grant other users access to database resources. All of these authorizations are set through a collection of statements that control the resources users can access. For more information about the privileges associated with these resources, see:

- **Schema Privileges** (page 136)
- **Table Privileges** (page 137)
- **View Privileges** (page 137)
- **Projection Privileges** (page 138)
- **External Procedure Privileges** (page 138)
- **Metadata Privileges** (page 139)

Use the GRANT statements (GRANT (Database), GRANT (Schema), GRANT (Table), GRANT (View), and GRANT (Procedure)) to assign privileges to users and the Revoke statements REVOKE (Database), (REVOKE (Schema), (REVOKE (Table), (REVOKE (View), and REVOKE (Procedure)) to repeal them. See the SQL Reference Manual for more information about these statements.

Implementing Client Authentication

This section describes the authentication methods supported at the database server layer. For communication layer authentication between server and client, refer to *Implementing SSL* (page 130).

When attempting to connect to a database server, the client application is required to provide the user name for the account established for its use on the server. Vertica uses client authentication to determine whether the client application (or the user who is running the client application) is permitted to connect to the server using the database user name provided. Vertica supports the following client authentication methods:

- trust — Authenticates clients based on valid user names only. You might want to implement trust if a user connection has already been authenticated through some external means such as SSL or a fire wall.
- reject — Rejects the connection and prevents additional records from being evaluated for the client. Use this setting to filter out clients that match this record. For example, this is useful for rejecting specific clients based on user name or IP address.
- krb5 — Authenticates the client using Kerberos version 5. This is useful if users have already been provisioned for Kerberos.
- gss — Authenticates the client using GSS-encoded Kerberos tokens. (Vertica follows RFC 1964.) This is useful if your application uses the Generic Security Services Application Programming Interface (GSS-API).
- ldap — Authenticates the client using Lightweight Directory Access Protocol (LDAP). This is useful if your application uses LDAP to query directory services.
- md5 — Requires the client to supply an MD5-hashed password across the network for authentication. By default, all account passwords are encrypted using Message-Digest algorithm 5 (MD5). The server provides the client with salt (random bytes included in the hash to prevent replay attacks).
- password — Requires the client to supply the password in clear text. Do not use this setting on untrusted networks.

The method Vertica uses to authenticate a particular client connection can be automatically selected on the basis of the connection type, client IP address, and user name.

Note: If you do not choose a client authentication method, Vertica defaults to using user name and password (if supplied) to grant access to the database.

Password Authentication

The simplest method of authenticating a client is to assign the user account a password in Vertica. If a user account has a password set, then a user or client using the account to connect to the database must supply the correct password. If the user account does not have a password set and Vertica is not configured to use another form of client authentication, the user account is always allowed to log in.

Passwords are stored in the database in an encrypted format to prevent others from potentially stealing them. However, the transmission of the password to Vertica is in plain text. This means it is possible for a "man in the middle" attack to intercept the password. In order to secure the login, you should consider *implementing SSL security* (page 130) or MD5 authentication.

Passwords are assigned to user accounts when they are created or afterwards (see the CREATE USER and ALTER USER statements in the SQL Reference Manual). Passwords can be changed using ALTER USER or the vsql **password** (page 378) command. The superuser can set any user account's password. Users can change their own passwords.

To make password authentication more effective, you can enforce password policies that control how often users are forced to change passwords and the required content of a password. These policies are set using *Profiles* (page 115).

Profiles

You set password policies using profiles. A profile is a group of parameters that set requirements for user's passwords. You assign users to a profile to set their password policy.

A profile controls:

- How often users must change their passwords.
- How many times users must change their passwords before they can reuse an old password.
- How many times users can fail to log in before their account is locked.
- The required length and content of the password (maximum and minimum amount of characters and the minimum number of letters, capital letters, lowercase letters, digits, and symbols that must be in a password).

You can create multiple profiles to enforce different password policies for different users. For example, you may choose to create one profile for interactive users that requires them to frequently change their passwords and another profile for user accounts that applications use to access the database that aren't required to change passwords.

You create profiles using the CREATE PROFILE statement and change them using ALTER PROFILE. You can assign a user to a profile when you create the user (CREATE USER) and afterwards the using the ALTER USER statement. A user is assigned to just one profile at a time.

When the database is created, it contains an initial profile named DEFAULT. This is the profile that users are assigned if you do not explicitly assign them a profile when creating them. They are also assigned to the DEFAULT profile if you drop the profile to which they are currently assigned. You can change the policy parameters in the DEFAULT profile, but you cannot delete it.

Note: When upgrading from versions of Vertica prior to 4.1, a DEFAULT profile is added to each database, and all users are assigned to it.

The profiles you create can inherit some or all of their policy parameters from the DEFAULT profile. When creating a profile using the CREATE PROFILE statement, any parameter you set to the special value DEFAULT or any parameter to which you do not assign a value inherits its value from the DEFAULT profile. Making a change to a parameter in the the DEFAULT profile changes that parameter's value in every profile that inherited the parameter from DEFAULT.

When you assign users to a profile (or alter an existing profile that has users assigned to it), the profile's policies for password content (maximum and minimum length and number of specific types of characters) do not have an immediate effect on the users—Vertica does not test user's passwords to ensure they comply with the new password criteria. These settings only affect the users the next time they change their password. If you want to ensure users comply with the new password policy, you can use the ALTER USER statement to expire user passwords. Users with expired passwords are prompted to their change passwords when they next log in.

Note: Only the profile settings for how many failed login attempts trigger account locking and how long accounts are locked have an effect on external password authentication methods such as LDAP or Kerberos. All password complexity, reuse, and lifetime settings only have an effect on passwords managed by Vertica.

Password Expiration

You can use profiles to control how often users must change their passwords. Initially, the DEFAULT profile is set so that passwords never expire. You can change this default value, or you can create additional profiles that set time limits for passwords and assign users to them.

When a password expires, the user is required to change his or her password when next logging in, unless the profile to which the user is assigned has a PASSWORD_GRACE_TIME set. In that case, the user is allowed to log in after the expiration, but will be warned about the password expiration. Once the grace period elapses, the user is forced to change passwords, unless he or she has manually changed the password during the grace period.

Password expiration has no effect on any of the user's current sessions.

Note: You can expire a user's password immediately using the ALTER USER statement's PASSWORD_EXPIRE argument. Expiring a password useful when you want to force users to comply with a change to their password policy, or when setting a new password for users who have forgotten their old one.

Account Locking

One password policy you can set in a profile is how many consecutive failed login attempts (giving the wrong password when trying to log in) a user account is allowed before the user account is locked. You set this value using the FAILED_LOGIN_ATTEMPTS parameter in the CREATE PROFILE or ALTER PROFILE statement.

Vertica locks any user account that has more sequential failed login attempts than the than the value to which you set FAILED_LOGIN_ATTEMPTS. A locked account is not allowed to log in, even if the user supplies the correct password.

There are two ways an account can be unlocked:

- Manually unlocked by the superuser using the ALTER USER command.
- Vertica automatically unlocks the account after the number of days set in the PASSWORD_LOCK_TIME parameter of the user's profile have passed. However, if this parameter is set to UNLIMITED, account is never automatically unlocked and must be manually unlocked).

This locking mechanism helps prevent dictionary-style brute-force attempts to crack users' passwords.

Note: The superuser account cannot be locked, since it is the only user that can unlock accounts. For this reason, you should ensure that you choose a very secure password for the superuser account. See *Password Guidelines* (page 117) for suggestions on choosing a secure password.

The following examples demonstrates failing to login to an account whose profile is set to lock accounts that fail to login after three tries:

```
> vsql -U dbuser
Password:
vsql: FATAL:  Invalid username or password
> vsql -U dbuser
Password:
vsql: FATAL:  Invalid username or password
> vsql -U dbuser
Password:
vsql: FATAL:  The user account "dbuser" is locked due to too many invalid logins
HINT:  Please contact the database administrator
> vsql -U dbuser
Password:
vsql: FATAL:  The user account "dbuser" is locked due to too many invalid logins
HINT:  Please contact the database administrator
```

In the last attempt, the correct password is given for the dbuser account.

Password Guidelines

For passwords to be effective, they must be hard to guess by both dictionary-style brute-force attacks and by having knowledge of the password holder (family names, dates of birth, etc.). You can use profiles to enforce some good password practices (password length and required content), but others (such as the use of personal information) are something you will need to communicate to database users.

Consider these guidelines published by the Internet Engineering Task Force (IETF) when creating passwords:

- Use a password with mixed-case characters.
- Use a password containing non-alphabetic characters (for example, digits and punctuation).
- Use a password that is easy to remember, so that you don't need to write it down.
- Use a password that you can type quickly without having to look at the keyboard.

Avoid the following:

- Do not use your login or user name in any form (as-is, reversed, capitalized, doubled, and so on).

- Do not use your first, middle, or last name in any form.
- Do not use your spouse's, significant other's, children's, friend's, or pet's name in any form.
- Do not use other information easily obtained about you, including your date of birth, license plate number, telephone number, social security number, make of your automobile, house address, and so on.
- Do not use a password of all digits or all the same letter.
- Do not use a word contained in English or foreign language dictionaries, spelling lists, acronym or abbreviation lists, or other lists of words.
- Do not use a password containing fewer than six characters.
- Do not give your password to another person for any reason.

Configuring External Authentication Methods in `vertica.conf` file

External authentication methods are configured using records inserted into the `vertica.conf` file through the Administration Tools. To provide options for client sessions that might require a variety of authentication settings, Vertica supports using multiple records. Each record establishes the authentication method to use based on the connection type, client IP address range, and user name for the client that is attempting to access the database. For example, you could use multiple records to have application logins authenticated using Vertica-based passwords, and interactive users authenticated using LDAP.

The first record with a matching connection type, client address, and user name is used to perform the authentication for that connection.

If the authentication fails, access is denied. Access is also denied if no records match the client session. If, however, there are no records, Vertica reverts to using the user name and password (if created) to control access to the database.

Creating Records

To make it easier to implement external authentication methods, Vertica provides an editing environment within the Administration Tools that enables you to create, edit, and maintain authentication records. It also verifies that the records are correctly formed, inserts them into the configuration file, `vertica.conf`, and implements them cluster-wide.

The actual editing of the records is performed by the text editor set in your Linux or UNIX account's VISUAL or EDITOR environment variable. If no environmental variables are set, the editor Vertica uses is vim (`vi`).

Note: Vertica Systems, Inc. does not recommend editing `vertica.conf` directly, since Administration Tools performs error checking on your entries before adding them to the `vertica.conf`.

To configure client authentication, all the hosts in the cluster must be up. Once you have verified that the hosts are all running, use the Administration Tools to create authentication records for various client sessions.

Creating and Editing Records

To create or edit records:

- 1 On the Main Menu in the Administration Tools, select **Configuration Menu**, and click **OK**.

- 2 On the **Configuration Menu**, select **Edit Authentication**, and click **OK**.
- 3 Select the database for which you want to establish client authentication and click **OK**.
The editor is displayed where you can specify the authentication method used for various sessions.
- 4 Enter one or more records.
See *Creating Records* (page 119) for a description of the content required to create a record. See also *Formatting Rules for Records* (page 121).
- 5 When you have finished entering records, exit the editor. For example, in vi, press the Esc key and type :wq to complete your editing session.
- 6 The authentication tool verifies that the records are correctly formed and does one of the following:
 - If the records are properly formed, they are inserted into the `vertica.conf` file and you are prompted to restart the database. Click OK and go to step 7.
 - If the records are not properly formed, the display indicates why they weren't properly formed and gives you the opportunity to: edit your errors (e), abort without saving your changes (a), or save and implement your changes anyway (s). Saving your changes is not generally recommended because it can cause client authentication to fail.
- 7 **Restart the database** (page 142).

Using GNU Emacs as the Editor of Choice

To switch the editor from vi to GNU Emacs, enter the following before running the Administration Tools:

```
export EDITOR=/usr/bin/emacs
```

Record Content

To specify a record, enter the following:

```
ClientAuthentication = <connection type> <user name> <address> <method>
```

Where:

- *<connection type>* is the method the client uses to connect to an instance. Valid values are:
 - `local` — Matches connection attempts made using local domain sockets. When using the local connection type, do not specify the *<address>* parameter.
 - `host` — Matches connection attempts made using TCP/IP. Connection attempts can be made using a plain (non-SSL) or SSL-wrapped TCP socket.
 - `hostssl` — Matches a SSL TCP connection only.
 - `hostnossl` — Matches a plain TCP socket only.

For client connections: Avoid using `-h hostname` from the client if a "local" connection type is specified and you want to match the client authentication entry. See the `vsq` command line option ***h hostname*** (page 362).

- *<user name>* identifies the client's user name. Valid user names are:
 - `all` — Matches all users.
 - One or more specific user names.

User Name accepts either a single value or concatenated values. To concatenate values, use a plus sign between the values. For example: `user1+user2`

- `<address>` identifies the client machine IP address range. Use a format of `<IP Address>/<netmask value>`. IP address must be specified numerically, not as domain or host names. Vertica supports the following formats:

- `w.x.y.z/mask` format (For example, `10.10.0.25/24`.)

The mask length indicates the number of high-order bits of the client IP address that must match. Do not insert any white space between the IP address, the slash (/), and the Classless Inter-Domain Routing (CIDR) mask length.

- Separate dotted-IP address and mask values (For example, `10.10.0.25 255.255.255.0`.)

Note: If working with a cluster with multiple nodes, be aware that any IP/netmask settings in host-based ClientAuthentication parameters (`host`, `hostssl`, or `hostnossl`) must match all nodes in the cluster. This allows the database owner to authenticate with and administer every node in the cluster. For example, specifying `10.10.0.8/30` would allow a CIDR address range of `10.10.0.8 - 10.10.0.11`.

- Method identifies the authentication method to be used for clients that match this record. Use one of the following:

- `trust` — Authenticates clients based on valid user names only. You might want to implement `trust` if a user connection has already been authenticated through some external means such as SSL or a fire wall.
- `reject` — Rejects the connection and prevents additional records from being evaluated for the client. Use this setting to filter out clients that match this record. For example, this is useful for rejecting specific clients based on user name or IP address.
- `krb5` — Authenticates the client using Kerberos version 5. This is useful if users have already been provisioned for Kerberos.
- `gss` — Authenticates the client using GSS-encoded Kerberos tokens. (Vertica follows RFC 1964.) This is useful if your application uses the GSS API.
- `ldap` — Authenticates the client using Lightweight Directory Access Protocol (LDAP). This is useful if your application uses LDAP to query directory services.
- `md5` — Requires the client to supply an MD5-hashed password across the network for authentication. By default, all account passwords are encrypted using `md5`. The server provides the client with salt (random bytes included in the hash to prevent replay attacks).
- `password` — Requires the client to supply the password in clear text. Do not use this setting on untrusted networks.

Note for client connections: Use `-h` hostname from the client if either a `"gss"` or `"krb5"` (Kerberos) connection method is specified. See the `vsq` command line option **`h hostname`** (page 362).

Specifying Records for LDAP

If you specify LDAP as the authentication method, you need to include the URL for LDAP in the ClientAuthentication parameter.

Examples

The following example uses the Administration Tools to specify LDAP as the authentication method:

```
ClientAuthentication = host all 10.0.0.0/8 ldap
"ldap://summit.vertica.com;/cn=;,dc=vertica,dc=com"
```

Where:

- The URL for the LDAP server is ldap://summit.vertica.com.
For connections over SSL, you can use S_HTTP. For example: ldaps://.
- For vsql, the prefix (cn=) and suffix (,dc=vertica,dc=com) are used to construct the entire Distinguished Name (DN) for the user. This means that the client only has to supply the user name.
For example the DN for user "jsmith" would be: cn=jsmith,dc=vertica,dc=com.
For ODBC, the SQLConnect function sends the user name and password combination to the database for authentication. If the client IP address and user name combination matches an LDAP ClientAuthentication entry in vertica.conf, the LDAP server is contacted.
For JDBC, the java.sql.DriverManager.getConnection() function passes the user name and password combination to the database for authentication. If the client IP address and user name combination matches an LDAP ClientAuthentication entry in vertica.conf, the LDAP server is contacted.

The following example uses a configuration parameter to specify the same LDAP authentication method:

```
=> SELECT SET_CONFIG_PARAMETER('ClientAuthentication',
'host all 10.0.0.0/8 ldap
"ldap://summit.vertica.com;cn=;,dc=vertica,dc=com"');
```

Formatting Rules for Records

When specify one or more records, follow these rules:

- Only one record is allowed per line.
- Do not use a carriage return to force records to span lines. A single record must be on one line unless it runs out of space and spills onto a second line.
- Fields that make up the record can be separated by white space or tabs.
- Other than IP addresses and mask columns, field values cannot contain white space.

See **Example Records** (page 121).

Example Records

The following examples illustrate how to specify client authentication through the Administration Tools. For examples of using configuration parameters to modify records, see **Modifying Records** (page 122).

Specifying One Record

The following example specifies a record for the trust method:

```
ClientAuthentication = hostnossl dbadmin 0.0.0.0/0 trust
```

Specifying One Record that Uses the LDAP Method

The following example specifies a record for the LDAP method:

```
ClientAuthentication = host all 10.0.0.0/8 ldap
"ldap://summit.vertica.com;cn=;,dc=vertica,dc=com"
```

Specifying Three Records

The following example specifies three records. Note that each record is placed on a separate line:

```
ClientAuthentication = hostnossl dbadmin 0.0.0.0/0 trust
ClientAuthentication = hostnossl all 0.0.0.0/0 md5
ClientAuthentication = local all trust
```

Modifying Records

To modify records, you can either use the Administration Tools to edit an existing record or use a configuration parameter.

Using the Administration Tools

The advantages of using this tool are:

- You do not have to connect to the database.
- The editor verifies that records are correctly formed.
- The editor maintains records, so they are available to you for editing at another point in time.

Note: You must restart the database to implement your changes.

For information about using the Administration Tools to create and edit records, see **Creating Records** (page 118).

Using a Configuration Parameter

The advantage of using a configuration parameter is that the changes are implemented immediately across all nodes within the database cluster. You do not need to restart the database.

However, all the database nodes must be up and you must connect to the database prior to setting this parameter. Additionally, this method does not verify that records are correctly formed and it does not maintain the records so they can be modified at a later point.

To configure client authentication through a connection parameter, use the `set_config_parameter` function as follows:

```
SELECT SET_CONFIG_PARAMETER('ClientAuthentication,' '<connection type> <user
name> <address> <method>');
```

When specifying records be sure to adhere to the following guidelines:

- Fields that make up the record can be separated by white space or tabs.
- Other than IP addresses and mask columns, field values cannot contain white space.

For more information, see **Record Content** (page 119).

Examples

The following example specifies a record for the trust method:

```
SELECT SET_CONFIG_PARAMETER('ClientAuthentication',
'hostnossl dbadmin 0.0.0.0/0 trust');
```

The following example specifies a record for the LDAP method:

```
SELECT SET_CONFIG_PARAMETER('ClientAuthentication',
'host all 10.0.0.0/8 ldap "ldap://summit.vertica.com;cn=;,dc=vertica,dc=com"');
```

The following example specifies three records. Note that each record is separated by a comma:

```
SELECT SET_CONFIG_PARAMETER('ClientAuthentication',
'hostnossl dbadmin 0.0.0.0/0 trust, hostnossl all 0.0.0.0/0 md5, local all trust');
```

Authenticating Using LDAP or Kerberos

Instead of using Vertica's own password features, you can choose to authenticate users via an external means, such as an LDAP or Kerberos server.

Before configuring Vertica to use an external client authentication system, you must first set up the service you want to use. See the documentation for your authentication service.

General Prerequisites

If an authentication method requires access to a remote server (such as Kerberos and LDAP), the server must be available or clients cannot be authenticated using this method. If clients cannot be authenticated, do not grant them access to the database.

Kerberos Prerequisites

- Both the client identity and the Vertica server must be configured as Kerberos principals in the centralized user store or Kerberos Key Distribution Center (KDC).
- To participate in authentication through Kerberos, ODBC and JDBC clients must contain code written directly to the Kerberos API. It is not sufficient for them to pass a username/domain name and password to the server. (See ***Kerberos Client Code Example Written in C*** (page 127) and ***Kerberos Client Code Example Written in JAVA*** (page 124).) vsql does not have this restriction because it is already Kerberos-aware.

LDAP Prerequisites

The LDAP directory must contain a record for each client identity.

Configuring Authentication Through Kerberos and GSS

To enable authentication through Kerberos or GSS, Kerberos- and GSS- enabled clients require knowledge about the authentication protocol in use. If you are using Kerberos or GSS as an authentication method, specify the following parameters.

Parameter	Description
KerberosRealm	A string that provides the Kerberos domain name. Usually it consists of all uppercase letters. Example: KerberosRealm=VERTICA.COM

KerberosHostname	A string that provides the Kerberos host name for the KDC server where Kerberos is running. This parameter is optional. If not specified, Vertica uses the return value from the function <code>gethostname()</code> . Example: <code>KerberosHostname=Host1</code>
KerberosKeytabFile	A string that provides the location of the keytab file. By default, the keytab file is located in <code>/etc/krb5.keytab</code> . The keytab file requires read and write permission only for the file owner who is running the process. Under Linux, for example, file permissions would be: <code>0600</code> . Example: <code>KerberosKeytabFile=/etc/krb5.keytab</code>
KerberosServiceName	A string that provides the Kerberos service name. By default, the service name is <code>'vertica'</code> . To construct the principal, follow the format: <code>KerberosServiceName/Kerberos Hostname@Kerberos Realm</code> Example: <code>KerberosServiceName='vertica'</code>

Note: The same parameters and syntax apply for both Kerberos and GSS.

To specify a parameter, set the configuration parameter, as follows:

`ClientAuthentication = Kerberos_Parameter Value`

Where:

- *Kerberos_Parameter* is one of the following: `KerberosRealm`, `KerberosHostname`, `KerberosKeytabFile`, or `KerberosServiceName`.
- *Value* is the value of the parameter.

Example

```
ClientAuthentication = KerberosRealm .VERTICA.COM
```

Kerberos Client Code Example Written in Java

To participate in authentication through Kerberos, JDBC clients must contain code written to directly to the Kerberos API. It is not sufficient for them to pass a username/domain name and password to the server. The following example Java client code is written directly to the Kerberos API to initialize the Kerberos context and authenticate using Kerberos. It is provided for the MIT implementation of Kerberos only.

```
import org.ietf.jgss.Oid;
import org.ietf.jgss.GSSContext;
import org.ietf.jgss.GSSCredential;
import org.ietf.jgss.GSSException;
import org.ietf.jgss.GSSManager;
import org.ietf.jgss.GSSName;
/**
 * This sample code indicates how to create a custom Java application
 * (perhaps in concert with JDBC) to use Kerberos GSS authentication
 * with the Vertica analytic database.
 */
```

```
* Note that GSSAPI applications must have access to a previously-
* created authentication token. In this case, the user must run
* kinit to create a Kerberos TGT (ticket granting ticket) prior to
* invoking this client.
*/

public class SampleGSSClient
{
private static String GSS_API_KRB_OID_STR = "1.2.840.113554.1.2.2";
private static Oid GSS_API_KRB_OID;
private static GSSManager _manager;
private String _host;
private String _port;
private String _username;
private String _dbname;
private int port;
public static void main (String[] args)
{
if (args.length != 4)
{
System.err.println ("Usage: SampleGSSClient <host> 5433 <username>
<databasename>");
return;
}
try {
GSS_API_KRB_OID = new Oid(GSS_API_KRB_OID_STR);
} catch (GSSException ge) {
System.err.println ("Could not instantiate Kerberos GSSAPI Oid; out-of-date
JRE?");
ge.printStackTrace();
return;
}
_manager = GSSManager.getInstance ();
SampleGSSClient client = new SampleGSSClient ();
client._host = args[0];
client._port = args[1];
client._username = args[2];
client._dbname = args[3];
client.go();
}
/**
* Main entry point for GSSAPI functionality.
*/
private void go()
{
try {
GSSContext ctx = getGSSContext();
ctx.requestConf(true);
ctx.requestInteg(true);
ctx.requestReplayDet(true);
if (!establishContext(ctx))
{
System.err.println("");
return;
}
}
}
```

```
}
} catch (GSSEException ge) {
System.err.println("Failed to create GSS context: " + ge);
ge.printStackTrace();
}
return;
}
/**
 *
 */
private GSSContext getGSSContext() throws GSSEException
{
GSSCredential clientcred = getGSSCredential();
if (clientcred == null)
{
return null;
}
System.out.println("Created GSS credential: " + clientcred);
GSSName server = getGSSServerName();
if (server == null)
{
return null;
}
System.out.println("Created GSS server name: " + server);
return _manager.createContext (server,
GSS_API_KRB_OID,
clientcred,
GSSCredential.DEFAULT_LIFETIME);
}
/**
 *
 */
private GSSCredential getGSSCredential () throws GSSEException
{
GSSName client = getGSSClientName();
if (client == null)
{
return null;
}
System.out.println("Created GSS client name: " + client);
return _manager.createCredential (client,
GSSCredential.INDEFINITE_LIFETIME,
GSS_API_KRB_OID,
GSSCredential.INITIATE_ONLY);
}
/**
 *
 */
private GSSName getGSSClientName () throws GSSEException
{
return _manager.createName (_username,
GSSName.NT_USER_NAME);
}
/**
```

```

* This class creates the server Kerberos principal, which
* is usually of the form:
* <service name>/<hostname>@<realm name>
*
* Note that these three parameters correspond, respectively,
* to the KerberosServiceName, KerberosHostname, and KerberosRealm
* parameters. For this example, we use the default service
* name value of "vertica".
*/
private GSSName getGSSServerName () throws GSSException
{
return _manager.createName ("vertica/" + _host,
GSSName.NT_HOSTBASED_SERVICE,
GSS_API_KRB_OID);
}
/**
*
*/
private boolean establishContext (GSSContext ctx) throws GSSException
{
byte[] token = new byte[0];
while (!ctx.isEstablished())
{
token = ctx.initSecContext(token,
0, // offset
token.length); // length
if (token != null)
{
System.out.println("Sending token: " + token);
}
}
return true;
}
} // end SampleGSSClient

```

Kerberos Client Code Example Written in C

To participate in authentication through Kerberos, ODBC clients must contain code written to directly to the Kerberos API. It is not sufficient for them to pass a username/domain name and password to the server. The following example C client code is written directly to the Kerberos API to initialize the Kerberos context and authenticate using Kerberos. It is provided for the MIT implementation of Kerberos only.

```

#include <krb5.h>
#include <com_err.h> /* Optional error code -> error string handling */
static krb5_context k5ctx;
static krb5_keytab k5keytab;
static krb5_ccache k5cache;
static krb5_principal k5princ;
static char* k5name = NULL;

/*
* To initialize your Kerberos context:
*/

```

```
int
krb5_init(const char* hostname,
          const char* realm,
          const char* keytab)
{
    krb5_error_code k5err;
    if (!keytab || !hostname || !realm)
    {
        /* Error handling */
    }

    k5err = krb5_init_context(&k5ctx);
    if (k5err)
    {
        /* Error handling */
    }

    k5err = krb5_cc_default(k5ctx, &k5cache);
    if (k5err)
    {
        krb5_free_context(k5ctx);
        /* Error handling */
    }

    k5err = krb5_cc_get_principal(k5ctx, k5cache, &k5princ);
    if (k5err)
    {
        krb5_cc_close(k5ctx, k5cache);
        krb5_free_context(k5ctx);
        /* Error handling */
    }

    k5err = krb5_kt_resolve(k5ctx, keytab, &k5keytab);
    if (k5err)
    {
        krb5_free_context(k5ctx);
        /* Error handling */
    }

    k5err = krb5_unparse_name(k5ctx, k5princ, &k5name);
    if (k5err)
    {
        krb5_free_principal(k5ctx, k5princ);
        krb5_cc_close(k5ctx, k5cache);
        krb5_free_context(k5ctx);
        /* Error handling */
    }

    /*
     * Optionally call krb5_an_to_ln, assuming your principal
     * identifier before either '/' or '@' is your database username.
     */
    return 0;
}
```

```

/*
 * To authenticate using Kerberos; assumes prior initialization above.
 */
int
krb5_auth(int socket, const char* hostname, const char* svcname) {
    int flags;
    krb5_auth_context k5authctx;
    krb5_principal k5srvprinc;
    krb5_error_code k5err;
    krb5_error* k5errstr;

    /* MIT Kerberos requires a non-blocking socket */
    flags = fcntl(socket, F_GETFL);
    flags |= O_NONBLOCK;
    if (fcntl(socket, flags))
    {
        /* Error handling */
    }

    k5err = krb5_sname_to_principal(k5ctx,
                                   hostname,
                                   svcname,
                                   KRB5_NT_SRV_HST,
                                   &k5srvprinc);

    if (k5err)
    {
        /* Error handling
    }

    k5err = krb5_sendauth(k5ctx,
                           &k5authctx,
                           (krb5_pointer)&socket,
                           PG_KRB_SRVNAM,
                           svcname,
                           server,
                           AP_OPTS_MUTUAL_REQUIRED,
                           NULL, /* Don't need to
supply any other authentication data */
                           0, /* Don't
need to supply creds; use credential cache; this assumes prior kinit execution */
                           k5cache,
                           &k5errstr,
                           NULL, /* Don't need
apreq material */
                           NULL); /* Don't need
returned creds */
    krb5_free_principal(k5ctx, k5svrprinc);
    flags &= ~O_NONBLOCK;
    fcntl(socket, flags); /* Assumes will succeed */

    if (k5err || k5errstr)
    {

```

```
    krb5_free_principal(k5ctx, k5svrprinc);
    /* Error handling */
    if (k5errstr)
        krb5_free_error(k5ctx, k5errstr);
}

return 0;
}
```

Implementing SSL

Vertica supports Secure Socket Layer (SSL) v3/Transport Layer Security (TLS) 1.0 traffic between the database and its clients. SSL/TLS provides for the privacy and integrity of data exchanged between Vertica and its clients. Vertica supports the following under SSL/TLS:

- **SSL Server Authentication** — Allows the client to confirm the server's identity by verifying that the server's certificate and public key are valid and were issued by a certificate authority (CA) listed in the client's list of trusted CAs. See "Required Prerequisites for SSL Server Authentication and SSL encryption" section under **SSL Prerequisites** (page 130) and **Configuring SSL** (page 134).
- **SSL Client Authentication** — (Optional) Allows the server to confirm the client's identity by verifying that the client's certificate and public key are valid and were issued by a certificate authority (CA) listed in the server's list of trusted CAs. Client authentication is optional because Vertica can achieve authentication at the application protocol level through user name and password credentials. See "Additional Prerequisites for SSL Server and Client Mutual Authentication" section under **SSL Prerequisites** (page 130).
- **Encryption** — Encrypts data sent between the client and database server to significantly reduce the likelihood that the data can be read if the connection between the client and server is compromised. Encryption works both ways whether or not SSL Client Authentication is enabled. See "Required Prerequisites for SSL Server Authentication and SSL encryption" section under **SSL Prerequisites** (page 130) and **Configuring SSL** (page 134).
- **Data Integrity** — Verifies that data sent between the client and server has not been altered during transmission.

Note: For server authentication, Vertica supports using RSA with ephemeral Diffie-Hellman (DH). DH is the key agreement protocol.

SSL Prerequisites

Before you implement SSL security, obtain the appropriate certificate signed by a certificate authority (CA) and private key files and then copy them to your system. (See the **OpenSSL** (<http://www.openssl.org>) documentation.) These files must be in Privacy-Enhanced Mail (PEM) format.

Required Prerequisites for SSL Server Authentication & SSL encryption

Follow these steps to set up SSL authentication of the server by the clients, which is also required in order to provide encrypted communication between server and client.

- 1 On each server host in the cluster, copy the server certificate file (`server.crt`) and private key (`server.key`) to the Vertica catalog directory. (See **Distributing Certifications and Keys** (page 133).)

The public key contained within the certificate and the corresponding private key allow the SSL connection to encrypt the data and ensure its integrity.

Note: The `server.key` file must have read and write permissions for the `dbadmin` user only. Do not provide any additional permissions or extend them to any other users. Under Linux, for example, file permissions would be `0600`.

- 2 On each client, if you want clients to be able to verify the server's certificate, copy the `root.crt` file to the client. If you are using `vsql`, copy the file to: `/home/dbadmin/.vsql/`. This ability is not available for ODBC client at this time.

The `root.crt` file contains either the server's certificate or the CA that issued the server certificate.

Note: If you do not perform this step, the SSL connection is set up and ensures message integrity and confidentiality via encryption; however, the client cannot authenticate the server and is, therefore, susceptible to problems where a fake server with the valid certificate file masquerades as the real server. If the `root.crt` is present but does not match the CA used to sign the certificate, the database will not start.

Additional Prerequisites for SSL Server and Client Mutual Authentication (Optional)

Follow these additional steps to optionally configure authentication of clients by the server.

Setting up client authentication by the server is optional because the server can use alternative techniques, like database-level password authentication, to verify the client's identity. Follow these steps only if you want to have both server and client mutually authenticate themselves with SSL keys.

- 1 On each server host in the cluster, copy the `root.crt` file to the Vertica catalog directory. (See ***Distributing Certifications and Keys*** (page 133).)

The `root.crt` file has the same name on the client and server. However, these files do not need to be identical. They would be identical only if the client and server certificates were used by the same root certificate authority (CA).
- 2 On each client, copy the client certificate file (`client.crt`) and private key (`client.key`) to the client. If you are using `vsql`, copy the files to: `/home/dbadmin/.vsql/`.

If you are using either ODBC or JDBC, you can place the files anywhere on your system and provide the location in the connection string (ODBC/JDBC) or ODBCINI (ODBC only). See ***Configuring SSL for ODBC clients*** (page 134) and ***Configuring SSL for JDBC clients*** (page 135).

Note: If you're using ODBC, the private key file (`client.key`) must have read and write permissions for the `dbadmin` user only. Do not provide any additional permissions or extend them to any other users. Under Linux, for example, file permissions would be `0600`.

Generating Certifications and Keys

For testing purposes, you can create and use simple self-signed certificates. For production, you need to use certificates signed by a certificate authority (CA) so the client can verify the server's identity.

This section illustrates how to create certificate authority (CA) keys and self-signed certificates for testing purposes. It uses the CA private keys to sign "normal" certificates and to generate the server's and client's private key files. For detailed information about creating signed certificates, refer to the **OpenSSL** (<http://www.openssl.org>) documentation.

The server and client keys can be rooted in different CAs.

1 Create the CA private key:

```
$>openssl genrsa -des3 -out servercakey.pem
```

The output file name can vary.

2 Create the CA public certificate:

```
$>openssl req -new -x509 -key servercakey.pem -out root.crt
```

The output file name can vary.

Important: The following is an example of the certificate's contents. When you create a certificate, there must be one unique name (a Distinguished Name (DN)), which is different for each certificate that you create. The examples in this procedure use the Organizational Unit Name for the DN.

```
Country Name (2 letter code) [GB]:US
State or Province Name (full name) [Berkshire]:Massachusetts
Locality Name (e.g., city) [Newbury]:Billerica
Organization Name (e.g., company) [My Company Ltd]:Vertica
Organizational Unit Name (e.g., section) []:Support_CA
Common Name (e.g., your name or your server's hostname) []:myhost
Email Address []:myhost@vertica.com
```

3 Create the server's private key file:

```
$>openssl genrsa -out server.key
```

Note that Vertica supports only unencrypted key files, so there is no `-des3` argument.

4 Create the server certificate request:

```
$>openssl req -new -out reqout.txt -key server.key
```

This step was not required for the CA because CA certificates are self-signed.

You are prompted to enter information that is incorporated into your certificate request. In this example, the Organizational Unit Name contains the unique DN (`Support_server`):

```
Country Name (2 letter code) [GB]:US
State or Province Name (full name) [Berkshire]:Massachusetts
Locality Name (e.g., city) [Newbury]:Billerica
Organization Name (e.g., company) [My Company Ltd]:Vertica
Organizational Unit Name (e.g., section) []:Support_server
Common Name (e.g., your name or your server's hostname) []:myhost
Email Address []:myhost@vertica.com
```

5 Use the CA private key file to sign the server's certificate:

```
$>openssl x509 -req -in reqout.txt -days 3650 -sha1 -CAcreateserial -CA
root.crt -CAkey servercakey.pem -out server.crt
```

6 Create the client's private key file:

```
$>openssl genrsa -out client.key
```

Vertica supports only unencrypted key files, so there is no `-des3` argument.

7 Create the client certificate request:

```
$>openssl req -new -out reqout.txt -key client.key
```

This step was not required for the CA because CA certificates are self-signed.

You are prompted to enter information that is incorporated into your certificate request. In this example, the Organizational Unit Name contains the unique DN (`Support_client`):

```
Country Name (2 letter code) [GB]:US
State or Province Name (full name) [Berkshire]:Massachusetts
Locality Name (e.g., city) [Newbury]:Billerica
Organization Name (e.g., company) [My Company Ltd]:Vertica
Organizational Unit Name (e.g., section) []:Support_client
Common Name (e.g., your name or your server's hostname) []:myhost
Email Address []:myhost@vertica.com
```

8 Use the CA private key file to sign the client's certificate:

```
$>openssl x509 -req -in reqout.txt -days 3650 -sha1 -CAcreateserial -CA
root.crt -CAkey servercakey.pem -out client.crt
```

JDBC Certificates

If you are using JDBC, perform the following steps after you have generated the key and self-signed certificate:

1 Convert the Vertica server certificate to a form that JAVA understands:

```
openssl x509 -in server.crt -out server.crt.der -outform der
```

2 Create a new truststore and imported the certificate into it:

```
keytool -keystore verticastore -alias verticasql -import -file
server.crt.der
```

Distributing Certifications and Keys

Once you have created the prerequisite certifications and keys for one host, you can easily distribute them cluster-wide by using the Administration Tools. Client files cannot be distributed through Administration Tools.

To distribute certifications and keys to all hosts in a cluster:

1 Log on to a host that contains the certifications and keys you want to distribute and start the Administration Tools.

See *Using the Administration Tools* (page 329) for information about accessing the Administration Tools.

2 On the Main Menu in the Administration Tools, select Configuration Menu, and click OK.**3 On the Configuration Menu, select Distribute Config Files, and click OK.****4 Select SSL Keys and click OK.****5 Select the database where you want to distribute the files and click OK.****6 Fill in the fields with the directory `/home/dbadmin/.vsq1/` using the `root.crt`, `server.crt` and `server.key` files to distribute the files.****7 Configure SSL (page 134).**

Configuring SSL

Configure SSL for each server in the cluster.

To enable SSL:

- 1 Ensure that you have performed the steps listed in **SSL Prerequisites** (page 130) minimally for server authentication and encryption, and optionally for mutual authentication.
- 2 Set the EnableSSL parameter to 1. By default, EnableSSL is set to 0 to disable it.

```
=> SELECT SET_CONFIG_PARAMETER('EnableSSL', '1');
```

Note: Vertica fails to start if SSL has been enabled and the server certificate files (`server.crt`, `server.key`) are not in the expected location.

- 3 **Restart the database** (page 142).
- 4 If you are using either ODBC or JDBC, configure SSL for the appropriate client:
 - **Configuring SSL for ODBC Clients** (page 134)
 - **Configuring SSL for JDBC Clients** (page 135)

vsqI automatically attempts to make connections using SSL. If a connection fails, vsqI attempts to make a second connection over clear text.

See Also

Configuration Parameters (page 25) in the Administrator's Guide

Configuring SSL for ODBC Clients

Configuring SSL for ODBC clients requires that you set the SSLMode parameter. If you want to configure optional SSL client authentication, you also need to configure the SSLKeyFile and SSLCertFile parameters.

The method you use to configure the DSN depends on the type of client operating system you are using:

- Linux and UNIX — Enter the parameters in the `odbc.ini` file. See [Creating an ODBC DSN for Linux and Solaris Clients](#).
- Microsoft Windows — Enter the parameters in the Windows Registry. See [Creating an ODBC DSN for Windows Clients](#).

SSLMode Parameter

Set the SSLMode parameter to one of the following for the DSN:

- `always` — Requires the server to use SSL. If the server cannot provide an encrypted channel, the connection fails.
- `prefer` (the default) — Prefers the server to use SSL. If the server does not offer an encrypted channel, the client requests one. The first connection to the database tries to use SSL. If that fails, a second connection is attempted over a clear channel.
- `allow` — Makes a connection to the server whether the server uses SSL or not. The first connection to the database tries to use SSL. If that fails, a second connection is attempted over a clear channel.

- `disable` — Never connects to the server using SSL. This setting is typically used for troubleshooting.

SSLKeyFile Parameter

To configure optional SSL client authentication, set the `SSLKeyFile` parameter to the file path and name of the client's private key. This key can reside anywhere on the client.

SSLCertFile Parameter

To configure optional SSL client authentication, set the `SSLCertFile` parameter to the file path and name of the client's public certificate. This file can reside anywhere on the client.

Configuring SSL for JDBC Clients

To configure JDBC:

- 1 Enable the driver for SSL.
- 2 Configure troubleshooting if desired.

To enable the driver for SSL

For JDBC, the driver must be enabled for SSL. Use a connection parameter when connecting to the database to force a connection using SSL. You can specify a connection parameter within a connection URL or by using an additional properties object parameter to `DriverManager.getConnection`.

- Using a Connection URL

The following example forces a connection using SSL by setting the `ssl` connection parameter to `true`:

```
String url = "jdbc:vertica://VerticaHost://DatabaseName?user=username"
    +
    "&password=password&ssl=true";
Connection conn = DriverManager.getConnection (url);
```

Note: If the server is not SSL enabled, the connection fails. This differs from `vsq`, which can try an unencrypted connection.

- Using an Additional Properties Object Parameter

The following code fragment forces a connection using SSL by establishing an `ssl` connection property:

```
String url = "jdbc:vertica://VerticaHost/DatabaseName";
Properties props = new Properties();
props.setProperty("user", "username"); props.setProperty("password",
    "password");
props.setProperty("ssl", "true");
Connection conn = new Connection(url, props);
```

Note: For compatibility with future versions, specify a value, even though the `ssl` property does not require that a value be associated with it. Specifying a `ssl` property, even without a value of `"true,"` automatically forces a connection using SSL.

To configure troubleshooting

To enable troubleshooting, configure the keystore file that contains trusted certificate authority (CA) certificates:

```
-Djavax.net.debug=ssl
```

-Djavax.net.ssl.trustStore=<keystore file> In the above command:

- Configuring `-Djavax.net.debug=ssl` is optional.
- The keystore file is the same keystore that was updated as part of **Generating Certifications and Keys** (page 131) (JDBC Certificates). Normally, the keystore file is `$HOME/.keystore`. The `keytool` utility takes `server.crt.der` and places it in the keystore.

For details, see "Customizing the Default Key and Trust Stores, Store Types, and Store Passwords" on the java.sun.com

<http://java.sun.com/j2se/1.5.0/docs/guide/security/jsse/JSSERefGuide.html#CustomizingStores> web site.

Implementing Database Authorization

Database authorization controls what a user account has access to in the database. The superuser can grant several types of privileges to users, allowing them to access and manipulate schemas, tables, projections, sequences, external procedures, and metadata.

Schema Privileges

A schema is a namespace for tables that also provides security. By default, only the superuser can create a schema or give a user the right to create a schema. (See GRANT (Database).)

Note: Unlike other RDBMSs, a schema in a Vertica database is not a collection of objects bound to one user.

Only the superuser and the schema owner (typically the person who creates a schema) have access to create objects within the schema. Additionally, only the schema owner and super user can drop or alter a schema. See DROP SCHEMA and ALTER SCHEMA.

Note: The schema owner is typically the user who creates the schema. However, the super user can create a schema and assign ownership of the schema to a different user at creation.

All other access to the schema and its objects must be explicitly granted to specific users by the super user or schema owner as needed. This prevents unauthorized users from accessing the schema and its objects. A user can be granted one of the following privileges through the GRANT SCHEMA statement:

Privilege	Description
USAGE	Allows the user access to look up metadata (e.g. system tables) for those objects contained within the schema for which the user has been granted at least one privilege to select or modify. See Metadata Privileges (page 139) and the GRANT TABLE, GRANT VIEW, and GRANT PROCEDURE statements.

	Tip: Usage by itself does not allow the user the access to select or modify the data in the objects. To select or modify the data in any individual object, the user must additionally be granted the specific privilege pertaining to desired type of access, for example, SELECT or DELETE.
CREATE	Allows the user read access to the schema and the right to create tables and views within the schema.

Note: The PUBLIC schema is present in any newly created Vertica database. However, only the superuser has access to it by default. Newly created users have USAGE privilege on PUBLIC but must be explicitly granted CREATE and individual object privileges for them to be able to create or see tables in the PUBLIC schema.

See the GRANT SCHEMA statement in the SQL Reference Manual for more information about all available schema privileges.

See *Projection Privileges* (page 138) for the schema privilege requirements to create projections.

Table Privileges

Access to tables is strictly controlled. By default, only the super user has access to all tables, and the super user has full privileges on these tables. The person who creates a table (the table owner) also has full privileges on the table he or she creates. The ability to drop or alter a table is reserved for the super user or table owner. See ALTER TABLE and DROP TABLE.

All other users, including the schema owner, must be explicitly granted specific privileges on individual tables within the schema by either the super user or the users who create these tables. See the GRANT TABLE statement for a complete description of all the privileges that can be assigned to users and the operations they allow.

To create a table, a user must be a superuser or have the following privileges:

- CREATE on the schema in which the table is created.

To view the metadata or to query the data in another user's table, a user must be a superuser or have the following privileges:

- USAGE on the schema containing the table.
- (At least) SELECT on the table.

To see which privileges have been assigned, use the following command:

```
=> SELECT * FROM GRANTS;
```

View Privileges

Access to views is strictly controlled. By default, only the super user has access to all views, and the super user has full privileges on these views. The person who creates a view (the view owner) also has full privileges on the view he or she creates.

To create a view, the user must be a superuser or have the following privileges:

- CREATE on the schema in which the view is created.

- SELECT on all the tables and views referenced within the view's defining query.
- USAGE on all the schemas that contain the tables and views referenced within the view's defining query

To drop a view, the user must be a superuser or the person who created it.

Since a view is a stored query but does not have stored data, it is read-only. That means the only privilege that applies to using a view is SELECT. By default, only the superuser and the user who creates a view has SELECT privilege on a view. All other users must be granted SELECT on a view by either the superuser or the user who created the view. See the GRANT View statement for more information.

To see which privileges have been assigned, use the following command:

```
=> SELECT * FROM GRANTS;
```

Projection Privileges

To create a projection, the user must be a superuser or have the following privileges:

- CREATE on the schema in which the projection is created
- SELECT on all the tables referenced by the projection
- USAGE on all the schemas that contain the tables referenced by the projection

Note: When a user issues a query, Vertica checks that the user has SELECT privilege on the tables in the query and automatically chooses the appropriate projection. No separate privileges are required to be granted on the projection object.

Sequence Privileges

Access to sequences is strictly controlled. By default, only the super user has access to all sequences, and the super user has full privileges on these sequences. The person who creates a sequence (the sequence owner) also has full privileges on the sequence he or she creates.

All other users, including the schema owner, must be granted privileges to use individual sequences within the schema by either the super user or the users who create these sequences. See the GRANT_SEQUENCE statement for a complete description of all the privileges that can be assigned to users. Additionally, all users must have USAGE privileges on the schema that the sequence belongs to.

To see which privileges have been assigned, use the following command:

```
=> SELECT * FROM GRANTS;
```

The ability to drop or alter a sequence is reserved for the super user or sequence owner. See ALTER SEQUENCE and DROP SEQUENCE.

External Procedure Privileges

Only the superuser is allowed to create or drop an external procedure. By default, users cannot execute external procedures. However, the superuser can grant users this right. Additionally, users must have USAGE privileges on the schema that contains the procedure.

See the GRANT Procedure statement for more information.

Note: A procedure cannot be executed as root.

Metadata Privileges

Superuser has unrestricted access to all database metadata. Users have significantly reduced access to metadata based on their privileges, as follows:

Type of Metadata	User Access
Catalog objects: <ul style="list-style-type: none"> ▪ Tables ▪ Columns ▪ Constraints ▪ Sequences ▪ External Procedures ▪ Projections ▪ ROS containers ▪ WOS 	<p>Users must possess USAGE privilege on the schema and any type of access (SELECT) or modify privilege on the object to see catalog metadata about the object. See also Schema Privileges (page 136).</p> <p>For internal objects like Projections, ROS containers and WOS that don't have access privileges directly associated with them, the user must possess the requisite privileges on the associated Schema and Table object(s) instead. For example, to see whether a table has any data in the WOS, you need to have USAGE on the table schema and at least SELECT on the table itself. See also Table Privileges (page 137) and Projection Privileges (page 138).</p>
User sessions and functions, and system tables related to these sessions	<p>Users can access only information about their own, current sessions. The following functions provide restricted functionality to users:</p> <ul style="list-style-type: none"> ▪ CURRENT_DATABASE ▪ CURRENT_SCHEMA ▪ CURRENT_USER ▪ HAS_TABLE_PRIVILEGE ▪ SESSION_USER (same as CURRENT_USER) <p>The system table, SESSIONS, provides restricted functionality to users.</p>

Operating the Database

Managing Your License Key

Vertica license keys provide full product functionality for a specific period of time or forever if you purchase a perpetual license. There is a grace period during which Vertica continues to work after the license has expired. The length of the grace period depends on the license.

Obtaining a License Key File

To obtain a license key, request one from **Technical Support** (on page 1).

Tips: Do not open the license key file in an editor or e-mail client as special characters may be introduced, which though not visible, would invalidate the license. Similarly, take care when copying between Windows and Linux. Check that the copied file size is identical to that of the one you received from Vertica. For ease of installation, Vertica recommends copying the license file to `/tmp/vlicense.key` on the Administration host.

Installing a New or Upgrade Installation License Key

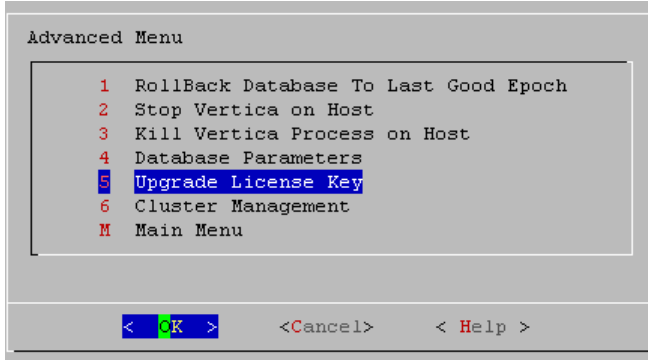
- 1 Obtain a license key as described above.
- 2 Install Vertica as described in the Installation Guide.
The first time you log in as the Database Administrator and run the Vertica Administration Tools, the user interface displays the EULA (license agreement).
- 3 Type "accept" to proceed. (Do not type the double quotes.)
- 4 A form asking for the path to the license key file that you downloaded from the Vertica Systems, Inc. Web site. The default path is `/tmp/vlicense.key`. If this is correct, click **OK**. Otherwise, enter the absolute path of the file in the bottom field of the form and click **OK**.

Installing a Renewal License Key

When a license is nearing expiration Vertica logs warning messages indicating the expiration date. When a license expires (the grace period has expired) Vertica logs an invalid license error and stops running queries. In that case:

- 1 Obtain a license key as described above.
- 2 Start a database.

- 3 In the Administration Tools, select Advanced > Upgrade License Key and click **OK**.



Examining Your License Key

Use the SQL `DISPLAY_LICENSE` function described in the SQL Reference Manual to display the license information. For example:

```
=> SELECT DISPLAY_LICENSE();
           display_license
```

```
-----
Vertica Systems, Inc.
1/1/2008
12/31/2008
30
50TB
```

```
(1 row)
```

Starting the Database

Starting a K-safe database is supported when up to K nodes are down or unavailable. See **Failure Recovery** (page 235) for a discussion on various scenarios encountered during database shutdown, startup and recovery.

To start a Vertica database:

- 1 Use **View Database Cluster State** (page 336) to make sure that all nodes are down and that no other database is running. If all nodes are not down, see **Shutdown Problems** (page 237).
- 2 Open the Administration Tools. See **Using the Administration Tools** (page 329) for information about accessing the Administration Tools.
- 3 On the **Main Menu**, select **Start Database**, and then select **OK**.
- 4 Select the database to start, and then click **OK**.

Warning: Vertica Systems, Inc. strongly recommends that you start only one database at a time. If you start more than one database at any time, the results are unpredictable. Users could encounter resource conflicts or perform operations in the wrong database.

- 5 Enter the database password, and then click **OK**.
- 6 When prompted that the database started successfully, click **OK**.
- 7 Check the log files to make sure that no startup problems occurred. See **Monitoring Vertica Using Ganglia** (page 218).

If the database does not start successfully, see **Startup Problems** (page 241).

Stopping the Database

Stopping a K-safe database is supported when up to K nodes are down or unavailable. See **Failure Recovery** (page 235) for a discussion on various scenarios encountered during database shutdown, startup and recovery.

To stop a running Vertica database:

- 1 Use **View Database Cluster State** (page 336) to make sure that all nodes are up. If all nodes are not up, see **Restarting a Node** (page 341).
- 2 Inform all users having open connections that the database is going to shut down and instruct them to close their sessions.

Tip: A simple way to prevent new client sessions from being opened while you are shutting down the database is to set the **MaxClientSessions** (page 25) configuration parameter to 0. Be sure to restore the parameter to its original setting once you've restarted the database.

```
=> SELECT SET_CONFIG_PARAMETER ('MaxClientSessions', 0);
```

- 3 Close any remaining user sessions. (Use the **CLOSE_SESSION** and **CLOSE_ALL_SESSIONS** functions.)
- 4 Open the Administration Tools. See **Using the Administration Tools** (page 329) for information about accessing the Administration Tools.
- 5 On the **Main Menu**, select **Stop Database**, and then click **OK**.
- 6 Select the database you want to stop, and click **OK**.

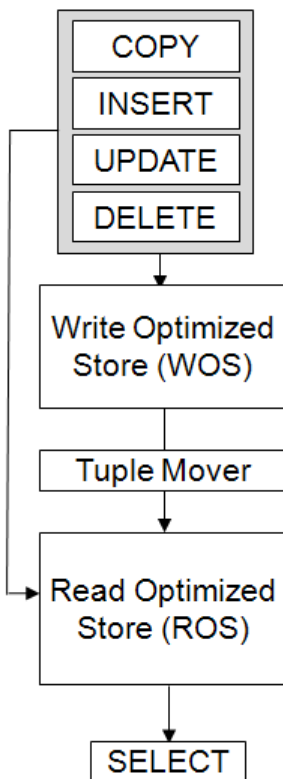
7 Enter the password if asked, and click **OK**.

8 When prompted that the database has been successfully stopped, click **OK**.

If the database does not stop successfully, see ***Shutdown Problems*** (page 237).

Loading and Modifying Data

Vertica's hybrid storage model provides a great deal of flexibility for loading and modifying data.



Bulk Loading

This section describes different methods for bulk loading data into a Vertica database.

Performing the Initial Database Load

Use COPY...DIRECT from vsql to load the database for the first time.

Tip: If you have a *Star schema* (page 37), load the dimension tables before you load the fact tables.

Extracting Data from an Existing Database

If possible, export the data in text form to a local file or attached disk.

ETL products typically use ODBC or JDBC to extract data, which gives them program-level access to modify column values, as needed, for the load files.

Database systems provide a variety of export methods.

Tip: Oracle does not provide a tool that can export to text. To export data, run a SELECT query in Oracle's SQL*Plus command line query tool using a specified column delimiter, suppressed headers, and so forth. Redirect the output to a local file.

Smaller tables generally fit into a single load file. Split any large tables into 250-500GB load files. For example, a 10 TB fact table would require 20-40 load files.

Tip: When working with large data size, Vertica recommends that you test the load process using smaller load files as described in **Configuration Procedure** (page 11) to avoid compatibility or file formatting issues.

Choose a column-value delimiter character that does not appear in any CHAR(N) or VARCHAR(N) data values. The vertical bar (|) might be a good choice and is the default. You can use a query on the source database to test for the existence of a certain character in a column. For example:

```
SELECT COUNT(*) FROM T WHERE X LIKE '%|%'
```

If only a few rows contain |, you can eliminate them from the load file using a WHERE clause and load them separately using a different delimiter.

Tip: Oracle has a REGEX_REPLACE function that can substitute one substring with another, but this slows down the unload operation significantly. A better approach is to use a WHERE clause to avoid problem rows in the main load file, and the negated WHERE clause with REGEX_REPLACE for just the problem rows.

Moving Data from an Existing Database to Vertica Nodes

To move data from an existing database to Vertica, consider using:

- USB 2.0 (or possibly SATA) disks.
- A fast local network connection.

Deliver chunks of data to the different Vertica nodes by connecting the transport disk or by writing files from network copy.

Loading From a Local Hard Disk

USB 2.0 disks can deliver data at about 30 MB per second, or 108 GB per hour, which is fast enough. USB 2.0 disks are easy to use for transporting data from Linux to Linux. Set up an ext3 filesystem on the disk and write large files there. Linux 2.6 has USB plug-and-play support, so a USB 2.0 disk is instantly usable on various Linux systems.

For other variants of UNIX, if there is no common filesystem format available, you can use the disk without a filesystem for a single large file. For example:

```
$ cp bigfile /dev/sdc1
```

Even without a filesystem on the disk, plug-and-play support still works on Linux to provide a device node for the disk. To find out the assigned device, plug in the disk and enter:

```
$ dmesg | tail -40
```

SATA disks are usually internal, but can be external, or unplugged safely internally.

Loading Over the Network

A 1Gbps (gigabit per second) network can deliver about 50 MB/s, or 180GB/hr. Vertica can load about 30-50GB/hour/node for a 1-safe projection design. Thus use a dedicated 1Gbps LAN. Slower LANs are proportionally slower, and non-local networks are probably untenable because the delays over distance slow down the TCP protocol to a small fraction of its apparent bandwidth, even without competing traffic.

Note: The actual load rates you obtain might be higher or lower depending on the properties of the data, number of columns, number of projections, and hardware and network speeds. Load speeds can be further improved by using multiple parallel streams.

Loading From Windows

For loading files directly from Windows to Linux, use NTFS. Although Red Hat Linux as originally installed can read Windows FAT32 file systems, this is not recommended.

Using the COPY and LCOPY Statements

The COPY and LCOPY statements are typically used to write multiple rows into physical storage. By **default**, COPY and LCOPY:

- Load data into WOS and if the WOS is full, writes the data directly to ROS. Use the default COPY behavior for smaller bulk loads and **trickle loads** (page 148) (more than one per day). For large data files (>100MB), Vertica recommends loading data directly into the ROS using the DIRECT option.
- Commit the current transaction and commit themselves.

Note: LCOPY is not a SQL command but a programmatic interface to COPY available only via ODBC, where the input data file is placed on the client machine and not the database server nodes.

Loading Data Directly to ROS

To load data directly into the ROS, use the DIRECT keyword with the COPY or LCOPY statement.

```
COPY a FROM stdin DIRECT;
```

Note: A large initial bulk load could temporarily affect query performance while Vertica organizes the data.

Loading Data without Committing it

Use the NO COMMIT keyword with the COPY statement to prevent the current transaction from committing. This is useful for executing multiple COPY commands in a single transaction. For example, all the rows in the following sequence commit in the same transaction.

```
COPY... NO COMMIT;  
COPY... NO COMMIT;  
COPY... NO COMMIT;  
COMMIT;
```


Tip: Use the NO COMMIT keyword to incorporate detection of constraint violations into the load process. Vertica checks for constraint violations when queries are run, not when data is loaded. To avoid constraint violations, load data without committing it and then test it using ANALYZE_CONSTRAINTS. If you find any constraint violations, you can roll back the load because you have not committed it. See *Analyzing Constraints (Detecting Constraint Violations)* (page 55) for detailed instructions.

Using COPY Interactively

The recommended way to use COPY is in script files, as described in *Using Load Scripts* (page 148). You can, however, use this command interactively by piping a text file to vsql and executing COPY with the standard input stream as the input file. For example:

```
$ cat fact_table.tbl | vsql -c "COPY FACT_TABLE FROM STDIN DELIMITER '|' DIRECT"
$ cat fact_table.tbl | vsql -c "COPY FACT_TABLE FROM STDIN DELIMITER '|' DIRECT"
```

Tracking Load Status

To view load metrics for each load stream on each node, use the system table LOAD_STREAMS. You can also use the GET_NUM_ACCEPTED_ROWS and GET_NUM_REJECTED_ROWS functions to obtain the number of accepted and rejected rows for the last completed load within the current session.

Note: When using multiple long running COPY operations, use the optional STREAM NAME parameter of COPY to make it easy to identify each operation in the LOAD_STREAMS table.

If you're using an ODBC or JDBC client, you can obtain the following data for the last completed load:

- The number of rows that were accepted or rejected.
- The row numbers for every rejected row.

See the Programmer's Guide for client-specific documentation.

Trickle Loading

Once you have a working database and have bulk loaded your initial data, you can use trickle loading to load additional data on an ongoing basis. By default, Vertica uses the transaction isolation level of `READ COMMITTED`, which allows users to see the most recently committed data without holding any locks. This allows new data to be loaded while concurrent queries are running.

See *Change Transaction Isolation Levels* (page 25).

Using INSERT, UPDATE, and DELETE

The SQL data manipulation language (DML) commands `INSERT`, `UPDATE`, and `DELETE` perform the same functions that they do in any ACID compliant database. The `INSERT` statement is typically used to load individual rows into physical memory or load a table using `INSERT AS SELECT`. `UPDATE` and `DELETE` are used to modify the data.

You can intermix the `INSERT`, `UPDATE`, and `DELETE` commands. Vertica follows the SQL-92 transaction model. In other words, you do not have to explicitly start a transaction but you must use a `COMMIT` or `ROLLBACK` command (or `COPY`) to end a transaction. Canceling a DML statement causes the effect of the statement to be rolled back.

Vertica differs from traditional databases in two ways:

- `DELETE` does not actually delete data from disk storage; it marks rows as deleted so that they can be found by historical queries.
- `UPDATE` writes two rows: one with new data and one marked for deletion.

Like `COPY`, by default, `INSERT`, `UPDATE` and `DELETE` commands write the data to the WOS and on overflow write to the ROS. For large `INSERTS` or `UPDATES`, you can use the `DIRECT` keyword to force the Vertica write rows directly to the ROS. Loading large number of rows as single row inserts are not recommended for performance reasons. Use `COPY` instead.

WOS Overflow

Loading data into the WOS is a tradeoff of speed versus potential memory overflow. Writing to the WOS is much faster than writing to the ROS, but writing too much data too quickly can overrun the amount of memory available. When that happens, Vertica automatically spills to the ROS until the Tuple Mover can catch up and move data from the WOS to the ROS.

Using the COPY Command

This section describes how to use the `COPY` command to design your load process. For detailed syntax of the various options see the SQL Reference Manual.

Using Load Scripts

This section describes how to write and run a load script using the `COPY` command using the simplest text-delimited file format.

Writing a Load Script

The `COPY` command requires an absolute path for a data file. It does not accept relative paths. However, you can specify the locations of your data files relative to your Linux working directory using `vsq` variables.

- 1 Create a vsql variable containing your Linux current directory.

```
\set t_pwd `pwd`
```
- 2 Create another vsql variable that uses a path relative to the Linux current directory variable for a specific data file.

```
\set input_file '\':t_pwd'/Date_Dimension.tbl\''
```
- 3 Use the second variable in the COPY statement.

```
COPY Date_Dimension FROM :input_file DELIMITER '|';
```
- 4 Repeat steps 2 and 3 for all data files. Load the dimension tables before the fact table.

Running a Load Script

You can run a load script on any host, as long as the data files are on that host.

- 1 Change your Linux working directory to the location of the data files.

```
$ cd /opt/vertica/doc/retail_example_database
```
- 2 Run the Administration Tools.

```
$ /opt/vertica/bin/admintools
```
- 3 Connect to the database.
- 4 Run the load script.

For information about other load formats see *Advanced formats for Loading Data* (page 157).

Using Parallel Load Streams

You can use multiple parallel load streams to load a Vertica database. There are two options:

- Issue multiple separate COPY commands that load different files from different nodes.
 This option lets you use vsql, ODBC, or JDBC. You can also use server-side files or client-side files (LCOPY).
- Issue a single multi-node COPY command that loads different files from different nodes by specifying the <nodename> for each file.
 This option is possible only using the vsql command, and not all options of COPY are supported; however, significantly higher performance and efficient resource usage can result from this option.

See COPY in the SQL Reference Manual for details.

The optimal number of load streams depends on several factors, including the number of nodes, the physical and logical schemas, host processors, memory, disk space, and so forth. Too many load streams can cause systems to run out of memory. See *Best Practices for Workload Management* (page 301) for advice on configuring load streams.

Loading Data into Character Data Types

Character Set

Vertica expects data files to be in the Unicode UTF-8 format. ASCII data is compatible with UTF-8 and can be loaded, however, character sets like ISO 8859-1 (Latin1), which are not compatible with UTF-8 are not supported.

- 1 Use the file command to check the type of a data file. For example:

```
$ file Date_Dimension.tbl
Date_Dimension.tbl: ASCII text
```

The file command could indicate ASCII TEXT even though the file contains multibyte characters.

2 Use the wc command to check for this problem. For example:

```
$ wc Date_Dimension.tbl
 1828   5484 221822 Date_Dimension.tbl
```

If the wc command returns an error such as Invalid or incomplete multibyte or wide character, the data file is using an incompatible character set.

Using Escaped Characters as Literals

Use the escape character to escape data characters that would otherwise be taken as special characters. In particular, the following characters must be preceded by the escape character if they appear as part of a column value:

- The COPY ... DELIMITER character. Default is the vertical bar character (|).
- The COPY ... NULL string. Default is an empty string ('').
- The backslash character itself
- Newline and other control characters

By default, the escape character is the backslash character (\). To change the escape character, use the ESCAPE AS parameter with the COPY statement. See COPY in the SQL Reference Manual.

Examples

In the examples that follow, the DELIMITER is comma for readability.

<pre>,1,2,3, ,1,2,3 1,2,3,</pre>	<p>Leading and trailing delimiters are ignored. Thus, the rows all have three columns.</p>
<pre>123, '\\n', \\n, 456</pre>	<p>Using a non-default null string, the row would be interpreted as:</p> <pre>123 newline \n 456</pre>
<pre>123,this\, that\, or the other,something else,456</pre>	
	<p>The row would be interpreted as:</p> <pre>123 this, that, or the other something else 456</pre>

Loading Data into Binary Data Types

Binary data types are similar to character data types except in how values are padded and translated on input, and also in the functions, operators, and casts supported.

On input, strings are translated from hexadecimal representation to a binary value using the `HEX_TO_BINARY` function. Strings are translated from bitstring representation to binary values using the `BITSTRING_TO_BINARY` function. Both functions take a `VARCHAR` argument and return a `VARBINARY` value.

The following formats are also allowed to load binary data:

- **Hexadecimal:** A prefix of `'0x'` is a good indicator that the value is hexadecimal, not decimal. (Not all hexadecimal values use A-F; for example, 5396). The `0x` prefix is ignored when the copy operation loads the inputs, so it is optional. Hexadecimal is similar in format to the `HEX_TO_BINARY` function.

If there are an odd number of characters in the hexadecimal value, the first character is treated as the low nibble of the first (furthest to the left) byte.

- **Octal:** Requires that each byte be represented by an octal code, which is exactly three digits. The first digit must be in the range `[0,3]` and the second and third must both be in the range `[0,7]`.

If the length of an octal value is not a multiple of three, or if one of the three digits is not in the proper range, then the value is invalid and the row in which the value appears is rejected.

- **Bitstring:** Each character must be a zero or one. Bitstring is similar in format to the `BITSTRING_TO_BINARY` function.

If the bitstring value is not a multiple of eight, then the first n characters are treated as the low bits of the first (furthest to the left) byte, where n is the remainder of the value's length divided by eight. The bitstring format is not used as often as hexadecimal or octal formats.

Notes

There is no copy format that loads binary data byte for byte because the column and record separators that appear in the data would have to be escaped. Binary data can be translated into the formats that Vertica supports.

The hexadecimal, octal, and bitstring formats can be used to load binary columns only. These column formats are specified using the `COPY` command's `FORMAT` argument:

```
COPY t1 (oct FORMAT 'octal', hex FORMAT 'hex', ...) FROM STDIN delimiter ',';
```

The same default format used to input binary data is used to load binary data. The `'\'` (backslash) character is the `COPY` operator's escape character, so octal inputs must be escaped. For example, the byte `'\141'` must appear as `'\\141'`.

You can also use the escape character to represent the (decimal) byte 92 by escaping it twice; for example, `'\\\\'`. Note that `vsq` inputs the escaped backslash as four backslashes. Equivalent inputs are hex value `'0x5c'` and octal value `'\134'` ($134 = 1 \times 8^2 + 3 \times 8^1 + 4 \times 8^0 = 92$).

A delimiter value can be loaded if you escape it with a backslash. For example, given delimiter `'|'`, `'\\001\\|\\002'` is loaded as `{1, 124, 2}`, which can also be represented in octal format as `'\\001\\174\\002'`.

If you supply an invalid octal value, the system returns an error; for example:

```
SELECT '\\000\\387'::binary(8);
ERROR: invalid input syntax for type binary
```

Rows that contain binary values with invalid octal representations are also rejected. For example, '\\008' is rejected because of the base-8 numbering system. Octal numbering processes as 000, 001, 002, 003, 004, 005, 006, 007, and then numbering restarts at 010. '\\ 008' does not exist.

If you insert a value with more bytes than fit into the target column, the system returns an error. For example, where column `c1` is `VARBINARY(1)`:

```
=> INSERT INTO t (c1) values ('ab');
      ERROR: 2-byte value too long for type Varbinary(1)
```

If you implicitly or explicitly cast a value with more bytes than fit the target data type, the value is silently truncated. For example:

```
=> SELECT 'abcd'::binary(2);
      binary
-----
      ab
(1 row)
```

Examples

The following example shows `VARBINARY HEX_TO_BINARY(VARCHAR)` and `VARCHAR TO_HEX(VARBINARY)` usage.

Table `t` and its projection are created with binary columns:

```
=> CREATE TABLE t (c BINARY(1));
=> CREATE PROJECTION t_p (c) AS SELECT c FROM t;
```

Insert minimum and maximum byte values, including an IP address represented as a character string:

```
=> INSERT INTO t values(HEX_TO_BINARY('0x00'));
=> INSERT INTO t values(HEX_TO_BINARY('0xFF'));
=> INSERT INTO t values (V6_ATON('2001:DB8::8:800:200C:417A'));
```

Binary values can be formatted in hexadecimal on output using the `TO_HEX` function:

```
=> SELECT TO_HEX(c) FROM t;
      to_hex
-----
      00
      ff
      20
(3 rows)
```

The next example illustrates the use of the three alternative load formats.

Create a table with columns that represent the three load formats and then insert the same value, byte sequence {0x61, 0x62, 0x63, 0x64, 0x65}.

```
=> CREATE TABLE t(oct VARBINARY(5), hex VARBINARY(5), bitstring VARBINARY(5));
```

Create a projection for table `t`:

```
=> CREATE PROJECTION t_p(oct, hex, bitstring) AS SELECT * FROM t;
```

Issue the COPY command and specify the format for the binary columns. Note that the copy is from STDIN, not a file.

```
=> COPY t (oct FORMAT 'octal',
>>         hex FORMAT 'hex',
>>         bitstring FORMAT 'bitstring')
>> FROM STDIN delimiter ',';
>> Enter the data to be copied, which you end with a backslash and a period on a
line by itself:
>> 141142143144145,0x6162636465,0110000101100010011000110110010001100101
>> \.
```

And now issue the SELECT statement to see the results:

```
=> SELECT * FROM t;
oct      | hex      | bitstring
-----+-----+-----
abcde   | abcde   | abcde
(1 row)
```

See Also

COPY in the SQL Reference Manual

Binary Data Types and Formatting Functions in the SQL Reference Manual

ASCII (<http://en.wikipedia.org/wiki/Ascii>) topic in Wikipedia for a quick reference table on these load format values.

Loading Data into Pre-join Projections

A pre-join projection stores rows of a fact table joined with rows of dimension tables. To insert a row into the fact table of a pre-join projection, the associated values of the dimension table's columns must be looked up. Thus, an insert into a pre-join projection shares some of the qualities of a query. The following sections describe the behaviors associated with loading into pre-join projections.

Foreign and Primary Key Constraints

To ensure referential integrity, foreign and primary key constraints are enforced on inserts into fact tables of pre-join projections. If a fact row attempts to reference a row that does not exist in the dimension table, the load is automatically rolled back. The load is also rolled back if a fact row references more than one dimension row.

The following tables and SQL examples highlight these concepts.

- **Fact Table:** Employees
- **Dimension Table:** HealthPlans
- **Pre-join Projection:** Joins Employees to HealthPlans using the PlanID column

```
CREATE PROJECTION EMP_HEALTH (EmployeeID, FirstName, LastName, Type)
AS (SELECT EmployeeID, FirstName, LastName, Type
    FROM Employees, HealthPlans
    WHERE Employees.HealthPlanID = HealthPlans.PlanID)
```

Employees (Fact Table)

EmployeeID(PK)	FirstName	LastName	PlanID(FK)
1000	David	Taylor	01
1001	Sunil	Ray	02
1002	Janet	Hildreth	02
1003	Pequan	Lee	01

HealthPlans (Dimension Table)

PlanID(PK)	Description	Type
01	PlanOne	HMO
02	PlanTwo	PPO

The following sequence of commands generate a missing foreign key error that results in a rollback because the reference is to a non-existent dimension row.

```
INSERT INTO Employees (EmployeeID, First, Last, PlanID) VALUES (1004, 'Ben', 'Smith', 04);
```

The following sequence of commands generate a foreign key error that results in a rollback because a duplicate row in the HealthPlans dimension table is referenced by an insert in the Employees fact table. The error occurs when the Employees fact table references the HealthPlans dimension table.

```
INSERT INTO HealthPlan VALUES(02, 'MyPlan', 'PPO');  
INSERT INTO Employee VALUES(1005, 'Juan', 'Hernandez', 02);
```

Concurrent Loads into Pre-join projections

Vertica supports concurrent inserts where two transactions can simultaneously insert rows into the same table. A transaction inserting records into a pre-join projection can run concurrently with another transaction inserting records into either the fact table or a dimension table of the pre-join projection. A load into a pre-join projection cannot run concurrently with updates or deletes on either the fact or the dimension tables.

When concurrently loading fact and dimension tables, the state of the dimension tables is fixed at the start of the insert or load into the fact table. Rows that are added to a dimension table after the start of an insert or load into a fact table are not available for joining because they are not visible to the fact table. The client is responsible for ensuring that all values in dimension tables are present before issuing the insert or load statement.

The following examples illustrate these behaviors.

- **Fact Table:** Sales
- **Dimension Table:** Employees
- **Pre-join Projection:** sales join employees on sales.seller=employees.empno

Success

Session A	Session B	Description
<pre>INSERT INTO EMPLOYEES (EMPNO, NAME) VALUES (1, 'Bob');</pre>		

COPY INTO SALES (AMT, SELLER) 5000 1 3500 1 . .		Records loaded by this COPY command all refer to Bob's sales (SELLER = 1)
	INSERT INTO EMPLOYEES (EMPNO, NAME)VALUES (2, 'Frank');	
7234 1	COPY INTO SALES (AMT,SELLER) 50 2 75 2 . .	Records loaded by this COPY command all refer to Frank's sales (SELLER = 2).
COMMIT;	COMMIT;	Both transactions are successful.

Failure

Session A	Session B	Description
INSERT INTO EMPLOYEES (EMPNO, NAME) 1 Bob		
2 Terry	COPY INTO SALES (AMT,SELLER) 5000 1	The transaction in Session B fails because the value inserted into the dimension table in Session A was not visible before the COPY into the pre-join in Session B was initiated.

Specifying Default Values for Columns

If a column is used in an insert or bulk load operation and no value is specified for it, the value of the column defaults to null. However, you can specify your own defaults for table columns by using variable-free expressions set through column constraints when you create or alter a table. See the DEFAULT key word in COPY in the SQL Reference Manual.

Transforming Data During Loads

To promote a consistent database and reduce the need for scripts to transform data at the source, Vertica supports the transformation of data when it is loaded into the target database. This is useful for computing values to be inserted into a column in the target database from other columns in the source.

To transform data, use the following syntax to specify the target column, for which you want to compute values, as an expression:

```
COPY [schema-name.]table[(
```

```
[Column as Expression] / column[FORMAT 'format']  
[ ,...]]]  
FROM ...
```

The following example computes and loads values for the year, month, and day columns in the target database based on the timestamp column in the source input. It also loads the parsed column, timestamp, from the source input to the target database.

```
CREATE TABLE t (  
    year VARCHAR(10),  
    month VARCHAR(10),  
    day VARCHAR(10),  
    k timestamp  
);  
  
CREATE PROJECTION tp (  
    year,  
    month,  
    day,  
    k)  
AS SELECT * from t;  
COPY t(year AS TO_CHAR(k, 'YYYY'),  
    month AS TO_CHAR(k, 'Month'),  
    day AS TO_CHAR(k, 'DD'),  
    k FORMAT 'YYYY-MM-DD') FROM STDIN NO COMMIT;
```

```
2009-06-17  
1979-06-30  
2007-11-26  
\.
```

```
SELECT * FROM t;  
year | month | day | k  
-----+-----+-----+-----  
2009 | June | 17 | 2009-06-17 00:00:00  
1979 | June | 30 | 1979-06-30 00:00:00  
2007 | November | 26 | 2007-11-26 00:00:00  
(3 rows)
```

See also **Using Sequences** (page 60) for how to generate an auto-incrementing value for columns.

See the COPY statement in the SQL Reference Manual for detailed information regarding requirements, restrictions, and usage.

Ignoring Columns and Fields in the Load File

When performing a load, you can instruct Vertica not to load a column and the fields it contains if the column does not exist in the destination table. This is useful for:

- Omitting columns that you do not want to transfer into a table.
- Transforming data from a source column and then loading the transformed data to a destination table without loading the original, untransformed source column (parsed column). (See **Transforming Data During Loads** (page 155) in this guide and the COPY command in the SQL Reference Manual.)

To skip a column, designate it as a filler column by using the FILLER key word in the COPY statement, as follows:

```
COPY [schema-name.]table[(
    [Expression as column] / column[FORMAT 'format'] [FILLER datatype]
    [ ,...]])]
FROM ...
```

The following example derives and loads the value for the timestamp column in the target database from the year, month, and day columns in the source input. The year, month, and day columns are not loaded because the FILLER key word skips them.

```
CREATE TABLE t (k TIMESTAMP);
CREATE PROJECTION tp (k) AS SELECT * FROM t;
COPY t(year FILLER VARCHAR(10),
    month FILLER VARCHAR(10),
    day FILLER VARCHAR(10),
    k AS TO_DATE(YEAR || MONTH || DAY, 'YYYYMMDD'))
FROM STDIN NO COMMIT;
2009|06|17
1979|06|30
2007|11|26
\.
SELECT * FROM t;
      k
-----
2009-06-17 00:00:00
1979-06-30 00:00:00
2007-11-26 00:00:00
(3 rows)
```

See the COPY statement in the SQL Reference Manual for detailed information regarding requirements, restrictions, and usage.

About Load Errors

Depending upon the type of error encountered, Vertica either rejects the row or rolls back the entire load:

- **Reject rows** — When Vertica encounters an error parsing records in the input file, it rejects the offending row and continues to load the database. For example, Vertica rejects a row if it contains any of the following: incompatible data types, missing fields, or missing delimiters.
- **Load rollback** — The following types of errors result in a load rollback:
 - Server-side errors (such as lack of memory)
 - Violations of primary key or foreign key constraints
 - Loading NULL data into a not NULL column

When an error results in a load rollback, the load is aborted and the data is rolled back. The result is that no data is loaded.

If you specify ABORT ON ERROR with the COPY command, the load is automatically canceled and rolled back immediately if any row is rejected or an error occurs.

Advanced Formats for Loading Data

Vertica provides three formats to load data:

- In a text format with delimiters, which is the default COPY command. Binary data types are translated on input. See **Using Load Scripts** (page 148) and **Loading Data into Binary Data Types** (page 150) for examples.
- In a native binary format using the NATIVE keyword to COPY command.
- In a native varchar format using the NATIVE VARCHAR keyword to COPY command.

See also COPY in the SQL Reference Manual, NATIVE and NATIVE VARCHAR keywords.

Native (Binary) Format

Loading data through a binary-format file is often faster than normal text mode, because it does not require the use and processing of delimiters, so it saves the database the extra work of converting integers, dates, and timestamps from text to their native storage format. Native (binary) format data files can be bigger than their text equivalents; however, you can reduce the space usage by compressing binary data using gzip or bzip. Native (binary) format loading can be used when developing plug-ins to ETL applications, as well as by batch inserts issued from ODBC and JDBC.

Binary-format files must meet exacting specifications. See **Creating Native-format Files to Load Data** (page 158).

Native Varchar Format

The Native Varchar format uses a similar file format to native binary, but all fields are represented as strings in CHAR or VARCHAR. Conversion to the actual table data type is done on the database server; thus, NATIVE VARCHAR does not provide the same efficiency as NATIVE BINARY. However, NATIVE VARCHAR provides the convenience of not having to use delimiters or escape special characters, such as quotes, which can make working with client applications easier.

See **Creating Native-Format Files to Load Data** (page 158) for the file specifications.

Batch inserts done via the Vertica ODBC and JDBC drivers automatically use either the NATIVE BINARY or NATIVE VARCHAR formats. NATIVE BINARY is used if the application data types match the actual table data types exactly (including maximum lengths of CHAR/VARCHAR and precision/scale of numeric data types), which provides best possible load performance. If there is any data type mismatch, NATIVE VARCHAR is used.

Note: Concatenated BZIP and GZIP files are not supported for NATIVE (Binary) and NATIVE VARCHAR formats.

Creating Native-Format Files to Load Data

COPY NATIVE requires a load file to use the following format, consisting of a file header followed by a sequence of records to be loaded.

Note: All integers are in little-endian format.

File Header

The file header described in Table 1 consists of 15 bytes of fixed fields, followed by a variable-length header extension area.

Table 1: Format specification for file header

Offset	Length (bytes)	Field	Comments
0	11	<i>Signature</i>	11-byte sequence <code>NATIVE\n\377\r\n\0</code> . The signature is designed to allow easy identification of files that have been corrupted by a non-8-bit-clean transfer. The signature is changed by end-of-line-translation filters, dropped zero bytes, dropped high bits, or parity changes.
11	4	<i>Header area length</i>	32-bit integer, which is the length in bytes of remainder of header, not including <i>Signature</i> and self.
15	2	<i>Version</i>	16-bit integer, which is the version number of the file format. The only valid value in Vertica 4.0 is the integer 1. Future changes to the format would be assigned different numbers to maintain backward compatibility.
17	1	<i>Filled</i>	The value is 0.
N/A	2	<i>NumFields</i>	16-bit integer that contains the number of fields in each record in the file.
N/A	4* <i>NumFields</i>	<i>Field Lengths</i>	Array of 32-bit integers, with the size of the fields to be loaded. For variable-length fields, use -1 as a 32-bit integer (0xFF 0xFF 0xFF 0xFF).

Records

Following the file header is a sequence of records to be stored in the file. Each record starts with a header described in Table 2.

Table 2: Format specification for the header of each record

Length (bytes)	Name	Comments
4	<i>Row Size</i>	A 32-bit integer, which is the length of the record. It includes the size of data only.
<code>CEILING(NumFields / (sizeof(uint8)*8));</code>	<i>Null bits</i>	A variable-length array of unsigned 8-bit integers, interpreted as a sequence of bits, where the i^{th} bit, if set, indicates that the i^{th} field in the row has NULLs. The first field is represented in the most significant bit of the 0 th entry in the array. If a field is NULL, then there should be no data for the field in the row; that is, data takes up 0 bytes in the row, regardless of data type.

Following the record header, the actual record is stored. The format of each field typically depends on the data type of the corresponding column in the database (unless it is a filler used to derive an expression to be stored in the actual column). Table 3 describes the format used for each data type.

Note: Some of the data types support different lengths. For example, integers are supported in 1, 2, 4, and 8-byte lengths. However, these are not variable-length data types; you cannot change the length of an integer field within different records in a file. If a field is represented as a 2-byte integer, then the length must be set to 2 in the *Field Lengths* section in the file header, and every record in the file should have a 2-byte integer for this field. In contrast, VARCHAR and VARBINARY data types are variable-length data types, where each record can have a different sized data value for the field.

Table 3: Specification of data fields of various data types

Data Type	Length (bytes)	Comments
INTEGER	1, 2, 4, 8	8, 16, 32, and 64-bit integers are supported.
BOOLEAN	1	Boolean needs only 1 byte.
FLOAT	8	Encoded in IEEE-754 format.
CHAR	Fixed-length string	<p>The length should be specified in the file header in the <i>Field Lengths</i> entry for the field.</p> <ul style="list-style-type: none"> ▪ Unlike the VARCHAR data type, the data should not be preceded by a length field. ▪ The field in the record should contain <i>length</i> number of bytes. ▪ Strings shorter than the specified length must be right-padded with spaces. ▪ Strings should not be null-terminated. ▪ Character encoding should be UTF-8. <p>Note: For strings containing multi-byte characters, the length should be calculated using number of bytes and not number of characters.</p>
VARCHAR	4-byte integer (length) + data	<p>File header should contain -1 in the <i>Field Lengths</i> entry for this field.</p> <ul style="list-style-type: none"> ▪ In each record, a VARCHAR string must be preceded by a 32-bit integer denoting the length of the string. ▪ The string should not be null-terminated. ▪ Character encoding should be UTF-8. <p>Note: For strings containing multi-byte characters, the length should be calculated using number of bytes and not number of characters.</p>
DATE	8	64-bit integer containing the Julian day since Jan 01 2000 (J2451545)
TIME	8	64-bit integer with number of microseconds since midnight in the UTC time zone.

TIMETZ	8	64-bit integer where <ul style="list-style-type: none"> Upper 40 bits contain the number of microseconds since midnight. 24 bits contain time zone as the UTC offset in microseconds calculated as follows: Time zone is logically from -24hrs to +24hrs from UTC. Instead it is represented here as a number between 0hrs to 48hrs. Therefore, 24hrs should be added to the actual time zone to calculate it.
TIMESTAMP	8	64-bit integer with number of microseconds since Julian day: Jan 01 2000 00:00:00.
TIMESTAMPTZ	8	A 64-bit integer that contains the number of microseconds since Julian day: Jan 01 2000 00:00:00 in the UTC timezone.
INTERVAL	8	64-bit integer with the number of microseconds in the interval.
BINARY	Constant length per file	Similar to CHAR. The length should be specified in the file header in the <i>Field Lengths</i> entry for the field. The field in the record must contain <i>length</i> number of bytes. If the value is smaller than the specified length, it must be right padded with INT(0).
VARBINARY	4-byte integer + data	Stored just like VARCHAR but data is interpreted as bytes rather than UTF-8 characters.
NUMERIC	(precision, scale) precision / 19 + 1	A constant-length data type. Length is determined by the precision, assuming that a 64-bit unsigned integer can store roughly 19 decimal digits. The data consists of a sequence of 64-bit integers, each stored in little-endian format, with the most significant integer first. Data in the integers is stored in base 2 ⁶⁴ . 2's complement is used for negative numbers. If there is a scale, then the numeric is stored as numeric * 10 ^{scale} ; that is, all real numbers are stored as integers, ignoring the decimal point. It is required that the scale matches that of the target column in the database table. Another option is to use FILLER columns to coerce the numeric to the scale of the target column.

Notes

You cannot mix binary and ASCII source files in the same COPY statement.

Example

The example below shows a table with all possible data types and a sample row of data. Table 4 shows what the binary load file should look like (each byte of data is show in octal).

```
=> CREATE TABLE allTypes (
    intcol INTEGER,
    floatcol FLOAT,
    charcol CHAR(10),
```

```

    varcharcol VARCHAR,
    boolcol BOOLEAN,
    datecol DATE,
    timestampcol TIMESTAMP,
    timestampTzcol TIMESTAMPTZ,
    timecol TIME,
    timeTzcol TIMETZ,
    varbincol VARBINARY,
    bincol BINARY,
    numcol NUMERIC,
    intervalcol INTERVAL
);
=> COPY allTypes FROM stdin delimiter '|' DIRECT;
 1|-1.11|one|ONE|1|1999-01-08|1999-02-23 03:11:52.35|1999-01-08
07:04:37|07:09:23|15:12:34|abcd|abcd|1234532|03:03:03
 2|-1.11|two|TWO|1|1999-01-08|1999-02-23 03:11:52.35|1999-01-08
07:04:37|07:09:23|15:12:34|abcd|abcd|1234532|03:03:03
 \.
=> \pset expanded
=> SELECT * from allTypes;
-[ RECORD 1 ]-----
intcol      | 1
floatcol    | -1.11
charcol     | one
varcharcol  | ONE
boolcol     | t
datecol     | 1999-01-08
timestampcol | 1999-02-23 03:11:52.35
timestampTzcol | 1999-01-08 07:04:37-05
timecol     | 07:09:23
timeTzcol   | 15:12:34-04
varbincol   | abcd
bincol      | a
numcol      | 1234532.0000000000000000
intervalcol | 03:03:03
-[ RECORD 2 ]-----
intcol      | 2
floatcol    | -1.11
charcol     | two
varcharcol  | TWO
boolcol     | t
datecol     | 1999-01-08
timestampcol | 1999-02-23 03:11:52.35
timestampTzcol | 1999-01-08 07:04:37-05
timecol     | 07:09:23
timeTzcol   | 15:12:34-04
varbincol   | abcd
bincol      | a
numcol      | 1234532.0000000000000000
intervalcol | 03:03:03

```

Description

Contents

File Header	NATIVE \n 377 \r \n\0
-------------	-----------------------

Header Length	= \0 \0 \0
Header Version	001 \0
Filler	\0
NumFields	016 \0
Field Length (intcol)	\b \0 \0 \0
Field Length (Float)	\b \0 \0 \0
Field Length (Char(10))	\n \0 \0 \0
Field Length (Varchar(10))	377 377 377 377
Field Length (Boolcol)	001 \0 \0 \0
Field Length (Datecol)	\b \0 \0 \0
Field Length (Timestampcol)	\b \0 \0 \0
Field Length (TimestampTZcol)	\b \0 \0 \0
Field Length (Timecol)	\b \0 \0 \0
Field Length (TimeTZcol)	\b \0 \0 \0
Field Length (varbinary)	377 377 377 377
Field Length (binary)	003 \0 \0 \0
Field Length (Numeric)	003 \0 \0 \0
Field Length (Interval)	\b \0 \0 \0
Row Size	~ \0 \0 \0
Null Bits	\0 \0
Int8col (1)	001 \0 \0 \0 \0 \0 \0 \0
Float (-1.11)	303 365 (\ 217 302 361 277
Char(10) (ONE)	o n e 7 spaces here
Varchar(10) (ONE)	003 \0 \0 \0 O N E
Boolcol(10) (ONE)	001
Datecol(1999-01-08)	377 377 377 377 377 377 376 232
Timestampcol (1999-02-23 03:11:52.35)	377 377 347 ~ 0 256 . #
TimestampTZcol (1999-01-08 07:04:37)	377 377 343 350 d > 037 @
Timecol (07:09:23)	\0 \0 \0 005 377 230 . 300
TimeTZCol (15:12:34)	020 360 y 360 200 001 227 320
Varbinary (abcd)	\0 \0 \0 002 253 315
Binary(3) (abcd)	253 315 \0
Numeric (1234532)	\0 022 326 d

Interval (03:03:03)	\0 \0 \0 002 216 243 G 300
---------------------	----------------------------

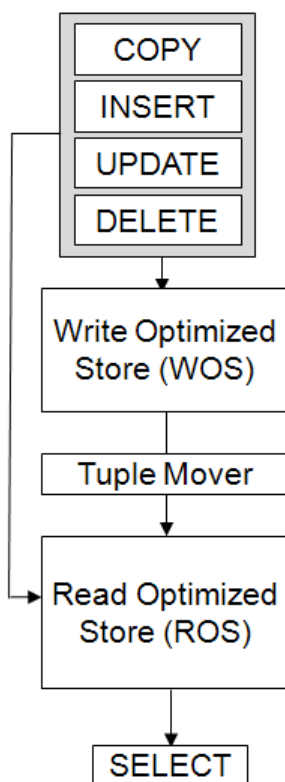
See Also

COPY in the SQL Reference Manual

Tuple Mover

Runs in the background, optimizing the data layout. Tasks include moving data from memory (WOS) to disk (ROS), combining small ROS containers into larger ones, and purging deleted data. Under ordinary circumstances, the operations performed by the Tuple Mover (TM) are automatic and transparent, and are therefore of little or no concern to the database administrator. However, when loading data, certain conditions require that you stop the Tuple Mover, perform some operations manually, and restart it.

This section discusses ***Tuple Mover operations*** (page 165) and ***how to perform TM tasks manually*** (page 171).



Understanding the Tuple Mover

The Tuple Mover performs two operations:

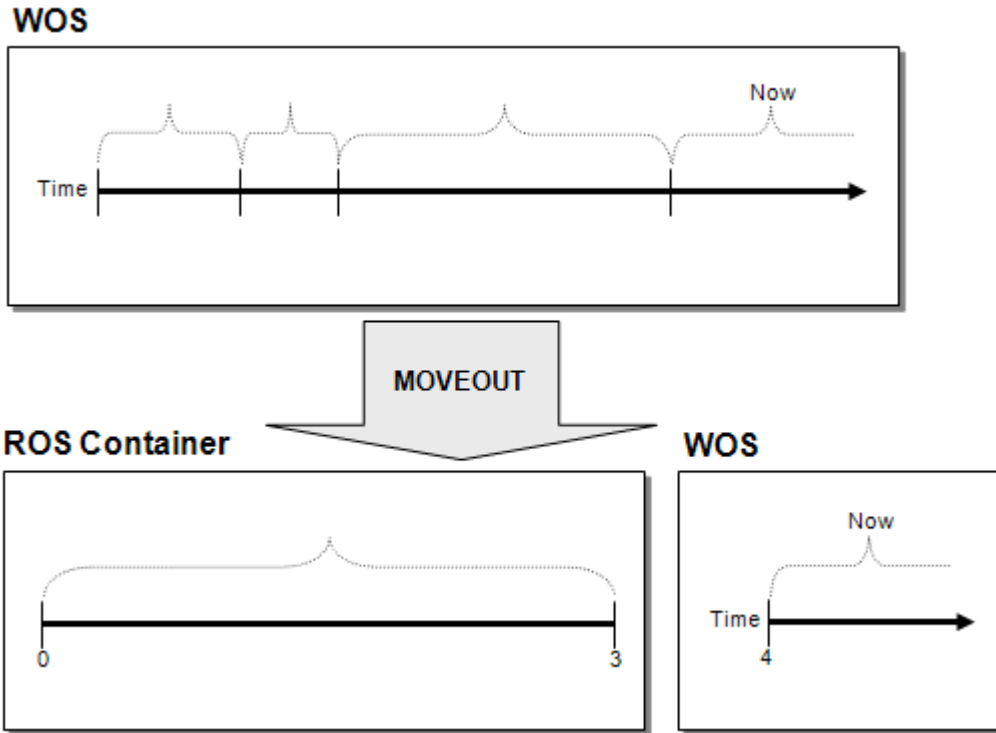
- ***Moveout*** (page 166)
- ***Mergeout*** (page 167)

Each of these operations occurs at different intervals across all nodes.

Moveout

Moveout operations move all epochs but the current epoch from memory (WOS) into a new ROS container. A moveout "flushes" all historical data from the WOS to the ROS.

The following illustration shows the effect of a projection moveout on a single node:



ROS Containers

Subsets of the Read Optimized Store (ROS) and sometimes referred to as ROSs, ROS containers are created by changes to the data stored within a projection as a result of bulk loads and DML. The Tuple Mover periodically merges ROS containers to maximize performance.

There is not necessarily a one-to-one correspondence between ROS containers and projection segments. For example, consider the following projection:

```
CREATE PROJECTION P1 (A, B, C, D) AS
SELECT A, B, C, D
FROM T1
SEGMENTED BY D
NODE S1 VALUES LESS THAN 5
NODE S2 VALUES LESS THAN MAXVALUE;
```

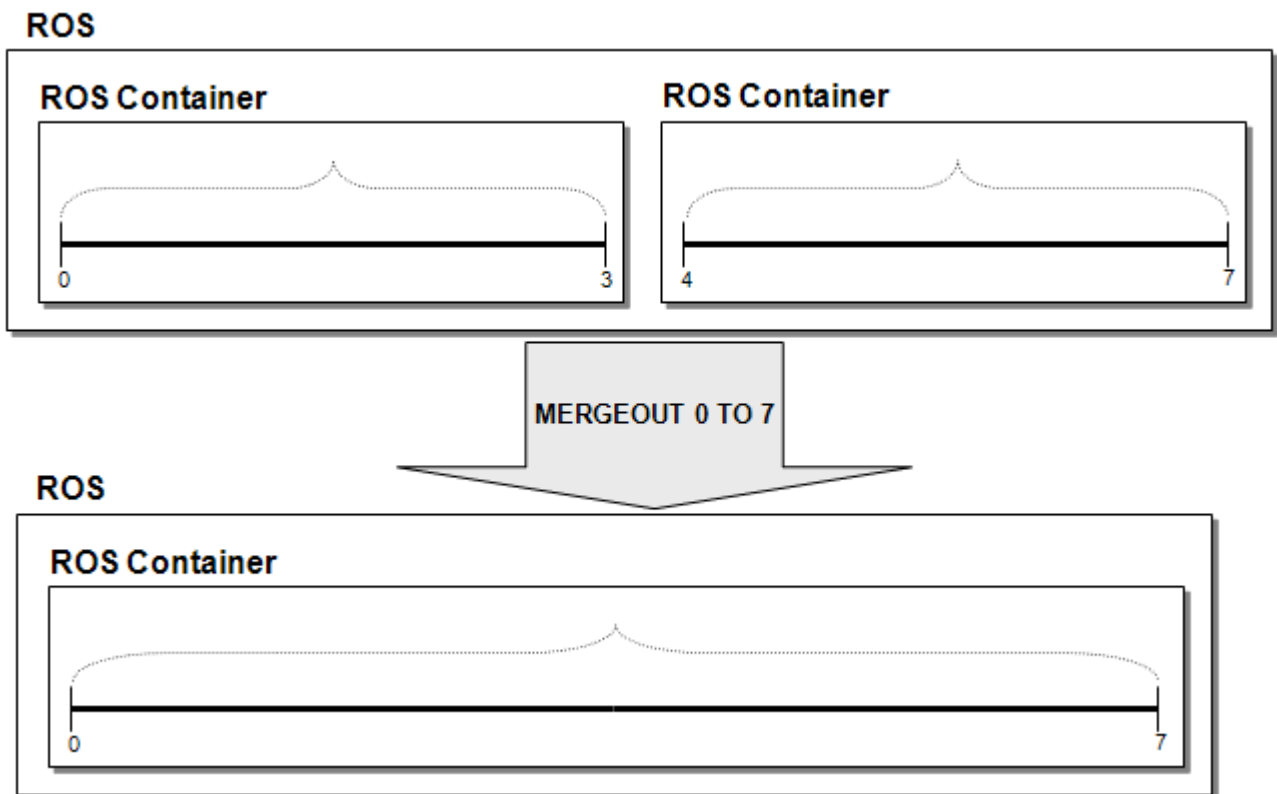
Inserting a tuple with a segmentation column value of 9 creates a new ROS container on node S2 but not on node S1.

Mergeout

A mergeout is the process of consolidating ROS containers and purging deleted records. Subsets of the Read Optimized Store (ROS) and sometimes referred to as ROSs, ROS containers are created by changes to the data stored within a projection as a result of bulk loads and DML. The Tuple Mover periodically merges ROS containers to maximize performance.

Over time, the number of ROS containers increases to a degree that it becomes necessary to merge some of them in order to avoid performance degradation. At that point, the Tuple Mover performs an automatic mergeout, which combines two or more ROS containers into a single container. This process can be thought of as "defragmenting" the ROS.

The following illustration shows the effect of a projection mergeout on a single node:



Tuning the Tuple Mover

The Tuple Mover comes preconfigured to work for most common workloads. However there are a few cases where you might need to tune behavior by changing its configuration parameters. The following section explains these parameters, and the remainder of this section explains how to use them to tune the Tuple Mover for several situations.

Tuple Mover Configuration Parameters

The following configuration parameters control how the Tuple Mover operates. You can use them to tweak its operation to suit your needs, as described in the following sections.

Parameters	Description	Default	Example
ActivePartitionCount	<p>Sets the number of partitions that are to be loaded at the same time. By default, the Tuple Mover assumes that data is only inserted into the most recent partition of a partitioned table. If this is not the case, then set this parameter to the number of partitions that are receiving data at the same time.</p> <p>Note: this parameter's value is ignored if <code>EnableStrataBasedMrgOutPolicy</code> is disabled. See Table Partitioning.</p>	1	<pre>SELECT SET_CONFIG_PARAMETER ('ActivePartitionCount', 2);</pre>
EnableStrataBasedMrgOutPolicy	<p>When set to 1 (the default) enables Vertica 4.0 mergeout behavior. Set this parameter to disabled (value 0) if you want to revert back to 3.5 mergeout behavior.</p> <p>Note: This parameter is deprecated and will be removed in a future release. Avoid using it except under the guidance of Technical Support (on page 1).</p>	1	<pre>SELECT SET_CONFIG_PARAMETER ('EnableStrataBasedMrgOutPolicy', 0);</pre>
MaxMrgOutROSSizeMB	<p>Sets the largest size in MB that a mergeout job can make on a non-partitioned ROS. By setting this to a small value, you can prevent the Tuple Mover from trying to merge large ROS containers, which requires more time to process. Raise this value during periods of lower activity, so the Tuple Mover can consolidate the larger ROS containers.</p>	100	<pre>SELECT SET_CONFIG_PARAMETER ('MaxMrgOutROSSizeMB', 150);</pre>
MergeOutInterval	<p>The number of seconds the Tuple Mover waits between checks for new ROS files to merge out. If ROS containers are added frequently, this value might need to be decreased.</p>	600	<pre>SELECT SET_CONFIG_PARAMETER ('MergeOutInterval', 1200);</pre>

MergeOutPolicySizeList	A list of the file sizes, in bytes, that a ROS file must reach before the Tuple Mover selects it for mergeout. This parameter has an effect only if <code>EnableStrataBasedMrgOutPolicy</code> is disabled. Note: This parameter is deprecated and will be removed in a future release. Avoid using it except under the guidance of Technical Support.	10, 100, 1000, 10000, 100000, 1000000, 10000000, 100000000, 1000000000, 10000000000, 100000000000, 0, 50000000000, 0, 100000000000	
MoveOutInterval	The number of seconds the Tuple mover waits between checks for new data in the WOS to move to ROS.	300	<code>SELECT SET_CONFIG_PARAMETER ('MoveOutInterval', 600);</code>
MoveOutMaxAgeTime	Forces the WOS to be written to disk at the specified interval (in seconds). The default is 30. Tip: If you have been running the <code>force_moveout.sh</code> script, you no longer need to run it.	30	<code>SELECT SET_CONFIG_PARAMETER ('MoveOutMaxAgeTime', 20);</code>
MoveOutSizePct	The percentage of the WOS that can be filled with data before the Tuple Mover performs a moveout operation.	0	<code>SELECT SET_CONFIG_PARAMETER ('MoveOutSizePct', 50);</code>

Resource Pool Settings

The Tuple Mover draws its resources from the `TM` resource pool. Adding more resources and more concurrency to this pool can make the Tuple Mover more effective in dealing with high load rates. The concurrency setting determines how many merges can occur at once. The Tuple Mover dedicates some threads to aggressively address small ROS containers as a side effect of this setting, as some threads are reserved to work only on merges of ROS containers in the lower strata.

```
=> ALTER RESOURCE POOL tm MEMORYSIZE '4096M' PLANNEDCONCURRENCY 4 MAXCONCURRENCY 5;
```

The settings for the `WOSDATA` resource pool also indirectly effect the Tuple Mover. In automatic mode, `INSERT` and `COPY` commands use the concurrency setting to determine whether data is small enough to store in WOS or if it should be written to ROS. Therefore, set this value to be the number of concurrent loads you expect to perform into your database. This pool also determines how much RAM can be used by the WOS.

```
=> ALTER RESOURCE POOL wosdata MAXMEMORYSIZE '4M' PLANNEDCONCURRENCY 3;
```

See *Managing Workloads* (page 292) and *Resource Pool Architecture* (page 294) in this guide and `ALTER RESOURCE POOL` and Built-in Pools in the SQL Reference Manual.

Loading Data

Vertica automatically decides whether the data should be placed in WOS or stored directly in ROS containers based on the amount of data processed by a COPY or INSERT command. Vertica stores large loads directly to disk and stores smaller loads in memory, which it later moves to disk.

For low-latency access to data, use small loads. The automatic Tuple Mover settings are the best option for handling such smaller loads. One exception is for single-node deployments, where a system failure would cause in-memory data to be lost. In this case, you might want to force all data loads to go directly to disk.

For high load rates, you might want the Tuple Mover to check for jobs more frequently by changing the `MergeOutInterval` and `MoveOutInterval` configuration parameters. Reduce the `MoveOutInterval` if you expect the peak load rate to fill the WOS quickly. Reduce `MergeOutInterval` if you anticipate performing many DIRECT loads or inserts.

See COPY [DIRECT] and INSERT in the SQL Reference Manual

Using More Threads

If your database is receiving a large volume of data to load or if it is performing many DIRECT loads or inserts, you should consider allowing the Tuple Mover to perform multiple operations concurrently by increasing the TM resource pool until it can keep up with the anticipated peak load rate. For example:

```
=> ALTER RESOURCE POOL tm MEMORYSIZE '4096M' PLANNEDCONCURRENCY 4 MAXCONCURRENCY 5;
```

See ALTER RESOURCE POOL and Built-in Pools in the SQL Reference Manual.

Active Data Partitions

By default, the Tuple Mover assumes that all loads and updates for partitioned tables are going to the same "active" partition. For example, if a table is partitioned by month, the Tuple Mover expects that after the start of a new month, no data is loaded into the partition for the prior month.

If loads and updates occur to more than one partition, set the `ActivePartitionCount` parameter to reflect the number of partitions that will be loading data. For example, if your database receives data for the current month as well as updates to the prior month, set `ActivePartitionCount` to 2. For tables partitioned by non-temporal attributes, set `ActivePartitionCount` to reflect the number of partitions that will be loaded simultaneously.

See Table Partitioning in this guide.

Scheduling Large Tuple Mover Operations

Tuple Mover uses more disk, CPU, and memory resources when it merges larger ROS containers. You might want to postpone large merge operations during periods of peak query activity, such as during business or trading hours, to keep the system more responsive to queries. You limit the size of ROS containers the Tuple Mover is allowed to merge by setting the configuration parameter `MaxMrgOutROSSizeMB` to a small value (for example 5GB) during the peak times. This limits the Tuple Mover to merging smaller ROS containers, which consumes less disk space and CPU. Reset the value of `MaxMrgOutROSSizeMB` during off hours to allow the Tuple Mover to merge the larger ROS containers.

Reverting to Vertica Version 3.5 Tuple Mover Behavior

The Tuple Mover in Vertica 4.1 handles many situations much more effectively than the Tuple Mover in version 3.5, primarily due to its use of multiple threads. However, if you have invested considerable time tuning the Tuple Mover in version 3.5, you can force the Tuple Mover to use the older behavior by setting the `EnableStrataBasedMrgOutPolicy` configuration parameter to 0. When the strata-based mergeout policy is disabled, the Tuple Mover uses the version 3.5 configuration parameter `MergeOutPolicySizeList`. The Tuple Mover ignores this parameter when `EnableStrataBasedMrgOutPolicy` is enabled.

Note: Both `EnableStrataBasedMrgOutPolicy` and `MergeOutPolicySizeList` are deprecated and will be removed in a future version. Use these parameters only until you can tune your database to use the new Tuple Mover behavior.

See Also

Best Practices for Workload Management (page 301)

Performing Tuple Mover Operations Manually

Vertica recommends that you use the Tuple Mover at all times. However, if you are directed to do so by ***Technical Support*** (on page 1), use the following sequence of operations.

Note: Running the Tuple Mover while performing manual moveout/mergeout operations concurrently is not supported.

- 1 Use the `manageTupleMover` script to stop the Tuple Mover.
From the command line:

```
$ /opt/vertica/bin/manageTupleMover stop wait
```

 From vsql:

```
# \! /opt/vertica/bin/manageTupleMover stop wait
```
- 2 Perform manual tuple mover operations as instructed by ***Technical Support*** (on page 1):
 - `SELECT DO_TM_TASK`
 - `ALTER PROJECTION`
 - `SELECT ANALYZE_STATISTICS`
 See the SQL Reference Manual for descriptions of these commands.
- 3 Restart the Tuple Mover.
From the command line:

```
$ /opt/vertica/bin/manageTupleMover start
```

 From vsql:

```
# \! /opt/vertica/bin/manageTupleMover start
```
- 4 Restart the database if directed to do so.

Collecting Statistics

The Vertica cost-based query optimizer relies on representative statistics on the data. These statistics are used in the optimizer's algorithms to choose between multiple available plans in which to execute a query. Various optimizer decisions rely on having up-to-date statistics, including:

- Choosing between multiple eligible projections to answer the query
- Choosing the best order in which to perform joins
- Choosing between plans involving different algorithms, such as `HASH JOIN` versus `MERGE JOIN` or `HASH GROUP BY` versus `PIPELINED GROUP BY`
- Choosing between distribution algorithms; for example, broadcast and re-segmentation

Without reasonably accurate statistics, the optimizer could choose a suboptimal projection or a suboptimal join order for a query. See ***Statistics Collection Guidelines*** (page 172).

See Also

`ANALYZE_STATISTICS`, `DROP_STATISTICS`, `EXPORT_STATISTICS`, and `IMPORT_STATISTICS` in the SQL Reference Manual

Statistics Used by the Query Optimizer

Vertica uses the estimated values of the following statistics in its cost model:

- Number of rows in the projection (or table)
- Number of distinct values of each column
- Minimum/maximum values of each column
- An equi-height histogram of the distribution of values each column
- Space occupied by the column on disk

Notes

- The Vertica query optimizer and the Database Designer both use the same set of statistics.
- When there are ties, the optimizer chooses the projection that was created earlier.

Statistics Collection Guidelines

Vertica provides two ways to collect statistics:

ANALYZE ROW COUNT

The `ANALYZE ROW COUNT` operation is automatically invoked every 60 seconds to collect a minimal set of statistics for each projection. This lightweight operation aggregates row counts calculated during loads. For example, to set the interval to 1 hour (3600 seconds), issue the following command:

```
=> SELECT SET_CONFIG_PARAMETER('AnalyzeRowCountInterval', 3600);
```

To reset the interval to the default of 1 minute (60 seconds):

```
=> SELECT SET_CONFIG_PARAMETER('AnalyzeRowCountInterval', 60);
```

See **Configuration Parameters** (page 25) in the Administrator's Guide for additional information. This function can be invoked manually, if needed, using the `DO_TM_TASK('analyze_row_count')` function.

ANALYZE_STATISTICS

The `ANALYZE_STATISTICS` function computes full statistics and must be explicitly invoked by the user. It can be invoked on all objects or on a per-table or per-projection basis, although there is no benefit to running it per projection.

Notes

- Even if `ANALYZE_STATISTICS()` is invoked on a projection, it calculates the statistics using the same procedure it used for the table object, so it is more efficient to invoke `ANALYZE_STATISTICS()` on the table object.
- Statistics computation is a cluster-wide operation, which accesses data using a historical query (at epoch latest) without any locks. Once computed, statistics are stored in the catalog and replicated on all nodes. This operation requires an exclusive lock on the catalog for a very short duration, similar to a DDL operation.

How Statistics are Computed

Vertica does not compute statistics incrementally, nor does it update full statistics during load operations.

For large tables exceeding 250,000 rows, histograms for minimum, maximum, and column value distribution are calculated on a sampled subset of rows. The default maximum number of samples for each column is approximately 2^{17} (131702) samples or the number of rows that fits within 1GB of memory, whichever is smaller; for example, the number of samples used for large VARCHAR columns could be less.

Notes

- Vertica does not provide a configuration setting to change the number of samples.
- Statistic collection functions consider data in the ROS but not in the WOS.

Best Practices for Statistics Collection

The query optimizer requires representative statistics; however, for most applications statistics do not have to be accurate to the minute. `DO_TM_TASK('analyze_row_count')` collects partial statistics automatically by default and can be sufficient for many optimizer choices. For example, the following command analyzes the row count on the Vmart Schema database:

```
=> SELECT DO_TM_TASK('analyze_row_count');
        DO_TM_TASK
-----
row count analyze for projection 'call_center_dimension_DBD_27_seg_temp_init_temp_init'
row count analyze for projection 'call_center_dimension_DBD_28_seg_temp_init_temp_init'
row count analyze for projection 'online_page_dimension_DBD_25_seg_temp_init_temp_init'
row count analyze for projection 'online_page_dimension_DBD_26_seg_temp_init_temp_init'
row count analyze for projection 'online_sales_fact_DBD_29_seg_temp_init_temp_init'
row count analyze for projection 'online_sales_fact_DBD_30_seg_temp_init_temp_init'
row count analyze for projection 'customer_dimension_DBD_1_seg_temp_init_temp_init'
row count analyze for projection 'customer_dimension_DBD_2_seg_temp_init_temp_init'
row count analyze for projection 'date_dimension_DBD_7_seg_temp_init_temp_init'
row count analyze for projection 'date_dimension_DBD_8_seg_temp_init_temp_init'
```

```
row count analyze for projection 'employee_dimension_DBD_11_seg_temp_init_temp_init'  
row count analyze for projection 'employee_dimension_DBD_12_seg_temp_init_temp_init'  
row count analyze for projection 'inventory_fact_DBD_17_seg_temp_init_temp_init'  
row count analyze for projection 'inventory_fact_DBD_18_seg_temp_init_temp_init'  
row count analyze for projection 'product_dimension_DBD_3_seg_temp_init_temp_init'  
row count analyze for projection 'product_dimension_DBD_4_seg_temp_init_temp_init'  
row count analyze for projection 'promotion_dimension_DBD_5_seg_temp_init_temp_init'  
row count analyze for projection 'promotion_dimension_DBD_6_seg_temp_init_temp_init'  
row count analyze for projection 'shipping_dimension_DBD_13_seg_temp_init_temp_init'  
row count analyze for projection 'shipping_dimension_DBD_14_seg_temp_init_temp_init'  
row count analyze for projection 'vendor_dimension_DBD_10_seg_temp_init_temp_init'  
row count analyze for projection 'vendor_dimension_DBD_9_seg_temp_init_temp_init'  
row count analyze for projection 'warehouse_dimension_DBD_15_seg_temp_init_temp_init'  
row count analyze for projection 'warehouse_dimension_DBD_16_seg_temp_init_temp_init'  
row count analyze for projection 'store_dimension_DBD_19_seg_temp_init_temp_init'  
row count analyze for projection 'store_dimension_DBD_20_seg_temp_init_temp_init'  
row count analyze for projection 'store_orders_fact_DBD_23_seg_temp_init_temp_init'  
row count analyze for projection 'store_orders_fact_DBD_24_seg_temp_init_temp_init'  
row count analyze for projection 'store_sales_fact_DBD_21_seg_temp_init_temp_init'  
row count analyze for projection 'store_sales_fact_DBD_22_seg_temp_init_temp_init'  
(1 row)
```

Running full `ANALYZE_STATISTICS` on a table is an efficient but potentially long-running operation that analyzes each unique column exactly once across all projections. It can be run concurrently with queries and loads in a production environment; however given that it takes resources (CPU and memory) from queries and loads, Vertica recommends that you run it only when necessary.

A good rule of thumb is to run full `ANALYZE_STATISTICS` on a particular table whenever:

- The table is first **bulk loaded** (page 144).
- A new projection using that table is created and **refreshed** (page 280).
- Number of rows in the table changes by 50%.
- MIN/MAX values in the tables changes by 50%.
- New primary key values are added to tables with referential integrity constraints. Both the primary key and foreign key tables should be reanalyzed.
- Relative size of a table, compared to tables it is being joined to, has changed materially; for example, the table is now only five times larger than the other when previously it was 50 times larger.
- There is a significant deviation in the distribution of data, which would necessitate recalculation of histograms. For example, there is an event that caused abnormally high levels of trading for a particular stock. This is application specific.
- There is a down-time window when the database is not in active use.

Once your system is running well, Vertica recommends that you save exported statistics for all tables. In the unlikely scenario that statistics changes impact optimizer plans, particularly after an upgrade, you can always revert back to the exported statistics. See **Importing and Exporting Statistics** (page 174) for details.

Importing and Exporting Statistics

Use the `EXPORT_STATISTICS()` function to export statistics to a file.

The `IMPORT_STATISTICS()` function can be used to import saved statistics from a file into the catalog where the saved statistics override existing statistics for all projections on the table.

The `IMPORT` and `EXPORT` functions are lightweight because they operate only on metadata.

Removing Statistics

Use the `DROP_STATISTICS()` function to remove statistics.

Caution: Once dropped, it can be very time consuming to regenerate statistics.

Troubleshooting Issues Using Statistics

To help expedite the resolution of your issue, include the system diagnostics, schema (or table and projection definitions), output of the `EXPLAIN` plan, and the output of `EXPORT_STATISTICS()`.

- 1 Run the Diagnostics Utility using the following command.

```
# /opt/vertica/bin/diagnostics [ command ... ]
```
- 2 Send the resulting `.zip` file from the Diagnostics Utility command to **Technical Support** (on page 1).
- 3 Run the following two commands in `vsq`, which send the output files to `/tmp/export.sql` and `/tmp/stats.xml`, respectively:

```
=> SELECT EXPORT_CATALOG('/tmp/export.sql', 'design');  
=> SELECT EXPORT_STATISTICS('/tmp/stats.xml');
```

Bulk Deleting and Purging Data

Vertica provides multiple techniques to remove data from the database in bulk.

Command	Description
DROP TABLE	Permanently removes a table and its definition. Optionally removed associated views and projections as well.
DELETE FROM TABLE	Marks rows with delete vectors and stores them so data can be rolled back to a previous epoch. The data must be eventually purged before the database can reclaim disk space. See <i>Purging Deleted Data</i> (page 180).
TRUNCATE TABLE	Removes all storage and history associated with a table. The table structure is preserved for future use. The results of this command cannot be rolled back.
DROP_PARTITION	Removes one partition from a partitioned table. Each partition contains a related subset of data in the table. Partitioned data can be efficiently dropped, as well as providing query performance benefits. See <i>Partitioning Tables</i> (page 183).

The following table provides a quick reference for the different delete operations you can use. The "Saves History" column indicates whether data can be rolled back to an earlier epoch and queried at a later time.

Syntax	Performance	Commits Tx	Saves History
DELETE FROM base_table	Normal	No	Yes
DELETE FROM temp_table	High	No	No
DELETE FROM base_table WHERE	Normal	No	Yes
DELETE FROM temp_table WHERE	Normal	No	Yes
DELETE FROM temp_table WHERE temp_table ON COMMIT PRESERVE ROWS	Normal	No	Yes
DELETE FROM temp_table WHERE temp_table ON COMMIT DELETE ROWS	High	Yes	No
DROP base_table	High	Yes	No
TRUNCATE base_table	High	Yes	No
TRUNCATE temp_table	High	Yes	No
DROP PARTITION	High	Yes	No

Choosing the right technique for deleting data

- If your goal is to delete both table data and its definitions and start from scratch, use the DROP TABLE [CASCADE] command.

- If your goal is to drop data but preserve table definitions so you can quickly and easily reload data, use TRUNCATE TABLE. Note that unlike DELETE, TRUNCATE does not have to mark each row with delete vectors, so it runs much more quickly.
- If you plan to perform bulk delete operations on a regular basis, Vertica recommends using Partitioning.
- If you perform occasional small deletes or updates and would like the option to roll back or review history, use DELETE FROM TABLE. See *Best Practices for DELETE and UPDATE* (page 177).

For details on syntax and usage, see DELETE, DROP TABLE, TRUNCATE TABLE, CREATE TABLE and DROP_PARTITION in the SQL Reference Manual.

Best Practices for DELETE and UPDATE

Vertica is optimized for query intensive workloads, so deletes and updates might not achieve the same level of performance as queries. Deletes and updates go to the WOS by default, but if the data is sufficiently large and would not fit in memory, Vertica automatically switches to using the ROS. See *Using INSERT, UPDATE, and DELETE* (page 148).

The topics that follow discuss best practices when using delete and update operations in Vertica.

Performance Considerations for Deletes and Updates

Query Performance after Large Deletes

A large number of (un-purged) deleted rows could negatively affect query and recovery performance.

To eliminate the rows that have been deleted from the result, a query must do extra processing. It has been observed if 10% or more of the total rows in a table have been deleted, the performance of a query on the table slows down. However your experience may vary depending upon the size of the table, the table definition, and the query. The same problem can also happen during the recovery. To avoid this, the delete rows need to be purged in Vertica. For more information, see Purge Procedure.

See *Optimizing Deletes and Updates for Performance* (page 178) for more detailed tips to help improve delete performance.

Concurrency

Deletes and updates take exclusive locks on the table. Hence, only one delete or update transaction on that table can be in progress at a time and only when no loads (or INSERTs) are in progress. Deletes and updates on different tables can be run concurrently.

Pre-join Projections

Avoid pre-joining dimension tables that are frequently updated. Deletes and updates to Pre-join projections cascade to the fact table causing a large delete or update operation.

Optimizing Deletes and Updates for Performance

The process of optimizing a design for deletes and updates is the same. Some simple steps to optimize a projection design or a delete or update statement can increase the query performance by tens to hundreds of times. The following section details several proposed optimizations to significantly increase delete and update performance.

Note: For large bulk deletion, Vertica recommends using *Partitioned Tables* (page 183) where possible because it can provide the best delete performance and also improve query performance.

Designing Delete- or Update-Optimized Projections

When all columns required by the delete or update predicate are present in a projection, the projection is optimized for deletes and updates. Delete and update operations on such projections are significantly faster than on non-optimized projections. Both simple and pre-join projections can be optimized.

Example

```
CREATE TABLE t (a integer, b integer, c integer);
CREATE PROJECTION p1 (a ENCODING RLE,b,c) as select * from t order by a;
CREATE PROJECTION p2 (a, c) as select a,c from t order by c, a;
```

In the following example, both p1 and p2 are eligible for delete and update optimization because the a column is available:

```
DELETE from t WHERE a = 1;
```

In the following example, only p1 is eligible for delete and update optimization because the b column is not available in p2:

```
DELETE from t WHERE b = 1;
```

Delete and Update Considerations for Sort Order of Projections

You should design your projections so that frequently used delete or update predicate columns appear in the SORT ORDER of all projections for large deletes and updates.

For example, suppose most of the deletes you perform on a projection look like the following example:

```
DELETE from t where time_key < '1-1-2007'
```

To optimize the deletes, you would make “time_key” appear in the ORDER BY clause of all your projections. This schema design enables Vertica to optimize the delete operation.

Further, add additional sort columns to the sort order such that each combination of the sort key values uniquely identifies a row or a small set of rows. See *Choosing Sort-orders for Low Cardinality Predicates* (page 102). You can use the EVALUATE_DELETE_PERFORMANCE function to analyze projections for sort order issues.

The following three examples demonstrate some common scenarios for delete optimizations. Remember that these same optimizations work for optimizing for updates as well.

In the first scenario, the data is deleted given a time constraint, in the second scenario the data is deleted by a single primary key and in the third scenario the original delete query contains two primary keys.

Scenario 1: Delete by Time

This example demonstrates increasing the performance of deleting data given a date range. You may have a query that looks like this:

```
delete from trades
where trade_date between '2007-11-01' and '2007-12-01';
```

To optimize this query, start by determining whether all of the projections can perform the delete in a timely manner. Issue a `SELECT COUNT(*)` on each projection, given the date range and notice the response time. For example:

```
SELECT COUNT(*) FROM [projection name i.e., trade_p1, trade_p2]
WHERE trade_date BETWEEN '2007-11-01' AND '2007-12-01';
```

If one query is slow, check the uniqueness of the `trade_date` column and determine if it needs to be in the projection's `ORDER BY` clause and/or can be Run Length Encoded (RLE). RLE replaces sequences of the same data values within a column by a single value and a count number.

If the number of unique columns is unsorted, or the average number of repeated rows is less than ten, `trade_date` is too close to being unique and cannot be RLE. If you find this to be the case, add a new column to minimize the search scope.

In this example, add a column for `trade_year = 2007`. However, first determine if the `trade_year` returns a manageable result set. The following query returns the data grouped by trade year.

```
SELECT DATE_TRUNC('year', trade_date), count(*)
FROM trades
GROUP BY DATE_TRUNC('year', trade_date);
```

Assuming that `trade_year = 2007` is near 8k (8k integer is 64k), a column for `trade_year` can be added to the `trades` table. The final `DELETE` statement then becomes:

```
DELETE FROM trades
WHERE trade_year = 2007
AND trade_date BETWEEN '2007-11-01' AND '2007-12-01';
```

Vertica makes the populating of extra columns easier with the ability to define them as part of the `COPY` statement.

Scenario 2: Delete by a Single Primary Key

This example demonstrates increasing the performance of deleting data given a table with a single primary key. Suppose you have the following query:

```
DELETE FROM [table]
WHERE pk IN (12345, 12346, 12347,...);
```

You begin optimizing the query by creating a new column called `'buckets'`, which is assigned the value of one the primary key column divided by 10k; in the above example, `buckets=(int) pk/10000`. This new column can then be used in the query to limit the search scope. The optimized delete would be:

```
DELETE FROM [table]
WHERE bucket IN (1,...)
AND pk IN (12345, 12346, 12347,...);
```

Scenario 3: Delete by Multiple Primary Keys

This example demonstrates deleting data given a table with multiple primary keys. Suppose you have the following query:

```
DELETE FROM [table]
WHERE (pk1, pk2) IN ((12345,5432), (12346,6432), (12347,7432), ...);
```

Similar to the previous example, you create a new column called 'buckets', which is assigned the value of one of the primary key column values divided by 10k; in the above example, `buckets=(int) pk1/10000`. This new column can then be used in the query to limit the search scope.

In addition, you can further optimize the original search by reducing the primary key `IN` list from two primary key columns to one column by creating a second column. For example, you could create a new column named 'pk1-2' that contains the concatenation of the two primary key columns. For example, `pk1-2 = 'pk1' || '-' || 'pk2'`.

Your optimized delete statement would then be:

```
DELETE FROM [table]
WHERE bucket IN (1,.. .)
AND pk1-2 IN ('12345-5432', '12346-6432', '12347-7432',...);
```

Caution: Remember that Vertica does not remove deleted data immediately but keeps it as history for the purposes of historical query. A large amount of history can result in slower query performance. See *Purging Deleted Data* (page 180) for information on how to configure the appropriate amount of history to be retained.

Purging Deleted Data

In Vertica, delete operations do not remove rows from physical storage. Unlike most databases, the `DELETE` command in Vertica marks rows as deleted so that they remain available to historical queries. These deleted rows are called historical data. Retention of historical data also applies to the `UPDATE` command, which is actually a combined `DELETE` and `INSERT` operation.

The cost of retaining deleted data in physical storage can be measured in terms of:

- Disk space for the deleted rows and delete markers
- A performance penalty for reading and skipping over deleted data

A purge operation permanently removes deleted data from physical storage so that the disk space can be reused. Vertica gives you the ability to control how much deleted data is retained in the physical storage used by your database by performing a purge operation using one of the following techniques:

- **Setting a Purge Policy** (page 181)
- **Manually purging data** (page 182)

Both methods set the Ancient History Mark (AHM), which is an epoch that represents the time until which history is retained. History older than the AHM are eligible for purge.

Note: Large delete and purge operations in Vertica could take a long time to complete, so use them sparingly. If your application requires deleting data on a regular basis, such as by month or year, Vertica recommends that you design tables that take advantage of **table partitioning** (page 183). If partitioning tables is not suitable, consider the procedure described in **Rebuilding a Table** (page 290). The `ALTER TABLE..RENAME` command lets you build a new table from the old table, drop the old table, and rename the new table in its place.

Setting a Purge Policy

The preferred method for purging data is to establish a policy that determines which deleted data is eligible to be purged. Eligible data is automatically purged when the Tuple Mover performs mergeout operations.

Vertica provides two methods for determining when deleted data is eligible to be purged:

- Specifying the time for which delete data is saved
- Specifying the number of epochs that are saved

Specifying the time for which delete data is saved

Specifying the time for which delete data is saved is the preferred method for determining which deleted data can be purged. By default, Vertica saves historical data only when nodes are down.

To change the the specified time for saving deleted data, use the `HistoryRetentionTime` **configuration parameter** (page 30):

```
=> SELECT SET_CONFIG_PARAMETER('HistoryRetentionTime', '{ <seconds> | -1 }');
```

In the above syntax:

- `seconds` is the amount of time (in seconds) for which to save deleted data.
- `-1` indicates that you do not want to use the `HistoryRetentionTime` configuration parameter to determine which deleted data is eligible to be purged. Use this setting if you prefer to use the other method (`HistoryRetentionEpochs`) for determining which deleted data can be purged.

The following example sets the history epoch retention level to 240 seconds:

```
=> SELECT SET_CONFIG_PARAMETER('HistoryRetentionTime', '240');
```

Specifying the number of epochs that are saved

Unless you have a reason to limit the number of epochs, Vertica recommends that you specify the time over which delete data is saved.

To specify the number of historical epoch to save through the `HistoryRetentionEpochs` configuration parameter:

- 1 Turn off the `HistoryRetentionTime` configuration parameter:

```
=> SELECT SET_CONFIG_PARAMETER('HistoryRetentionTime', '-1');
```

- 2 Set the history epoch retention level through the `HistoryRetentionEpochs` configuration parameter:

```
=> SELECT SET_CONFIG_PARAMETER('HistoryRetentionEpochs',  
    '{<num_epochs>|-1}');
```

- *num_epochs* is the number of historical epochs to save.
- *-1* indicates that you do not want to use the `HistoryRetentionEpochs` configuration parameter to trim historical epochs from the epoch map. By default, `HistoryRetentionEpochs` is set to *-1*.

The following example sets the number of historical epochs to save to 40:

```
=> SELECT SET_CONFIG_PARAMETER('HistoryRetentionEpochs', '40');
```

Modifications are immediately implemented across all nodes within the database cluster. You do not need to restart the database.

Note: If both `HistoryRetentionTime` and `HistoryRetentionEpochs` are specified, `HistoryRetentionTime` takes precedence.

See *Epoch Management Parameters* (page 30) for additional details.

Disabling Purge

If you want to preserve all historical data, set the value of both historical epoch retention parameters to *-1*, as follows:

```
=> SELECT SET_CONFIG_PARAMETER('HistoryRetentionTime', '-1');  
=> SELECT SET_CONFIG_PARAMETER('HistoryRetentionEpochs', '-1');
```

Manually Purging Data

Manually purging deleted data consists of the following series of steps:

- 1 **Determine the point in time** to which you want to purge deleted data.
- 2 **Set the Ancient History Mark (AHM)** to this point in time using one of the following SQL functions (described in the SQL Reference Manual):
 - `SET_AHM_TIME()` sets the AHM to the epoch that includes the specified `TIMESTAMP` value on the initiator node.
 - `SET_AHM_EPOCH()` sets the AHM to the specified epoch.
 - `GET_AHM_TIME()` returns a `TIMESTAMP` value representing the AHM.
 - `GET_AHM_EPOCH()` returns the number of the epoch in which the AHM is located.
 - `MAKE_AHM_NOW()` sets the AHM to the greatest allowable value (now), and lets you drop pre-existing projections. This purges all deleted data.

When you use `SET_AHM_TIME` or `GET_AHM_TIME`, keep in mind that the timestamp you specify is mapped to an epoch, which has (by default) a three-minute granularity. Thus, if you specify an AHM time of '2008-01-01 00:00:00.00' the resulting purge could permanently remove as much as the first three minutes of 2008, or could fail to remove the last three minutes of 2007.

Note: The system prevents you from setting the AHM beyond the point at which it would prevent recovery in the event of a node failure.

- 3 **Manually initiate a purge** using one of the following SQL functions (described in the SQL Reference Manual):
 - `PURGE_PROJECTION()` purges a specified projection.

- `PURGE_TABLE()` purges all projections on the specified table.
- `PURGE()` purges all projections in the physical schema.

The Tuple Mover performs a mergeout operation to purge the data.

Notes:

- Manual purge operations can take a long time.
- For purge operation on a non-partitioned table, all ROS containers are combined into a single container. Non-partitioned tables cannot be re-partitioned into multiple ROS containers.
- A purge operation on a partitioned table also results in a single ROS. To re-partition a partitioned table into multiple ROS containers, use the `PARTITION_TABLE()` function.
- If you have partitioned tables that need to be explicitly purged, use `MERGE_PARTITIONS()` instead of `PURGE()`, targeting any partitions with large numbers of deleted rows. Use the `deleted_row_count` column in the `STORAGE_CONTAINERS` system table to find the partitions that need to be merged for the purge effect.

Partitioning Tables

Vertica supports data partitioning at the table level, which divides one large table into smaller pieces. Partitions are a property of the table.

A common use for partitions is to split a table that contains decades of data by year. Partitioning this data lets you quickly rotate out old data.

Table partitioning applies to all projections of a given table. It is used to segregate the data within each node to facilitate dropping partitions, by letting you discard old partitions of data to make room for new data (rolling window). Partitions can improve parallelism during query execution and also enable some optimizations that can improve query performance.

The basic operations for working with partitions are described in the sections that follow:

- **Defining partitions** (page 184)
- **Loading data** (page 144), and engaging in other normal operations
- Forcing partitioned data, if needed, and **dropping partitions** (page 186)

There is a distinction between partitioning at the table level and the segmentation (hash or range) associated with a projection. Table partitioning segregates data into different partitions within each node, while segmentation distributes data for a projection across multiple nodes in a cluster. Different projections on the same table have identical partitioning but can have different segmentation clauses. Both methods provide opportunities for parallelism during query processing. In short:

- **Segmentation**—defined by the projection for distributed computing.
- **Partitioning**—defined by the table for fast data purges and query performance. An added bonus is that a partition can be dropped.

See also **Partitioning and Segmenting Data** (page 188) in this book and Projection Segmentation in the Concepts Guide.

Tip: Partitioning tables allows for fast data deletion. When a storage container contains data for a single partition, that storage can just be thrown away when the partition is dropped using the `DROP_PARTITION()` function.

Vertica provides the following functions that let you manage your partitions and obtain additional information about them. See the SQL Reference Manual for details:

- `DROP_PARTITION` drops all data belonging to the partition with the specified partition value.
- `PARTITIONS` system table displays partition metadata, one row per partition key per ROS container.
- `MERGE_PARTITIONS` merges ROSs that have data belonging to partitions in a specified partition key range.
- `PARTITION_PROJECTION` forces a split of ROS containers of the specified projection.
- `PARTITION_TABLE` forces the system to break up any ROSs that contain multiple distinct values of the partitioning expression.

Defining Partitions

The first step in defining data partitions is to establish the relationship between the data and partitions. To illustrate, consider the following table called `trade`, which contains unpartitioned data for trade date, ticker symbol, and trade time.

Table 1: Unpartitioned data

tdate	tsymbol	ttime
2006-01-02	AAA	13:00:00
2007-02-04	BBB	14:30:00
2008-09-18	AAA	09:55:00
2007-05-06	AAA	11:14:30
2006-12-22	BBB	15:30:00

(5 rows)

If you wanted to discard data once a year, a logical way to partition the table would be by year. The partition expression `PARTITION BY EXTRACT(year FROM tdate)` provides the partitioning shown in Table 2:

Table 2: Data partitioned by year

2006			2007			2008		
tdate	tsymbol	ttime	tdate	tsymbol	ttime	tdate	tsymbol	ttime
01/02/06	AAA	13:00:00	02/04/07	BBB	14:30:00	09/18/08	AAA	09:55:00
12/22/06	BBB	15:30:00	05/06/07	AAA	11:14:30			

Unlike some databases, which require that partition boundaries be defined explicitly in the `CREATE TABLE` statement, Vertica selects a partition for each row based on the result of a partitioning expression provided in the `CREATE TABLE` statement. Partitions do not have explicit names associated with them. Vertica internally creates a partition for each distinct value in the `PARTITION BY` expression.

After you specify a partition expression, Vertica processes the data by applying the partition expression to each row and then assigning partitions.

The following syntax generates the partitioning for this example, with results shown in Table 3. It creates a table called `trade` and partitions it by year. For additional information, see `CREATE TABLE` in the SQL Reference Manual.

```
CREATE TABLE trade (
  tdate DATE NOT NULL,
  tsymbol VARCHAR(8) NOT NULL,
  ttime TIME)
PARTITION BY EXTRACT (year FROM tdate);
CREATE PROJECTION trade_p (tdate, tsymbol, ttime) AS
SELECT * FROM trade
ORDER BY tdate, tsymbol, ttime UNSEGMENTED ALL NODES;

INSERT INTO trade VALUES ('01/02/06' , 'AAA' , '13:00:00');
INSERT INTO trade VALUES ('02/04/07' , 'BBB' , '14:30:00');
INSERT INTO trade VALUES ('09/18/08' , 'AAA' , '09:55:00');
INSERT INTO trade VALUES ('05/06/07' , 'AAA' , '11:14:30');
INSERT INTO trade VALUES ('12/22/06' , 'BBB' , '15:30:00');
```

Table 3: Partitioning expression and results

tdate	tsymbol	ttime	EXTRACT (year FROM tdate)	tdate	tsymbol	ttime
01/02/06	AAA	13:00:00.00	2006	01/02/06	AAA	13:00:00.00
02/04/07	BBB	14:30:00.00	2007	12/22/06	BBB	15:30:00.00
09/18/08	AAA	09:55:00.00	2008	05/06/07	AAA	11:14:30.00
05/06/07	AAA	11:14:30.00	2007	02/04/07	BBB	14:30:00.00
12/22/06	BBB	15:30:00.00	2006	09/18/08	AAA	09:55:00.00

Restrictions on Partitioning Expressions

- The partitioning expression can reference one or more columns from the table.
- The partitioning expression cannot evaluate to NULL for any row, so do not include nullable columns in the `CREATE TABLE..PARTITION BY` expression.
- SQL functions used in the partitioning expression must be immutable, which means they return the exact same value regardless of when it is invoked and independently of session or environment settings, such as `LOCALE`. For example, the `TO_CHAR` function is dependent on locale settings and cannot be used. `RANDOM` produces different values on each invocation and cannot be used.
- Vertical functions cannot be used in partitioning expressions.
- Single quotes are optional for `INT` and `FLOAT` data types when specifying arguments in `CREATE TABLE`. Quotes are required for all other data types.
- All projections anchored on a table must include all columns referenced in the `PARTITION BY` expression; this allows the partition to be calculated.
- You cannot modify partition expressions once a partitioned table is created. If you want modified partition expressions, create a new table with a new `PARTITION BY` clause, and then `INSERT...SELECT` from the old table to the new table. Once your data is partitioned the way you want it, you can safely drop the old table.

Best Practices for Partitioning

- Try to minimize the number of partitions you create. The maximum recommended number of partitions varies with the number of columns in the table, as well as system RAM. Vertica recommends a maximum of 20 partitions and, ideally, no more than 12.
- Do not apply partitioning to tables used as dimension tables in pre-join projections. You can apply partitioning to tables used as large single (fact) tables in pre-join projections.

Dropping Partitions

Internally, Vertica attempts to keep data from different partitions segregated into different ROS containers. This separation makes it easier to drop partitions, as ROS containers that have data from the partition are also dropped.

Occasionally a ROS container could contain rows that belong to more than one partition, such as after a MERGE_PARTITIONS operation. The table below, for example, shows three ROS containers, two of which contain data for the appropriate partition (2007 and 2008, respectively) and one ROS container with both 2007 and 2008 data:

	ROS 1	ROS 1		ROS 3	ROS 3		ROS 2
	tdate	tdate		tdate	tdate		tdate
	-----	-----		-----	-----		-----
2007 data	08/15/07	08/15/07	2008 data	03/06/08	03/06/08	2007 and	12/22/07
	09/01/07	09/01/07		03/10/08	03/10/08	2008 data	12/29/07
	09/12/07	09/12/07		04/12/08	04/12/08		01/03/08
	10/02/07	10/02/07		05/30/08	05/30/08		02/04/08

To drop the 2007 partition from ROS2, use the DROP_PARTITION() function, which forces the partition of data into two ROS containers (one for 2007 and one for 2008) and then drops the specified partition.

Notes and Restrictions

Any of the following operations could result in a ROS container with mixed partitions:

- Manual invocation of certain Tuple Mover operations, including:
 - DO_TM_TASK, which runs a Tuple Mover task (moveout) on n number of projections defined over the specified table.
 - MERGE_PARTITIONS, which merges ROS containers that have data belonging to partitions in a specified partition key range.
 - PURGE(), which purges all projections in the physical schema. Remember that the PURGE() function creates a single ROS container.
- Refresh and recovery operations can generate ROS containers with mixed partitions under some conditions. See **Auto Partitioning** (page 192).

The number of partitions that contain data is restricted by the number of ROS containers that can comfortably exist in the system.

In general, if a ROS container has data that belongs to $n+1$ partitions and you want to drop a specific partition, the DROP_PARTITION operation:

- 1 Forces the partition of data into two containers where

- one container holds the data that belongs to the partition that is to be dropped
- another container holds the remaining n partitions

2 Drops the specified partition.

You can also use the MERGE_PARTITIONS function to merges ROS containers that have data belonging to partitions in a specified partition key range; for example, [partitionKeyFrom, partitionKeyTo].

DROP_PARTITION forces a moveout if there is data in the WOS (WOS is not partition aware).

DROP_PARTITION acquires an exclusive lock on the table to prevent DELETE | UPDATE | INSERT | COPY statements from affecting the table, as well as any SELECT statements issued at SERIALIZABLE isolation level.

Users must be the table owner to drop a partition. They must have MODIFY (INSERT | UPDATE | DELETE) permissions in order to:

- Partition a projection/table
- Merge partitions
- Run mergeout, moveout or purge operations on a projection

DROP_PARTITION operations cannot be performed on tables with projections that are not up to date (have not been refreshed).

Examples

Using the example schema in *Defining Partitions* (page 184), the following command explicitly drops the 2006 partition key from table trade:

```
SELECT DROP_PARTITION('trade', 2006);
DROP_PARTITION
-----
Partition dropped
(1 row)
```

Here, the partition key is specified:

```
SELECT DROP_PARTITION('trade', EXTRACT('year' FROM '2006-01-01'::date));
DROP_PARTITION
-----
Partition dropped
(1 row)
```

The following example creates a table called dates and partitions the table by year:

```
CREATE TABLE dates (
    year INTEGER NOT NULL,
    month VARCHAR(8) NOT NULL)
PARTITION BY year * 12 + month;
```

The following statement drops the partition using a constant for Oct 2007 (2007*12 + 10 = 24094):

```
SELECT DROP_PARTITION('dates', '24094');
DROP_PARTITION
-----
Partition dropped
```

(1 row)

Alternatively, the expression can be placed in line: `SELECT DROP_PARTITION('dates', 2007*12 + 10);`

See Also

`DROP_PARTITION` in the SQL Reference Manual

Partitioning and Segmenting Data

Partitioning and segmentation have completely separate functions in Vertica. It is important to clarify the differences because the concepts are similar, and these terms are often used interchangeably for other databases.

In Vertica, segmentation defines how data is spread among cluster nodes, while partitioning specifies how data is organized within the individual nodes. Segmentation is defined by the projection, and partitioning is defined by the table. Logically, the partition clause is applied after the segmented by clause. See `CREATE TABLE` in the SQL Reference Manual for details.

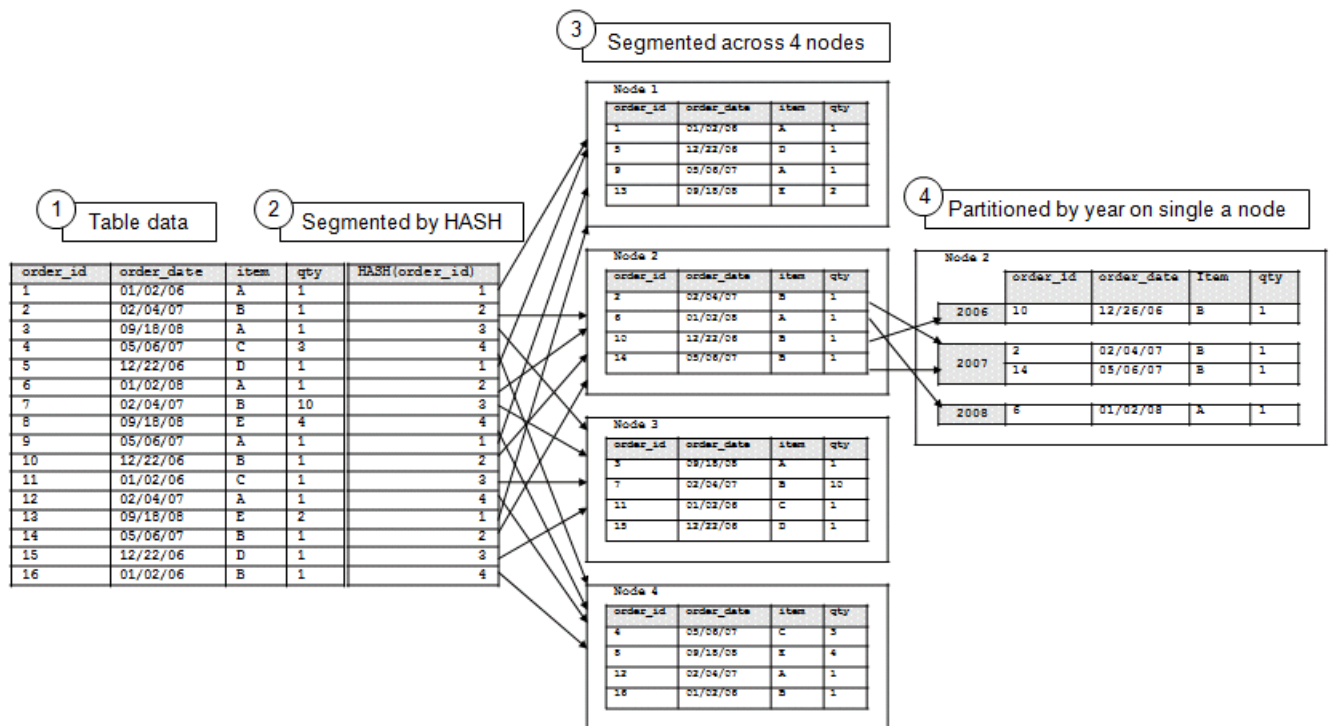
Segmentation and partitioning have opposite goals regarding data localization. Partitioning attempts to introduce hot spots within the node, allowing for a convenient way to drop data and reclaim the disk space. Segmentation (by hash) distributes the data evenly across all nodes in a Vertica cluster.

Partitioning by year, for example, makes sense if you intend to retain and drop data at the granularity of a year. On the other hand, segmenting the data by year would be an extremely bad choice, as the node holding data for the current year would likely answer far more queries than the other nodes.

The following table shows segmentation and partitioning combined with the following flow:

- 1 Example table data
- 2 Data segmented by `HASH(order_id)`
- 3 Data segmented across four nodes
- 4 Data partitioned by year on a single node

Note: In the following example, the partitioning process occurs on all four nodes, but the illustration shows partitioned data on just one node for simplicity



See Also

Reclaiming Disk Space (page 290)

Using Identically Segmented Projections in the Programmer's Guide

Projection Segmentation in the Concepts Guide

CREATE TABLE in the SQL Reference Manual

Partitioning and Data Storage

Partitions and ROS Containers

- Data is automatically split into partitions during load / refresh / recovery operations.
- The Tuple Mover maintains physical separation of partitions.
- Each ROS container contains data for a single partition, though there can be multiple ROS containers for a single partition.

Partition Pruning

When a query predicate includes one or more columns in the partitioning clause, queries look only at relevant ROS containers. See **Partition Elimination** (page 194) for details.

Managing Partitions

Vertica provides several functions that let you manage and monitor your partitions.

Note: Partitioning functions take immutable functions only, in order that the same information be available across all nodes.

PARTITION_TABLE

The `PARTITION_TABLE()` function physically separates partitions into separate containers. Only ROS containers with more than one distinct value participate in the split.

Vertica internal operations (mergeout, refresh, and recovery) maintain partition separation except in certain cases:

- Recovery of a projection when the buddy projection from which the partition is recovering is identically sorted. If the projection is undergoing a full rebuild, it is recovered one ROS container at a time. The projection ends up with a storage layout identical to its buddy and is, therefore, properly segmented.

Note: In the case of a partial rebuild, all recovered data goes into a single ROS container and must be partitioned manually.

- Manual tuple mover operations often output a single storage container, combining any existing partitions; for example, after executing any of the `PURGE()` operations.

The following example creates a simple table called `states` and partitions data by state.

```
=> CREATE TABLE states (
    year INTEGER NOT NULL,
    state VARCHAR NOT NULL)
    PARTITION BY state;
=> CREATE PROJECTION states_p (state, year) AS
    SELECT * FROM states
    ORDER BY state, year UNSEGMENTED ALL NODES;
```

Now issue the command to partition table `states`:

```
=> SELECT PARTITION_TABLE('states');
           PARTITION_TABLE
-----
partition operation for projection 'states_p_node0004'
partition operation for projection 'states_p_node0003'
partition operation for projection 'states_p_node0002'
partition operation for projection 'states_p_node0001'
(1 row)
```

MERGE_PARTITIONS

The `MERGE_PARTITIONS()` function merges partitions between the specified values to a single ROS container. For example:

```
MERGE_PARTITIONS ( table_name , partition_key_from , partition_key_to )
```

The edge values of the partition key are included in the range, and `partition_key_from` must be less than or equal to `partition_key_to`. Inclusion of partitions in the range is based on the application of less than(<)/greater than(>) operators of the corresponding data type.

Note: No restrictions are placed on a partition key's data type.

If `partition_key_from` is the same as `partition_key_to`, all ROS containers of the partition key are merged into one ROS.

Users must be the table owner to drop a partition. They must have `MODIFY (INSERT | UPDATE | DELETE)` permissions in order to:

- Partition a projection/table
- Merge partitions

Run mergeout, moveout or purge operations on a projection The following series of statements show how to merge partitions:

```
=> SELECT MERGE_PARTITIONS('T1', '200', '400');
=> SELECT MERGE_PARTITIONS('T1', '800', '800');
=> SELECT MERGE_PARTITIONS('T1', 'CA', 'MA');
=> SELECT MERGE_PARTITIONS('T1', 'false', 'true');
=> SELECT MERGE_PARTITIONS('T1', '06/06/2008', '06/07/2008');
=> SELECT MERGE_PARTITIONS('T1', '02:01:10', '04:20:40');
=> SELECT MERGE_PARTITIONS('T1', '06/06/2008 02:01:10', '06/07/2008 02:01:10');
=> SELECT MERGE_PARTITIONS('T1', '8 hours', '1 day 4 hours 20 seconds');
```

PARTITIONS

You can also displays partition metadata, one row per partition key, per ROS container, via the `PARTITIONS` system table.

Given a projection named `p1` with three ROS containerS (`RC1`, `RC2` and `RC3`), the values are defined as follow:

COLUMN NAME	RC1	RC2	RC3
PARTITION_KEY	(20,30,40)	(20)	(30,60)
ROS_ID	45035986273705000	45035986273705001	45035986273705002
SIZE	1000	20000	30000
ROW_ROW_COUNT	100	200	300
NODE_NAME	node01	node01	node01

The `PARTITIONS` function returns six rows:

```
=> SELECT PARTITION_KEY, PROJECTION_NAME, ROS_ID, ROS_SIZE_BYTES, ROS_ROW_COUNT,NODE_NAME
FROM PARTITIONS;
```

PARTITION_KEY	PROJECTION_NAME	ROS_ID	ROS_SIZE_BYTES	ROS_ROW_COUNT	NODE_NAME
20	p1	45035986273705000	10000	100	node01
30	p1	45035986273705000	10000	100	node01
40	p1	45035986273705000	10000	100	node01
20	p1	45035986273705001	20000	200	node01
30	p1	45035986273705002	30000	300	node01
60	p1	45035986273705002	30000	300	node01

Notes

There are just a few more things worth mentioning in helping you manage your partitions.

- To prevent too many ROS containers, be aware that delete operations must open all the containers; thus, ideally create fewer than 20 partitions and avoid creating more than 50. You can use the MERGE_PARTITIONS() function to merge old partitions to a single ROS container.
- Non-deterministic functions cannot be used in the PARTITION BY expression. One example is TIMESTAMP WITH TIME ZONE, because the value depends on user settings.
- A dimension table in a pre-join projection cannot be partitioned.

Auto Partitioning

Vertica attempts to keep data from each partition stored separately. Auto partitioning occurs when data is written to disk, such as during COPY DIRECT or moveout operations.

Separate storage provides two benefits: Partitions can be dropped quickly, and **partition elimination** (page 194) can eliminate storage that need not participate in a query plan.

Vertica internal operations (mergeout, refresh, and recovery) maintain partition separation except in certain cases:

- Recovery of a projection when the buddy projection from which the partition is recovering is identically sorted. If the projection is undergoing a full rebuild, it is recovered one ROS container at a time. The projection ends up with a storage layout identical to its buddy and is, therefore, properly segmented.
Note: In the case of a partial rebuild, all recovered data goes into a single ROS container and must be partitioned manually.
- Manual tuple mover operations often output a single storage container, combining any existing partitions; for example, after executing any of the PURGE () operations.

Notes

- If you use INSERT...SELECT into a partitioned table, Vertica sorts the data before writing it to disk, even if the source of the SELECT has the same sort order as the destination.
- Refer to Partitioning Errors in the Troubleshooting Guide if the system returns 'Too many ROS' errors.

Examples

The examples that follow use this simple schema. First create a table named `t1` and partition the data on the `c1` column:

```
CREATE TABLE t1 (  
  c1 INT NOT NULL,
```

```

    c2 INT NOT NULL)
SEGMENTED BY c1 ALL NODES
PARTITION BY c2;

```

Create two identically-segmented buddy projections:

```

CREATE PROJECTION t1_p AS SELECT * FROM t1 SEGMENTED BY HASH(c1) ALL NODES OFFSET 0;
CREATE PROJECTION t1_p1 AS SELECT * FROM t1 SEGMENTED BY HASH(c1) ALL NODES OFFSET 1;

```

Now insert some data:

```

INSERT INTO t1 VALUES (10,15);
INSERT INTO t1 VALUES (20,25);
INSERT INTO t1 VALUES (30,35);
INSERT INTO t1 VALUES (40,45);

```

Query the table to verify the inputs:

```

SELECT * FROM t1;
  c1 | c2
-----
  10 | 15
  20 | 25
  30 | 35
  40 | 45
(4 rows)

```

Now perform a moveout operation on the projections in the table:

```

SELECT DO_TM_TASK('moveout','t1');
      do_tm_task
-----
moveout for projection 't1_p1'
moveout for projection 't1_p'
(1 row)

```

Query the `PARTITIONS` system table, and you'll see that the four partition keys reside on two nodes, each in its own ROS container (see the `ros_id` column). The `PARTITION BY` clause was used on column `c2`, so Vertica auto partitioned the input values during the `COPY` operation:

```

SELECT partition_key, projection_name, ros_id, ros_size_bytes, ros_row_count, node_name
FROM PARTITIONS WHERE projection_name like 't1_p1';
  partition_key | projection_name |      ros_id      | ros_size_bytes | ros_row_count |      node_name
-----+-----+-----+-----+-----+-----
  15            | t1_p1          | 49539595901154617 | 78            | 1            | node0002
  25            | t1_p1          | 54043195528525081 | 78            | 1            | node0003
  35            | t1_p1          | 54043195528525069 | 78            | 1            | node0003
  45            | t1_p1          | 49539595901154605 | 79            | 1            | node0002
(4 rows)

```

Vertica does not auto partition when you refresh with the same sort order. If you create a new projection, Vertica returns a message telling you to refresh the projections; for example:

```

CREATE PROJECTION t1_p2 AS SELECT * FROM t1 SEGMENTED BY HASH(c1) ALL NODES OFFSET 2;
WARNING: Projection <public.t1_p2> is not available for query processing.
Execute the select
  start_refresh() function to copy data into this projection.
  The projection must have a sufficient number of buddy projections and all nodes
  must be up before starting a refresh.

```

Run the `START_REFRESH` function:

```
SELECT START_REFRESH();
           start_refresh
```

```
-----
Starting refresh background process.
(1 row)
```

Query the PARTITIONS system table again. The partition keys now reside in two ROS containers, instead of four, which you can tell by looking at the values in the `ros_id` column. The `ros_row_count` column holds the number of rows in the ROS container:

```
SELECT partition_key, projection_name, ros_id, ros_size_bytes, ros_row_count, node_name
FROM PARTITIONS WHERE projection_name like 't1_p2';
partition_key | projection_name |      ros_id      | ros_size_bytes | ros_row_count |      node_name
-----+-----+-----+-----+-----+-----
15            | t1_p2          | 54043195528525121 |          80    |          2    | node0003
25            | t1_p2          | 58546795155895541 |          77    |          2    | node0004
35            | t1_p2          | 58546795155895541 |          77    |          2    | node0004
45            | t1_p2          | 54043195528525121 |          80    |          2    | node0003
(4 rows)
```

The following command more specifically queries ROS information for the partitioned tables. In this example, the query counts two ROS containers each on two different nodes for projection `t1_p2`:

```
SELECT ros_id, node_name, COUNT(*) FROM PARTITIONS
WHERE projection_name LIKE 't1_p2' GROUP BY ros_id, node_name;
ros_id      | node_name | COUNT
-----+-----+-----
54043195528525121 | node0003 |      2
58546795155895541 | node0004 |      2
(2 rows)
```

This command returns a result of four ROS containers on two different nodes for projection `t1_p1`:

```
SELECT ros_id,node_name, COUNT(*) FROM PARTITIONS
WHERE projection_name LIKE 't1_p1' GROUP BY ros_id, node_name;
ros_id      | node_name | COUNT
-----+-----+-----
49539595901154605 | node0002 |      1
49539595901154617 | node0002 |      1
54043195528525069 | node0003 |      1
54043195528525081 | node0003 |      1
(4 rows)
```

See Also

DO_TM_TASK and PARTITIONS and START_REFRESH in the SQL Reference Manual

Partition Elimination

Vertica eliminates unneeded ROS containers (of partitioned tables) from being processed by queries, by comparing query predicates to partition-related metadata that is associated with the ROS containers. What this means is that Vertica eliminates a scan on partitions where there can be no matching values.

This pruning occurs at query time and requires a query predicate on the partitioning column.

Each ROS of each partition expression column keeps track of the minimum and maximum values of data stored in that ROS. Those values are then used to potentially eliminate ROS containers from query planning. For example, if a ROS does not contain data that satisfies a given query, that ROS is eliminated from query planning. After non-participating ROS containers are eliminated, queries that use partitioned tables run more quickly.

Assume a table that is partitioned by year (2007, 2008, 2009) into three ROS containers, one for each year. Given the following series of commands, the two ROS containers that contain data for 2007 and 2008 fall outside the boundaries of the requested year (2009) and can be eliminated.

```
PARTITION BY EXTRACT(year FROM date);
SELECT ... WHERE date = '12-2-2009';
```

date	amount	date	amount	date	amount
11/11/09	6	03/13/08	96	07/12/07	43
06/05/09	12	04/21/08	17	03/02/07	45
...
12/02/09	8	12/07/08	7	12/02/07	68
Min: 01/01/09 Max: 12/31/09		Min: 01/01/08 Max: 12/31/08		Min: 01/01/07 Max: 12/31/07	

Making past partitions eligible for elimination

On any database that has been upgraded from version 3.5 and prior to Vertica 4.1, ROS containers are ineligible for partition elimination because they do not contain the minimum/maximum partition key values required. These ROS containers need to be recreated or merged by the Tuple Mover.

The following optional procedure lets you take advantage of making past partitions eligible for elimination.

The easiest way to guarantee that all ROS containers are eligible is to:

- 1 Create a new fact table with the same projections as the existing table.
- 2 Use INSERT..SELECT to populate the new table.
- 3 Drop the original table and rename the new table.

If there is not enough disk space for a second copy of the fact table, an alternative is to:

- 1 Verify that the Tuple Mover has finished all post-upgrade work; for example, when the following command shows no mergeout activity:

```
SELECT * FROM TUPLE_MOVER_OPERATIONS;
```

- 2 Identify which partitions need to be merged to get the ROS minimum/maximum values:

```
SELECT DISTINCT table_schema, projection_name, partition_key
FROM partitions p LEFT OUTER JOIN vs_ros_min_max_values v
ON p.ros_id = v.delid
WHERE v.min_value IS null;
```

- 3 Insert a record into each partition that has ineligible ROS containers and commit.
- 4 Delete each inserted record and commit again.

At this point, the Tuple Mover automatically merges ROS containers from past partitions.

To verify the merge occurred:

- 1 Query the TUPLE_MOVER_OPERATIONS table again:

```
SELECT * FROM TUPLE_MOVER_OPERATIONS;
```

- 2 Check again for any partitions that need to be merged:

```
SELECT DISTINCT table_schema, projection_name, partition_key
FROM partitions p LEFT OUTER JOIN vs_ros_min_max_values v
ON p.ros_id = v.rosid
WHERE v.min_value IS null;
```

Examples

Assume a table that is partitioned by time and queries that restrict data on time.

```
CREATE TABLE time (
  tdate DATE NOT NULL,
  tnum INTEGER)
PARTITION BY EXTRACT(year FROM tdate);
CREATE PROJECTION time_p (tdate, tnum) AS
SELECT * FROM time
ORDER BY tdate, tnum UNSEGMENTED ALL NODES;
```

Note: Projection sort order has no effect on partition elimination.

```
INSERT INTO time VALUES ('03/15/04' , 1);
INSERT INTO time VALUES ('03/15/05' , 2);
INSERT INTO time VALUES ('03/15/06' , 3);
INSERT INTO time VALUES ('03/15/06' , 4);
```

The data inserted in the previous series of commands would be loaded into three ROS containers, one per year, since that is how the data is partitioned:

```
SELECT * FROM time ORDER BY tnum;
  tdate      | tnum
-----+-----
 2004-03-15 |    1  --ROS1 (min 03/01/04, max 03/15/04)
 2005-03-15 |    2  --ROS2 (min 03/15/05, max 03/15/05)
 2006-03-15 |    3  --ROS3 (min 03/15/06, max 03/15/06)
 2006-03-15 |    4  --ROS3 (min 03/15/06, max 03/15/06)
(4 rows)
```

In the first query, Vertica eliminates ROS2 because it is only looking for year 2004:

```
SELECT COUNT(*) FROM time WHERE tdate = '05/07/2004';
```

In the next query, Vertica eliminates both ROS1 and ROS3:

```
SELECT COUNT(*) FROM time WHERE tdate = '10/07/2005';
```

This query has an additional predicate on the `tnum` column for which no minimum/maximum values are maintained. In addition, the use of logical operator `OR` is not supported, so no ROS elimination occurs:

```
SELECT COUNT(*) FROM time WHERE tdate = '05/07/2004' OR tnum = 7;
```

Monitoring the Database

This section describes some of the ways in which you can monitor the health of your Vertica database.

Monitoring the Log Files

When a Database is Running

When a Vertica database is running, each node in the cluster writes messages into a file named `vertica.log`. For example, the Tuple Mover and the transaction manager write INFO messages into `vertica.log` at specific intervals even when there is no WOS activity.

To monitor a running database in real time:

- 1 Log in to the database administrator account on any or all nodes in the cluster.
- 2 In a terminal window (such as `vsq`) enter:

```
$ tail -f catalog-path/database-name/node-name catalog/vertica.log
```

<code>catalog-path</code>	The catalog pathname specified when you created the database. See Creating a Database (page 342) in the Administrator's Guide.
<code>database-name</code>	The database name (case sensitive)
<code>node-name</code>	The node name, as specified in the database definition. See Viewing a Database (page 344) in the Administrator's Guide.

When the Database / Node is starting up

During startup before the `vertica.log` has been initialized to write messages, each node in the cluster writes messages into a file named `dbLog`. This log is useful to diagnose situations where database fails to start before it can write messages into `vertica.log`. The `dblog` can be found at the following path, using `catalog-path` and `database-name` as described above:

```
catalog-path/database-name/dbLog
```

See Also

Rotating the Log Files (page 197)

Rotating the Log Files

The `logrotate` utility, which is included with most Linux distributions, helps simplify log file administration on systems that generate many log files. `Logrotate` allows for automatic rotation, compression, removal, and mailing of log files and can be configured to perform these tasks at specific intervals or when the log file reaches a particular size.

If logrotate is present when Vertica is installed (which is typical for most Linux distributions), then Vertica automatically sets logrotate to look for configuration files in the `/opt/vertica/config/logrotate` directory. The utility also creates the file `vertica` in the `/etc/logrotate.d/` directory, which includes the line:

```
include /opt/vertica/config/logrotate
```

If logrotate is not present but installed at a later time, either reinstall the Vertica RPM on every node in the cluster or add a file in the `/etc/logrotate.d/` directory that instructs logrotate to include the logrotate directory contents. For example:

- 1 Create the file `/etc/logrotate.d/vertica`.

- 2 Add the following line:

```
include /opt/vertica/config/logrotate
```

When a database is created, Vertica creates database-specific logrotate configurations which are used by the logrotate utility. For example, a file `/opt/vertica/config/logrotate/<dbname>` is created for each individual database.

Using Administration Tools Logrotate Utility

The administration tools provide a logrotate option to help configure logrotate scripts for a database and to distribute it across the cluster. Only a few basic options are supported - how often to rotate logs, how large the log can get before rotation and how long to keep the logs. For other options, you can manually create logrotate scripts as described later in this topic.

Example:

The following example sets up log rotation on a weekly schedule and keeps for 3 months (12 logs).

```
$ admintools -t logrotate -d <dbname> -r weekly -k 12
```

See **Writing Administration Tools Scripts** (page 351) for full usage description.

Manually Rotating Logs

To perform manual log rotation, use the following procedure to implement a custom log rotation process. No log messages are lost during the procedure.

- 1 Rename or archive the `vertica.log` file that is produced. For example:

```
$ mv vertica.log vertica.log.1
```

- 2 Send the Vertica process the USR1 signal. For example:

```
$ killall -USR1 vertica
```

or

```
$ ps -ef | grep -i vertica
```

```
$ kill -USR1 process-id
```

Manually Creating Logrotate Scripts

If your needs are not met by the administration tools logrotate utility, you may create your own scripts. The following script is an example:

```
/mydb/site01_catalog/vertica.log {
```

```

# rotate weekly
weekly
# and keep for 52 weeks
rotate 52
# no complaining if vertica did not start yet
missingok
# compress log after rotation
compress
# no creating a new empty log, vertica will do that
ncreate
# if set, only rotates when log size is greater than X
size 10M
# delete files after 90 days (not all logrotate pkgs support this keyword)
# maxage 90
# signal vertica to reopen and create the log
postrotate
    kill -USR1 `head -1 /mydb/site01_catalog/vertica.pid 2> /dev/null` 2>
/dev/null || true
endscript
}

```

The following script is an example of the typical default setting for the dbLog file:

```

/mydb/dbLog {
# rotate weekly
weekly
# and keep for 52 weeks
rotate 52
# no complaining if vertica did not start yet
missingok
# compress log after rotation
compress
# this log is stdout, so rotate by copying it aside and truncating
copytruncate
}

```

For details about additional settings, issue the `man logrotate` command.

See Also

Monitoring the Log Files (page 197)

Using the SQL Monitoring API

Vertica provides an API (application programming interface) for monitoring various features and functions within a database in the form of system tables. You can write queries against system tables with full `SELECT` support the same way you perform query operations on base and temporary tables. Queries against system tables may use expressions, predicates, aggregates, analytics, subqueries, joins, and historical query syntax. It is also possible to save the results of a system table query into a user table for future analysis using, for example, `INSERT INTO <user_table> SELECT * FROM <system_table>;`

System tables are grouped into the following schemas:

- `V_CATALOG` – information about persistent objects in the catalog

- `V_MONITOR` – information about transient system state

These schemas reside in the default search path so there is no need to specify `schema.table` in your queries unless you **change the search path** (page 44) to exclude `V_MONITOR` or `V_CATALOG` or both.

Notes and Restrictions

- You can use external monitoring tools or scripts to query the system tables and act upon the information, as necessary. For example, when a host failure causes the K-safety level to fall below a desired level, the tool or script can notify the database administrator and/or appropriate IT personnel of the change, typically in the form of an e-mail.

Note: When a cluster is a recovering state, the database refuses connection requests and cannot be monitored using the SQL monitoring API.

- To view all of the system tables issue the following command:
- DDL and DML operations are not supported on system tables.
- With the exception of the `PROJECTION_REFRESHES` table, system tables do not hold historical data.
- Vertica reserves some memory to help monitor busy systems. Using simple system table queries makes it easier to troubleshoot issues. See also `sysquery` and `sysdata` pools under Built-in pools topic in SQL Reference Manual.
- In `V_CATALOG.TABLES`, columns `TABLE_SCHEMA` and `TABLE_NAME` are case sensitive when equality (`=`) predicates are used in queries. For example, given the following schema:

```
=> CREATE SCHEMA SS;
=> CREATE TABLE SS.TT (c1 int);
=> INSERT INTO ss.tt VALUES (1);
```

If you run a query using the `=` predicate, Vertica returns 0 rows:

```
=> SELECT table_schema, table_name FROM v_catalog.tables WHERE
       table_schema = 'ss';
 table_schema | table_name
-----+-----
(0 rows)
```

Use the case-insensitive `ILIKE` predicate to return the expected results:

```
=> SELECT table_schema, table_name FROM v_catalog.tables WHERE
       table_schema ILIKE 'ss';
 table_schema | table_name
-----+-----
SS           | TT
```

(1 row)

List of System Tables

The system tables that make up the monitoring API are listed here for convenience and are described fully in the SQL Reference Manual. You may also use the following command to view all the system tables and their schema:

```
=> SELECT * FROM system_tables;
```

Monitor Tables	Description	Schema
ACTIVE_EVENTS	Displays all the active events in the cluster.	V_MONITOR
COLUMN_STORAGE	Returns the amount of disk storage used by each column of each projection on each node.	V_MONITOR
COLUMNS	Provides information about columns.	V_CATALOG
CONFIGURATION_PARAMETERS	Provides information about configuration parameters currently in use by the system.	V_MONITOR
CURRENT_SESSION	Returns information about the current active session.	V_MONITOR
DELETE_VECTORS	Holds information on deleted rows to speed up the delete process.	V_MONITOR
DISK_RESOURCE_REJECTIONS	Returns requests for resources that are rejected due to disk space shortages.	V_MONITOR
DISK_STORAGE	Returns the amount of disk storage used by the database on each node.	V_MONITOR
DUAL	A single-column "dummy" table with one record whose value is X.	V_CATALOG
EVENT_CONFIGURATIONS	Returns configuration information about current events.	V_MONITOR
EXECUTION_ENGINE_PROFILES	Returns information regarding query execution runs.	V_MONITOR
FOREIGN_KEYS	Provides foreign key information.	V_CATALOG
GRANTS	Provides grant information.	V_CATALOG
HOST_RESOURCES	Returns information about host profiling.	V_MONITOR
LOAD_STREAMS	Returns load metrics for each load stream on each node.	V_MONITOR
LOCKS	Monitors lock grants and requests for all nodes.	V_MONITOR
NODE_RESOURCES	Provides a snapshot of the node. This is useful for regularly polling the node with automated tools or scripts.	V_MONITOR
PARTITIONS	Displays partition metadata, one row per partition key, per ROS container.	V_MONITOR
PASSWORDS	Contains password information.	V_CATALOG
PRIMARY_KEYS	Provides primary key information.	V_CATALOG
PROFILE_PARAMETERS	Defines what user profiles contain.	V_CATALOG
PROFILES	Provides user profile information.	V_CATALOG
PROJECTION_COLUMNS	Provides projection column information.	V_CATALOG

PROJECTION_REFRESHES	Returns information about refresh operations for projections.	V_MONITOR
PROJECTION_STORAGE	Returns the amount of disk storage used by each projection on each node.	V_MONITOR
PROJECTIONS	Provides information about projections.	V_CATALOG
QUERY_METRICS	Monitors the sessions and queries executing on each node.	V_MONITOR
QUERY_PROFILES	Provides information regarding queries that have run.	V_MONITOR
RESOURCE_ACQUISITIONS	Provides details of resources (memory, open file handles, threads) acquired by each running request for each resource pool in the system.	V_MONITOR
RESOURCE_ACQUISITIONS_HISTORY	Provides details of resources (memory, open file handles, threads) acquired by any profiled query for each resource pool in the system.	V_MONITOR
RESOURCE_POOL_STATUS	Provides resource pool usage information.	V_MONITOR
RESOURCE_POOLS	Provides configuration of resource pools, both user-defined and built-in.	V_CATALOG
RESOURCE_QUEUES	Provides information about queries waiting for resources	V_MONITOR
RESOURCE_REJECTIONS	Returns requests for resources that are rejected by the resource manager.	V_MONITOR
RESOURCE_USAGE	Returns system resource management on each node.	V_MONITOR
SESSION_PROFILES	Provides basic session parameters and lock time out data.	V_MONITOR
SESSIONS	Monitors external sessions.	V_MONITOR
STORAGE_CONTAINERS	Monitors information about each storage container in the database.	V_MONITOR
STRATA	Provides information of strata used in Tuple Mover, one row per stratum. (Vertica Internal use only)	V_MONITOR
STRATA_STRUCTURES	Provides information of strata structures used in Tuple Mover, one row per strata structure. (Vertica Internal use only)	V_MONITOR
SYSTEM	Monitors the overall state of the database.	V_MONITOR
SYSTEM_TABLES	Displays a list of all system table names.	V_CATALOG
TABLE_CONSTRAINTS	Provides information about table constraints.	V_CATALOG
TABLES	Provides information about all tables in the database.	V_CATALOG
TUPLE_MOVER_OPERATIONS	Monitors the status of the Tuple Mover on each node.	V_MONITOR

TYPES	Provides information about supported data types.	V_CATALOG
USER_FUNCTIONS	Returns metadata about user-defined SQL Macros, which store commonly used SQL expressions in a function.	V_CATALOG
USER_PROCEDURES	Provides information about external procedures that have been defined for Vertica	V_CATALOG
USERS	Provides information about users.	V_CATALOG
VIEW_COLUMNS	Provides view attribute information.	V_CATALOG
VIEWS	Provides information about all views within the system.	V_CATALOG
WOS_CONTAINER_STORAGE	Monitors information about WOS storage, which is divided into regions.	V_MONITOR

Examples

The following query uses the vmart schema to obtain the number of rows and size occupied by each table in the database.

```
=> SELECT t.table_name AS table_name,
        SUM(ps.wos_row_count + ps.ros_row_count) AS row_count,
        SUM(ps.wos_used_bytes + ps.ros_used_bytes) AS byte_count
FROM tables t
JOIN projections p ON t.table_id = p.anchor_table_id
JOIN projection_storage ps on p.projection_name = ps.projection_name
WHERE (ps.wos_used_bytes + ps.ros_used_bytes) > 500000
GROUP BY t.table_name
ORDER BY byte_count DESC;
  table_name      | row_count | byte_count
-----+-----+-----
online_sales_fact |    200000 | 11920371
store_sales_fact  |    200000 |  7621694
product_dimension |    240000 |  7367560
customer_dimension |    200000 |  6981564
store_orders_fact |    200000 |  5126330
(5 rows)
```

The rest of the examples show simple ways to use system tables in queries.

```
=> SELECT table_name FROM columns WHERE data_type ILIKE 'Numeric' GROUP BY
table_name;
  table_name
-----
n1
(1 row)
```

```
=> SELECT current_epoch, designed_fault_tolerance, current_fault_tolerance FROM
SYSTEM;
  current_epoch | designed_fault_tolerance | current_fault_tolerance
-----+-----+-----
          492 |                1 |                1
(1 row)
```

```
=> SELECT node_name, total_user_session_count, executed_query_count FROM
query_metrics;
```

node_name	total_user_session_count	executed_query_count
node01	53	42
node02	53	0
node03	42	120
node04	53	0

(4 rows)

```
=> AT EPOCH LATEST SELECT table_schema FROM primary_keys;
table_schema
```

```
-----
public
public
public
public
public
public
public
public
public
public
store
online_sales
online_sales
```

(12 rows)

Configuring PROJECTION_REFRESHES History

Information about a refresh operation—whether successful or unsuccessful—is maintained in the PROJECTION_REFRESHES system table until either the CLEAR_PROJECTION_REFRESHES() function is executed or the storage quota for the table is exceeded. The PROJECTION_REFRESHES.IS_EXECUTING column returns a boolean value that indicates whether the refresh is currently running (t) or occurred in the past (f).

To immediately purge this information, use the CLEAR_PROJECTION_REFRESHES() function:

```
=> SELECT clear_projection_refreshes();
clear_projection_refreshes
```

```
-----
CLEAR
(1 row)
```

Note: Only the rows where the PROJECTION_REFRESHES.IS_EXECUTING column equals *false* are cleared.

See Also

CLEAR_PROJECTION_REFRESHES and PROJECTION_REFRESHES in the SQL Reference Manual

Querying Case-sensitive data in System Tables

Some system table data may be stored in mixed case. For instance, Vertica stores mixed-case identifier names as they were specified in the CREATE statement, even though the case is ignored when they are referenced in queries. See Identifiers. When these object names appear as data in the system tables, it is error prone to retrieve them with the equality (=) predicate because the case must match exactly to what is stored. It is much easier to use the case-insensitive operator ILIKE instead.

Example:

Given the following schema:

```
=> CREATE SCHEMA SS;
=> CREATE TABLE SS.TT (c1 int);
=> CREATE PROJECTION SS.TTP1 AS SELECT * FROM ss.tt UNSEGMENTED ALL NODES;
=> INSERT INTO ss.tt VALUES (1);
```

If you run a query using the = predicate, Vertica returns 0 rows:

```
=> SELECT table_schema, table_name FROM v_catalog.tables WHERE table_schema = 'ss';
table_schema | table_name
-----+-----
(0 rows)
```

Using the case-insensitive ILIKE predicate returns the expected results:

```
=> SELECT table_schema, table_name FROM v_catalog.tables WHERE table_schema ILIKE
'ss';
table_schema | table_name
-----+-----
SS           | TT
(1 row)
```

Monitoring Processes

You can use ps to monitor the database and Spread processes running on each node in the cluster. For example:

```
$ ps aux | grep /opt/vertica/bin/vertica
$ ps aux | grep /opt/vertica/sbin/spread
```

You should see exactly one Vertica process and exactly one Spread process on each node. To monitor Administration Tools and connector processes:

```
$ ps aux | grep vertica
```

There could be many connection processes but, at most, one Administration Tools process.

Monitoring Events

To help you monitor your database system, Vertica traps and logs significant events that impact database performance and functionality if you do not address their root causes. This section describes where events are logged, the types of events that Vertica logs, how to respond to these events, the information that Vertica provides for these events, and how to configure event monitoring.

Event Logging Mechanisms

Vertica posts events to the following:

Mechanism	Description
vertica.log	All events are automatically posted to <code>vertica.log</code> . See Monitoring the Log Files (page 197).
ACTIVE_EVENTS	This SQL system table provides information about all open events. See Using the SQL Monitoring API (page 199) and ACTIVE_EVENTS.
SNMP	To post traps to SNMP, enable global reporting in addition to each individual event you want trapped. See Configuring Event Reporting (page 210).
Syslog	To log events to syslog, enable event reporting for each individual event you want logged. See Configuring Event Reporting (page 210).

Event Types

Event names are sensitive to case and spaces. Vertica logs the following events:

Event Name	Event Type	Description	Action
Low Disk Space	0	The database is running out of disk space or a disk is failing or there is a I/O hardware failure.	It is imperative that you add more disk space or replace the failing disk or hardware as soon as possible. Check <code>dmesg</code> to see what caused the problem. Also, use the DISK_RESOURCE_REJECTIONS system table to determine the types of disk space requests that are being rejected and the hosts on which they are being rejected. See Managing Disk Space (page 283) within the Database Administrator's Guide for more information about low disk space.
Read Only File System	1	The database does not have write access to the file system for the data or catalog paths. This can sometimes occur if Linux remounts a drive due to a kernel issue.	Modify the privileges on the file system to give the database write access.
Loss Of K Safety	2	The database is no longer K-Safe because there are insufficient nodes functioning within the cluster. Loss of K-Safety causes the database	If a system shuts down due to loss of K-Safety, you need to recover the system. See Failure Recovery (page 235) in the Troubleshooting Guide.

		to shut down. In a four-node cluster, for example, K-Safety=1. If one node fails, the fault tolerance is at a critical level. If two nodes fail, the system loses K-Safety.	
Current Fault Tolerance at Critical Level	3	One or more nodes in the cluster have failed. If the database loses one more node, it is no longer K-Safe and it shuts down. (For example, a four-node cluster is no longer K-safe if two nodes fail.)	Restore any nodes that have failed or been shut down.
Too Many ROS Containers	4	Due to heavy data load conditions, there are too many ROS containers. This occurs when the Tuple Mover falls behind in performing mergeout operations. The resulting excess number of ROS containers can exhaust all available system resources. To prevent this, Vertica automatically rolls back all transactions that would load data until the Tuple Mover has time to catch up.	See Too many ROS containers in the Troubleshooting Guide for more information about what causes excess ROS containers and how to process them.
WOS Over Flow	5	The WOS cannot hold all the data that you are attempting to load. This means that the copy fails and the transaction rolls back. Note: This event does not occur in Vertica 4.0.	Consider loading the data to disk (ROS) instead of memory (WOS) or splitting the fact table load file into multiple pieces and then performing multiple loads in sequence. You might also consider making the Tuple Mover's moveout operation more aggressive. See Tuning the Tuple Mover (page 167) in Administrator's Guide.
Node State Change	6	The node state has changed.	Check the status of the node.
Recovery Failure	7	The database was not restored to a functional state after a hardware or software related failure.	The reason for recovery failure can vary. See the event description for more information about your specific situation.
Recovery Error	8	The database encountered an error while attempting to recover. If the number of recovery errors exceeds Max Tries, the Recovery Failure event is triggered. See Recovery Failure within this	The reason for a recovery error can vary. See the event description for more information about your specific situation.

		table.	
Recovery Lock Error	9	A recovering node could not obtain an S lock on the table. If you have a continuous stream of COPY commands in progress, recovery might not be able to obtain this lock even after multiple re-tries.	Either momentarily stop the loads or pick a time when the cluster is not busy to restart the node and let recovery proceed.
Recovery Projection Retrieval Error	10	Vertica was unable to retrieve information about a projection.	The reason for a recovery projection retrieval error can vary. See the event description for more information about your specific situation.
Refresh Error	11	The database encountered an error while attempting to refresh.	The reason for a refresh error can vary. See the event description for more information about your specific situation.
Refresh Lock Error	12	The database encountered a locking error during refresh.	The reason for a refresh error can vary. See the event description for more information about your specific situation.
Tuple Mover Error	13	The database encountered an error while attempting to move the contents of the Write Optimized Store (WOS) into the Read Optimized Store (ROS).	The reason for a Tuple Mover error can vary. See the event description for more information about your specific situation.
Timer Service Task Error	14	An error occurred in an internal scheduled task.	Internal use only
Stale Checkpoint	15	Data in the WOS has not been completely moved out in a timely manner. An UNSAFE shutdown could require reloading a significant amount of data.	Be sure that Moveout operations are executing successfully. Check the <code>vertica.log</code> files for errors related to Moveout. Contact Technical Support (on page 1) for assistance.

Event Data

To help you interpret and solve the issue that triggered an event, each event provides a variety of data depending upon the event logging mechanism used. The following table describes the event data and indicates where it is used.

<u>vertica.log</u>	<u>ACTIVE_EVENTS</u> (column names)	<u>SNMP</u>	<u>Syslog</u>	<u>Description</u>
N/A	NODE_NAME	N/A	N/A	The node where the event occurred.
Event Code	EVENT_CODE	Event Type	Event Code	A numeric ID that indicates the type of event. See Event Types in the previous table for a list of event type

				codes.
Event Id	EVENT_ID	Event OID	Event Id	A unique numeric ID that identifies the specific event.
Event Severity	EVENT_SEVERITY	Event Severity	Event Severity	The severity of the event from highest to lowest. These events are based on standard syslog severity types: 0 - Emergency 1 - Alert 2 - Critical 3 - Error 4 - Warning 5 - Notice 6 - Info 7 - Debug
PostedTimestamp	EVENT_POSTED_TIMESTAMP	N/A	PostedTimestamp	The year, month, day, and time the event was reported. Time is provided as military time.
ExpirationTimestamp	EVENT_EXPIRATION	N/A	ExpirationTimestamp	The time at which this event expires. If the same event is posted again prior to its expiration time, this field gets updated to a new expiration time.
EventCodeDescription	EVENT_CODE_DESCRIPTION	Description	EventCodeDescription	A brief description of the event and details pertinent to the specific situation.
ProblemDescription	EVENT_PROBLEM_DESCRIPTION	Event Short Description	ProblemDescription	A generic description of the event.
N/A	REPORTING_NODE	Node Name	N/A	The name of the node within the cluster that reported the event.
DatabaseName	N/A	Database Name	DatabaseName	The name of the database that is impacted by the event.
N/A	N/A	Host Name	Hostname	The name of the host within the cluster that reported the event.
N/A	N/A	Event Status	N/A	The status of the event. It can be either: 1 - Open 2 - Clear

Configuring Event Reporting

Event reporting is automatically configured for `vertica.log`, and current events are automatically posted to the `ACTIVE_EVENTS` system table. You can also configure Vertica to post events to **syslog** (page 210) and **SNMP** (page 212).

Configuring Reporting for syslog

Syslog is a network-logging utility that issues, stores, and processes meaningful log messages. It is designed so DBAs can keep machines up and running, and it is a useful way to get heterogeneous data into a single data repository.

To log events to syslog, enable event reporting for each individual event you want logged. Messages are logged, by default, in `/var/log/messages`.

Configuring event reporting to syslog consists of:

- 1 Enabling Vertica to trap events for syslog.
- 2 Defining which events Vertica traps for syslog.
Vertica strongly suggests that you trap the Stale Checkpoint event.
- 3 Defining which syslog facility to use.

Enabling Vertica to Trap Events for syslog

To enable event trapping for syslog, issue the following SQL command:

```
=> SELECT SET_CONFIG_PARAMETER('SyslogEnabled', 1 );
      SET_CONFIG_PARAMETER
-----
Parameter set successfully
(1 row)
```

To disable event trapping for syslog, issue the following SQL command:

```
=> SELECT SET_CONFIG_PARAMETER('SyslogEnabled', 0 );
      SET_CONFIG_PARAMETER
-----
Parameter set successfully
(1 row)
```

Defining Events to Trap for Syslog

To define events that generate a syslog entry, issue the following SQL command:

```
=> SELECT SET_CONFIG_PARAMETER('SyslogEvents', 'Event_Name' , 'Event_Name');
      SET_CONFIG_PARAMETER
-----
Parameter set successfully
(1 row)
```

Where `Event_Name` is one of the following events:

- Low Disk Space

- Read Only File System
- Loss Of K Safety
- Current Fault Tolerance at Critical Level
- Too Many ROS Containers
- WOS Over Flow
- Node State Change
- Recovery Failure
- Recovery Error
- Recovery Lock Error
- Recovery Projection Retrieval Error
- Refresh Error
- Refresh Lock Error
- Tuple Mover Error
- Timer Service Task Error

Stale Checkpoint The following example generates a syslog entry for low disk space and recovery failure:

```
=> SELECT SET_CONFIG_PARAMETER('SyslogEvents', 'Low Disk Space, Recovery
Failure');
      SET_CONFIG_PARAMETER
-----
Parameter set successfully
(1 row)
```

Defining the SyslogFacility to Use for Reporting

The syslog mechanism allows for several different general classifications of logging messages, called facilities. Typically, all authentication-related messages are logged with the `auth` (or `authpriv`) facility. These messages are intended to be secure and hidden from unauthorized eyes. Normal operational messages are logged with the `daemon` facility, which is the collector that receives and optionally stores messages.

The `SyslogFacility` directive allows all logging messages to be directed to a different facility than the default. When the directive is used, *all* logging is done using the specified facility, both authentication (secure) and otherwise.

To define which SyslogFacility Vertica uses, issue the following SQL command:

```
=> SELECT SET_CONFIG_PARAMETER('SyslogFacility' , 'Facility_Name');
```

Where the facility-level argument `<Facility_Name>` is one of the following:

- | | |
|---|--------------------------------------|
| ▪ <code>auth</code> | ▪ <code>uucp</code> (UUCP subsystem) |
| ▪ <code>authpriv</code> (Linux only) | ▪ <code>local0</code> (local use 0) |
| ▪ <code>cron</code> | ▪ <code>local1</code> (local use 1) |
| ▪ <code>daemon</code> | ▪ <code>local2</code> (local use 2) |
| ▪ <code>ftp</code> (Linux only) | ▪ <code>local3</code> (local use 3) |
| ▪ <code>lpr</code> (line printer subsystem) | ▪ <code>local4</code> (local use 4) |

- mail (mail system)
- news (network news subsystem)
- user (default system)
- local5 (local use 5)
- local6 (local use 6)
- local7 (local use 7)

See Also

Event Reporting Examples (page 214) and **Configuration Parameters** (page 25) in the Administrator's Guide

Configuring Reporting for SNMP

Configuring Event Reporting for SNMP consists of:

- 1 Configuring Vertica to enable event trapping for SNMP as described below.
- 2 Importing the Vertica MIB file into the SNMP monitoring device.

The Vertica MIB file allows the SNMP trap receiver to understand the traps it receives from Vertica. This, in turn, allows you to configure the actions it takes when it receives traps.

Vertica supports the SNMP V1 trap protocol, and it is located in `/opt/vertica/sbin/VERTICA-MIB`. See the documentation for your SNMP monitoring device for more information about importing MIB files.

- 3 Configuring the SNMP trap receiver to handle traps from Vertica.

SNMP trap receiver configuration differs greatly from vendor to vendor. As such, the directions presented here for configuring the SNMP trap receiver to handle traps from Vertica are generic.

Vertica traps are single, generic traps that contain several fields of identifying information. These fields equate to the event data described in **Monitoring Events** (page 205). However, the format used for the field names differs slightly. Under SNMP, the field names contain no spaces. Also, field names are pre-pended with "vert". For example, Event Severity becomes vertEventSeverity.

When configuring your trap receiver, be sure to use the same hostname, port, and community string you used to configure event trapping in Vertica.

Examples of network management providers:

- HP Software Network Node Manager; openview.hp.com/products/nnm/index.html
- IBM Tivoli
- AdventNet
- Net-SNMP (Open Source)
- Nagios (Open Source)
- Open NMS (Open Source)

Configuring Event Trapping in Vertica

Configuring Vertica to trap events for SNMP consists of:

- 1 Enabling Vertica to trap events.
- 2 Defining where Vertica sends traps.

- 3** Defining which events Vertica traps if you do not want all the events trapped (the default). Vertica strongly suggests that you trap the Stale Checkpoint event even if you decide to reduce the number events Vertica traps.

Note: Once you have performed steps 1 and 2 to enable event trapping and identify the location where traps are sent, Vertica automatically traps the following events: Low Disk Space, Read Only File System, Loss of K Safety, Current Fault Tolerance at Critical Level, Too Many ROS Containers, WOS Over Flow, Node State Change, Recovery Failure, and Stale Checkpoint. Perform step 3 only if you want to redefine the events Vertica traps.

To enable event trapping for SNMP, use the following SQL command:

```
SELECT SET_CONFIG_PARAMETER('SnmpTrapsEnabled', 1 );
```

To define where Vertica send traps, use the following SQL command:

```
SELECT SET_CONFIG_PARAMETER('SnmpTrapDestinationsList', 'host_name port
CommunityString' );
```

Where host_name and port identify the computer where SNMP resides, and CommunityString acts like a password to control Vertica's access to the server.

For example:

```
SELECT SET_CONFIG_PARAMETER('SnmpTrapDestinationsList', 'localhost 162 public'
);
```

To define which events Vertica traps, use the following SQL command:

```
SELECT SET_CONFIG_PARAMETER('SnmpTrapEvents', 'Event_Name, Event_Name');
```

Where Event_Name is one of the following events:

- Low Disk Space
- Read Only File System
- Loss Of K Safety
- Current Fault Tolerance at Critical Level
- Too Many ROS Containers
- WOS Over Flow
- Node State Change
- Recovery Failure
- Recovery Error
- Recovery Lock Error
- Recovery Projection Retrieval Error
- Refresh Error
- Tuple Mover Error
- Stale Checkpoint

Note: These values are case sensitive.

The following is an example that uses two different event names:

```
$ SELECT SET_CONFIG_PARAMETER('SnmpTrapEvents', 'Low Disk Space, Recovery
Failure');
```

Verifying SNMP Configuration

To create a set of test events that checks SNMP configuration:

- 1 Set up SNMP trap handlers to catch Vertica events.
- 2 Test your setup with the following command:

```
SELECT SNMP_TRAP_TEST();
       SNMP_TRAP_TEST
-----
Completed SNMP Trap Test
(1 row)
```

See Also

Configuration Parameters (page 25) in the Administrator's Guide

Event Reporting Examples

Vertica.log

The following example illustrates a Too Many ROS Containers event posted and cleared within vertica.log:

```
08/14/08 15:07:59 thr:nameless:0x45a08940 [INFO] Event Posted:
Event Code:4 Event Id:0 Event Severity: Warning [4] PostedTimestamp:
2008-08-14 15:07:59.253729 ExpirationTimestamp: 2008-08-14 15:08:29.253729
EventCodeDescription: Too Many ROS Containers ProblemDescription:
Too many ROS containers exist on this node. DatabaseName: QATESTDB
Hostname: fc6-1.verticacorp.com
08/14/08 15:08:54 thr:Ageout Events:0x2aaab0015e70 [INFO] Event Cleared:
Event Code:4 Event Id:0 Event Severity: Warning [4] PostedTimestamp:
2008-08-14 15:07:59.253729 ExpirationTimestamp: 2008-08-14 15:08:53.012669
EventCodeDescription: Too Many ROS Containers ProblemDescription:
Too many ROS containers exist on this node. DatabaseName: QATESTDB
Hostname: fc6-1.verticacorp.com
```

SNMP

The following example illustrates a Too Many ROS Containers event posted to SNMP:

```
Version: 1, type: TRAPREQUEST
Enterprise OID: .1.3.6.1.4.1.31207.2.0.1
Trap agent: 72.0.0.0
Generic trap: ENTERPRISESPECIFIC (6)
Specific trap: 0
.1.3.6.1.4.1.31207.1.1 ---> 4
.1.3.6.1.4.1.31207.1.2 ---> 0
.1.3.6.1.4.1.31207.1.3 ---> 2008-08-14 11:30:26.121292
.1.3.6.1.4.1.31207.1.4 ---> 4
.1.3.6.1.4.1.31207.1.5 ---> 1
.1.3.6.1.4.1.31207.1.6 ---> site01
.1.3.6.1.4.1.31207.1.7 ---> suse10-1
.1.3.6.1.4.1.31207.1.8 ---> Too many ROS containers exist on this node.
.1.3.6.1.4.1.31207.1.9 ---> QATESTDB
```

```
.1.3.6.1.4.1.31207.1.10 ---> Too Many ROS Containers
```

Syslog

The following example illustrates a Too Many ROS Containers event posted and cleared within syslog:

```
Aug 14 15:07:59 fc6-1 vertica: Event Posted: Event Code:4 Event Id:0 Event Severity:
Warning [4] PostedTimestamp: 2008-08-14 15:07:59.253729 ExpirationTimestamp:
2008-08-14 15:08:29.253729 EventCodeDescription: Too Many ROS Containers
ProblemDescription:
Too many ROS containers exist on this node. DatabaseName: QATESTDB Hostname:
fc6-1.verticacorp.com
Aug 14 15:08:54 fc6-1 vertica: Event Cleared: Event Code:4 Event Id:0 Event
Severity:
Warning [4] PostedTimestamp: 2008-08-14 15:07:59.253729 ExpirationTimestamp:
2008-08-14 15:08:53.012669 EventCodeDescription: Too Many ROS Containers
ProblemDescription:
Too many ROS containers exist on this node. DatabaseName: QATESTDB Hostname:
fc6-1.verticacorp.com
```

Monitoring Linux Resource Usage

It is recommended to monitor system resource usage on any or all nodes in the cluster.

Note:

Vertica recommends that you install `pstack` and `sysstat` to help monitor Linux resources.

The `SYSSTAT` package contains utilities for monitoring system performance and usage activity, such as `sar`, as well as tools you can schedule via `cron` to collect performance and activity data. See the ***SYSSTAT Web page***

<http://pagesperso-orange.fr/sebastien.godard/> for details.

The `pstack` utility lets you print a stack trace of a running process. See the ***PSTACK man page*** ***<http://linux.die.net/man/1/pstack>*** for details.

- 1 Log in to the database administrator account on any node.

- 2 Run the `top` utility

```
$ top
```

A high CPU percentage in `top` indicates that Vertica is CPU-bound. For example:

```
top - 16:08:52 up 7 days, 4:52, 9 users, load average: 0.91, 0.97, 0.81
Tasks: 123 total, 1 running, 122 sleeping, 0 stopped, 0 zombie
Cpu(s): 26.9% us, 1.3% sy, 0.0% ni, 71.8% id, 0.0% wa, 0.0% hi, 0.0% si
Mem: 4053136k total, 3882020k used, 171116k free, 407688k buffers
Swap: 4192956k total, 176k used, 4192780k free, 1526436k cached
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
8785	dbadmin	1	0	1374m	678m	55m	S	99.9	17.1	80:41.14	vertica
3557	root	16	0	32152	11m	2508	S	1.0	0.3	53:30.44	X
1	root	16	0	4748	552	456	S	0.0	0.0	0:00.75	init
2	root	RT	0	0	0	0	S	0.0	0.0	0:00.14	migration/0
3	root	34	19	0	0	0	S	0.0	0.0	0:00.04	ksoftirqd/0
4	root	RT	0	0	0	0	S	0.0	0.0	0:00.10	migration/1
5	root	34	19	0	0	0	S	0.0	0.0	0:00.03	ksoftirqd/1
6	root	RT	0	0	0	0	S	0.0	0.0	0:00.09	migration/2
7	root	34	19	0	0	0	S	0.0	0.0	0:00.03	ksoftirqd/2

Some possible reasons for high CPU usage are:

- The Tuple Mover runs automatically and thus consumes CPU time even if there are no connections to the database. If you believe this to be a problem, contact **Technical Support** (on page 1).
- The `pdflush` process (a set of worker threads for writing back dirty filesystem data) is consuming a great deal of cpu time, possibly driving the load way up. Adding RAM appears to make the problem worse. Log in to root and change the Linux parameter `swappiness` to 0.

```
# echo 0 > /proc/sys/vm/swappiness
```

- Some information sources:

TechRepublic

<http://techrepublic.com.com/5206-6230-0.html?forumID=36&threadID=175191&start=0>

Red Hat https://bugzilla.redhat.com/bugzilla/show_bug.cgi?id=150653

Indiana University Unix Systems Support Group

<http://www.ussg.iu.edu/hypermail/linux/kernel/0404.3/0744.html>

- 3 Run the `iostat` utility. A high idle time in `top` at the same time as a high rate of blocks read in `iostat` indicates that Vertica is disk-bound. For example:

```
$ /usr/bin/iostat
```

```
Linux 2.6.9-22.ELsmp (qa0) 07/13/2007
```

```
avg-cpu:  %user   %nice   %sys %iowait   %idle
           0.83    0.00    0.13    0.02   99.03
```

```
Device:            tps    Blk_read/s    Blk_wrtn/s    Blk_read
   Blk_wrtn
hda                 0.37           3.40           10.37       2117723
 6464640
sda                 0.46           1.94           18.96       1208130
11816472
sdb                 0.26           1.79           15.69       1114792
9781840
```

sdc	0.24	1.80	16.06	1119304
10010328				
sdd	0.22	1.79	15.52	1117472
9676200				
md0	8.37	7.31	66.23	4554834
41284840				

Monitoring Vertica Using Ganglia

The Vertica® Analytic Database is integrated with Ganglia, a web-based administration console and monitoring tool that lets you observe the status of a Vertica cluster and its running databases from your client's browser.

Ganglia Architecture

Ganglia architecture comprises the components listed in this section.

gmond

The **G**anglia **MON**itor **D**aemon is a data-collecting agent that must be installed on every node in a cluster. Gmond gathers metrics about the local node and sends information to other nodes via XML to a browser window. Gmond is portable and collects system metrics, such as CPU, memory, disk, network and process data. The Gmond configuration file `/etc/gmond.conf` controls the Gmond daemon and resides on each node where Gmond is installed.

gmetad

The **G**anglia **META**data **D**aemon is a data-consolidating agent that provides a query mechanism for collecting historical information about groups of machines. Gmetad is typically installed on a single, task-oriented server (the monitoring node), though very large clusters could require more than one Gmetad daemon. Gmetad collects data from other Gmetad and Gmond sources and stores their state in indexed RRDtool (round-robin) databases, where a Web interface reads and returns information about the cluster. The Gmetad configuration file `/etc/gmetad.conf` controls the Gmetad daemon and resides on the monitoring node.

RRDtool

RRDTool is an open-source data logging and graphing system that Ganglia uses to store the collected data and to render the graphs for Web-based reports. Cron jobs that run in the background to collect information from Vertica monitoring system tables are stored in the RRD database.

PHP-based Web interface

The PHP-based Web interface comprises a collection of scripts that are used by the Ganglia Web reporting front end and by the Vertica extensions. The Web server starts these scripts, which then collect Vertica-specific metrics from the RRD database and generate the XML graphs. These scripts provide access to Vertica health across the cluster, as well as on each host.

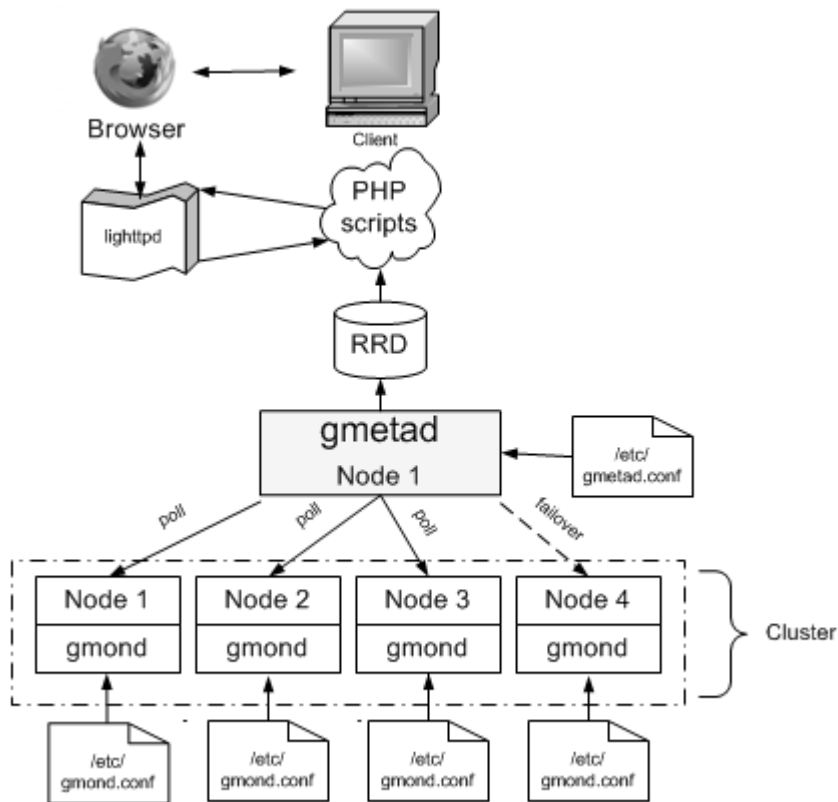
Web server

The Web server uses `lighttpd`, a lightweight http server and can be any Web server that supports PHP, SSL, and XML. The Ganglia web front end displays the data stored by Gmetad in a graphical web interface using PHP.

Advanced tools

Gmetric, an executable, is added during Ganglia installation. Gmetric provides additional statistics and is used to store user-defined metrics, such as numbers or strings with units.

The following diagram illustrates Ganglia architecture on a four-node cluster.



Ganglia Prerequisites

To use the Vertica-Ganglia monitoring package, the following must be installed on the server:

- Vertica® Analytic Database (available as a download on the Vertica **Downloads page** http://www.vertica.com/v-zone/download_vertica)
- Required package dependencies for *all* Linux distributions:
 - php5
 - php5-gd
 - ganglia-gmetad
 - ganglia-gmond
 - rrdtool

Other packages might be required for Ganglia to work properly. These dependencies come preinstalled on most Linux distributions, though some could be missing on older Linux distributions and use php4 instead of php5. See **Required Packages** (page 220).

This guide describes the primary required packages and dependencies, but it cannot account for all possible missing dependencies on all distributions. In the event the packages cannot be installed, the install script fails with an error message. See **Installing the Vertica Monitoring Package** (page 221) for information on obtaining missing dependencies.

What's in the Package

The Vertica Monitoring package contains the following files:

- `install-vertica-ganglia` – Provided as an assistant to the installer for the prerequisites packages. This script attempts to install dependencies from your configured package repositories or from a small set of packages available from `vertica.com`.
- `vertica-ganglia-<version>.<arch>.rpm` – The distribution-specific version of the `vertica-ganglia` monitoring component.

Important Notes:

Before you download the Vertica Monitoring package, make sure the version of the rpm matches the Vertica installation on your server. For example:

- Vertica 4.1.x databases require `vertica-ganglia-4.1.x` (ganglia only package)
- Vertica 4.0.x databases require `vertica-ganglia-4.0.x` (ganglia only package)
- Vertica 3.5.x databases require `vertica-ganglia-3.5.x` (ganglia only package)
- Vertica 3.0.x databases require `vertica-web-3.0.x` (ganglia-webmin package)

Note: See *Upgrading the Vertica Monitoring Package* (page 232) for required upgrade paths, particularly if you are upgrading from the `ganglia-webmin` package to a `ganglia-only` package.

Before you install the Vertica Monitoring Package, you must install `php5`, if it is not installed already. See the *Required Packages* (page 220).

RHEL5 users: A Red Hat subscription is required to access and download dependencies from the Red Hat site. See the *Red Hat Web site* http://www.redhat.com/licenses/rhel_us_3.html for details.

Required Packages

A number of dependencies must be installed before you can install the `vertica-ganglia` package, either from your distribution's package repository or manually.

IMPORTANT!

If the server does not have an Internet connection at the time you install the `vertica-ganglia` package, you must obtain the required dependencies manually, using the package manager for your distribution (`yum` on Red Hat, for example, or `yast` on SUSE) before you proceed. Read ***Servers without Internet Access*** (page 224) before you begin the install process.

Vertica does not provide installation instructions for the individual packages. See their respective Web sites for details. Some links have been provided for your convenience and have been validated at the time of publication. Note that these links could change over time.

RHEL5 users: A Red Hat subscription is required to access and download dependencies from the Red Hat site. See the ***Red Hat Web site*** http://www.redhat.com/licenses/rhel_us_3.html for details.

The following top-level package dependencies are required for *all* Linux distributions and should be installed in the order listed. Additional package dependencies could be required, depending on how the top-level package was built.

- `php5`
- `php5-gd`
- `ganglia-gmetad`
- `ganglia-gmond`
- `rrdtool`

Due to variations in the exact version available for each Linux distribution, specific version numbers are not listed for each package. Vertica recommends that you install the latest version for your distribution. The provided `install-vertica-ganglia` script attempts to install the dependencies automatically, if possible. However due to your specific repository configuration, you might need to install additional packages manually. This guide makes some attempt to help you resolve those missing dependencies but cannot account for every possible scenario on all distributions.

In general, the basic flow for resolving dependencies is to install `php5`, run the `install-vertica-ganglia` script (except on SuSE), install missing dependencies that the script could not resolve, and run the `install-vertica-ganglia` script again.

For distribution-specific instructions, see the following topics:

- ***RHEL5*** (page 222)
- ***SUSE SLE 10 and 11*** (page 223)

Installing the Vertica Monitoring Package

On all Linux distributions, the following is the basic installation path:

- 1 Obtain the required packages/dependencies.
- 2 Install the Vertica Monitoring Package.

- 3 Install gmetad and gmond on the monitoring node.
- 4 Install gmond on all nodes in the Vertica cluster.

IMPORTANT!

To download the Vertica Monitoring Package and readily access the required packages/dependencies, the distribution-specific instructions that follow assume a server with Internet access. If the server does not have an Internet connection, see **Servers without Internet Access** (page 224) and then refer to the instructions for your particular distribution.

Plan to install the Vertica Monitoring package on the same node on which Vertica runs (the monitoring node).

RHEL5

Plan to install the Vertica Monitoring package on the same node on which Vertica runs (the monitoring node).

Installing the Vertica Monitoring package:

- 1 **IMPORTANT:** Before you proceed, read **Ganglia Prerequisites** (page 219) and **Required Packages** (page 220).
- 2 Log in as root or sudo on the target server:

```
# su - root
```
- 3 Download the Vertica Monitoring package (vertica-ganglia-<version>.<arch>.tar.gz) from the Vertica **Downloads page** http://www.vertica.com/v-zone/download_vertica and save the package to a location on the server; for example to /tmp.

Note: Scroll to the bottom of the Downloads page to the section, "Other Software for Use with Vertica Analytic Database 4.1"

- 4 Change directory to the location of the rpm:

```
# cd /tmp
```
- 5 Extract the Vertica Monitoring package:

```
# tar xzvf vertica-ganglia-<version>.<arch>.tar.gz
```

In the above command, substitute the version and architecture variables with file information from the download; for example:

```
# tar xzvf vertica-ganglia-4.1.xx.x86_64.RHEL5.tar.gz
```
- 6 Run the `install-vertica-ganglia` script to aid the installer in finding and installing any missing dependencies.

```
# ./install-vertica-ganglia vertica-ganglia-<version>.<arch>.rpm
```

The above command:

 - Installs the Ganglia Web front end.

- Installs gmetad and gmond on the monitoring node.
- Creates a `/tmp/vertica-web-download` directory on the monitoring node, which contains files you need to perform additional installation and configuration operations, including:
 - `ganglia-gmetad.rpm`, installer package for the data consolidator
 - `ganglia-gmond.rpm`, installer package for the data collector

7 Verify that the packages are installed:

```
# rpm -qa | grep ganglia
```

Output should be:

```
ganglia-gmond
ganglia-gmetad
vertica-ganglia
```

8 If Step 6 found missing dependencies that the script could not resolve, install those packages manually. Refer to **Required Packages** (page 220), and then run the script again:

```
# ./install-vertica-ganglia vertica-ganglia-<version>.<arch>.rpm
```

The following is an example of a failed dependency:

```
error: Failed dependencies:
    ganglia-gmetad is needed by vertica-ganglia-1.1.0-1.noarch
    ganglia-gmond is needed by vertica-ganglia-1.1.0-1.noarch
    php is needed by vertica-ganglia-1.1.0-1.noarch
    php-gd is needed by vertica-ganglia-1.1.0-1.noarch
```

9 Proceed to **Configuring Ganglia** (page 225).

SuSE SLE 10 and SLE 11

Several packages are not readily available for SuSE SLE 10 and 11 distributions. You can obtain them from your distribution's media or from publicly-available repositories. See the searchable list on the **openSUSE Build Service** <http://software.opensuse.org/search> Web page.

Plan to install the Vertica Monitoring package on the same node on which Vertica runs (the monitoring node).

Installing the Vertica Monitoring package:

1 **IMPORTANT:** Before you proceed, read **Ganglia Prerequisites** (page 219) and **Required Packages** (page 220).

2 Log in as root or sudo on the target server:

```
# su - root
```

3 Download the Vertica Monitoring package

(`vertica-ganglia_<version>.<arch>.tar.gz`) from the Vertica **Downloads page** http://www.vertica.com/v-zone/download_vertica and save the package to a location on the server; for example to `/tmp`.

Note: Scroll to the bottom of the Downloads page to the section, "Other Software for Use with Vertica Analytic Database 4.0"

4 Change directory to the location of the rpm:

```
# cd /tmp
```

6 Install the required dependencies in the following order:

- libapr1
- rrdtool
- libmm14-1
- php5
- php5-gd
- php5-fastcgi
- libconfuse0
- libganglia
- ganglia-gmond
- ganglia-gmetad

7 Extract the Vertica Monitoring package:

```
# tar xzvf vertica-ganglia-<version>.<arch>.tar.gz
```

In the above command, substitute the version and architecture variables with file information from the download; for example:

```
# tar xzvf vertica-ganglia-4.0.12.x86_64.RHEL5.tar.gz
```

8 Install the Vertica Monitoring package.

```
# rpm -Uvh vertica-ganglia-<version>.<arch>.tar.gz
```

9 Proceed to **Configuring Ganglia** (page 225).

Servers without Internet Access

IMPORTANT!

This procedure is for servers that *do not* have an Internet connection; thus the required packages must be obtained manually, as described in this topic.

Vertica does not provide installation instructions for the individual packages. See their respective Web sites for details. Links are provided below for your convenience, though they could change between Vertica releases.

RHEL4 and RHEL5 users: A Red Hat subscription is required to access and download dependencies from the Red Hat site. See the **Red Hat Web site** http://www.redhat.com/licenses/rhel_us_3.html for details.

Plan to install the Vertica Monitoring package on the same node on which Vertica runs (the monitoring node).

Installing the Vertica Monitoring package on a server without Internet access:

1 **IMPORTANT:** Before you proceed, read **Ganglia Prerequisites** (page 219) and **Required Packages** (page 220).

- 2 On any workstation with Internet access, download the dependencies for your distribution and transfer them to the target system (the monitoring node).

Note: gmond is required on all nodes, so make note of the directory to which you downloaded the package; for example, /tmp.

- 3 Log in as root or sudo on the target server:

```
# su - root
```

- 4 Change directory to the location of the rpm:

```
# cd /tmp
```

- 5 Using your distributions package management system, install the dependencies in the required order. For example:

- php5
- php5-gd
- ganglia-gmetad
- ganglia-gmond
- rrdtool

- 6 Extract the vertica-ganglia package:

```
# tar xzf vertica-ganglia-<version>.<arch>.tar.gz
```

- 7 Install the vertica-ganglia package:

```
# rpm -Uvh vertica-ganglia-<version>.<arch>.rpm
```

- 8 Verify that the packages are installed:

```
# ./install-vertica-ganglia vertica-ganglia-<version>.<arch>.rpm
```

- 9 If Step 7 found missing dependencies, you must install those packages now. Refer to **Required Packages** (page 220), and then run the script again:

```
# ./install-vertica-ganglia vertica-ganglia-<version>.<arch>.rpm
```

The following is an example of a failed dependency:

```
error: Failed dependencies:
    ganglia-gmetad is needed by vertica-ganglia-1.1.0-1.noarch
    ganglia-gmond is needed by vertica-ganglia-1.1.0-1.noarch
    php is needed by vertica-ganglia-1.1.0-1.noarch
    php-gd is needed by vertica-ganglia-1.1.0-1.noarch
```

- 10 Proceed to **Configuring Ganglia** (page 225).

Configuring Ganglia

In the previous installation procedure, the final step installed gmetad and gmond on the monitoring node.

During configuration, you:

- 1 **Install gmond** (page 226) on *all nodes* in the Vertica cluster
- 2 **Modify the gmetad configuration file** (page 226) on the monitoring node
- 3 **Modify the gmond configuration file** (page 227) on the monitoring node and on *all nodes* in the Vertica cluster.

Installing Gmond on All Nodes

IMPORTANT!

gmond must be installed on every node in the cluster.

- 1 On the monitoring node, change directory to the location of the `ganglia-gmond` package (or to the location where you manually did the install):

```
# cd /tmp/vertica-web-downloads
```

- 2 Copy the `ganglia-gmond` package to all nodes in the Vertica cluster:

```
# scp ./ganglia-gmond.rpm <hostname>:/tmp/ganglia-gmond.rpm
```

- 3 Install gmond on all nodes:

```
# rpm -Uvh ./ganglia-gmond.rpm
```

The configuration file `gmond.conf` is created in `/etc`.

Configuring Gmetad on the Monitoring Node

The role of gmetad is to request summary information from gmond and save it. The saved data is used by the Web interface to produce the graphs, and the behavior of gmetad is controlled by a single configuration file, `/etc/gmetad.conf`.

The following procedure assumes you have already *installed the Vertica Monitoring package* (page 221).

- 1 On the monitoring node, use the text editor of your choice to open the `gmetad.conf` file:

```
# vi /etc/gmetad.conf
```

Note: The path on SuSE is `/etc/ganglia`.

- 2 Scroll or search for the keyword `data_source` and specify at least one data source name to include at least the monitoring node.

The `data_source` keyword specifies the host where `tcp_accept_channel` is defined and its port. The format of the `data_source` line is:

"<data source name>" <host 1> <host 2> .. <host n>. For example:

```
data_source "Vertica_Cluster" 192.168.0.1 192.168.0.2 192.168.0.3 192.168.0.4
```

- Vertica recommends that you use IP addresses instead of host names.
- You can list only a few hosts in the `data_source` setting. Listing numerous hosts does not mean that gmetad polls all of them for data. If gmetad cannot get data from the first host in the list you provide, it tries the next one. The order in which you list hosts does not matter.
- If you do not specify a port number, gmetad assumes the default Ganglia port is 8649.
- The data source name is case sensitive.

- 3 Add gmetad to the list of services to run:

```
/sbin/chkconfig --add gmetad
```

- 4 Configure the system-run levels on which to start gmetad:

```
/sbin/chkconfig --level 2345 gmetad on
```

- 5 Verify the configuration:

```
/sbin/chkconfig --list gmetad
```



```
GMETAD 0:off 1:off 2:on 3:on 4:on 5:on 6:off
```

6 Restart gmetad to make the changes effective:

```
/etc/init.d/gmetad restart
```

Note: If gmetad is not already running, the shutdown phase shows a FAILED message, which you can safely ignore.

Configuring Gmond on All Nodes

The role of gmond is to collect, send, and receive data. Once gmond has been installed on each node in the Vertica cluster, edit the configuration file on each node, as described in this section.

TIPS:

- You can edit `gmond.conf` on each node, or you can edit the file on the monitoring node and then copy `gmond.conf` to `/etc` on all other nodes in the cluster.
- On SuSE systems, the path is `/etc/ganglia/`.
- Remember to restart the service each time you edit the configuration file.

About the `gmond.conf` file

There are three important settings in the `gmond.conf` file. For details on all available settings, refer to the ***gmond.conf Linux man page*** <http://linux.die.net/man/5/gmond.conf>, which is documented below, in part, for your convenience.

- `udp_send_channel`. You can define as many `udp_send_channel` sections as you like within the limitations of memory and file descriptors. If gmond is configured to be mute, then these sections are ignored. The `udp_send_channel` has a total of five attributes: `mcast_join`, `mcast_if`, `host`, `port`, and `ttl`.

Note: All nodes require a `udp_send_channel` section, which notifies gmond where to send the data it has collected about the local node – even if the data it collects is about itself only. You can configure this setting to broadcast the information or send it to a particular host and port. If you specify a particular host, you might want all nodes to send data to the same location. You can also have each node send the same information to more than one location for redundancy.

- `udp_rcv_channel`. You can specify as many `udp_rcv_channel` sections as you like within the limits of memory and file descriptors, but at least one node must have a `udp_rcv_channel` section. The `udp_rcv_channel` section has the following attributes: `mcast_join`, `bind`, `port`, `mcast_if`, and `family`.

Data received by this section forms a snapshot of the state of all nodes. You can configure this setting to receive the data via broadcast or to receive it on a particular IP interface and port. More than one node could be receiving the same data.

If Ganglia is in use on multiple clusters in your network, you might need to filter which hosts are being reported on by configuring the `udp_rcv_channel` block in the `gmond.conf` file to use an access control list. For example:

```
udp_rcv_channel {
```

```
mcast_join = 239.2.11.71
bind       = 239.2.11.71
port       = 8649

acl {
  default = "deny"
  access {
    ip = 192.168.0.4
    mask = 32
    action = "allow"
  }
}
```

- **tcp_accept_channel**. You can specify as many `tcp_accept_channel` sections as you like within the limitations of memory and file descriptors. If `gmond` is configured to be mute, then these sections are ignored. The `tcp_accept_channel` has the following attributes: `bind`, `port`, `interface`, `family`, and `timeout`.

In order to get the most use out of Ganglia, at least one node that has `udp_rcv_channel` defined must have a `tcp_accept_channel` setting, as well. This setting describes a particular IP interface and port where a query can be sent. `gmond` returns an XML string of the summary information it has collected.

Edit the `gmond.conf` file

Perform the following steps *on each node* where `gmond` is installed, starting with the monitoring node.

- 1 On the monitoring node, use the text editor of your choice to open the `gmond.conf` file:

```
# vi /etc/gmond.conf
```
- 2 Change the name of the cluster to the (case-sensitive) name you specified in `gmetad.conf`.

```
cluster {
  name = "Vertica_Cluster"
  owner = "unspecified"
  latlong = "unspecified"
  url = "unspecified"
}
```
- 3 Add `gmond` to the list of services to run:

```
/sbin/chkconfig --add gmond
```
- 4 Configure the system-run levels on which to start `gmond`:

```
/sbin/chkconfig --level 2345 gmond on
```
- 5 Verify the configuration:

```
/sbin/chkconfig --list gmond
GMOND 0:off 1:off 2:on 3:on 4:on 5:on 6:off
```
- 6 Restart `gmond` to make the changes effective:

```
/etc/init.d/gmond restart
```

Note: If `gmond` is not already running, the shutdown phase shows a `FAILED` message, which you can safely ignore.

- 7 Run the following command to return an XML description of the state of the nodes in your cluster:

```
telnet localhost 8649
```

You can also use localhost 8651.

- 8 Edit `gmond.conf` on each node, or you can edit the file on the monitoring node and then copy `gmond.conf` to `/etc` on all other nodes in the cluster.

Tip: Restart the service each time you edit the configuration file.

Multicast IP Support

The server and network must be multicast-enabled to run Ganglia. To check, run `/sbin/ifconfig` on the server. If the network interface is flagged with MULTICAST, it is enabled.

If your machines are all on the same switch

If your machines are all on the same switch, proceed to *Edit the `gmond.conf` file* (page 228).

Note: Vertica recommends that all machines be on the same switch.

If the machines in the cluster are separated by a router

If the machines in the cluster are separated by a router, you must set the multicast Time-To-Live (TTL) option in `/etc/gmond.conf` higher than the default of 1.

- 1 Edit the `mcast ttl` setting to be one greater than the number of hops (routers) between the hosts.
- 2 Make sure that the routers are configured to pass along the multicast traffic. See the *Ganglia documentation* <http://sourceforge.net/apps/trac/ganglia/wiki> for details.

Configuring the Vertica Monitoring Package

This section describes how to configure the Vertica-specific extension files that are required for the Web-reporting front end.

Configuring and Starting `lighttpd`

The Vertica Monitoring package includes `lighttpd`, a lightweight http server. The package also installs the startup script `verttpd` to `/etc/init.d`.

- 1 On the node where `gmetad` is installed (the monitoring node), copy the `lighttpd.conf` file into the Vertica user directory for each user responsible for running the service, for example `dbadmin`:

```
# cp /opt/vertica/www/conf/lighttpd.conf /opt/vertica/config/users/dbadmin
```

- 2 Start the service:

```
# /etc/init.d/verttpd start
```

By default, the server starts on port 9090, but you can modify this setting in the `lighttpd` configuration file.

- 3 Access the URL to verify if the `lighttpd` is installed

<http://xx.xx.xx.xx:9090/>

where x is IP address (or host name) of the machine. Alternatively, specify the machine IP address on which lighttpd is installed.

If you encounter issues with lighttpd installation, see the *Lighttpd documentation* <http://redmine.lighttpd.net/projects/lighttpd/wiki>.

When the browser finishes loading, it displays the Vertica Console page with a link to Monitoring Tools (Ganglia).

Configuring Vertica Extension Files

Before you can monitor Vertica, you need to configure the Vertica extension files `vertica-dashboard.xml` and `verticadefines.php`. These files reside in the `/opt/vertica/www/htdocs/ganglia/verticaDashboard` folder.

Note: The default settings in `verticadefines.php` are adequate in most environments.

Configuration is now complete, and you can monitor the health of your Vertica cluster by clicking Vertica Monitoring from the Console page.

Edit the `vertica-dashboard.xml` file

The following procedure assumes you are still in the `/opt/vertica/www/htdocs/ganglia/verticaDashboard` folder.

- 1 Using the text editor of your choice, open `vertica-dashboard.xml`:

```
# vi vertica-dashboard.xml
```

- 2 Configure the following variables:

- **database.** Insert an XML tag that specifies the name of the database to be monitored, along with the password, if required. These variables are case sensitive. For example:

```
<databases>
  <database name="YourDBName" password=""></database>
</databases>
```

- **hostdetails.** Specify complete details about the host that maps the host name used by both Vertica and Ganglia:
 - Name, exactly as known by Vertica
 - Local IP address
 - Public IP address
 - Fully-qualified domain name (this is the name of the host as understood by Vertica)

Ensure that the information is correct or Vertica PHP scripts fails to locate the RRD databases and cannot display statistics. The following is an example.

```
<clusterdetails>
  <hostdetails name="host01" localip="10.0.0.1"
    publicip="xx.xx.xx.xx"
    fqdn="host01.vertica.com"></hostdetails>
</clusterdetails>
```

You need a `<hostdetails/>` block for each host in the cluster. If the hosts are on a private network, the `hostdetails` can be the `privateip`. List private network details under `localip` and public network details under `publicip`.

- **gmetric**. This executable is added during Ganglia installation and is used to store the data of the user-defined metrics:

```
<gmetric path="/usr/bin/gmetric"></gmetric>
```

- **cron-hostname**. Use the same name that Ganglia uses to refer to this node; for example:

```
<cron-hostname name="host01.vertica.com"></cron-hostname>
```

Vertica cron jobs run on the machine where the Web front end runs.

`cron-hostname` collects information about Vertica from system tables in Vertica.

To identify the name ganglia refers to on the node, check the

`/var/lib/ganglia/rrds/<cluster_name>` folders for a list of node names. Use the monitoring node name in the `cron-hostname` setting.

- **debug**. Set the debug enable to 1 if you want to enable the logging for cron jobs and for PHP scripts, specify the directory where the logs are collected, and provide the path where the `lighttpd` user has the sufficient privileges; for example:

```
<debug enable="1" path="/tmp/vertica-ganglia/"></debug>
```

- **fqdn**: Use the ganglia name identified as above with (or without) domain name qualification.

- 3 Log in as `dbadmin` (*not* root or the system returns errors), and verify that `gmond` is running on all the hosts where Vertica is installed and that information about all hosts is present in order to view the complete statistics about all the hosts:

```
$ /opt/vertica/bin/admintools -t list_db -d database_name
```

Note: An optional `--no-log` option, which must appear before `-t`, allows the Administration Tools to run silently (i.e., without logging anything). This setting is useful if, for example, you are running Ganglia dashboard scripts that run the Administrative Tools scripts frequently, which could cause the size of the `adminTools-dbadmin.log` file to rapidly increase. If you add the `--no-log` switch to `vertica-dashboard.xml`, logging is disabled.

- 4 Save and exit `vertica-dashboard.xml`.

Edit the `verticadefines.php` file [Optional]

This procedure is optional and included in the event you decide to edit the `verticadefines.php` file. In most environments, the default settings are adequate.

- 1 Using the text editor of your choice, open `verticadefines.php`:

```
# vi verticadefines.php
```

- 2 Configure the following variables.

- **vertica_path**. Location of the Vertica installation with a default value of `/opt/vertica/`.
- **admintools_path**. Location of the `admintools` installation with a default value of `/opt/vertica/bin/admintools`.
- **gangliadefault_url**. URL where the default Ganglia PHP scripts run; for example, `/ganglia`. The `gangliadefault_url` setting needs to be changed only if the defaults are not used.
- **refresh_time**. Time in seconds after which the Web page refreshes and displays Vertica statistics. The default is 300 seconds (5 minutes).

3 Save and exit `verticadefines.php`.

Add a cron job

In this procedure, create a cron job, which collects data from Vertica by running queries against system tables and returning system statistics in a graphical format.

1 Log in as the DBA user (not as root):

```
# su dbadmin
```

2 Using the text editor of your choice, insert the following line into the crontab for the DBA user:

```
crontab -e
```

3 Add the following line.

IMPORTANT! Despite how the following code fragment appears in the HTML or PDF output of this document, it is *one long line that must not contain returns*. If you copy the code from this document, paste it first into the text editor of your choice and remove all carriage returns before you add the line to your cron job. Also manually delete and retype the dash between `vertica-dashbard` to prevent the dash from becoming UTF-8 encoded.

```
*/* * * * * php /opt/vertica/www/htdocs/ganglia/verticaDashboard/  
cronjobs/vertica-dashboard.php -i /opt/vertica/www/htdocs/ganglia/  
verticaDashboard/ -c /opt/vertica/www/htdocs/ganglia/  
verticaDashboard/vertica-dashboard.xml >  
/tmp/vertica-ganglia/cronlogs.log 2>&1
```

The cron job is now configured to collect data from Vertica in one-minute increments. The `-i` switch represents the location of `verticaDashboard`, and the `-c` switch represents the location of the configuration file.

You are now ready to use Ganglia to monitor your Vertica cluster.

Upgrading the Vertica Monitoring Package

IMPORTANT

You cannot upgrade the Vertica-Ganglia Monitoring Package from the `vertica-web` package to the `vertica-ganglia` package

(`vertica-web-3.0.0-20090511050007.x86_64.RHEL5.rpm` to `vertica-ganglia-4.0.5-20091105151816.x86_64.RHEL5.rpm`).

The `vertica-web` rpm was the Webmin/Ganglia combination rpm that was introduced in Vertica 3.0, but in Vertica 3.0.7, the Ganglia and Webmin packages were separated into separate installation packages.

If you have the Vertica 3.0.7 Ganglia rpm installed, you can successfully upgrade to 4.0 Ganglia because it is a Ganglia-to-Ganglia upgrade, not a Webmin/Ganglia-to-Ganglia upgrade.

If you installed Ganglia after you installed Vertica 3.0.7:

Follow the installation instructions provided in *Installing the Vertica Monitoring Package* (page 221).

If you installed Ganglia before you installed Vertica 3.0.7:

- 1 Back up the `vertica-dashboard.xml` file by copying the file to a separate directory, such as `tmp`:

```
# cp /opt/vertica/www/htdocs/ganglia/verticaDashboard/vertica-dashboard.xml
/tmp
```

- 2 Uninstall the `webmin-ganglia` combined rpm; for example:

```
# rpm --erase vertica-web-<version>.<arch>.rpm
```

Note: In the above command, replace `<version>` with the current version of the rpm and `<arch>` with your system architecture (e.g., `x86_64.RHEL5`).

- 3 Download the current Ganglia rpm from the Vertica **Download Web site** http://www.vertica.com/v-zone/download_vertica.
- 4 Install the Vertica-Ganglia rpm by following the instructions provided in **Installing the Vertica Monitoring Package** (page 221).
- 5 Restore the `vertica-dashboard.xml` file; for example:

```
# cp /tmp/vertica-dashboard.xml
/opt/vertica/www/htdocs/ganglia/verticaDashboard
```

Uninstalling the Vertica Monitoring Package

Depending on which version of the Ganglia package you installed, choose one of the following paths:

If you installed the Webmin-Ganglia combined rpm (`vertica-web`) provided in Vertica 3.0:

```
# rpm --erase vertica-web-<version>.<arch>.rpm
```

In the above command, replace `<version>` with the version of the rpm (for example, `3.0.0-20090511050007`) and `<arch>` with your system architecture (for example, `x86_64.RHEL5`).

If you installed Vertica-Ganglia rpm (`vertica-ganglia`) provided in Vertica 3.0.7 or later:

```
# rpm --erase vertica-ganglia-<version>.<arch>.rpm
```

In the above command, replace `<version>.<arch>` with the version of the rpm and your system architecture; for example, `vertica-ganglia-4.0.12.x86_64.RHEL5.tar.gz`.

Recovering the Database

Recovering a database can consist of any of the following:

- **Restarting Vertica on a Host** (page 247).
- **Restarting the database** (page 248).
- **Recovering the Cluster from a Backup** (page 251).
- **Replacing Failed Disks** (page 285).
- **Copying a Database to Another Cluster** (page 258).
- **Exporting a Catalog** (page 251) for support purposes.

You can **monitor a recovery** (page 251) in progress by viewing log messages posted to the `vertica.log` file on each host.

See Also

Failure Recovery (page 235).

Failure Recovery

Failure recovery is the process of restoring the database to a fully-functional state after one or more nodes in the system has experienced a software or hardware related failure. Vertica recovers nodes by querying replicas of the data stored on other nodes. For example, a hardware failure could cause a node to lose database objects or to miss changes made to the database (INSERTs, UPDATEs, and so on) while offline. When the node comes back online, it recovers lost objects and catches up with changes by querying the other nodes.

Vertica uses the concept of K-Safety for failure recovery. The K value represents the maximum number of nodes in a database that can fail and recover with no loss of data. In Vertica, the value of K can be zero (0), one (1), or two (2). The Physical Schema design must meet certain requirements. To create designs that are K-Safe, Vertica recommends using the Database Designer.

The following table illustrates the number of nodes that can be down when the value of K is one (1) or two (2).

K Value	Nodes	Nodes DOWN	State of Database
1	3 or more	0	Safe.
		1	
		2	Unsafe. Automatic shutdown.
2	5 or more	0	Safe.
		1	
		2	
		3	Unsafe. Automatic shutdown.

Note: You can monitor the cluster state through the **View Database Cluster** state menu option.

Recovery Scenarios

Recovery comes into play when a node or the database is started. Depending upon how the node or database was shut down, there are three possibilities for a K-Safe database:

- **Recovery after failure of up to K Nodes:** This means that either one node has failed, for a database with a K value of one (1), or as many as two nodes have failed, for a database with a K value of two (2). Even in this state, the surviving 'UP' nodes of the database enable it to be fully operational and available for commands. The failed nodes can be restarted through the **Administration Tools** (page 336) using the **Restart Vertica on host (page 341)** option. The nodes being restarted show a status of 'RECOVERING' while they rebuild some of the data from the remaining nodes and, once finished, transition to an UP status. Transactions can continue to commit during the recovery process, except for a short period at the end of the recovery.

- **Recovery after a Clean Shutdown:** The database had been shut down cleanly via the Administration Tools Stop Database option. In this case, the database should be restarted using the **Start Database** (page 248) option. Upon restart all nodes that were 'UP' at the time of shutdown immediately transition to 'UP'. It is possible that at the time of shutdown, the database could have had at most K(1) or K(2) nodes down. These nodes go through the 'RECOVERING' state as described in 'Recovery after failure of up to K nodes' case above.
- **Recovery after an Unclean Shutdown (Manual Recovery):** The database was not shut down cleanly, which means that the database lost more than K nodes and possibly did not write all the data from the WOS to disk. This could happen due to a various reasons; for example:
 - An unexpected event, such as a power failure that causes all nodes to reboot
 - Vertica processes on the nodes exited due to a software or hardware failure
 - Additional nodes failed during recovery, violating k-safety

When the database is started through the Administration Tools Start Database option, recovery determines that a normal startup is not possible. It goes on to determine a point in time in which the data was consistent on all nodes. This is called the Last Good Epoch. As part of Start Database processing, the administrator is prompted to accept recovery with the suggested epoch. If accepted, the database recovers and any data changes made after the Last Good Epoch are lost. If not accepted, startup is aborted and the database is not started on any of the nodes.

Instead of accepting the given epoch, the administrator can instead choose to **recover from a backup** (page 251) or select an epoch for an even earlier point using the Roll Back Database to Last Good Epoch option in the Administration Tools Advanced Menu. This is useful in special situations, for example if the failure occurs during a batch of loads, for which it is easier to go back to the beginning of the batch, rather than starting in the middle, even though some of the work must be repeated. In most scenarios, it is sufficient and recommended to accept the given epoch.

Notes

- In Vertica 4.1, the default for the `HistoryRetentionTime` configuration parameter changed to 0, which means that Vertica only keeps historical data when nodes are down. This default setting effectively prevents the use of the Administration Tools 'Roll Back Database to Last Good Epoch' option because the AHM remains close to the current epoch and a rollback is not permitted to an epoch prior to the AHM. If you rely on the Roll Back option to remove recently loaded data, consider setting a day-wide window for removing loaded data; for example:

```
=> SELECT SET_CONFIG_PARAMETER ('HistoryRetentionTime', '86400');
```


See **Epoch Management Parameters** (page 30) in the Administrator's Guide.
- Starting in 4.0, manual recovery is possible even if up to K nodes are out of commission; for example, physically removed for repair or not reachable at the time of recovery. Once the nodes are back in commission, they recover and rejoin the cluster, as described in the "Recovery after failure of up to K nodes" section above.
- **IMPORTANT:** When a node is down, it can take a full minute or more for the Vertica processes to time out during its attempt to form a cluster when manual recovery is needed. Wait approximately one minute until the system returns the manual recovery prompt. Do not press CTRL-C during database startup.

For information on troubleshooting recovery problems, refer to the following sections in the Troubleshooting Guide:

- **Startup Problems** (page 241)
- **Shutdown Problems** (page 237)

See Also

High Availability and Recovery in the Concepts Guide.

Shutdown Problems

A Database shuts down when one of the following events occurs:

- The administrator uses the **Stop Database** (page 338) command.
- The cluster becomes unsafe and the database shuts down to prevent data loss.

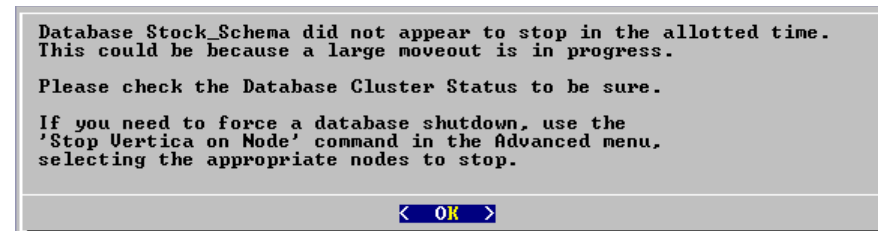
If a problem prevents the database from shutting down, the Administration Tools display a dialog containing the following error message:

```
Database ... did not appear to stop...
```

The message is followed by a description of the problem. This section describes some of the known problems that can occur when stopping a database.

... large moveout is in progress

If there is a Tuple Mover operation in progress, the Administration Tools displays a message similar to the one shown below:



This particular message indicates that the Tuple Mover needs more time to complete a moveout operation, which is an internal session. (See **Managing Sessions** (page 313) in the Administrator's Guide for more information.)

Note: Vertica Systems, Inc. recommends that you wait as long as possible before taking action. You can cause data loss by, for example, interrupting a database that is still performing a moveout.

- 1 If you cannot wait any longer, disconnect and select Advanced > **Stop Vertica on Host** (page 347) from the Administration Tools.
- 2 If Stop Vertica on Node fails, select Advanced > **Killing a Vertica Process on Host** (page 348).

This forces the cluster to go through recovery at startup.

... users are connected

Error

If users are connected during shutdown operations, the Administration Tools displays a message similar to the following:

```
Database Stock_Schema did not appear to stop in the allotted time.
NOTICE: Cannot shut down while users are connected
        shutdown
-----
Shutdown: aborting shutdown
(1 row)
```

If you need to force a database shutdown, use the 'Stop Vertica on Node' command in the Advanced menu, selecting the appropriate nodes to stop.

Description

The message indicates that there are active user connections (sessions). See *Managing Sessions* (page 313) in the Administrator's Guide for more information.

Resolution

The following examples were taken from a different database.

- 1 To see which users are connected, connect to the database and query the `SESSIONS` system table described in the SQL Reference Manual. For example:

```
=> \pset expanded
Expanded display is on.
=> SELECT * FROM SESSIONS;

-[ RECORD 1 ]
node_name      | site01
user_name      | dbadmin
client_hostname | 127.0.0.1:57141
login_timestamp | 2009-06-07 14:41:26
session_id     | rhel4-1-30361:0xd7e3e:994462853
transaction_start | 2009-06-07 14:48:54
transaction_id  | 45035996273741092
transaction_description | user dbadmin (select * from session;)
statement_start | 2009-06-07 14:53:31
statement_id    | 0
last_statement_duration | 1
current_statement | select * from sessions;
ssl_state       | None
authentication_method | Trust
-[ RECORD 2 ]
node_name      | site01
user_name      | dbadmin
client_hostname | 127.0.0.1:57142
login_timestamp | 2009-06-07 14:52:55
session_id     | rhel4-1-30361:0xd83ac:1017578618
transaction_start | 2009-06-07 14:53:26
transaction_id  | 45035996273741096
transaction_description | user dbadmin (COPY ClickStream_Fact FROM
'/data/clickstream/1g/ClickStream_Fact.tbl' DELIMITER '|' NULL '\n' DIRECT;)
statement_start | 2009-06-07 14:53:26
statement_id    | 17179869528
```

```

last_statement_duration | 0
current_statement       | COPY ClickStream_Fact FROM '/data/clickstream/1g/ClickStream_Fact.tbl'
DELIMITER '|' NULL '\\n' DIRECT;
ssl_state               | None
authentication_method   | Trust

```

The current statement column of Record 1 shows that session is the one you are using to query the system table. Record 2 shows the session that must end before the database can be shut down.

- 2 If a statement is running in a session, that session must be closed. Use the function `CLOSE_SESSION` or `CLOSE_ALL_SESSIONS` described in the SQL Reference Manual.

Note: `CLOSE_ALL_SESSIONS` is the more common command because it forcefully disconnects all user sessions.

```
=> SELECT * FROM SESSIONS;
```

```

-[ RECORD 1 ]
node_name           | site01
user_name           | dbadmin
client_hostname     | 127.0.0.1:57141
client_pid          | 17838
login_timestamp     | 2009-06-07 14:41:26
session_id          | rhel4-1-30361:0xd7e3e:994462853
client_label        |
transaction_start   | 2009-06-07 14:48:54
transaction_id      | 45035996273741092
transaction_description | user dbadmin (select * from sessions;)
statement_start     | 2009-06-07 14:53:31
statement_id        | 0
last_statement_duration_us | 1
current_statement   | select * from sessions;
ssl_state           | None
authentication_method | Trust
-[ RECORD 2 ]
node_name           | site01
user_name           | dbadmin
client_hostname     | 127.0.0.1:57142
client_pid          | 17839
login_timestamp     | 2009-06-07 14:52:55
session_id          | rhel4-1-30361:0xd83ac:1017578618
client_label        |
transaction_start   | 2009-06-07 14:53:26
transaction_id      | 45035996273741096
transaction_description | user dbadmin (COPY ClickStream_Fact FROM
'/data/clickstream/1g/ClickStream_Fact.tbl'
DELIMITER '|' NULL '\\n' DIRECT;)
statement_start     | 2009-06-07 14:53:26
statement_id        | 17179869528
last_statement_duration_us | 0
current_statement   | COPY ClickStream_Fact FROM
'/data/clickstream/1g/ClickStream_Fact.tbl'
DELIMITER '|' NULL '\\n' DIRECT;
ssl_state           | None
authentication_method | Trust

```

```
=> SELECT CLOSE_SESSION('rhel4-1-30361:0xd83ac:1017578618');
```

```

-[ RECORD 1 ]
close_session | Session close command sent. Check sessions for progress.
=> SELECT * FROM SESSIONS;
```

```

-[ RECORD 1 ]
node_name           | site01
user_name           | dbadmin

```

```

client_hostname      | 127.0.0.1:57141
client_pid          | 17838
login_timestamp     | 2009-06-07 14:41:26
session_id          | rhe14-1-30361:0xd7e3e:994462853
client_label        |
transaction_start   | 2009-06-07 14:48:54
transaction_id      | 45035996273741092
transaction_description | user dbadmin (select * from sessions;)
statement_start     | 2009-06-07 14:54:11
statement_id        | 0
last_statement_duration_us | 98
current_statement   | select * from sessions;
ssl_state           | None
authentication_method | Trust
    
```

3 Query the SESSIONS table again. For example, two columns have changed:

- stmtid is now 0, indicating that no statement is in progress.
- stmt_duration now indicates how long the statement ran in milliseconds before being interrupted.

The SELECT statements that call these functions return when the interrupt or close message has been delivered to all nodes, not after the interrupt or close has completed.

4 Query the SESSIONS table again. When the session no longer appears in the SESSION table, disconnect and run the **Stop Database** (page 338) command.

Controlling Sessions

The database administrator must be able to disallow new incoming connections in order to shut down the database. On a busy system, database shutdown is prevented if new sessions connect after the CLOSE_SESSION or CLOSE_ALL_SESSIONS() command is invoked — and before the database actually shuts down.

One option is for the administrator to issue the SHUTDOWN('true') command, which forces the database to shut down and disallow new connections. See SHUTDOWN in the SQL Reference Manual.

Another option is to modify the MaxClientSessions parameter from its original value to 0, in order to prevent new non-dbadmin users from connecting to the database.

1 Determine the original value for the MaxClientSessions parameter by querying the V_MONITOR.CONFIGURATIONS_PARAMETERS system table:

```

=> SELECT CURRENT_VALUE FROM CONFIGURATION_PARAMETERS WHERE
      parameter_name='MaxClientSessions';
      CURRENT_VALUE
-----
          50
(1 row)
    
```

2 Set the MaxClientSessions parameter to 0 to prevent new non-dbadmin connections:

```

=> SELECT SET_CONFIG_PARAMETER('MaxClientSessions', 0);
    
```

Note: The previous command allows up to five administrators to log in.

3 Issue the CLOSE_ALL_SESSIONS() command to remove existing sessions:

```

=> SELECT CLOSE_ALL_SESSIONS();
    
```

4 Query the SESSIONS table:

```
=> SELECT * FROM SESSIONS;
```

When the session no longer appears in the `SESSIONS` table, disconnect and run the **Stop Database** (page 338) command.

5 Restart the database.

6 Restore the `MaxClientSessions` parameter to its original value:

```
=> SELECT SET_CONFIG_PARAMETER('MaxClientSessions', 50);
```

See Also

`CLOSE_SESSION`, `CLOSE_ALL_SESSIONS`, `CONFIGURATION_PARAMETERS`, and `SESSIONS` in the SQL Reference Manual

Managing Sessions (page 313) and *Configuration Parameters* (page 25) in the Administrator's Guide

No running statement, that session is idle

Error

No running statement, that session is idle

Description

The `INTERRUPT_STATEMENT` function failed. The session exists but is not running a statement and the session ID can't be found.

Resolution

Not required.

Startup Problems

This section describes some of the known problems that can occur when starting a database. Startup fails on one or more nodes either due to communication problems between nodes or when recovery fails to recover the data on the node(s) for any reason.

Note: If a database fails to start before it can write messages into `vertica.log`, check the file `catalog-path/database-name/dbLog`.

Startup successful, but some nodes are recovering

Error

Startup successful, but some nodes are recovering. You can use the View Database Cluster State option to check progress. Press RETURN to continue

Description

This message typically indicates an abnormal shutdown of one or more nodes. If any nodes in the database are UP, the database is fully operational. Note however that if you start load processing on the database the recovery node could take additional time to become operational

Resolution

None required

Error starting database, no nodes are up

Error

Error starting database, no nodes are up
Press RETURN to continue

Description

An unknown problem is preventing the database from starting.

Resolution

Reboot the hosts and try to start the database. If unsuccessful, run diagnostics and contact **Technical Support** (on page 1).

Database startup successful, but it could be incomplete

Error

Database startup successful, but it may be incomplete.

Description

Some nodes are in a transitional state: not up but not recovering.

Resolution

If this error persists, try using the **Stop Vertica on Host** (page 347) command in the **Advanced** (page 346) menu to stop Vertica on all nodes. Then use the Roll Back Database With Catalog Version command.

Error

Database startup successful, but it may be incomplete.
Some nodes remain in a transitional state.
See Database Cluster State in Main Menu for details.
Press RETURN to continue

Description

This message indicates that the Database Administrator chose not to wait when prompted and that the database cannot start.

Resolution

If this error persists, contact **Technical Support** (on page 1).

Database did not start cleanly on initiator node!

Error

ERROR: Database did not start cleanly on initiator node!
Stopping all nodes

Issuing shutdown command to database

Description

Configuration problems can cause this error.

Resolution

- 1 Check hostname resolution as described in Configure Hostname Resolution section of the Installation Guide.
- 2 Examine `/etc/hosts` on each node and specify a fully qualified domain name and an unqualified hostname. For example:
`192.168.1.99 node01.fqdomain.com node01`
- 3 Verify that there is no firewall running.

TIMEOUT ERROR: Could not login with SSH

Error

```
TIMEOUT ERROR: Could not login with SSH. Here is what SSH said:  
Last login: Sat Dec 15 18:05:35 2007 from node01
```

Description

Installing Vertica on a host that is missing the mount point `/dev/pts` could result in the error when creating a database.

Resolution

Make sure that `/dev/pts` is mounted.

Good epoch logs are available on all nodes

Error

```
Database startup failed. Good epoch logs are available on all nodes.  
WARNING: if you say 'yes.' changes made to the database after  
'2007-07-04 03:58:03-04' (epoch 265) will be permanently lost.  
Do you really want to restart the database from '2007-07-04 03:58:03-04' (epoch  
265)?
```

Explanation

A startup attempt failed due to database inconsistency across the cluster. Vertica has determined that it can probably restart and continue running at an earlier epoch.

Workaround

Restarting from the suggested epoch erases any changes made to the database subsequent to that epoch, across the cluster. It is likely that these changes were incomplete and erasing them allows the cluster to proceed normally using the data saved prior to the epoch.

No good epoch log available on node

Error

Database startup failed. No good epoch log available on node `stock_multi_node_0`. Please run diagnostics and contact Vertica

Description

There are a number of possible reasons for this error message, including an abnormal startup or shutdown. Every node in the cluster must be started with the same recovery epoch.

Non-matching recovery epochs occur when a cluster has experienced an unsafe shutdown.

Resolution

- 1 Make sure that all nodes are powered on.
- 2 **Start the database** (page 337) again.
- 3 Make sure that all nodes have Spread running. If necessary, restart Spread where it is not running and **start the database** (page 337).
- 4 On each node that did not start up, examine **dbLog** (page 197) for the cause of the failure.
- 5 If the cause cannot be determined, it is likely that a node has no catalog version or epoch log from which to recover. Run diagnostic tests (see Using Diagnostic Tools) and contact **Technical Support** (on page 1).

Nodes stuck in INITIALIZING state

Error

In rare cases, some or all nodes can get stuck in the INITIALIZING state when trying to start the database.

Description

This issue is known to happen due to network configuration problems, corrupted catalogs or disks, missing database directories or some other fatal problem in the database setup. Incorrect use of the Administration Tools Advanced Menu options could also lead to this condition.

Resolution

- 1 Open the Administration Tools.
- 2 Select Advanced > **Stopping Vertica on Host** (page 347) to stop all nodes.
- 3 Go back to the Main Menu and click **Start Database** (page 337).
- 4 If the error persists, run diagnostics and contact **Technical Support** (on page 1)

Node does not recover because of lock timeouts

Error

Upon starting a node, it stays in RECOVERING state for a long time and eventually shuts down again. Examination of the `vertica.log` on the node reveals an error:

```
Locking failure: Timed out S locking Table.
```

Description

The final stage of recovering a node requires a S lock on the table. If you have a continuous stream of COPY commands in progress, recovery might not be able to get this lock even after multiple re-tries.

Resolution

If you see this situation, either momentarily stop the loads or pick a time when the cluster is not busy to restart the node and let recovery proceed.

Spread Problems

This section describes some of the known problems that can occur when using spread.

Spread is not running

Error

```
admintools View Cluster State shows "Could not connect to spread.
Spread is configured as part of database creation".
spread dead but pid file exists
```

Resolution

Verify that spread is not running and then restart the spread daemon.

- 1 Verify that spread is not running:


```
ps ax | grep spread
```
- 2 Examine `/tmp/spread*.log` and `/var/log/spreadd.log` for problems. Permission problems and syntax problems are identified in these log files.
- 3 Issue the `ifconfig` command to check the current IP addresses of the hosts and verify that those IP addresses are listed in `/opt/vertica/config/vspread.conf`.
- 4 Check for Vertica processes that might be running, even though spread is down:


```
ps ax | grep vertica
```
- 5 Kill the Vertica process. Alternatively, use the Admintools Advanced Menu > **Kill Vertica Process on Host**.
- 6 Restart spread:
 1. Log in as root:


```
$ su - root
password: <root-password>
#
```

You can use `sudo` (if enabled) if you do not have the root password.
 2. Restart the spread daemon:


```
# /etc/init.d/spreadd restart
```
 3. Ensure the daemon is running:


```
# ps ax | grep spread
```

Administration Tools shows node state as UNKNOWN

Error

The Administration Tools 'View Cluster State' shows one or more nodes with an UNKNOWN status

Description

Under certain conditions, Vertica nodes can go into the UNKNOWN state yet still be processing. In most cases, after some time, they return to UP status. However, if you see a persistent UNKNOWN state that does not resolve to an UP state after several minutes, follow the instructions in this section.

Resolution

Likely cause of this issue is a sub-optimally configured I/O subsystem leading to high contention that causes Vertica to be unresponsive to messages from the spread daemon. You might notice this problem occurs more readily when running several large join queries that spill to disk. Check for high I/O waits and other symptoms of I/O problems and if unable to resolve, contact **Technical Support** (on page 1) with I/O statistics and sar data.

See Also

Spread is not running (page 245) for restarting spread

Diagnosing spread problems

Spread Panic

Error

```
Error while starting/enabling multicasting to all hosts  
Spread panic during re-init on the following hosts: ['vertica01']
```

Description

Vertica automatically sets up a spread configuration for the cluster when you use the **Create Database** (page 342) command in the Administration Tools, and starts spread. Various other configuration errors can cause the spread startup to fail.

Resolution

Do not attempt to change the spread configuration. Contact **Technical Support** (on page 1).

Spread Dead but pid File Exists

Error

```
spread dead but pid file exists
```

Description

If spread ends abnormally, the pid and lock file are left behind.

Resolution

Restart spread.

- 1 Log in as root:

```
$ su - root
password: <root-password>
#
```

You can use sudo (if enabled) if you do not have the root password.

- 2 Restart the spread daemon:

```
# /etc/init.d/spreadd restart
```

- 3 Ensure the daemon is running:

```
# ps ax | grep spread
```

To diagnose issues related to starting Spread, "status" option has been enhanced to provide guidance .

Example

The following example checks on the spread status.

```
$ sudo /etc/init.d/spreadd status
```

```
spread is stopped
```

If you are having trouble starting spread, check `/opt/vertica/config/vspread.conf` and spread logs in `/tmp/spread_*` and `/var/log/spreadd.log`.

```
$ sudo /etc/init.d/spreadd start
```

```
Starting spread daemon: spread (pid 24290) is running...
```

```
[ OK ]
```

```
$ sudo /etc/init.d/spreadd status
```

```
spread (pid 24290) is running...
```

Restarting Vertica on a Host

When one node in a running database cluster fails, or if any files from the catalog or data directories are lost from any one of the nodes, you can check the status of failed nodes using the Administration Tools.

- 1 Run Administration Tools.
- 2 From the Main Menu, select **Restart Vertica on Host** and click **OK**.
- 3 Select the database host you want to recover and click **OK**.

Note: You might see additional nodes in the list, which are used internally by the Administration Tools. You can safely ignore these nodes.

- 4 Verify recovery state by selecting **View Database Cluster State** from the **Main Menu**.

After the database is fully recovered, you can check the status at any time by selecting **View Database Cluster State** from the Administration Tools **Main Menu**.

Restarting the Database

If you lose the Vertica process on more than one node (for example, due to power loss), or if the servers are shut down without properly shutting down the Vertica database first, the database cluster indicates that it did not shut down gracefully the next time you start it.

The database automatically detects when the cluster was last in a consistent state and then shuts down, at which point an administrator can restart it.

From the Main Menu in the Administration Tools:

- 1 Verify that the database has been stopped by clicking **Stop Database**.
A message displays: No databases owned by <dbadmin> are running
- 2 Start the database by selecting **Start Database** from the Main Menu.
- 3 Select the database you want to restart and click **OK**.

If you are starting the database after an unclean shutdown, messages display, which indicate that the startup failed. Press **RETURN** to continue with the recovery process.

```

*** Starting database: QATESTDB ***
  Participating hosts:
    rhel5-1
    rhel5-2
    rhel5-3
    rhel5-4

  Checking vertica version on host rhel5-1
  Checking vertica version on host rhel5-2
  Checking vertica version on host rhel5-3
  Checking vertica version on host rhel5-4
  Processing host rhel5-1
  Processing host rhel5-2
  Processing host rhel5-3
  Processing host rhel5-4

  Node Status: site01: (INITIALIZING) site02: (INITIALIZING) site03: (INITIALIZING)
  Node Status: site01: (INITIALIZING) site02: (INITIALIZING) site03: (INITIALIZING)
  Node Status: site01: (LOSTCONTACT) site02: (LOSTCONTACT) site03: (LOSTCONTACT) sit
  Node Status: site01: (LOSTCONTACT) site02: (LOSTCONTACT) site03: (LOSTCONTACT) sit
  Node Status: site01: (LOSTCONTACT) site02: (LOSTCONTACT) site03: (LOSTCONTACT) sit
  Node Status: site01: (LOSTCONTACT) site02: (LOSTCONTACT) site03: (LOSTCONTACT) sit
  Node Status: site01: (LOSTCONTACT) site02: (LOSTCONTACT) site03: (LOSTCONTACT) sit
  Node Status: site01: (LOSTCONTACT) site02: (LOSTCONTACT) site03: (LOSTCONTACT) sit
  Node Status: site01: (LOSTCONTACT) site02: (LOSTCONTACT) site03: (LOSTCONTACT) sit
  Node Status: site01: (LOSTCONTACT) site02: (LOSTCONTACT) site03: (LOSTCONTACT) sit
  Node Status: site01: (DOWN) site02: (DOWN) site03: (DOWN) site04: (DOWN)
  Node Status: site01: (DOWN) site02: (DOWN) site03: (DOWN) site04: (DOWN)
  Node Status: site01: (DOWN) site02: (DOWN) site03: (DOWN) site04: (DOWN)
  Node Status: site01: (DOWN) site02: (DOWN) site03: (DOWN) site04: (DOWN)
  Node Status: site01: (DOWN) site02: (DOWN) site03: (DOWN) site04: (DOWN)
  Node Status: site01: (DOWN) site02: (DOWN) site03: (DOWN) site04: (DOWN)
  Node Status: site01: (DOWN) site02: (DOWN) site03: (DOWN) site04: (DOWN)
  Node Status: site01: (DOWN) site02: (DOWN) site03: (DOWN) site04: (DOWN)
  Node Status: site01: (DOWN) site02: (DOWN) site03: (DOWN) site04: (DOWN)
  Node Status: site01: (DOWN) site02: (DOWN) site03: (DOWN) site04: (DOWN)
  Node Status: site01: (DOWN) site02: (DOWN) site03: (DOWN) site04: (DOWN)
  Node Status: site01: (DOWN) site02: (DOWN) site03: (DOWN) site04: (DOWN)
  Node Status: site01: (DOWN) site02: (DOWN) site03: (DOWN) site04: (DOWN)
  Error starting database, no nodes are up
  Press RETURN to continue █

```

An epoch represents committed changes to the data stored in a database between two specific points in time. When starting the database, Vertica searches for last good epoch.

- 4 Upon determining the last good epoch, you are prompted to verify that you want to start the database from the good epoch date. Select **Yes** to continue with the recovery.

```
Database startup failed. Good epoch logs are available on all nodes.
WARNING: if you say 'yes', changes made to database after
'2008-01-28 16:49:31-05' (epoch 4203) will be permanently lost.

Do you really want to restart the database from '2008-01-28 16:49:31-05' (epoch 4203)?

< Yes > < No >
```

Caution: If you do not want to start from the last good epoch, you may instead restore the data from a backup and attempt to restart the database. For this to be useful, the backup must be more current than the last good epoch.

Vertica continues to initialize and recover all data prior to the last good epoch.

```
*** Restarting database QATESTDB at epoch 4203 ***
  Participating hosts:
    rhel5-1
    rhel5-2
    rhel5-3
    rhel5-4
  Checking vertica version on host rhel5-1
  Checking vertica version on host rhel5-2
  Checking vertica version on host rhel5-3
  Checking vertica version on host rhel5-4
  Processing host rhel5-1
  Processing host rhel5-2
  Processing host rhel5-3
  Processing host rhel5-4
  Node Status: site01: (RECOVERING) site02: (RECOVERING) site03: (RECOVERING) si
  Node Status: site01: (RECOVERING) site02: (RECOVERING) site03: (RECOVERING) si
  Node Status: site01: (RECOVERING) site02: (RECOVERING) site03: (RECOVERING) si
  Node Status: site01: (UP) site02: (UP) site03: (UP) site04: (UP)
```

If recovery takes more than a minute, you are prompted to answer <Yes> or <No> to "Do you want to continue waiting?"

When all the nodes' status have changed to RECOVERING or UP, selecting <No> lets you exit this screen and monitor progress via the Administration Tools Main Menu. Selecting <Yes> continues to display the database recovery window.

Note: Be sure to reload any data that was added after the last good epoch date to which you have recovered.

Recovering the Cluster from a Backup

To recover a cluster from a backup, refer to the following topics in this guide:

- **Backing Up the Database** (page 253)
- **Restoring the Database From a Backup** (page 255)

Monitoring Recovery

During recovery, Vertica adds logging information to the `vertica.log` on each host.

You can monitor your recovery progress by viewing these messages. Recovery status messages are identified by the string ' [Recover] '. For example:

```
$ tail -f catalog-path/database-name/node-name_catalog/vertica.log
01/23/08 10:35:31 thr:Recover:0x2a98700970 [Recover] <INFO> Changing
host node01 startup state from INITIALIZING to RECOVERING
01/23/08 10:35:31 thr:CatchUp:0x1724b80 [Recover] <INFO> Recovering to
specified epoch 0x120b6
01/23/08 10:35:31 thr:CatchUp:0x1724b80 [Recover] <INFO> Running 1 split
queries
01/23/08 10:35:31 thr:CatchUp:0x1724b80 [Recover] <INFO> Running query:
ALTER PROJECTION proj_tradesquotes_0 SPLIT node01 FROM 73911;
```

When recovery has completed:

- 1 Launch Administration Tools.
- 2 From the Main Menu, select **View Database Cluster State** and click **OK**.
The utility reports your node's status as UP.

See Also

Monitoring the Database (page 197)

Exporting a Catalog

When you export a catalog you can quickly move a catalog to another cluster. Exporting a catalog transfers schemas, tables, constraints, projections, and views. System tables are not exported.

Exporting catalogs can also be useful for support purposes.

See the `EXPORT_CATALOG` function in the SQL Reference Manual for details.

Backup and Restore

Vertica provides utilities to backup and restore the database in various ways.

You can perform full backups as well as incremental backups. A backup could be restored back to the original database or used to create a new cluster for Disaster Recovery (DR) purposes.

When to Back Up the Database

In addition to any guidelines established by the IT department within your company, Vertica recommends that you back up your database:

- Before you upgrade Vertica to another release.
- Before you drop a partition.
- After you load a large volume of data.
- Before and after you add a node to your database cluster.

Backing Up the Database

Vertica provides a utility to perform full and incremental database backups using a script called `backup.sh`. This file is located in `/opt/vertica/scripts` and uses the file synchronization utility, `rsync`, to intelligently copy only the files that have been added and remove the unused ones.

Vertica never modifies data files; it adds files to and deletes them from the database.

Requirements

- `rsync` 3.0 or later must be installed on the database nodes.
- The script must be run on one of the database nodes you intend to back up.
- The script must be run as the Linux user who is the database administrator and who has passwordless SSH access from each of the source nodes to the backup node.
- The nodes being backed up must be resolvable using their public IP address from the backup node. (See [Configure Hostname Resolution in the Installation Guide](#).)
- The source database must be UP; otherwise the script generates errors.

Syntax

```
backup.sh -s source_host1, source_host2, source_host3
          -i initiator_host
          -b backup_host
          -B backup_dir
          -D database_directory_path
          [ -d database_name ]
          [ -p database_port ]
          [ -u dbadmin_user ]
          [ -w dbadmin_password ]
          [ -S snapshot_name ]
          [ -T temporary_directory_path ]
```

Parameters

-s	The hosts from the source database to include in the backup.
-i	The host to initiate any SQL issued by the backup.
-b	The host on which to copy the backup files.
-B	The directory path on which to copy the backup files.
-D	The absolute path to the parent directory that contains the subdirectories for each node's catalog. You can find this path by viewing the database details (in adminTools, select Advanced > View Database (page 344)). The path you need is shown under Catalog Directory, minus the last directory in the path (which is where the catalogs for the individual nodes are stored). For example, if the path shown under Category Directory is <code>/data/dbs/vmart/v_vmart_node001_catalog</code> , then the path you want is <code>/data/dbs/vmart/</code> .
-d	The name of the database (optional).
-p	The database port (optional).

-u	The user name for the database administrator (optional, if not specified, uses current user).
-w	The password for the database administrator (optional, if not specified, will be prompted).
-s	The snapshot name used to identify files that are used in the process of moving the database. If you do not supply this parameter, the script creates a name based on the current date. Important: This value of the parameter is output by the script and is required by the restore process. Keep a record of it or create your own naming convention. For example, you could incorporate the date into the name. For incremental backups you must provide the same snapshot every time you run the backup script.

Notes

- The list of host names for `-s` must not have any spaces between them.
- The backup directory path must have write permissions by the administrator user.
- If you are using external procedures owned by users other than the administrator user, the `backup.sh` script might not be able to retain the SUID permissions of the backup files. Upon restore, you must restore the permissions of the procedures manually. See Requirements for External Procedures in Programmer's Guide. The output of the restore script alerts you to such discrepancies.

Incremental Backups

The same `backup.sh` script can also be run repeatedly using the same parameters each time to create incremental backups. The catalog is copied anew each time, but only the data files that have changed are copied.

Example

The following example backs up all the nodes of a database presumed to be located at `/examples/vmartdb` on hosts named `wombat-5` through `wombat-8` to a directory `/scratch/qa/backup` on host named `raster-f1`.

```
/opt/vertica/scripts/backup.sh -s wombat-5,wombat-6,wombat-7,wombat-8
-i wombat-5 -b raster-f1 -D /examples/vmartdb -B /scratch_b/qa/backup
```

The following shows excerpts from the output of the `backup.sh` script. Note in particular the **SNAPSHOT** output by the script. The restore process requires this information. The 'Literal data' and 'Matched data' lines show whether the backup is full or incremental. In the case of an incremental backup, 'Matched data' is a subset of 'Literal data'.

```
SOURCE_HOSTS    = wombat-5 wombat-6 wombat-7 wombat-8
INITIATOR_HOST  = wombat-5
PORT            = 5433
BACKUP_HOST     = raster-f1
BACKUP_DIR      = /scratch_b/qa/backup
DATABASE_DIR    = /examples/vmartdb
SNAPSHOT       = snap-2010-04-30-1342
TEMP_DIR        = /tmp
...
Number of files: 761
Number of files transferred: 384
```

```
Total file size: 115.18M bytes
Total transferred file size: 115.18M bytes
Literal data: 115.18M bytes
Matched data: 0 bytes
File list size: 16.77K
File list generation time: 0.031 seconds
File list transfer time: 0.000 seconds
Total bytes sent: 10.57K
Total bytes received: 115.23M
...
```

Restoring the Database from a Backup

The method you use to restore a database from a backup depends on the reason you want to restore the database and the location where you want to restore it:

Restoring a database to the same cluster

This method is useful if you need to restore the database to an earlier time or recover data lost after events such as hardware corruption or human error.

See *Restoring to the Same Cluster* (page 255).

Restoring a database to a host that has failed and been repaired

You might choose this method if you are concerned that the host machine might lose a disk. In this case the machine has the same name and IP address. Once you have replaced the disk, you could restore the data on the disk to the most recent backup and then let the host automatically query other hosts to recover changes since the last backup. This method is faster than letting the host completely restore itself.

See *Replacing Nodes* (page 275).

Restoring the database to replacement node within the same cluster

This method is useful for swapping an existing host for a new host. Once you have swapped the host, you could restore the data to the most recent backup and then let the host automatically query other hosts to recover changes made since the last backup.

See *Replacing Nodes* (page 275).

Restoring to the Same Cluster

Restoring a database on the same cluster from a backup consists of following these general steps:

- 1 **Stopping the database** (page 142) you intend to restore.

Note: If you restore data to a single node, the node has already stopped. You do not need to stop the database.

- 2 **Restoring the database** (page 256).

- 3 **Starting the database** (page 142) from the Administration Tools.

Note: If you restored all the nodes using the backup, use *Manual Recovery* (page 235). Administration Tools returns the message, "Database startup failed," after you attempt to restart the database and then offers to restart the database from an earlier epoch. Click **Yes**.

- 4 After the database starts, connect to it through the Administration Tools and verify that it was successfully restored by running some queries.

Restoring the Database

Vertica provides a `restore.sh` script to restore the database from the backup created by `backup.sh`. The script is located in `/opt/vertica/scripts`.

Requirements

- The script must be run on one of the nodes of the database being backed up.
- The script requires `rsync 3.0` or later to be installed on the database nodes.
- The script must be run as the Linux user who is the database administrator. The hosts being restored must be resolvable using their public IP address from the backup host. (See [Configure Hostname Resolution in Installation Guide](#) for requirements on hostname resolution.) The Linux user for the database administrator must have passwordless `ssh` access from each of the hosts being restored to the backup host.
- The backup must have been created using the `backup.sh` script. It is important to note the snapshot name used by the backup script for use in restore operations.
- The database is being restored to the same cluster. See [Copying a Database to Another Cluster](#) (page 258) for restore to a different cluster.
- The database being restored must be shut down; otherwise the results could be unexpected.

Syntax

```
restore.sh -s host01, host02, host03
           -b backup_host
           -B backup_dir
           -D database_directory_path
           -S snapshot_name
           [ -T temporary_directory_path ]
           [ -c ]
           [ -q ]
```

Parameters

-s	The hosts to be included in the restore
-b	The host to copy the backup files from
-B	The directory path to copy the backup files from
-D	The absolute path to the parent directory that contains the subdirectories for each node's catalog. You can find this path by viewing the database details (in <code>adminTools</code> , select <code>Advanced > View Database</code> (page 344)). The path you need is shown under <code>Catalog Directory</code> , minus the last directory in the path (which is where the catalogs for the individual nodes are stored). For example, if the path shown under <code>Category Directory</code> is <code>/data/dbs/vmart/v_vmart_node001_catalog</code> , then the path you want is <code>/data/dbs/vmart/</code> .
-T	The directory path to use for temporary files during restore

-c	If provided, <code>vertica.conf</code> file will also be restored.
-q	If provided, uses quiet mode, which means the script will not ask for confirmation before starting the restore process
-S	The snapshot name used to identify files to restore. This is the snapshot name used when creating the backup or generated as output of the <code>backup.sh</code> script

Notes

- The list of host names for `-s` must not have any spaces between them.
- The Linux user for the administrator must have passwordless ssh access from the backup host to each of the source hosts.
- By default, `restore.sh` does not restore the `vertica.conf` file. This is useful if you have modified the database configuration since the database was backed up. Use the `restore.sh` script with the `-c` switch to restore the `vertica.conf` file. For example: `restore.sh -c`.
Certain files names are allowed to be present in the top-level catalog directory for a successful bootstrap/restore. Any other files cause the bootstrap/restore to fail, so be sure no files are present, with the exception of the following files, which are allowed and/or needed:
 - Anything starting with `vertica.log`
 - `Epoch.log`
 - `vertica.pid`
 - `not-yet-initialized`
 - `global`
 - `ErrorReport.txt`
 - `SAL`
 - `tmp`
 - `CopyErrorLogs`
 - `Snapshots`
 - Anything starting with `Catalog-old-`
 - `bootstrap-catalog.log`
 - Anything ending in `.conf`
- If you are using external procedures owned by users other than the administrator user, the `backup.sh` script might not have been able to retain the SUID permissions of the procedure executable files. Upon restore, you will need to restore the permissions of the procedure files manually. See Requirements for External Procedures in Programmer's Guide. The output of the backup script will alert you to such discrepancies.

Example

The following example restores the database backed up in the example in *Backing Up the Database* (page 253). Notice the `-S` switch that specifies the snapshot.

```
/opt/vertica/scripts/restore.sh -s wombat-5,wombat-6,wombat-7,wombat-8
-b raster-f1 -B /scratch_b/qa/backup -D /examples/vmartdb -S snap-2010-04-30-1342
```

```
RESTORE_HOSTS      = wombat-5 wombat-6 wombat-7 wombat-8
BACKUP_HOST        = raster-f1
BACKUP_DIR         = /scratch_b/qa/backup
DATABASE_DIR       = /examples/vmartdb
SNAPSHOT           = snap-2010-04-30-1342
TEMP_DIR           = /tmp
RESTORE_VERTICA_CONF = no
```

```
About to begin restore on host wombat-5 by removing catalog
/examples/vmartdb/v_vmartdb_node0013_catalog/Catalog
Continue with restore? [y/n] y
Begin restore on host wombat-5
```

```
Number of files: 746
Number of files transferred: 0
Total file size: 115.16M bytes
Total transferred file size: 0 bytes
Literal data: 0 bytes
Matched data: 0 bytes
File list size: 16374
File list generation time: 0.027 seconds
File list transfer time: 0.000 seconds
Total bytes sent: 45
Total bytes received: 16.40K
```

```
sent 45 bytes  received 16.40K bytes  32.89K bytes/sec
total size is 115.16M  speedup is 7003.01
creating directory /examples/vmartdb/v_vmartdb_node0013_catalog/global
... ok
creating configuration files ... ok
```

Catalog successfully bootstrapped

```
About to begin restore on host wombat-6 by removing catalog
```

```
...
```

Copying a Database to Another Cluster

To copy a database from one cluster to another, use the `copy_vertica_database.sh` script. This script is included in the server RPM located in the `/opt/vertica/scripts` directory.

Requirements

- The script must be run on one of the nodes of the database being backed up.
- The script requires that `rsync 3.0` or later be installed on the database nodes.
- The script must be run as the Linux user who is the database administrator.
- The hosts being copied must be resolvable using their public IP address from the target hosts. (See [Configure Hostname Resolution in Installation Guide](#) for requirements on hostname resolution.)
- The Linux user for the database administrator must have passwordless SSH access from each of the source hosts to the corresponding target hosts.

- The source database must be UP; otherwise the script gets errors.

Before using this script be sure to:

- Create a target database on another cluster.
The target database must use the same name as the source database, contain the same number of nodes and must have at least as much disk space as the source cluster. If the source database uses additional locations for storage (such as mount points), the target database requires an equivalent setup. Be sure to create a temporary directory on each node in the target cluster and assign them all the same name. The name must match the name used for the temporary directories on the source database.
- Stop the database on the target cluster.

It does not matter whether the target database already contains data before you perform the copy operation.

Syntax

```
copy_vertica_database.sh
    -s source_node
    -t target_node
        -d database_name
    | -D database_directory_path
    [ -S snapshot_name ]
    [ -T temporary_directory ]
    [ -u dbadmin_name ]
    [ -w dbadmin_password ]
    [ -o output_file ]
```

Parameters

-s	The name of one of the nodes in the cluster containing the data to be copied. You need specify only one node; the script automatically determines the names of the remaining nodes in the cluster.
-t	The name of a node in the cluster to which the data will be copied. The script determines the name of the remaining nodes in the cluster automatically. The target nodes are automatically paired with the nodes in the source cluster. For example, when copying a database from a cluster containing nodes named node01, node02, and node03 to a cluster containing nodes named node_a, node_b, and node_c, node01 is automatically paired with node_a, node02 is paired with node_b, and so on.
-d	The name of the database to copy. You must either supply the database name using this parameter, or give the path to the database files using the <code>-D</code> parameter.
-D	The absolute path to the parent directory that contains the subdirectories for each node's catalog. You can find this path by viewing the database details (in adminTools, select Advanced > View Database (page 344)). The path you need is shown under Catalog Directory, minus the last directory in the path (which is where the catalogs for the individual nodes are stored). For example, if the path shown under Category Directory is <code>/data/dbs/vmart/v_vmart_node001_catalog</code> , then the path you want is <code>/data/dbs/vmart/</code> . You must use either this parameter or the <code>-d</code> parameter to tell the script which database to copy.

-S	The snapshot name used to identify files that are used in the process of copying the database. If you don't supply this parameter, the script creates a name based on the current date. Tip: To get a full copy, use a unique name each time you run the script. For example, you could incorporate the date into the name. To get an incremental copy, use the same name every time you run the script.
-T	The temporary directory that resides on each node in the target cluster in which to store scripts and logs. This directory must exist on all of the nodes.
-u	The user name for the database administrator.
-w	The password for the database administrator.
-o	Redirects the output of the script to specified file.

Note

- The Linux user for the administrator must have passwordless SSH access from each of the source hosts to the backup host.
- The directory paths on target hosts must be writable by the Linux user for the database administrator.
- If specifying the database paths using the -D parameter, the paths must be identical on the source and target nodes or the script returns an error.
- If you are using external procedures owned by users other than the administrator user, the script might not be able to retain the SUID permissions of the procedure executable files. Upon restore, manually restore the permissions of these files. See Requirements for External Procedures in Programmer's Guide. The output of the script alerts you to such discrepancies.

While copying, you may see the following error message from rsync: `rsync: failed to set times on "/directory": Operation not permitted (1) rsync error: some files/attrs were not transferred (see previous errors)`

These errors occur when rsync attempts to update the datetime stamp on the parent directory of the directory where the data is being stored, but the database administrator doesn't own the parent directory. These errors have no effect on the data copy process and can be ignored.

Note: There is no way to specify the point in time up to which to copy the data; it always picks the latest committed epoch at the start of the script. Therefore, data backed up might not include data that is being loaded after the script starts executing.

Example

The following example copies data from the cluster containing node01 to the cluster containing node_a on the target database located at /home/dbadmin/database:

```
copy_vertica_database.sh -s node01 \
  -t node_a \
  -D /home/dbadmin/database \
  -S VMart \
  -T /tmp \
  -u bGlover \
  -w password6309
```

Best Practices for Disaster Recovery

In order to protect your database from site failures caused by disasters like fire, flood, earthquake or catastrophic power failure, you might consider maintaining an off-site replica of your database as a standby database. In case of disaster, the end-users can fail over to use the standby database.

The solution to employ depends upon two factors that you must determine for your application:

- **Recovery point objective (RPO):** How much data loss is tolerated upon recovery from the disaster
- **Recovery time objective (RTO):** How quickly do you need to make the database available after the disaster

Depending on your RPO and RTO, Vertica recommends choosing from the following solutions:

- 1 **Dual-load:** During the load process for the database, simultaneously load a second database. This could be easily achievable using off-the-shelf ETL software.
- 2 **Periodic Incremental Backups:** Use the procedure described in *Copying a Database to Another Cluster* (page 258) to periodically copy the data to the target database. Remember that the script will only copy files that have changed.
- 3 **Replication solutions provided by Storage Vendors:** If you are using a SAN, evaluate your storage vendor's replication (SRDF) solutions.

The following table discusses the RPO and RTO that can be achieved as well as the pros and cons of the three approaches:

Dual Load	Periodic Incremental Backups	Storage Replication
<p>RPO: Up to the minute data</p> <p>RTO: Available at all times</p>	<p>RPO: Up to last backup</p> <p>RTO: Available except when backup in progress</p>	<p>RPO: Recover to the minute</p> <p>RTO: Available at all times</p>
<p>Pros:</p> <ul style="list-style-type: none"> • Standby database can have different configuration • Can use the standby database for queries • No additional Vertica license costs 	<p>Pros:</p> <ul style="list-style-type: none"> • Built-in scripts • Fast due to compressed file transfers • No additional Vertica license costs 	<p>Pros:</p> <ul style="list-style-type: none"> • Transparent to the database
<p>Cons:</p> <ul style="list-style-type: none"> • Possibly incur addition ETL licenses • Needs application logic to 	<p>Cons:</p> <ul style="list-style-type: none"> • Need identical standby system 	<p>Cons:</p> <ul style="list-style-type: none"> • More expensive • Media corruptions are also

handle errors		replicated
---------------	--	------------

Managing Nodes

Vertica provides the ability to ***add*** (page 264), ***remove*** (page 271), and ***replace*** (page 275) nodes on a live cluster that is actively processing queries. This ability lets you scale the database without interrupting users.

You might also consider ***refreshing*** (page 280) or ***dropping*** (page 281) projections.

Adding Nodes

There are many reasons for adding one more more nodes to an installation of Vertica:

- Increase system performance. Add additional nodes due to a high query load or load latency or increase disk space without adding storage locations to existing nodes.
- Increase storage. Permanently add single nodes to a cluster at some rate over time. That rate of adding new nodes is determined by the rate at which the available capacity decreases.
- Make the database K-safe (K-safety=1) or increase K-safety to 2. Increasing K-safety to one (1) allows the database to lose a node and keep running as if nothing happened. Increasing K-safety to two (2) allows Vertica to run normally if up to two nodes fail. See **Failure Recovery** (page 235) for details.
- Swap a node for maintenance: Use a spare machine to temporarily take over the activities of an existing node that needs maintenance. The node that requires maintenance is known ahead of time so that when it is temporarily removed from service, the cluster is not vulnerable to additional node failures.
- Replace a node. Permanently add a node to remove obsoleted hardware.

IMPORTANT: If you installed Vertica on a single node without specifying the IP address or hostname (you used `localhost`), it is not possible to expand the cluster. You must reinstall Vertica and specify an IP address or hostname.

Adding nodes consists of the following general tasks:

1 Backing up the database (page 253).

Vertica strongly recommends that you back up the database before you perform this significant operation because it entails creating new projections, refreshing them, and then deleting the old projections.

The process of migrating the projection design to include the additional nodes could take a while; however during this time, all user activity on the database can proceed normally, using the old projections.

2 Configuring the hosts you want to add to the cluster.

See Before you Install in the Installation Guide.

3 Adding one or more hosts to the cluster (page 265).

4 Adding the hosts (page 267) you added to the cluster (in step 3) to the database

Note: When a host is added to the database, it becomes a node.

5 Distributing configuration files to the new host (page 267).

6 Rebalancing (page 268) the projection design data to be distributed across all nodes.

If going from a single node database to a cluster, consider increasing K-safety.

7 Testing the modified database design (page 282).

8 Dropping the original, unused projections (page 281) (if following a manual process).

Adding Hosts to a Cluster

After you have backed up the database and configured the hosts you want to add to the cluster, you can now add hosts to the cluster using the `update_vertica` script.

Prerequisites and Restrictions

- Follow the instructions in Configure Hostname Resolution to ensure that the newly-added hosts are reachable by the existing nodes in the cluster.
- If you installed Vertica on a single node without specifying the IP address or hostname (you used localhost), it is not possible to expand the cluster. You must reinstall Vertica and specify an IP address or hostname.
- If your database has more than one node already, you can add a node without stopping the server. However, if you are adding a node to a single-node, non-localhost installation, you must shut down both the database and spread. If you do not, the system returns an error like the following:

```
[dbadmin@node01 tmp]$ sudo /opt/vertica/sbin/update_vertica -A node0x0
-rvertica-3.5.1-20091002010202.x86_64.RHEL5.DBG.rpm -u release -p
  V3rtlqa -PV3rtlqa
Vertica Analytic Database 3.5.1-20091002010202 Installation Tool
Starting installation tasks...
Getting system information for cluster (this may take a while)....
Spread is running on ['node01']. Vertica and spread must be stopped before
  adding nodes to a 1 node cluster.
Use the admin tools to stop the database, if running, then use the
  following command to stop spread:
    /etc/init.d/spread stop (as root or with sudo)
Installation completed with errors.
Installation failed.
```

Procedure to Add Hosts

From one of the existing cluster hosts, run the `update_vertica` script with a minimum of the `-A` parameter (where `host` is the hostname or IP address of the system you are adding to the cluster) and the `-r` parameter:

```
# /opt/vertica/sbin/update_vertica -A hostname -r rpm_package
```

Note: See The `install_vertica` Script for the full list of parameters.

The `update_vertica` script uses all the same options as `install_vertica` and:

- Installs the Vertica RPM on the new host.
- Performs post-installation checks, including RPM version and N-way network connectivity checks.
- Modifies spread to encompass the larger cluster.
- Configures the **Administration Tools** (page 336) to work with the larger cluster.

Important Tips:

- A host can be specified by the hostname or IP address of the system you are adding to the cluster.
- Do not use include spaces in the hostname list provided with `-A` if you specified more than one host.
- If a new RPM is specified, Vertica first installs it on the existing cluster hosts before the newly-added hosts.
- Use the same command line parameters you used when you installed the original cluster. Specifically, if you used non-default values for the database administrator username, password, or directory path, provide the same when you add new hosts; otherwise the procedure will not work. It may be beneficial to create a properties file to save the parameters during install and then re-using it on subsequent install and update operations. See *Installing Vertica Silently*.
- If you are installing using `sudo`, the database administrator user (`dbadmin`) must already exist on the hosts you are adding and must be configured with passwords and home directory paths identical to the existing hosts. Vertica sets up passwordless ssh from existing hosts to the new hosts, if needed.
- If you initially used the `-T` option to configure spread to use direct, point-to-point communication between nodes on the subnet, and you want to continue to do so, use the `-T` option when you add the new host. Otherwise, the entire cluster is reconfigured to use the default UDP broadcast.

Examples:

```
-A host01
-A 192.168.233.101
-A host02,host03
```


Adding Hosts to a Database

Once you have added one or more hosts to the cluster, you can add them to the database.

To add one or more hosts to a database:

- 1 Open the Administration Tools. (See *Using the Administration Tools* (page 329).)
- 2 On the **Main Menu**, select **View Database Cluster State** to verify that the database is running. If it is not, start it.
- 3 From the **Main Menu**, select **Advanced** and click **OK**.
- 4 In the **Advanced Menu**, select **Cluster Management** and click **OK**.
- 5 In the **Cluster Management** menu, select **Add Host(s)** and click **OK**.
- 6 Select the database to which you want to add one or more hosts, and then select **OK**.
A list of unused hosts is displayed.
- 7 Select the hosts you want to add to the database and click **OK**.
- 8 When prompted, click **Yes** to confirm that you want to add the hosts.
- 9 When prompted, enter the password for the database, and then select **OK**.
- 10 When prompted that the hosts were successfully added, select **OK**.
- 11 **Rebalance your data** (page 268) across all the nodes.

Distributing Configuration Files to the New Host

Once you have created the new host, you need to distribute the configuration file (`vertica.conf`) to the new host. If you are using Secure Socket Layer (SSL) you also need to distribute the SSL configuration to the new host.

To distribute these files to the new host:

- 1 Log on to a host that contains these files and start the Administration Tools.
See *Using the Administration Tools* (page 329) for information about accessing the Administration Tools.
- 2 On the **Main Menu** in the Administration Tools, select **Configuration Menu** and click **OK**.
- 3 On the **Configuration Menu**, select **Distribute Config Files** and click **OK**.
- 4 Select **Database Configuration**.
- 5 Select the database in which you want to distribute the files and click **OK**.
The `vertica.conf` file is distributed to all the other hosts in the database. If it previously existed on a host, it is overwritten.
- 6 On the **Configuration Menu**, select **Distribute Config Files** and click **OK**.
- 7 Select **SSL Keys**.
The certifications and keys for the host are distributed to all the other hosts in the database. If they previously existed on a host, they are overwritten.
- 8 On the **Configuration Menu**, select **Distribute Config Files** and click **OK**.
Select **AdminTools Meta-Data**.
The Administration Tools metadata is distributed to every host in the cluster.
- 9 **Restart the database** (page 248).

Rebalancing Data Across Nodes

If you used the Administration Tools UI to **add hosts to your database** (page 267), the UI next prompts you to rebalance the existing data across all nodes, including newly-added nodes.

In one simple click, data rebalancing redistributes your database projections' data across all nodes, refreshes projections, sets the Ancient History Mark, and drops projections that are no longer needed.

The Administration Tools UI offers two choices:

- **Automatically rebalance the data** (page 270) immediately using the Administration Tools.
 - For segmented projections, creates new (renamed), segmented projections that are identical in structure to the existing projections, but data is distributed cross all nodes; then it refreshes them, sets the Ancient History Mark (AHM) to the greatest allowable epoch (now), and drops old segmented projections.
 - For unsegmented projections, leaves existing projections unmodified, creates new projections on the new nodes, and refreshes them. No projections are dropped.
- **Generate a script** (page 270) that you can run manually at a later time.

Note: Even if you decide to rebalance the data immediately, the system still generates a script called `extend_catalog_rebalance.sql`.

The following code fragment is an example based on the vmart database, after a single host was added to a three-node cluster. If your projections are segmented, when you run the script, it creates new, segmented projections, refreshes all projections, sets the AHM, and drops the old projections.

```
CREATE PROJECTION call_center_dimension_DBD_32_seg_rebalance_design_rebalance_design_v1 (
  call_center_key ENCODING COMMONDELTA_COMP,
  cc closed date,
  cc open date,
  cc_name,
  cc_class,
  cc_employees,
  cc_hours,
  cc manager,
  cc address,
  cc_city,
  cc_state,
  cc region
)
AS
SELECT call_center_dimension.call_center_key,
  call_center_dimension.cc_closed_date,
  call_center_dimension.cc_open_date,
  call_center_dimension.cc_name,
  call_center_dimension.cc_class,
  call_center_dimension.cc_employees,
  call_center_dimension.cc_hours,
  call_center_dimension.cc_manager,
  call_center_dimension.cc_address,
  call_center_dimension.cc_city,
  call_center_dimension.cc_state,
  call_center_dimension.cc_region
FROM online_sales.call_center_dimension
```

```

ORDER BY call_center_dimension.call_center_key
SEGMENTED BY hash(call_center_dimension.call_center_key) ALL NODES OFFSET 0;

CREATE PROJECTION call_center_dimension_DBD_8_seg_rebalance_design_rebalance_design_v1(
  call_center_key ENCODING COMMONDELTA_COMP,
  cc_closed_date,
  cc_open_date,
  cc_name,
  cc_class,
  cc_employees,
  cc_hours,
  cc_manager,
  cc_address,
  cc_city,
  cc_state,
  cc_region
)
AS
  SELECT call_center_dimension.call_center_key,
         call_center_dimension.cc_closed_date,
         call_center_dimension.cc_open_date,
         call_center_dimension.cc_name,
         call_center_dimension.cc_class,
         call_center_dimension.cc_employees,
         call_center_dimension.cc_hours,
         call_center_dimension.cc_manager,
         call_center_dimension.cc_address,
         call_center_dimension.cc_city,
         call_center_dimension.cc_state,
         call_center_dimension.cc_region
  FROM online_sales.call_center_dimension
  ORDER BY call_center_dimension.call_center_key
  SEGMENTED BY hash(call_center_dimension.call_center_key) ALL NODES OFFSET 1;

select refresh('online_sales.call_center_dimension');

select make_ahm_now();

DROP PROJECTION online_sales.call_center_dimension_DBD_32_seg_rebalance_design_rebalance_design,
online_sales.call_center_dimension_DBD_8_seg_rebalance_design_rebalance_design CASCADE;
...

```

Notes and Restrictions

- Only the superuser has permissions to rebalance data.
- Before data rebalancing completes, Vertica operates with the existing k-safe value. After rebalance completes, Vertica operates with the k-safe value specified during the rebalance operation.
- You can maintain existing K-safety or specify a new value (0 to 2) for the modified database cluster. Vertica does not support downgrading K-safety and returns a warning if you attempt to reduce it from its current value: Design k-safety cannot be less than system k-safety level.
- Rebalancing data does not work on range segmented projections. If such projections exist and you upgrade K-safety using the Re-Balance Data utility, the system attempts to rebalance the data across all nodes, but it cannot be marked K-safe. You must manually rebalance range segmented projections and then reset the system K-safety. See ***Modifying Database Designs for Updated Nodes*** (page 279).

- If a failure occurs during a data rebalance, the superuser can rebalance again (using either the Administration Tools or the script), and if the issues were resolved, the rebalance operation continues from where it failed. A failed data rebalance could leave projections in a non up-to-date state, which are not automatically removed. You must remove those projections manually using the DROP PROJECTION function. See **Refreshing Projections** (page 280) and **Dropping Projections** (page 281) in this guide.
- To find only the non up-to-date projections, query the V_CATALOG.PROJECTIONS system table. For example:

```
=> SELECT projection_name, anchor_table_name, is_prejoin, is_up_to_date  
      FROM projections WHERE is_up_to_date = false;
```

Rebalancing Data Using the Administration Tools UI

The following procedure assumes you are continuing from the final step in **Adding Hosts to a Database** (page 267) and that you are rebalancing your data now.

- 1 In the **Re-balance Data** window, select **Automatically re-balance data across all nodes** and click **OK**.
- 2 Select the database and click **OK**.
- 3 Enter the directory for the Database Designer outputs (for example `/tmp`) and click **OK**.
- 4 Accept the proposed K-safety value or provide a new value. Valid values are 0 to 2.
- 5 Review the message and click **Proceed** to begin rebalancing data.

The Database Designer modifies existing projections in order to rebalance data across all nodes in the database with the K-safety you provided. A script to rebalance data, which you can run manually at a later time, is also generated and resides in the path you specified; for example `/tmp/extend_catalog_rebalance.sql`.

IMPORTANT: Rebalancing data can take some time, depending on the number of projections and the amount of data in them. Vertica recommends that you allow the process to complete. If you must cancel the operation, use Ctrl+C.

The terminal window notifies you when the rebalancing operation is complete.

- 6 Press **Enter** to return to the Administration Tools.

Generating a Rebalance Script for Later Use

To generate a script for later use:

- 1 In the **Re-balance Data** window, select **Generate the script ...** and click **OK**.
- 2 Select the database and click **OK**.
- 3 Enter the directory for the Database Designer outputs (for example `/tmp`) and click **OK**.
- 4 Accept the proposed K-safety value or provide a new value. Valid values are 0 to 2.
- 5 Review the message and click **Proceed** to generate the script.

The script is saved to the path you specified in the previous step. For example `/tmp/extend_catalog_rebalance.sql`.

Removing Nodes

Although less common than adding a node, permanently removing a node is useful if the host system is obsolete or over-provisioned.

Removing one or more nodes consists of the following general steps:

- 1 **Backing Up the Database** (page 253).
Vertica recommends that you back up the database before performing this significant operation because it entails creating new projections, deleting old projections, and reloading data.
- 2 Creating projections that exclude the nodes that you want to remove from the database, and then refreshing data within these projections. See **Modifying Database Designs for Updated Nodes** (page 279).
- 3 **Testing the modified database design** (page 282).
- 4 Dropping the original, unused projections as described in **Dropping Projections** (page 281).
Note: Be sure to follow this procedure or Vertica might not allow you to drop some projections.
- 5 **Removing hosts from the database** (page 272).
- 6 **Removing hosts from the cluster** (page 273) if they are not used by any other databases.

Tip: You can perform steps 2 - 5 incrementally for each projection set you need to replace.
For example, you might want to create, test, and drop one projection and its buddy projections at a time. This would reduce the amount of disk space required to perform the overall operation to the space required by the largest projection set.

Removing Hosts from a Database

Prerequisites

- The node must be empty, in other words there should be no projections referring to the node. Ensure you have followed the steps listed in **Removing Nodes** (page 271) to modify your database design.
- The database must be UP.

Procedure to Remove Hosts

To remove one or more unused hosts from the database:

- 1 Open the Administration Tools. See **Using the Administration Tools** (page 329) for information about accessing the Administration Tools.
- 2 On the **Main Menu**, select **View Database Cluster State** to verify that the database is running. If the database isn't running, start it.
- 3 From the **Main Menu**, select **Advanced**, and then select **OK**.
- 4 In the **Advanced** menu, select **Cluster Management**, and then select **OK**.
- 5 In the **Cluster Management** menu, select **Remove Host(s) from Database**, and then select **OK**.
- 6 When warned that you must redesign your database and create projections that exclude the hosts you are going to drop, select **Yes**.
- 7 Select the database from which you want to remove the hosts, and then select **OK**.
A list of all the hosts that are currently being used is displayed.
- 8 Select the hosts you want to remove from the database, and then select **OK**.
- 9 When prompted, select **OK** to confirm that you want to remove the hosts.
- 10 Enter the password for the database, and select **OK**.
- 11 When prompted that the hosts were successfully removed, select **OK**.

Removing Hosts from a Cluster

If a host that you removed from the database is not used by any other database, you can remove it from the cluster using the `update_vertica` script. You can leave the database running (UP) during this operation.

Prerequisites

The host must not be used by any database

Procedure to Remove Hosts

From one of the hosts in the cluster, run `update_vertica` with the `-R` switch, where `-R` specifies a comma-separated list of hosts to remove from an existing Vertica cluster. A host can be specified by the hostname or IP address of the system.:

```
# /opt/vertica/sbin/update_vertica -R host
```

For example:

```
# /opt/vertica/sbin/update_vertica -R R host01,Host01.vertica.com
```

Note: See The `install_vertica` Script for the full list of parameters.

The `update_vertica` script uses all the same options as `install_vertica` and:

- Modifies the spread to match the smaller cluster.
- Configures the Administration Tools to work with the smaller cluster.

Important Tips:

- A host does not need to be functional, or even exist, to be removed as long as the database design no longer includes a node on it. Specify the hostname or IP address that you used originally for the installation. Adding hosts to and removing them from VM-based clusters can lead to a situation in which a host doesn't exist.
- If you have identified a node using various forms of the hostname and IP address, you must identify all the forms you used. For example, you might identify a node with both short and fully-qualified names. Use a comma-separated list to identify two or more forms of the host name.
- Do not include spaces in the hostname list provided with `-R` if you specified more than one host.
- If a new RPM is specified, Vertica will first install it on the existing cluster hosts before proceeding.
- Use the same command line parameters as those used when you installed the original cluster. Specifically if you used non-default values for the database administrator username, password, or directory path, provide the same when you remove hosts; otherwise, the procedure fails. Consider creating a properties file in which you save the parameters during the installation, which you can reuse on subsequent install and update operations. See [Installing Vertica Silently](#).

Examples:

```
-R host01  
-R 192.168.233.101  
-R host01,Host01.vertica.com
```


Replacing Nodes

If you have a K-Safe database, you can replace nodes as necessary without bringing the system down. For example, you might want to replace an existing node if you:

- Need to repair an existing host system that no longer functions and restore it to the cluster
- Want to exchange an existing host system for another more powerful system

Note: Replacing a node is not supported with a k-safety=0 database. Use the procedures to **add** (page 264) and **remove** (page 271) nodes instead.

The process you use to replace a node depends on whether you are replacing the node with:

- A host that uses the same name and IP address
- A host that uses a different name and IP address

Prerequisites:

- Configure the replacement hosts for Vertica. See *Before you Install* in the Installation Guide.
- Read the Important Tips sections under **Adding Hosts to a Cluster** (page 265) and **Removing Hosts from a Cluster** (page 273).
- Ensure that the database administrator user exists on the new host and is configured identically to the existing hosts. Vertica will setup passwordless ssh as needed.
- Ensure that the directories for Catalog Path, Data Path and any storage locations added to the database are created and/or mounted correctly on the new host and have read and write access permissions for the database administrator user. Also ensure that there is sufficient disk space.
- Follow the best practice procedure below for introducing the failed hardware back into the cluster to avoid spurious full-node rebuilds.

Best Practice for Restoring Failed Hardware

Following this procedure will prevent Vertica from misdiagnosing missing disk or bad mounts as data corruptions, which would result in a time-consuming, full-node recovery.

If a server fails due to hardware issues, for example a bad disk or a failed controller, upon repairing the hardware:

- 1 Reboot the machine into runlevel 1, which is a root and console-only mode.
Runlevel 1 prevents network connectivity and keeps Vertica from attempting to reconnect to the cluster.
- 2 In runlevel 1, validate that the hardware has been repaired, the controllers are online, and any RAID recover is able to proceed.

Note: You do not need to initialize RAID recover in runlevel 1; simply validate that it can recover.

- 3 Once the hardware is confirmed consistent, only then reboot to runlevel 3 or higher.

At this point, the network activates, and Vertica rejoins the cluster and automatically recovers any missing data.

Replacing a Node with a Host that Uses the Same Name and IP Address

To replace a node with a host system that has the same IP address and host name as the original:

- 1 From a functioning node in the cluster, run the `install_vertica` script with the `-s` parameter:

```
# /opt/vertica/sbin/install_vertica -s host
```

Where `host` is the hostname or IP address of the system you are restoring to the cluster; for example:

```
-s host01
```

```
-s 192.168.233.101
```

The installation script verifies system configuration and that Vertica, spread, and the Administration Tools metadata are installed on the host.

- 2 Use the procedure in ***Distributing configuration files to the new host*** (page 267) to transfer metadata to the new host.
- 3 Use the Administration Tools to restart the host you just replaced.

The node automatically joins the database and recovers its data by querying the other nodes within the database. It then transitions to an UP state.

Note: Do not connect two hosts with the same name and IP address to the same network. If this occurs, traffic is unlikely to be routed properly.

Replacing a Failed Host with a Host that Uses a Different Name and IP Address

Replacing a failed node with a host system that has a different IP address and host name from the original consists of the following steps:

- 1 ***Back up the database*** (page 253).

Vertica recommends that you back up the database before you perform this significant operation because it entails creating new projections, deleting old projections, and reloading data.

- 2 Run `update_vertica` with the `-A`, `-R`, `-E` and `-r` parameters to replace the failed host:

```
# /opt/vertica/sbin/update_vertica -A NewHostName -R OldHostName -E -r rpm_package
```

Where:

- `NewHostName` is the hostname or IP address of the system you are adding to the cluster.
- `OldHostName` is the hostname or IP address of the system you are removing from the cluster.
- The `-E` parameter forces Vertica to drop the failed node from the cluster.
- `-r` is the name of the rpm package; for example `-r vertica_4.1.x.x86_64.RHEL5.rpm`

Note: The `update_vertica` script uses all the same options as `install_vertica`. See **The `install_vertica` Script** for the full list of parameters.

- 3 Use the Administration Tools to replace the original host with the new host. If you are using more than one database, replace the original host in all the databases in which it is used. See ***Replacing Hosts*** (page 277).

- 4 Use the procedure in ***Distributing Configuration Files to the New Host*** (page 267) to transfer metadata to the new host.
- 5 Run `update_vertica` again with just the `-R` parameter to clear the node that you replaced from the Administration Tools metadata.

```
# /opt/vertica/sbin/update_vertica -R OldHostName
```

`OldHostName` is the hostname or IP address of the system you removed from the cluster.

- 6 Use the Administration Tools to restart Vertica on the host. On the **Main Menu**, select **Restart Vertica on Host**, and click **OK**. See ***Starting a Database*** (page 337) for more information.

Once you have completed this process, the replacement node automatically recovers the data that was stored in the original node by querying other nodes within the database.

Replacing a Functioning Node with a Host that Uses a Different Name and IP Address

Replacing a node with a host system that has a different IP address and host name from the original consists of the following general steps:

- 1 ***Back up the database*** (page 253).

Vertica recommends that you back up the database before you perform this significant operation because it entails creating new projections, deleting old projections, and reloading data.

- 2 ***Add the replacement hosts to the cluster*** (page 265).

At point, both the original host that you want to remove and the new replacement host are members of the cluster.

- 3 Use the Administration Tools to shut down the original host.

- 4 Use the Administration Tools to ***replacing the original host*** (page 277) with the new host. If you are using more than one database, replace the original host in all the databases in which it is used.

- 5 ***Remove the host from the cluster*** (page 273).

- 6 Restart Vertica on the host.

Once you have completed this process, the replacement node automatically recovers the data that was stored in the original node by querying the other nodes within the database. It then transitions to an UP state.

Note: If you do not remove the original host from the cluster and you attempt to restart the database, the host is not invited to join the database because its node address does not match the new address stored in the database catalog. Therefore, it remains in the INITIALIZING state.

Replacing Hosts

If you are replacing a node with a host that uses a different name and IP address, use the Administration Tools to replace the original host with the new host.

To replace the original host with a new host:

- 1 Open the Administration Tools.

- 2 On the **Main Menu**, select **View Database Cluster State** to verify that the database is running. If it's not running, use the Start Database command on the Main Menu to restart it.
- 3 On the **Main Menu**, select **Advanced Menu**.
- 4 In the **Advanced Menu**, select **Stop Vertica on Host**.
- 5 Select the host you want to replace, and then click **OK** to stop the node.
- 6 When prompted if you want to stop the host, select **Yes**.
- 7 In the **Advanced Menu**, select **Cluster Management**, and then click **OK**.
- 8 In the **Cluster Management** menu, select **Replace Host**, and then click **OK**.
- 9 Select the database that contains the host you want to replace, and then click **OK**.
A list of all the hosts that are currently being used displays.
- 10 Select the host you want to replace, and then click **OK**.
- 11 Select the host you want to use as the replacement, and then click **OK**.
- 12 When prompted, enter the password for the database, and then click **OK**.
- 13 When prompted, click **Yes** to confirm that you want to replace the host.
- 14 When prompted that the host was successfully replaced, click **OK**.
- 15 In the **Main Menu**, select **View Database Cluster State** to verify that all the hosts are running. You might need to start Vertica on the host you just replaced. Use **Restart Vertica on Host**. The node enters a RECOVERING state.

Caution: If you are using a K-Safe database, keep in mind that the recovering node counts as one node down even though it might not yet contain a complete copy of the data. This means that if you have a database in which $K \text{ safety}=1$, the current fault tolerance for your database is at a critical level. If you lose one more node, the database shuts down. Be sure that you do not stop any other nodes.

Modifying Database Designs for Updated Nodes

As previously discussed, you can modify a database design in one simple step using the Administration Tools UI or a script that the Administration Tools generates. See **Rebalancing Data Across Nodes** (page 268) for details.

There are situations, however, in which automatic data rebalancing does not work:

- A node has been removed from the database cluster (projections cannot include the removed node)
- The design was not created by the Database Designer using the Administration Tools UI
- You want to add custom projections to your design
- You want to export a design to recreate it on another cluster
- You have custom projections (such as for a specific query) that you want to add to an existing database design
- Range segmented projections exist on one or more nodes in the cluster.

For each of the above scenarios, you need to modify the database design manually using the procedures described in this section:

- 1 **Create new projections** (page 279)
- 2 **Refresh projections** (page 280)
- 3 **Drop unused projections** (page 281)

Creating New Projections

- 1 Create new projections.
 1. For each existing table with segmented projections, create a new projection that spans all the nodes within the database.
 2. For each existing table with replicated projections, add an unsegmented (replicated) projection on each additional node being added.
 3. Copy existing projection statements, modify them as necessary, and run the new projection statements.

TIP: For syntax and examples of creating both segmented and replicated projections on a new node, see **Add Node to a Database** (page 110). (See also **Creating Custom Designs** (page 91) for a more comprehensive overview.)

- 2 Generate a script that contains the catalog, system schemas, system views, system tables, and projections for the tables.

```
SELECT EXPORT_CATALOG('filename','design_all')
```

For `filename`, specify the name of the file you want to create.

- 3 Open the script and locate each segmented and unsegmented projection.

Projections take the form:

```
CREATE PROJECTION projection_name (...)
AS SELECT ...
[ SEGMENTED | UNSEGMENTED ] ALL NODES;
```

- 4 Create a copy of each projection for each additional node using the following form:

```
CREATE PROJECTION projection_name_<NewNodeName>( ... )
AS SELECT ...
[ SEGMENTED | UNSEGMENTED ] [ ALL NODES | NODE <NewNodeName> ] ;
```

Where `NewNodeName` is the name of the node you added to the cluster, which you can find by querying the `NODE_RESOURCES` system table, described in the SQL Reference Manual.

- 5 Examine the projections in the `PROJECTIONS` system table and ensure that the `VERIFIED_FAULT_TOLERANCE` column is greater than what you require. If it is not then check that each projection has the appropriate buddy projections defined. This is particularly relevant if you are increasing k-safety.
- 6 Load data into the new projections as described in *Refreshing Projections* (page 280).

Refreshing Projections

New projections are out-of-date (not available for query processing) until they are refreshed.

- 1 Use the Administration Tools to verify that all nodes are up. (On the Administration Tool's **Main Menu**, select **View Database Cluster State**, and then click **OK**.) If they are not up, bring them up before you proceed.
- 2 Use the `START_REFRESH` function described in the SQL Reference Manual to copy data into the new projection from other projections.
A refresh runs simultaneously on all nodes. During a refresh, the new projection:
 - Cannot participate in query execution.
 - Cannot be used as a buddy of another projection.
- 3 Optionally monitor the progress of the refresh operation by viewing the following system tables:
 - The `IS_UP_TO_DATE` column in the `PROJECTIONS` table
 - The `REFRESH_STATUS` and `IS_EXECUTING` columns in the `PROJECTION_REFRESHES` tableThe refresh status can be *queued*, *refreshing*, *refreshed*, or *failed*.
`IS_EXECUTING` indicates if the refresh is currently running or if the entry pertains to a prior attempt that could have succeeded or failed.
You can also use the `LOCKS` system table to determine if a refresh has been blocked on a table lock.
- 4 Optionally use the `REFRESH` function, which invokes refresh synchronously rather than as a background process.

Notes

Information about a refresh operation—whether successful or unsuccessful—is maintained in the `PROJECTION_REFRESHES` system table until either the `CLEAR_PROJECTION_REFRESHES()` function is executed or the storage quota for the table is exceeded. The `PROJECTION_REFRESHES.IS_EXECUTING` column returns a boolean value that indicates whether the refresh is currently running (t) or occurred in the past (f).

When a new projection is created and refreshed, Vertica does not reconstruct historical data for that projection unless it is a buddy projection of an existing projection. The newly-refreshed projection maintains history only beyond the epoch in which the refresh operation commits. (See `SYSTEM.REFRESH_EPOCH` in the SQL Reference Manual.) Therefore, the query optimizer cannot choose the new projection for AT EPOCH queries that request historical data at epochs older than the refresh epoch.

Dropping Projections

To manually drop a projection that is no longer being used:

- 1 Use the Administration Tools to verify that all nodes are up. (On the Administration Tool's **Main Menu**, select **View Database Cluster State**, and then click **OK**.) If they are not up, bring them up before you proceed.
- 2 Call the `MAKE_AHM_NOW` function, described in the SQL Reference Manual.
 - `MAKE_AHM_NOW` sets the Ancient History Mark (AHM) to the greatest allowable value and lets you drop any projections that existed before the issue occurred.
 - If you do not set the AHM to the current epoch, Vertica could prevent the drop if it violates K-safety.
 - Optionally, you can use the `true` parameter (`SELECT MAKE_AHM_NOW(true);`) to allow the AHM to advance when nodes are down, but note that if the AHM advances after the last good epoch of the failed nodes, those nodes must recover all data from scratch. Use this option with care.
 - Once you have set the AHM, you can no longer perform historical queries based on any date that precedes the date you set for the AHM.
- 3 Use the `DROP_PROJECTION` statement to drop the projection. If it has buddies, drop the buddy projections as well.

For example, the following statement drops the `schema.fact_proj_a` and `schema.fact_proj_b` projections:

```
DROP PROJECTION schema1.fact_proj_a, schema1.fact_proj_b;
```

Testing Modified Database Designs

Testing a new database design consists of verifying that the new projections:

- Are safe and up-to-date.
Examine the projections using the PROJECTIONS system table and ensure that the VERIFIED_FAULT_TOLERANCE column is greater than what you require. If it is not then check that each projection has the appropriate buddy projections defined. This is particularly relevant if you are increasing K-safety.
- Provide the same results as the original projections. (See Comparing Projections below.)

Comparing Projections

To verify that the new projections produce the same results as the original projections:

- 1 Run reports using each set of projections and compare the reports. (Save the reports so you can use them for further comparisons after you delete the original projections.)
- 2 Create queries and run them on both the original and new projections.

For example:

```
SELECT "P1", COUNT(*) FROM P1
UNION ALL
SELECT "P2", COUNT(*) FROM P2;
```

In the above example, P1 is the name of the original projection and P2 is the name of the new projection.

Managing Disk Space

Vertica detects and reports low disk space conditions in the log file so that the issue can be addressed before serious problems occur. It also detects and reports low disk space conditions via **SNMP traps** (page 212) if enabled.

Critical disk space issues are reported sooner than other issues. For example, running out of catalog space is fatal; therefore, Vertica reports the condition earlier than less critical conditions. To avoid database corruption when the disk space falls beyond a certain threshold, Vertica begins to reject transactions that update the catalog or data.

Caution: A low disk space report indicates one or more hosts are running low on disk space or have a failing disk. It is imperative to add more disk space (or replace a failing disk) as soon as possible.

When Vertica reports a low disk space condition, use the `DISK_RESOURCE_REJECTIONS` system table to determine the types of disk space requests that are being rejected and the hosts on which they are being rejected.

These and the other **SQL Monitoring API** (page 199) system tables are described in detail in the SQL Reference Manual.

To add disk space, see **Adding Disk Space to a Node** (page 283) or **Adding Disk Space Across the Cluster** (page 284). To replace a failed disk, see **Replacing Failed Disks** (page 285).

Monitoring Disk Space Usage

You can use these system tables to monitor disk space usage on your cluster:

System Table	Description
<code>DISK_STORAGE</code>	Monitors the amount of disk storage used by the database on each node.
<code>COLUMN_STORAGE</code>	Monitors the amount of disk storage used by each column of each projection on each node.
<code>PROJECTION_STORAGE</code>	Monitors the amount of disk storage used by each projection on each node.

Adding Disk Space to a Node

This procedure describes how to add disk space to a node in the Vertica cluster:

- 1 Shut down Vertica on the host where disk space is being added.
- 2 Turn off the system, if required by your hardware environment.
- 3 Insert the new disk.
- 4 Power on the system, if required.
- 5 Partition, format, and mount the new disk, as required by the hardware environment.

For example:

```
mount /myNewPath/ on new volume
```

- 6 Create a data directory path on the new volume.

For example:

```
mkdir -p /myNewPath/myDB/host01_data2/
```

- 7 Restart Vertica on the host.

- 8 Open a database connection on the host where you installed the new disk and add a storage location. This adds the new data directory path to the host.

See **Adding Storage Locations** (page 286) in this guide and the ADD_LOCATION function in the SQL Reference Manual.

- 9 **Note:** ADD_LOCATION is a local command, which must be run on each node to which space is added.

Adding Disk Space Across the Cluster

Use the following procedures to add disk space to all sites in an optimal cluster environment.

If the cluster can be taken offline:

- 1 Shut down the cluster from the Administration Tools.
- 2 Use the following steps to add a disk on each host:
 1. Shut down Vertica.
 2. Power off system, if required by your hardware environment.
 3. Insert the new disk.
 4. Power on the system, if required.
 5. Partition/format/mount the new disk, if required by the hardware environment.
 6. Create a database path on the new volume.
- 3 Start the Vertica database from Administrative Tools.
- 4 For each host, open a database connection and add a storage location. This adds the new data directory path to the host.

See **Adding Storage Locations** (page 286) in this guide and the ADD_LOCATION function in the SQL Reference Manual.

Note: ADD_LOCATION is a local command, which must be run on each host to which space is added.

If the cluster cannot be taken offline (and assuming K-safety=1 or higher) :

Use the following procedure for each node, one node at a time:

- 1 Shut down Vertica on the host where disk space is being added.
- 2 Turn off the system, if required by your hardware environment.
- 3 Insert the new disk.
- 4 Power on the system, if required.
- 5 Partition, format, and mount the new disk, as required by the hardware environment.

For example:

```
mount /myNewPath/ on new volume
```

- 6 Create a data directory path on the new volume.

For example:

```
mkdir -p /myNewPath/myDB/host01_data2/
```

- 7 Restart Vertica on the host.
- 8 Open a database connection on the host where you installed the new disk and add a storage location. This adds the new data directory path to the host.
See **Adding Storage Locations** (page 286) in this guide and the ADD_LOCATION function in the SQL Reference Manual.
- 9 **Note:** ADD_LOCATION is a local command, which must be run on each node to which space is added.

Replacing Failed Disks

If the disk on which the data or catalog directory resides fails, causing full or partial disk loss, perform the following steps:

- 1 Replace the disk and recreate the data or catalog directory.
- 2 Distribute the configuration file (`vertica.conf`) to the new host. See **Distributing Configuration Files to the New Host** (page 267) for details.
- 3 Restart the database following the steps in **Restarting the Database** (page 248).

See **Catalog and Data Files** (page 285) for information about finding your DATABASE_HOME_DIR.

Catalog and Data Files

For the recovery process to complete successfully, it is essential that catalog and data files be in the proper directories.

In Vertica, the catalog is a set of files that contains information (metadata) about the objects in a database, such as the nodes, tables, constraints, and projections. The catalog files are replicated on all nodes in a cluster, while the data files are unique to each node. These files are installed by default in the following directories:

```
/DATABASE_HOME_DIR/DATABASE_NAME/v_db_nodexxxx_catalog/  
/DATABASE_HOME_DIR/DATABASE_NAME/v_db_nodexxxx_catalog/
```

Note: DATABASE_HOME_DIR is the path, which you can see from the Administration Tools. See **Using the Administration Tools** (page 329) in the Administrator's Guide for details on using the interface.

To view the path of your database:

- 1 Run the Administration Tools.
`#/opt/vertica/bin/admintools`
- 2 From the Main Menu, select **Configuration Menu** and click **OK**.
- 3 Select **View Database** and click **OK**.

- 4 Select the database you want would like to view and click **OK** to see the database profile.

The following example includes the kind of information typically returned: database path name, the host or hosts your database is running on (in this example a 4-node cluster), the port, and that your database is K-Safe:

```
Database: VMart_Schema
Database Log: /scratch_b/qa//VMart_Schema/dbLog,
/scratch_b/qa//VMart_Schema/v_vmart_schema_node0001_catalog/vertica.log
Hosts:
cent5-1.verticacorp.com,cent5-2.verticacorp.com,cent5-3.verticacorp.com,ce
nt5-4.verticacorp.com
Restart Policy: ksafe
Port: 5433
```

80%

< OK >

Creating and Configuring Storage Locations

Each node must contain a minimum of one storage location in which to store data. However, you can add and configure additional storage locations to provide additional storage space. (See **Prepare Disk Storage Locations** (page 12) in the Installation Guide for disk space requirements.)

By default, data is randomly distributed to all storage locations. However, you can control disk usage and increase I/O performance by isolating files that have different I/O or access patterns in different storage locations. For example, consider:

- Isolating execution engine temporary files from data files.
- Creating a tiered disk architecture in which columns are stored on different disks based on predicted or measured access patterns.

This section describes how to:

- **Add storage locations** (page 286)
- **Measure storage location performance** (page 287)
- **Set storage location performance** (page 288)
- **Modify storage locations** (page 289)
- **Retire storage locations** (page 289)
- **Restore retired storage locations** (page 290)
- **Drop storage locations** (page 290)

Adding Storage Locations

Configuring additional storage locations not only provides additional storage space, it provides you with the option to control disk usage and increase I/O performance by isolating files that have different I/O or access patterns.

Before adding a location:

- Verify that the directory you want to use for the location is an empty directory with write permissions for the Vertica process.
- Determine the type of information you want to store in the storage location:
 - DATA — Persistent data and temp table data
 - TEMP — Temporary files that are generate and dumped to disk such as those generated by sort, group by, join, and so on
 - DATA,TEMP — Both data and temp files (the default)

Tip: The advantage of storing temp files and data in different storage locations is that they have different disk I/O access patterns. Temp data is distributed across available storage locations based on available storage space. However, data can be stored on different storage locations based on predicted or measured access patterns.

To add a location:

- 1 use the `ADD_LOCATION` function to specify the new data directory path to the host, the node where the location is available (optional), and the type of information to be stored (optional).

The following example adds a location that is available on node2 to store data only:

```
SELECT ADD_LOCATION ('/secondVerticaStorageLocation/' , 'node2' ,
  'DATA');
```

The following example adds a location that is available on the initiator node to store data and temporary files:

```
SELECT ADD_LOCATION ('/secondVerticaStorageLocation/');
```

- 2 If the storage location is used to store data or data and temp files and you want to create a tiered disk architecture in which columns are stored on different disks based on predicted or measure access patterns, you need to:

1. **Measure the performance of the storage location** (page 287).
2. **Set the performance of the storage location** (page 288).

Note: Once you have created a storage location, you can modify the type of information it stores. See **Modifying Storage Locations** (page 289).

Measuring Location Performance

If you intend to create a tiered disk architecture in which columns are stored on different disks based on predicted or measured access patterns, you need to measure storage location performance for each location in which data is stored. Measuring location performance is not applicable for temp storage locations.

Note: Measuring Disk I/O is an intensive operation. Therefore, start this operation only when no other operations are running.

Storage location performance equates to the amount of time it takes to read a fixed amount of data from the disk. This read time equates to the disk throughput in MB per second plus the time it takes to seek data based on the number of seeks per second, as follows:

Read Time (seconds) = 1/Throughput (MB/second) + 1/Latency (seeks/second)

Therefore, a disk is faster than another disk if its Read Time is smaller.

Vertica provides two methods for measuring storage location performance depending upon whether or not the database is functioning. Both of these methods return the throughput and latency for the storage location. Write down the throughput and latency information because you need it to set the location performance.

Measuring Location Performance for a Functioning Vertica Database

To measure performance for a storage location on a functioning database, use `MEASURE_LOCATION_PERFORMANCE`. This function has the following requirements:

- The storage path must already exist in the database.
- You need RAM*2 free space available in a storage location to measure its performance. For example, if you have 16GB RAM, you need 32GB of available disk space. If you do not have enough disk space, the function errors out.

Use the virtual table `DISK_STORAGE` to obtain information about disk storage on each database node.

The following example measures the performance of a storage location on node2:

```
SELECT MEASURE_LOCATION_PERFORMANCE('/secondVerticaStorageLocation/', 'node2');
WARNING: measure_location_performance can take a long time. Please check logs for
progress
      measure_location_performance
-----
Throughput : 122 MB/sec. Latency : 140 seeks/sec
```

Measuring Location performance Before a Cluster is Set Up

Measuring disk performance before setting up a cluster is useful for verifying that the disk is functioning within normal parameters. This method requires only that Vertica be installed.

To measure disk performance, use the following vsql command:

```
opt/vertica/bin/vertica -m <path to disk mount>
```

For example:

```
opt/vertica/bin/vertica -m /secondVerticaStorageLocation/site01_data
```

Setting Location Performance

Once you have **measured the performance for each location that stores data files** (page 287), you need to set the performance for each of these locations.

To set the performance for a location, use the `SET_LOCATION_PERFORMANCE` function and specify the performance data you gathered from measuring the location's performance.

Note: Both the throughput and latency must be set to 1 or more.

The following example sets the performance of a storage location on node2 to a throughput of 122 MB/second and a latency of 140 seeks/second.

```
SELECT
SET_LOCATION_PERFORMANCE('node2', '/secondVerticaStorageLocation/', '122', '140');
```

How Vertica Uses Location performance Settings

Once set, Vertica automatically uses this information to determine how to store projection columns.

Vertica stores columns that are included in the sort order of projections on the fastest disks and columns that are not included within the sort order of projections on slower disks. It accomplishes this by ranking columns for each projection as follows:

- Columns in the sort order are given the highest priority (numbers > 1000).
- The last column in the sort order is given the rank number 1001.
- The next to the last column in the sort order is given the rank number 1002, and so on until the first column in the sort order is given 1000 + # of sort columns.
- The remaining columns are given numbers from 1000 - 1, starting with 1000 and working down from there.

Then it stores these columns on disk as follows:

Columns are stored on disk from the highest ranking to the lowest ranking in which the highest ranking columns are placed on the fastest disks and the lowest ranking columns are placed on the slowest disks.

Modifying Storage Locations

You can modify the type of files Vertica stores at any storage location. This is useful if you create additional storage locations and you want to isolate execution engine temporary files from data files.

At least one location must remain for storing data and temp files. These files can be stored in the same storage location or separate storage locations.

To modify a storage location, use the `ALTER_LOCATION_USE` function. When a storage location is altered, it stores only the type of information indicated from that point forward. If, for example, you modify a storage location so that it stores only temp files instead of data and temporary files, the data previously stored on that location is eventually merged out through the ATM as per its policies. You can also merge it out manually.

If you modify a storage location that stores temp files or temp and data files to store only data files, all currently running statements, such as queries and loads that use these temp files, continue to run. Subsequent statements no longer use this location for temp files.

The following example alters the storage location on node3 to store data only:

```
SELECT ALTER_LOCATION_USE ('/thirdVerticaStorageLocation/' , 'node3' , 'DATA');
```

Retiring Storage Locations

Retiring a location prevents Vertica from storing data or temp files to it. It does not remove the actual location. Any data previously stored in the retired location is eventually merged out by the ATM as per its policies.

Tip: If the location you are retiring was used to store temp files only, you can remove it. See [Dropping Storage Locations](#) (page 290).

Before retiring a location, be sure that at least one location remains for storing data and temp files. Data and temp files can be stored in either one storage location or separate storage locations.

To retire a storage location, use the `RETIRE_LOCATION` function.

The following example retires a storage location on node2:

```
SELECT RETIRE_LOCATION('/secondVerticaStorageLocation/' , 'node2');
```

Restoring Retired Storage Locations

If you determine that you want to use a storage location that you have retired, you can restore that location. Once restored, Vertica re-ranks the storage locations and use the restored location to process queries as determined by its rank.

To restore a retired storage location, use the `RESTORE_LOCATION` function.

The following example restores a retired storage location on node2:

```
=> SELECT RESTORE_LOCATION('/secondVerticaStorageLocation/' , 'node2');
```

Dropping Storage Locations

Only storage locations that are configured to store temp files can be dropped. You cannot drop storage locations that store data files.

If a location used to store data and you modified it to store only temp files, the location may still contain data files. If the storage location contains data files, Vertica will not allow you to drop it. You can manually merge out all the data in this location, wait for the ATM to mergeout the data files automatically, or you can **drop partitions** (page 186). Deleting data files will lead to database corruption.

Dropping a storage location is a permanent operation and cannot be undone. Therefore, Vertica recommends that you retire a storage location before dropping it. This will allow you to verify that you actually want to drop a storage location before doing so. Additionally, you can easily **restore a retired storage location** (page 290).

To drop a storage location, use the `DROP_LOCATION` function.

The following example drops a storage location on node3 that was used to store temp files:

```
=> SELECT DROP_LOCATION('/secondVerticaStorageLocation/' , 'node3');
```

Reclaiming Disk Space

You can reclaim the disk space held by deleted records by **purging the deleted records** (page 180), **rebuilding the table** (page 290) or **dropping a partition** (page 186).

Rebuilding a Table

When it is necessary to do large-scale disk reclamation operations, consider rebuilding the table by following sequence of operations:

- 1 Create a new table.
- 2 Create projections for the new table.
- 3 Populate the new table using `INSERT ... SELECT` to copy the desired table from the old table.

- 4 Drop the old table and its projections.
- 5 Use ALTER TABLE ... RENAME to give the new table the name of the old table.

Notes

- You must have enough disk space to contain the old and new projections at the same time. If necessary, you can drop some of the old projections before loading the new table. You must, however, retain at least one superprojection of the old table (or two buddy superprojections to maintain K-Safety) until the new table is loaded. (See **Prepare Disk Storage Locations** (page 12) in the Installation Guide for disk space requirements.)
- You can specify different names for the new projections or use the ALTER PROJECTION ... RENAME command to change the names of the old projections.
- The relationship between tables and projections does not depend on object names. Instead, it depends on object identifiers that are not affected by rename operations. Thus, if you rename a table, its projections continue to work normally.
- Manually purging a table continues to retain history for rows deleted after the Ancient History Mark. Rebuilding the table results in purging all the history of the table, which means you cannot do historical queries on any older epoch.
- Rather than dropping the old table in Step 4, you might rename it to a different name and use it as a backup copy. Note, however, that you must have sufficient disk space.

Managing Workloads

Vertica provides a sophisticated workload management scheme to enable diverse concurrent workloads to run efficiently against the database. For basic operations, Vertica provides a built-in GENERAL pool that is pre-configured based on RAM and machine cores and can be customized further to handle specific concurrency requirements.

For more advanced needs, the database administrator can establish new resource pools configured with limits on memory usage, concurrency, and priority. Each database user can optionally be restricted to use a specific resource pool and also be given quotas on memory usage to control resources used by their requests.

User-defined pools are useful when there are competing resource requirements across different classes of workloads. Some examples of scenarios that can make use of user-defined pools include:

- A large batch job takes up all of the server resources, leaving small jobs that update a web page to starve, thereby degrading user experience. To solve this problem, a dedicated resource pool can be created for web page requests to ensure they always get resources. Alternatively, a limited resource pool can be created for the batch job, so it cannot use up all the resources in the system.
- A certain application has lower priority and you would like to limit the amount of memory and number of concurrent users of this application. To solve this problem, a resource pool with an upper limit on the memory usage of a query can be created and associated with users of this application.

For detailed syntax of creating and managing resource pool see the following topics in the SQL Reference Manual:

Statements

- ALTER RESOURCE POOL alters a resource pool.
- ALTER USER associates a user with the RESOURCE POOL and MEMORYCAP parameters.
- CREATE RESOURCE POOL creates a resource pool.
- CREATE USER adds a name to the list of authorized database users and specifies that user's RESOURCE POOL and MEMORYCAP parameters.
- DROP RESOURCE POOL drops a user-created resource pool.
- SET SESSION MEMORYCAP sets the limit on amount of memory that any request issued by the session can consume.
- SET SESSION RESOURCE POOL associates a user session with specified resource pool.

System Tables

- RESOURCE_ACQUISITIONS provides details of resources (memory, open file handles, threads) acquired by each request for each resource pool in the system.
- RESOURCE_ACQUISITIONS_HISTORY provides details of resources (memory, open file handles, threads) acquired by any profiled query for each resource pool in the system.

- RESOURCE_POOL_STATUS provides configuration settings of the various resource pools in the system, including internal pools.
- RESOURCE_POOLS displays information about the parameters the resource pool was configured with.
- RESOURCE_QUEUES provides information about requests pending for various resource pools.
- RESOURCE_REJECTIONS monitors requests for resources that are rejected by the Resource Manager.

The Resource Manager

In a single-user environment, the system can devote all resources to a single query and, thereby, get the most efficient execution for that one query. However, in a environment where several concurrent queries are expected to run at once, there is tension between providing each query the maximum amount of resources (thereby getting fastest run time for that query) and serving multiple queries simultaneously with a reasonable run time. The Resource Manager (RM) provides options and controls for resolving this tension, while ensuring that every query eventually gets serviced and that true system limits are respected at all times.

When the system experiences resource pressure, the Resource Manager might queue queries until the resources become available (or a timeout value is reached). Also, by configuring various RM settings, the target memory of each query can be tuned based on the expected number of concurrent queries running against the system.

The following sections discuss the detailed architecture and operation of the Resource Manager.

Resource Manager Impact on Query Execution

The Resource Manager (RM) impacts individual query execution in various ways. When a query is submitted to the database, the following series of events occur:

- 1 The query is parsed and optimized to determine an execution plan.
- 2 The Resource Manager is invoked on the initiator node to estimate resources required to run the query. If the resource requirements violate true system limits, one of the following could occur:
 - If the memory required by the query alone would exceed the machine's physical memory, the query is rejected.
 - If the resource requirements are within limits that are not currently available, the query is queued.Otherwise the query is allowed to run.
- 3 Eventually the query either times out on the queue or is allowed to run.
- 4 When allowed to run, the query is distributed to other nodes, called executor nodes.

- 5 On the executor nodes, the plan is augmented to consider local ROS container distribution. Because the initiator node is not fully aware of the ROS layout on the other nodes and of the exact resource usage by queries running on other nodes, it is possible, though rare, that *actual* resources needed by the query are more than the *estimated* resources calculated at the initiator, or the *requested* resources are not available at the executor. If this happens, then query is rejected.

Notes

- No resources are reserved or held while the query is in the queue.
- Apportioning of resources for the query and maximum number of queries allowed to run depends on the resource pool configuration. See **Resource Pool Architecture** (page 294).
- When a query is rejected at the initiator node, the system returns an “Insufficient resources to initiate plan” message to the user. When a query is rejected at an executor node, the system returns an “Insufficient resources to execute localized plan” message to the user. The term ‘resource rejection’ describes these error conditions. See RESOURCE_REJECTIONS in the SQL Reference Manual, a table that monitors requests for resources that are rejected by the Resource Manager.

Resource Pool Architecture

The Resource Manager manages resources as one or more resource pools. A resource pool comprises a pre-allocated subset of the system resources, with an associated queue.

Out-of-the-box, Vertica comes pre-configured with a set of built-in pools, which Vertica uses to allocate resources to different types of requests. By default, Vertica provides the pools described in Built-in Pools in the SQL Reference Manual.

For basic operation of Vertica, the built-in GENERAL pool comes preconfigured for a certain concurrency level based on the RAM and cores in the machines. This pool can be customized further based on actual concurrency and performance requirements. See **Guidelines for Setting Pool Parameters** (page 301).

Advanced users can create custom pools to handle various classes of workloads. User requests can then be restricted to use these custom resource pools.

Note: A resource pool is created using the CREATE RESOURCE POOL command as described in the SQL Reference Manual.

The GENERAL Pool

Note: This topic provides a simplified description of the GENERAL pool concept. For a detailed list of built-in pools (including GENERAL) and their default configuration settings, see CREATE RESOURCE POOL in the SQL Reference Manual.

The GENERAL pool is a special, catch-all pool used to answer requests that have no specific resource pool associated with them. Any memory left over after memory has been allocated to all other pools is automatically allocated to the GENERAL pool. Any user-defined pool can be configured to “borrow” memory from the GENERAL pool to satisfy requests that need extra memory. If the pool is configured so that it cannot borrow any memory from the GENERAL pool, it is said to be standalone. When multiple pools request memory from the GENERAL pool, they are granted access to GENERAL pool memory according to their priority setting.

In this manner, the GENERAL pool provides some elasticity to account for point-in-time deviations from normal usage of individual resource pools.

Resource Tracking in a Pool

If you want to know what resources are tracked within a resource pool and how are the overall system limits determined, the Vertica Resource Manager tracks the following:

- Memory (KB) – memory used by the requests running against the pool.
- Threads – amount of concurrent execution.
- File handles – number of open files

The resource pool configuration parameters include only memory and concurrency related parameters, not threads and file handles. Vertica automatically apportions threads and file handle resources to the pools, in proportion to their memory usage.

The true system limits for various types of resources managed by the Resource Manager are as follows:

- Memory – set by default to 95% of physical RAM on the node. The assumption is that the remainder is used by the rest of the processes running on the machine, or by other components within Vertica itself. Memory is customized by setting the MAXMEMORYSIZE parameter of the GENERAL pool.
- Threads – $\text{/proc/sys/kernel/threads-max} * 3/4$ (Default thread-max 80896)
- File Handles – $\text{OS limit} * 7/8$ (Default OS limit 65536)

Query Queue/Rejection Process

Every resource pool has an internal memory threshold called the Queuing Threshold, such that if the memory usage of requests running against the pool exceeds this threshold, subsequent requests are queued. (See RESOURCE_POOL_STATUS.QUEUEING_THRESHOLD_KB in the SQL Reference Manual.)

Queries issued against a pool can be rejected for one of following reasons:

- If the absolute amount of resources required by the query exceeds the memory available in the system, the query is immediately rejected.

Note: This should be a fairly rare condition if the system has been well designed with adequate resources.

- The query stays on the queue but is unable to acquire resources even after a specified timeout period. (See the `QUEUE_TIMEOUT` parameter in `CREATE RESOURCE POOL` in the SQL Reference Manual) In this case, the query is rejected, and this is the most likely scenario for query rejection in Vertica 4.0.
- The query is admitted to run at the initiator node but then gets rejected at the executor node. This rejection can happen if the actual resources needed by the query are more than the estimated resources calculated at the initiator, or the requested resources are not available at the executor. The system returns an error like the following:

```
Query required more resources than initiator resource manager estimated,  
likely due to high storage container counts or a heavier workload on node  
XXX
```
- The number of queries currently running against the pool has reached its specified concurrency limit. (See the `MAX_CONCURRENCY` parameter in `CREATE RESOURCE POOL` in the SQL Reference Manual.)
Note: This limit applies only to queries actually running and not queries queued against the pool.
- The user had set up a `MEMORYCAP` on how much memory could be used by the query. See *User Profiles* (page 299).

See Also

`V_MONITOR.RESOURCE_POOL_STATUS` and
`V_MONITOR.RESOURCE_REJECTIONS.REASON` in the SQL Reference Manual

Target Memory Determination for Queries in Concurrent Environments

The resource pool parameters of `MEMORYSIZE` and `PLANNED_CONCURRENCY` (`CREATE RESOURCE POOL` in the SQL Reference Manual) provide the options that let you tune the target memory allocated to queries.

- If `MEMORYSIZE` is set to 0, in which case the pool borrows all memory as needed from the `GENERAL` pool, the target amount of memory for the query is calculated using the Queueing Threshold of the `GENERAL` pool / `PLANNED_CONCURRENCY`.
- If the resource pool for the query has the `MEMORYSIZE` parameter set, and the pool is standalone (i.e. cannot borrow from General pool) then the target memory is to use the Queueing Threshold of the pool / `PLANNED_CONCURRENCY` amount of memory.
- Otherwise, if `MEMORYSIZE` is set but the pool is not standalone, the target memory is set to `MEMORYSIZE / PLANNED_CONCURRENCY` of the pool.

Therefore, by carefully tuning the `MEMORYSIZE` and `PLANNED_CONCURRENCY` parameters, it is possible to restrict the amount of memory used by a query to a desired size.

See Also

`MEMORYCAP` setting in *User Profiles* (page 299)

`RESOURCE_POOL_STATUS.QUEUEING_THRESHOLD_KB` in the SQL Reference Manual

Monitoring Resource Pools and Resource Usage by Queries

The **Linux top command** <http://linux.die.net/man/1/top> can be used to determine the overall CPU usage and I/O waits across the system. However, resident memory size indicated by `top` is not a good indicator of actual memory use or reservation because of file system caching and so forth. Instead, Vertica provides several monitoring tables that provide detailed information about resource pools, their current memory usage, resources requested and acquired by various requests and the state of the queues.

The `RESOURCE_POOLS` table lets you view various resource pools defined in the system (both internal and user-defined), and the `RESOURCE_POOL_STATUS` table lets you view the current state of the resource pools.

Examples

The following command returns the various resource pools defined in the system.

```
=> SELECT * FROM V_CATALOG.RESOURCE_POOLS;
  name | is_internal | memorysize | maxmemorysize | priority | queuetimeout | plannedconcurrency
| maxconcurrency | singleinitiator
-----+-----+-----+-----+-----+-----+-----
general | t          |             | Special: 95%  |         0 |             300 |
|             | f          |             |             |         0 |             300 |
sysquery | t          | 64M         |             |        20 |             300 |
|             | f          |             |             |         0 |             300 |
sysdata  | t          | 100M        | 10%          |         0 |             300 |
|             |           |             |             |         0 |             300 |
wodata   | t          | 0%          | 25%          |         0 |             300 |
|             |           |             |             |         0 |             300 |
tm       | t          | 200M        |             |        10 |             300 |
|             |           |             |             |         0 |             300 |
refresh  | t          | 0%          |             |       -10 |             300 |
|             | t          |             |             |         0 |             300 |
recovery | t          | 0%          |             |        15 |             300 |
|             | t          |             |             |         0 |             300 |
dbd      | t          | 0%          |             |         0 |             300 |
|             | t          |             |             |         0 |             300 |
(8 rows)
```

To see only the user-defined resource pools, you can limit your query to return records where `IS_INTERNAL` is false.

Note: The user-defined pools below are used as examples in subsequent sections related to Workload Management.

The following command returns information on user-defined resource pools:

```
=> SELECT name, memorysize, maxmemorysize, priority, maxconcurrency
FROM V_CATALOG.RESOURCE_POOLS where is_internal = 'f';
  name | memorysize | maxmemorysize | priority | maxconcurrency
-----+-----+-----+-----+-----
load_pool | 0%         |               |        10 |
ceo_pool  | 250M       |               |        10 |
ad_hoc_pool | 200M      | 200M          |         0 |
billing_pool | 0%        |               |         0 | 3
web_pool  | 25M        |               |        10 | 5
batch_pool | 150M       | 150M          |         0 | 10
dept1_pool | 0%         |               |         5 |
dept2_pool | 0%         |               |         8 |
(8 rows)
```

The queries borrow memory from the GENERAL **pool** (page 294) and show the amount of memory in use from the GENERAL pool.

The following command uses the V_MONITOR.RESOURCE_POOL_STATUS table to return the current state of all resource pools on node0001:

```
=> SELECT pool_name, memory_size_kb, memory_size_actual_kb, memory_inuse_kb,
general_memory_borrowed_kb,
       running_query_count FROM V_MONITOR.RESOURCE_POOL_STATUS where node_name ilike '%node0001';
 pool_name | memory_size_kb | memory_size_actual_kb | memory_inuse_kb | general_memory_borrowed_kb
 | running_query_count
-----+-----+-----+-----+-----+-----
general   |      15108517 |      15108517 |          0 |          0
 |              |              |              |              |
sysquery  |         65536 |         65536 |          0 |          0
 |              |              |              |              |
sysdata   |        102400 |        102400 |         4096 |          0
 |              |              |              |              |
wosdata   |           0 |           0 |          0 |          0
 |              |              |              |              |
tm        |       204800 |       204800 |          0 |          0
 |              |              |              |              |
refresh   |           0 |           0 |          0 |          0
 |              |              |              |              |
recovery  |           0 |           0 |          0 |          0
 |              |              |              |              |
dbd       |           0 |           0 |          0 |          0
 |              |              |              |              |
(8 rows)
```

The following command uses the V_MONITOR.RESOURCE_ACQUISITIONS table to show all resources granted to the queries that are currently running:

Note: While running vmart_query_04.sql from the VMart example database, notice that the query uses memory_inuse_kb = 219270 from the GENERAL pool.

```
=> SELECT pool_name, thread_count, open_file_handle_count, memory_inuse_kb, queue_entry_timestamp,
acquisition_timestamp FROM V_MONITOR.RESOURCE_ACQUISITIONS WHERE node_name ILIKE '%node0001';

 pool_name | thread_count | open_file_handle_count | memory_inuse_kb | queue_entry_timestamp
 | acquisition_timestamp
-----+-----+-----+-----+-----+-----
sysquery  |           4 |           0 |          4103 | 2010-04-12 15:57:05.526678-04
 | 2010-04-12 15:57:05.526684-04
general   |           4 |           5 |       219270 | 2010-04-12 15:56:38.95516-04
 | 2010-04-12 15:56:38.956373-04
sysdata   |           0 |           0 |         4096 | 2010-04-12 12:58:06.063178-04
 | 2010-04-12 13:11:54.930346-04
wosdata   |           0 |           0 |           0 | 2010-04-12 15:22:33.454542-04
 | 2010-04-12 15:22:33.454548-04
(4 rows)
```

To determine how long a query waits in the queue before it is admitted to run, you can get the difference between the acquisition_timestamp and the queue_entry_timestamp using a query like the following:

```
=> SELECT pool_name, queue_entry_timestamp, acquisition_timestamp,
(acquisition_timestamp-queue_entry_timestamp)
 AS 'queue wait' FROM V_MONITOR.RESOURCE_ACQUISITIONS WHERE node_name ILIKE '%node0001';
 pool_name | queue_entry_timestamp | acquisition_timestamp | queue wait
-----+-----+-----+-----
sysquery  | 2010-04-14 10:43:45.931525-04 | 2010-04-14 10:43:45.931532-04 | 00:00:00.000007
```



```

billing_pool | 2010-04-14 10:39:24.295196-04 | 2010-04-14 10:39:24.296469-04 | 00:00:00.001273
ceo_pool    | 2010-04-14 10:40:07.281384-04 | 2010-04-14 10:40:07.29919-04 | 00:00:00.017806
sysdata    | 2010-04-12 12:58:06.063178-04 | 2010-04-12 13:11:54.930346-04 | 00:13:48.867168
wosdata    | 2010-04-12 15:22:33.454542-04 | 2010-04-12 15:22:33.454548-04 | 00:00:00.000006
(5 rows)

```

See the SQL Reference Manual for detailed descriptions of the monitoring tables described in this topic.

User Profiles

User profiles are attributes associated with a user that control that user's access to several system resources. These resources include:

- the resource pool to which a user is assigned (RESOURCE POOL)
- the maximum amount of memory a user's session can use (MEMORYCAP)
- the maximum amount of temporary file storage a user's session can use (TEMPSPACECAP)
- the maximum amount of time a user's query can run (RUNTIMECAP)

You can set these attributes when calling CREATE USER and changed later using ALTER USER.

There are two strategies for limiting a user's access to resources: setting attributes on the user directly to control resource use, or assigning the user to a resource pool. The first method allows you to fine tune individual users, while the second makes it easier to group many users together and set their collective resource usage. For examples, see the scenarios described in *Using User-defined Pools and User-Profiles for Workload Management* (page 303).

Example

Set the user's RESOURCE POOL attribute to assign the user to a resource pool. To create a user named user1 who has access to the resource pool my_pool, use the command:

```
=> CREATE USER user1 RESOURCE POOL my_pool;
```

To limit the amount of memory for a user without designating a pool, set the user's MEMORYCAP to either a particular unit or a percentage of the total memory available. For example, to create a user named user2 whose sessions are limited to using 200 megabytes memory each, use the command:

```
=> CREATE USER user2 MEMORYCAP '200M';
```

To limit the time a user's queries are allowed to run, set the RUNTIMECAP attribute. To prevent user2's queries from running more than 5 minutes, you can use the command:

```
=> ALTER USER user2 RUNTIMECAP '5 minutes';
```

To limit the amount of temporary disk space that the user's sessions can use, set the TEMPSPACECAP to either a particular size or a percentage of temporary disk space available. This example creates user3 who is limited to using 1 gigabyte of temporary space:

```
=> CREATE USER user3 TEMPSPACECAP '1G';
```

You can combine different attributes into a single command. For example, to limit a user3's MEMORYCAP and RUNTIMECAP, include both attributes in an ALTER USER command:

```
=> ALTER USER user3 MEMORYCAP '750M' RUNTIMECAP '10 minutes';
ALTER USER
```

```
=> \x
Expanded display is on.
=> SELECT * FROM USERS;
-[ RECORD 1 ]-----+-----
user_id      | 45035996273704962
user_name    | release
is_super_user | t
resource_pool | general
memory_cap_kb | unlimited
temp_space_cap_kb | unlimited
run_time_cap | unlimited
-[ RECORD 2 ]-----+-----
user_id      | 45035996273964824
user_name    | user1
is_super_user | f
resource_pool | my_pool
memory_cap_kb | unlimited
temp_space_cap_kb | unlimited
run_time_cap | unlimited
-[ RECORD 3 ]-----+-----
user_id      | 45035996273964832
user_name    | user2
is_super_user | f
resource_pool | general
memory_cap_kb | 204800
temp_space_cap_kb | unlimited
run_time_cap | 00:05
-[ RECORD 4 ]-----+-----
user_id      | 45035996273970230
user_name    | user3
is_super_user | f
resource_pool | general
memory_cap_kb | 768000
temp_space_cap_kb | 1048576
run_time_cap | 00:10
```

See Also

ALTER USER and CREATE USER in the SQL Reference Manual

Best Practices for Workload Management

This section provides general guidelines and best practices on how to set up and tune resource pools for various common scenarios.

Note: The exact settings for the pool parameters are heavily dependent on your query mix, data size, hardware configuration, and concurrency requirements. Vertica recommends performing your own experiments to determine the optimal configuration for your system.

Basic Principles for Scalability and Concurrency Tuning

A Vertica database runs on a cluster of commodity hardware. All loads and queries running against the database take up system resources, such as CPU, memory, disk I/O bandwidth, file handles, and so forth. The performance (run time) of a given query depends on how much resource it has been allocated.

When running more than one query concurrently on the system, both queries are sharing the resources; therefore, each query could take longer to run than if it was running by itself. In an efficient and scalable system, if a query takes up all the resources on the machine and runs in X time, then running two such queries would double the run time of each query to $2X$. If the query runs in $> 2X$, the system is not linearly scalable, and if the query runs in $< 2X$ then the single query was wasteful in its use of resources. Note that the above is true as long as the query obtains the minimum resources necessary for it to run and is limited by CPU cycles. Instead, if the system becomes bottlenecked so the query does not get enough of a particular resource to run, then the system has reached a limit. In order to increase concurrency in such cases, the system must be expanded by adding more of that resource.

In practice, Vertica should achieve near linear scalability in run times, with increasing concurrency, until a system resource limit is reached. When adequate concurrency is reached without hitting bottlenecks, then the system can be considered as ideally sized for the workload.

Note: Typically Vertica queries on segmented tables run on multiple (likely all) nodes of the cluster. Adding more nodes generally improves the run time of the query almost linearly.

Guidelines for Setting Pool Parameters

This section provides guidelines on setting the various parameters of any resource pool. When using Vertica with out-of-the-box resource pools, you typically need to tune only the GENERAL pool parameters, PLANNEDCONCURRENCY and MAXCONCURRENCY, using the guidelines described here. See *Using User-defined Pools and User-profiles for Workload Management* (page 303) for examples of situations where you might want to create your own pools.

Parameter	Guideline
MEMORYSIZE	<p>Ignore if tuning the GENERAL pool.</p> <p>For other pools, you can leave the setting to the default (0%), which allows the pool to borrow memory from the General pool, as needed. Consider setting this in the following situations:</p> <ul style="list-style-type: none">▪ To set aside memory for exclusive use of requests issued to that pool. This memory then cannot be used for other purposes, even if there are no requests made to that pool.▪ In combination with PLANNEDCONCURRENCY, to tune the memory

	<p>used by a certain query to a certain size, where applicable. This is for expert use only.</p> <p>See Target Memory Determination for Queries in Concurrent Environments (page 296).</p>
MAXMEMORYSIZE	<p>Ignore if tuning the GENERAL pool.</p> <p>For other pools, use this parameter to set up the resource pool as a standalone pool or to limit how much memory the resource pool can borrow from the GENERAL pool. This provides a mechanism to enforce a hard limit on the memory usage by certain classes of workload; for example, loads should take up no more than 4GB of total available memory.</p> <p>See Scenario: Restricting resource usage and concurrency of ad-hoc application (page 306)</p>
QUEUETIMEOUT	<p>Use this parameter to change the timeout of the pool from the 5-minute default.</p> <p>The timeout can be customized if you need different queuing durations for different classes of workloads. For example, long-running batch workloads could be configured to run on a pool with high timeout values, possibly unlimited, since completion of the task is more critical than response time. For interactive application queries, the timeouts could be set to low or 0 to ensure application gets an immediate error if the query cannot run.</p> <p>Note: Be mindful that increased timeouts will lead to longer queue lengths and will not necessarily improve the overall throughput of the system.</p>
PRIORITY	<p>Use this parameter to prioritize the use of GENERAL pool resources, either by requests made directly to the GENERAL pool, or by requests made to other pools borrowing memory from the GENERAL pool. For instance, requests made to the RecoveryPool have highest priority out of the box so that Vertica can expeditiously provide resources to recover nodes that are down.</p> <p>Note: The PRIORITY setting has no meaning for a standalone pool, which does not borrow memory from the GENERAL pool. See examples in Scenario: Periodic Batch Loads (page 303) and Scenario: Setting Priorities on Queries Issued by Different Users (page 309).</p>
PLANNEDCONCURRENCY	<p>Use this parameter to represent the typical number of queries running concurrently in the system. By default, this setting is configured using the number of cores and RAM in the system, so as to provide each query with its optimal amount of memory and use of at least one core.</p> <p>This parameter must be set by experimentation to ensure individual query performance meets requirements. As discussed in Query Memory in Concurrent Environments (page 296), there is a tradeoff between giving each query maximum amount of resources versus allowing many concurrent queries to run in a reasonable amount of time.</p> <p>Notes:</p> <ul style="list-style-type: none"> ▪ This parameter can be used in combination with MEMORYSIZE to tune the memory used by a query down to a specific size. ▪ If you created or upgraded your database in 4.0 or 4.1, the PLANNEDCONCURRENCY setting on the GENERAL pool defaults to

	<p>a too-small value for machines with large numbers of cores. To adjust to a more appropriate value:</p> <ul style="list-style-type: none"> ▪ => ALTER RESOURCE POOL general PLANNEDCONCURRENCY <#cores>; ▪ This parameter only needs to be set if you created a database before 4.1, patchset 1.
MAXCONCURRENCY	<p>Use this parameter if you want to impose a hard limit on the number of concurrent requests that are allowed to run against any pool, including the GENERAL pool.</p> <p>Instead of limiting this at the pool level, it is also possible to limit at the connection level using <i>MaxClientSessions</i> (page 25).</p>
SINGLEINITIATOR	<p>This parameter can be left to its default (false) value in most situations, unless advised otherwise by Vertica <i>Technical Support</i> (on page 1). SINGLEINITIATOR is set to 'true' by some built-in pools used for operations that are local to the node, such as Tuple Mover and Recovery.</p>

Using User-defined Pools and User-Profiles for Workload Management

The scenarios in this section describe some commonly encountered workload-management problems and provide some solutions with examples.

Scenario: Periodic Batch Loads

Scenario

We do batch loads every night, or occasionally during the day but infrequently. When loads are running, it is acceptable to reduce resource usage by queries, but at all other times we want all resources to be available to queries.

Solution

Create a separate resource pool for loads with a higher priority than the GENERAL pool.

During nightly loads, the loads get preference when borrowing memory from the GENERAL pool. When loads are not running, all memory is automatically available for queries.

If using the WOS, tune the PLANNEDCONCURRENCY parameter of the WOSDATA pool to the number of concurrent loads. This ensures that AUTO spill to ROS is configured in an optimal fashion.

Example

Create a resource pool with the PRIORITY of the pool set higher than the GENERAL pool.

For example, to create a pool designated for loads that has a higher priority than the GENERAL pool, set load_pool with a priority of 10:

```
CREATE RESOURCE POOL load_pool PRIORITY 10;
```

Edit the WOSDATA pool PLANNEDCONCURRENCY:

```
ALTER RESOURCE POOL WOSDATA PLANNEDCONCURRENCY 6;
```

Modify the user's resource pool:

```
ALTER USER load_user RESOURCE POOL load_pool;
```

Scenario: The CEO Query

Scenario

The CEO runs a report every Monday at 9AM. How can we ensure the report always runs?

Solution

To ensure that a certain query or class of queries always gets resources, you could create a dedicated pool for it as follows:

- 1 Using the PROFILE command, run the query that the CEO runs every week to determine how much memory should be allocated:

```
PROFILE SELECT DISTINCT s.product_key, p.product_description
FROM store.store_sales_fact s, public.product_dimension p
WHERE s.product_key = p.product_key AND s.product_version =
      p.product_version
AND s.store_key IN (
      SELECT store_key FROM store.store_dimension
      WHERE store_state = 'MA')
ORDER BY s.product_key;
```

- 2 At the end of the query, the system returns a notice with resource usage:

```
NOTICE: Statement is being profiled.
HINT:  select * from v_monitor.execution_engine_profiles where
transaction_id=45035996273751349 and statement_id=6;
NOTICE: Initiator memory estimate for query: [on pool general: 1723648
      KB,
minimum: 355920 KB]
```

- 3 Create a resource pool with MEMORYSIZE reported by the above hint to ensure that the CEO query has at least this memory reserved for it:

```
CREATE RESOURCE POOL ceo_pool MEMORYSIZE '110M' PRIORITY 10;
\x
```

Expanded display is on.

```
=> SELECT * FROM resource_pools WHERE name = 'ceo_pool';
```

```
-[ RECORD 1 ]-----+-----
name          | ceo_pool
is_internal   | f
memorysize    | 110M
maxmemorysize |
priority      | 10
queuetimeout  | 300
plannedconcurrency | 4
maxconcurrency |
singleinitiator | f
```

- 4 Assuming the CEO report user already exists, associate this user with the above resource pool using ALTER USER statement.

```
=> ALTER USER ceo_user RESOURCE POOL ceo_pool;
```

- 5 Issue the following command to confirm that the ceo_user is associated with the ceo_pool:

```
=> SELECT * FROM users WHERE user_name = 'ceo_user';
-[ RECORD 1 ]-+-----
user_id       | 45035996273713548
user_name     | ceo_user
is_super_user | f
resource_pool | ceo_pool
memory_cap_kb | unlimited
```

If the memory usage by the CEO query is too large, you could ask the Resource Manager to tune it down to fit within certain budget. See *Target Memory Determination for Queries in Concurrent Environments* (page 296).

Scenario: Preventing Run-away Queries

Scenario

Joe, a business analyst often runs big reports in the middle of the day that take up the whole machine. How can we prevent Joe from taking up more than 100MB of memory?

Solution

User Profiles (page 299) provides a solution to this scenario. To restrict the amount of memory business analyst Joe can use at one time, set a MEMORYCAP for Joe to 100MB using the ALTER USER command. If any query run by Joe takes up more than its cap, Vertica rejects the query.

Example

```
ALTER USER analyst_user MEMORYCAP '100M';
```

If Joe attempts to issue a query that exceeds 100M the system returns an error that the request exceeds the memory session limit. For example:

```
\i vmart_query_04.sql
vsq:vmart_query_04.sql:12: ERROR: Insufficient resources to initiate plan
on pool general [Request exceeds memory session limit: 137669KB > 102400KB]
```

Only the system database administrator (dbadmin) can increase only the MEMORYCAP setting. Users cannot increase their own MEMORYCAP settings.

If users attempts to increase the MEMORYCAP, the system returns a "permission denied" error:

```
ALTER USER analyst_user MEMORYCAP '135M';
ROLLBACK: permission denied
```

Scenario: Restricting Resource Usage of Ad-hoc Query Application

Scenario

We recently opened up our data warehouse to a large group of users who are not very experienced with poor SQL. Occasionally, many of them run reports that operate on a large number of rows and overwhelm the system. How can we throttle usage of the system by such users?

Solution

The simplest solution is to create a standalone resource pool for the ad-hoc applications so that the total MEMORYSIZE is fixed. Recall that in a standalone pool, MAXMEMORYSIZE is set equal to MEMORYSIZE so no memory can be borrowed from the GENERAL pool. Associate this user pool with the database user(s) from which the application uses to connect to the database.

Other solutions include limiting the memory usage of individual users such as in the **Scenario: Preventing run-away Queries** (page 305).

Tip: Besides adding limits such as the above, it is also a great idea to train the user community on writing good SQL.

Example

To create a standalone resource pool for the adhoc users, set the MEMORYSIZE equal to the MAXMEMORYSIZE:

```
CREATE RESOURCE POOL adhoc_pool MEMORYSIZE '200M' MAXMEMORYSIZE '200M'
PRIORITY 0 QUEUE_TIMEOUT 300 PLANNED_CONCURRENCY 4;
SELECT pool_name, memory_size_kb, queueing_threshold_kb
FROM V_MONITOR.RESOURCE_POOL_STATUS w
WHERE is_standalone = 'true' AND is_internal = 'false';
```

pool_name	memory_size_kb	queueing_threshold_kb
adhoc_pool	204800	153600

(1 row)

Once the pool has been created, associate the adhoc users with the adhoc_pool:

```
ALTER USER appl_user RESOURCE POOL adhoc_pool;
ALTER RESOURCE POOL adhoc_pool MEMORYSIZE '10M' MAXMEMORYSIZE '10M';
\i vmart_query_04.sql vsql:vmart_query_04.sql:12: ERROR: Insufficient resources
to initiate plan on pool adhoc_pool [Request Too Large:Memory(KB)
Exceeded: Requested = 84528, Free = 10240 (Limit = 10240, Used = 0)]
```

The query will not borrow memory from the GENERAL pool and gets rejected with a 'Request Too Large' message.

Scenario: Setting a Hard Limit on Concurrency For An Application

Scenario

For billing purposes, analyst Jane would like to impose a hard limit on concurrency for this application. How can she achieve this?

Solution

The simplest solution is to create a separate resource pool for the users of that application and set its MAXCONCURRENCY to the desired concurrency level. Any queries beyond MAXCONCURRENCY are rejected.

Tip: Vertica recommends leaving PLANNEDCONCURRENCY to the default level so the queries get their maximum amount of resources. The system as a whole thus runs with the highest efficiency.

Example

In this example, there are four billing users associated with the billing pool. The objective is to set a hard limit on the resource pool so a maximum of five concurrent queries can be executed at one time. All other queries will queue and complete as resources are freed.

```
=> CREATE RESOURCE POOL billing_pool MAXCONCURRENCY 5 QUEUETIMEOUT 2;
=> CREATE USER bill1_user RESOURCE POOL billing_pool;
=> CREATE USER bill2_user RESOURCE POOL billing_pool;
=> CREATE USER bill3_user RESOURCE POOL billing_pool;
=> CREATE USER bill4_user RESOURCE POOL billing_pool;

=> \x
Expanded display is on.

=> SELECT * FROM users WHERE resource_pool = 'billing_pool';
   user_id      | user_name | is_super_user | profile_name | is_locked | lock_time | resource_pool
| memory_cap_kb | temp_space_cap_kb | run_time_cap
-----+-----+-----+-----+-----+-----+-----
45035996273910978 | bill1_user | f             | default      | f         |          | billing_pool
| unlimited     | unlimited     | unlimited
45035996273910982 | bill2_user | f             | default      | f         |          | billing_pool
| unlimited     | unlimited     | unlimited
45035996273910986 | bill3_user | f             | default      | f         |          | billing_pool
| unlimited     | unlimited     | unlimited
45035996273910990 | bill4_user | f             | default      | f         |          | billing_pool
| unlimited     | unlimited     | unlimited
(4 rows)

=> SELECT reason, resource_type, rejection_count, first_rejected_timestamp,
last_rejected_timestamp, last_rejected_value
FROM RESOURCE_REJECTIONS
WHERE pool_name = 'billing_pool' AND node_name ilike '%node0001';

   reason                                     | resource_type | rejection_count |
first_rejected_timestamp | last_rejected_timestamp | last_rejected_value
-----+-----+-----+-----+-----+-----
Timedout waiting for resource request | Queries       | 16 | 2010-04-13
16:28:12.640383-04 | 2010-04-14 09:35:00.056489-04 | 1
(1 row)
```

If three queries are running and do not complete in the allotted time (default timeout setting is 5 minutes), the next query requested gets an error similar to the following:

```
ERROR: Insufficient resources to initiate plan on pool billing_pool
[Timeout waiting for resource request: Request exceeds limits:
Queries Exceeded: Requested = 1, Free = 0 (Limit = 3, Used = 3)]
```

The table below shows that there are three active queries on the billing pool.

```
=> SELECT pool_name, thread_count, open_file_handle_count, memory_inuse_kb,
queue_entry_timestamp, acquisition_timestamp
FROM RESOURCE_ACQUISITIONS
WHERE pool_name = 'billing_pool';
```

pool_name	thread_count	open_file_handle_count	memory_inuse_kb	queue_entry_timestamp	acquisition_timestamp
billing_pool	4	5	132870	2010-04-14	16:24:30.136789-04
billing_pool	4	5	132870	2010-04-14	16:24:28.119842-04
billing_pool	4	5	132870	2010-04-14	16:24:26.209174-04

(3 rows)

Scenario: Handling Mixed Workloads (Batch vs. Interactive)

Scenario

We have a web application with an interactive portal. Sometimes when IT is running batch reports, the web page takes ages to refresh and our users complain. How can we provide a better experience to our web site users?

Solution

The principles learned from the previous scenarios can be applied to solve this problem. The basic idea is to segregate the queries into two groups associated with different resource pools. (The prerequisite is that there are two distinct database users issuing the different types of queries. If this is not the case, do consider this a best practice for application design.)

There are two ways to do this.

- **METHOD 1:** Create a dedicated pool for the web page refresh queries where you:
 1. Size the pool based on the average resource needs of the queries and expected number of concurrent queries issued from the portal.
 2. Associate this pool with the database user that runs the web site queries. (See **Scenario: The CEO Query** (page 304) for detailed procedure on creating a dedicated pool.)

This ensures that the web site queries always run and never queue behind the large batch jobs. Leave the batch jobs to run off the GENERAL pool.

For example, the following pool is based on the average resources needed for the queries running from the web and the expected number of concurrent queries. It also has a higher PRIORITY to the web queries over any running batch jobs and assumes the queries are being tuned to take 250M each:

```
CREATE RESOURCE POOL web_pool MEMORYSIZE '250M' MAXMEMORYSIZE NONE
PRIORITY 10 MAXCONCURRENCY 5 PLANNEDCONCURRENCY 1
```

- **METHOD 2:** Create a standalone pool to limit the batch reports down to a fixed memory size so memory is always left available for other purposes. (See **Scenario: Restricting Resource Usage of Ad-hoc Query Application** (page 306).)

For example:

```
CREATE RESOURCE POOL batch_pool MEMORYSIZE '4G'
```

```
MAXMEMORYSIZE '4G' MAXCONCURRENCY 10:
```

The same principle can be applied if you have three or more distinct classes of workloads.

Scenario: Setting Priorities on Queries Issued by Different Users

Scenario

We would like user queries from one department to have a higher priority than another department.

Solution

The solution is very similar to the one discussed for *mixed workload case* (page 308). In this case, the issue is not to limit resource usage but to set different priorities. To do so, create two different pools each with MEMORYSIZE=0% and a different PRIORITY parameter. Both pools borrow from the GENERAL pool, however when competing for resources, the priority determine the order in which each pool's request is granted. For example:

```
CREATE RESOURCE POOL dept1_pool PRIORITY 5;
CREATE RESOURCE POOL dept2_pool PRIORITY 8;
```

If you find that this solution is not sufficient, or if one department continuously starves another department's users, you could add a reservation for each pool by setting MEMORYSIZE so some memory is guaranteed to be available for each department.

For example, since both resources are using the GENERAL pool for memory, you could allocate some memory to each resource pool by using the ALTER RESOURCE POOL command to change the MEMORYSIZE for each pool:

```
ALTER RESOURCE POOL dept1_pool MEMORYSIZE '100M';
ALTER RESOURCE POOL dept2_pool MEMORYSIZE '150M';
```

Scenario: Continuous Load and Query

Scenario

Our application would like to run continuous load streams, and many have up concurrent query streams. How can we ensure that performance is predictable?

Solution

The solution to this scenario will depend a lot on your query mix, however below are the general steps to take:

- 1 Determine the number of continuous load streams required. This may be related to the desired load rate if a single stream does not provide adequate throughput, or may be more directly related to the number of sources of data to load. Also determine if automatic storage is best, or if DIRECT is required. Create a dedicated resource pool for the loads, and associate it with the database user that will perform them. See CREATE RESOURCE POOL for details.

In general, the concurrency settings for the load pool should be less than the number of cores per node. Unless the source processes are slow, it is more efficient to dedicate more memory per load, and have additional loads queue. Adjust the load pool's QUEUETIMEOUT setting if queueing is expected.

- 2 If using automatic targeting of COPY and INSERT, set the PLANNEDCONCURRENCY parameter of the WOSDATA pool to the number of concurrent loads expected. Also, set MEMORYSIZE of the WOS to the expected size of the loaded data to ensure that small loads don't spill to ROS immediately. See *Built-in Pools* for details.
- 3 Run the load workload for a while and observe whether the load performance is as expected. If the Tuple Mover is not tuned adequately to cover the load behavior, see *Tuning the Tuple Mover* (page 167) in Administrator's Guide.
- 4 If there is more than one kind of query in the system (say some queries that must be answered quickly for interactive users, and others that are part of a batch reporting process), follow the advice in *Scenario: Handling Mixed Workloads* (page 308).
- 5 Let the queries run and observe the performance. If some classes of queries are not getting the desired performance, then it may be necessary to tune *the GENERAL pool* (page 294) as outlined in *Scenario: Restricting Resource Usage of Ad-hoc Query Application* (page 306), or to create further dedicated resource pools for those queries. See *Scenario: The CEO Query* (page 304) and *Scenario: Handling Mixed Workloads* (page 308).

See the sections on *Managing Workloads* (page 292) and CREATE RESOURCE POOL for additional details and tips for obtaining predictable results in mixed workload environments.

Tuning the Built-in Pools

The scenarios in this section describe how to tune the built-in pools.

Scenario: Restricting Vertica to Take Only 60% of Memory

Scenario

We have a single node application embedding Vertica, and some portion of the RAM needs to be devoted to the application process. Can we limit Vertica to use only 60% of the available RAM?

Solution

Set the MAXMEMORYSIZE parameter of the GENERAL pool to the desired memory size. See *Resource Pool Architecture* (page 294) for a discussion on resource limits.

Scenario: Tuning for Recovery

Scenario

We have a large database containing a single large table with two projections. With out-of-the-box settings, recovery is taking too long. Is there a way to give recovery more memory to improve speed?

Solution

Set the PLANNEDCONCURRENCY and MAXCONCURRENCY setting of the recovery pool to 1 so that recovery can take as much memory as possible from the GENERAL pool and run only one thread at once.

Note: This setting could slow down other queries in your system.

Scenario: Tuning for Refresh

Scenario

When refresh is running, the system performance is impacted and user queries get rejected. Can we reduce the memory usage of the refresh job?

Solution

Set the MEMORYSIZE parameter of the refresh pool to a fixed value. The Resource Manager then tunes the refresh query to only use this amount of memory.

Tip: Remember to reset the refresh pool MEMORYSIZE back to 0% after refresh is finished so the memory can be used for other operations.

Scenario: Tuning Tuple Mover Pool Settings

Scenario

During loads, we occasionally see spikes in number of ROS containers. How can we make the Tuple Mover more aggressive?

Solution

Increase the MAXCONCURRENCY parameter of the `TM` pool to 3 or higher. This setting ensures that the Tuple Mover can run more than one mergeout thread, so if a large mergeout is in progress, smaller ROS containers can also be merged, thus preventing a buildup of ROS containers.

Reducing Run-time of Queries

The run time of queries depends on the complexity of the query, the number of operators in the plan, data volumes, and projection design. If the system is bottlenecked on either I/O or CPU, queries could run more slowly than expected. In most cases, high CPU usage can be alleviated by better projection design, and high I/O is usually due to contention because of operations like joins and sorts that spill to disk. However, there is no one-size fix for high CPU or high I/O usage, so queries must be examined and tuned individually.

Two primary ways to determine why a query is slow are:

- Examine the query plan via EXPLAIN
- Examine the execution profile (EXECUTION_ENGINE_PROFILES).

Examining the plan will reveal one more more of the following:

- Suboptimal sort order
- Cases when predicate evaluation occurs on an unsorted or unencoded column
- Cases where data is uncompressed prematurely
- Occurrence of a partition hash join instead of merge join
- Presence of Group by Hash rather than pipeline

See **Creating Custom Designs** (page 91) to understand projection design techniques. The Database Designer automatically applies these techniques to suggest optimal designs for queries.

Real-time Profiling

Vertica provides profiling mechanisms that let you determine how well the database is performing. For example, Vertica can collect profiling data for a single statement, a single session, or for all sessions on all nodes.

Real-time profiling is always "on", without profiling being explicitly enabled.

For details, see Profiling Database Performance in the Troubleshooting Guide and, in particular:

- Profiling a Single Statement
- Real-time Profiling
- Viewing Profiling Data
- Viewing Real-time Profiling Data

See also EXECUTION_ENGINE_PROFILES in the SQL Reference Manual

Managing System Resource Usage

You can use the **SQL Monitoring APIs (system tables)** (page 199) to track overall resource usage on your cluster. These and the other system tables are described in the SQL Reference Manual.

If your queries are experiencing errors due to resource unavailability, you can use the following system tables to obtain more details:

System Table	Description
RESOURCE_REJECTIONS	Monitors requests for resources that are rejected by the Resource Manager.
DISK_RESOURCE_REJECTIONS	Monitors requests for resources that are rejected due to disk space shortages. See Managing Disk Space (page 283) for more information.

When requests for resources of a certain type are being rejected, do one of the following:

- Increase the resources available on the node by adding more memory, more disk space, and so on. See **Managing Disk Space** (page 283).
- Reduce the demand for the resource by reducing the number of users on the system (see **Managing Sessions** (page 313)), rescheduling operations, and so on.

The LAST_REQUEST_REJECTED_REASON field in RESOURCE_REJECTIONS indicates the cause of the problem. For example:

- The message `Usage of a single requests exceeds high limit` means that the system does not have enough of the resource available for the single request. A common example occurs when the file handle limit is set too low and you are loading a table with a large number of columns.

See [Increase the Maximum Number of Files Open in the Installation Guide](#) for more information.

- The message `Timed out or Canceled waiting for resource reservation` usually means that there is too much contention for the resource because the hardware platform cannot support the number of concurrent users using it. Contact **Technical Support** (on page 1) for assistance in this area.

See Also

Guidelines for Setting Pool Parameters (page 301)

Managing Sessions

Vertica provides powerful methods for database administrators to view and control sessions. The methods vary according to the type of session:

- External (user) sessions are initiated by `vsq` or programmatic (ODBC or JDBC) connections and have associated client state.
- Internal (system) sessions are initiated by the Vertica database process and have no client state.

You can view a list of currently active sessions (including internal sessions) and can interrupt or close external sessions when necessary, particularly when ***shutting down the database*** (page 338).

By default Vertica allows 50 client sessions and an additional 5 administrator sessions. You can modify connection settings with the `MaxClientSessions` parameter. For example, to increase the number of `MaxClientSessions` to 100, issue the following command at a `vsq` prompt:

```
=> SELECT SET_CONFIG_PARAMETER('MaxClientSessions', 100);
```

To prevent new non-dbadmin sessions from connecting, set `MaxClientSessions` to 0:

```
=> SELECT SET_CONFIG_PARAMETER('MaxClientSessions', 0);
```

Viewing Sessions

Vertica provides the `SESSIONS` table to view the session status of your database. `SESSIONS` contains information about external sessions and returns one row per session. This table is described in the SQL Reference Manual.

Note: Superuser has unrestricted access to all database metadata. Users have significantly reduced access to metadata based on their privileges. See ***Metadata Privileges*** (page 139).

Interrupting and Closing Sessions

- Interrupting a running statement returns an enclosing session to an idle state, meaning no statements or transactions are running, no locks are held, and the database is doing no work on behalf of the session. If no statement is running, you get an error.
- Closing a session interrupts the session and disposes of all state related to the session, including client socket connections for external sessions.

These actions are provided in the form of SQL functions, described in the SQL Reference Manual:

- `INTERRUPT_STATEMENT`

- CLOSE_SESSION
- CLOSE_ALL_SESSIONS
- SHUTDOWN

SELECT statements that call these functions return when the interrupt or close message has been delivered to all nodes, not after the interrupt or close has completed. This means there might be a delay after the statement returns and the interrupt or close taking effect throughout the cluster. To determine if the session or transaction has ended, you can monitor the SESSIONS system table.

Controlling Sessions

The database administrator must be able to disallow new incoming connections in order to shut down the database. On a busy system, database shutdown is prevented if new sessions connect after the CLOSE_SESSION or CLOSE_ALL_SESSIONS() command is invoked — and before the database actually shuts down.

One option is for the administrator to issue the SHUTDOWN('true') command, which forces the database to shut down and disallow new connections. See SHUTDOWN in the SQL Reference Manual.

Another option is to modify the MaxClientSessions parameter from its original value to 0, in order to prevent new non-dbadmin users from connecting to the database.

- 1 Determine the original value for the MaxClientSessions parameter by querying the V_MONITOR.CONFIGURATIONS_PARAMETERS system table:

```
=> SELECT CURRENT_VALUE FROM CONFIGURATION_PARAMETERS WHERE
      parameter_name='MaxClientSessions';
      CURRENT_VALUE
      -----
          50
(1 row)
```

- 2 Set the MaxClientSessions parameter to 0 to prevent new non-dbadmin connections:

```
=> SELECT SET_CONFIG_PARAMETER('MaxClientSessions', 0);
```

Note: The previous command allows up to five administrators to log in.

- 3 Issue the CLOSE_ALL_SESSIONS() command to remove existing sessions:

```
=> SELECT CLOSE_ALL_SESSIONS();
```

- 4 Query the SESSIONS table:

```
=> SELECT * FROM SESSIONS;
```

When the session no longer appears in the SESSIONS table, disconnect and run the **Stop Database** (page 338) command.

- 5 Restart the database.

- 6 Restore the MaxClientSessions parameter to its original value:

```
=> SELECT SET_CONFIG_PARAMETER('MaxClientSessions', 50);
```

See Also

Administrator's Guide

- **Configuration Parameters** (page 25)
- **Stop Database** (page 338)
- **Shutdown Problems** (page 237)

SQL Reference Manual

- SESSIONS
- CONFIGURATION_PARAMETERS
- CLOSE_ALL_SESSIONS
- CLOSE_SESSION
- INTERRUPT_STATEMENT
- SESSIONS
- SHUTDOWN

Troubleshooting Guide

- New Session Rejected Due to Limit
- ... **Users are connected** (page 238)

Managing Load Streams

You can use the **SQL Monitoring API** (page 199) to keep track of data being loaded on your cluster.

System Table	Description
LOAD_STREAMS	Monitors load metrics for each load stream on each node.

These and the other SQL Monitoring API system tables are described in detail in the SQL Reference Manual.

If a COPY ... DIRECT operation is in progress, the `ACCEPTED_ROW_COUNT` field could increase up to the maximum number of rows in the input file as the rows are being parsed. If COPY reads from many named pipes, `PARSE_COMPLETE_PERCENT` shows 0 until it receives an EOF from *all* named pipes. This can take a significant amount of time, and it is easy to mistake this state as a hang. Check your **system CPU and disk accesses** (page 215) to determine if any activity is in progress before canceling COPY or reporting a hang.

In a typical load, you might notice `PARSE_COMPLETE_PERCENT` creep up to 100% or jump to 100% if loading from named pipes or STDIN, while `SORT_COMPLETE_PERCENT` is at 0. Once `PARSE_COMPLETE_PERCENT` reaches 100%, `SORT_COMPLETE_PERCENT` creeps up to 100%. Depending on the data sizes, there could be significant lag between the time `PARSE_COMPLETE_PERCENT` reaches 100% and the time `SORT_COMPLETE_PERCENT` begins to increase.

Load Balancing

In Vertica, load balancing supports multiple client connections through a single Virtual IP (VIP) address that is shared among all nodes in a cluster. This is useful for balancing incoming client requests across nodes, as well as preventing node exclusion from clients in the case of node failure.

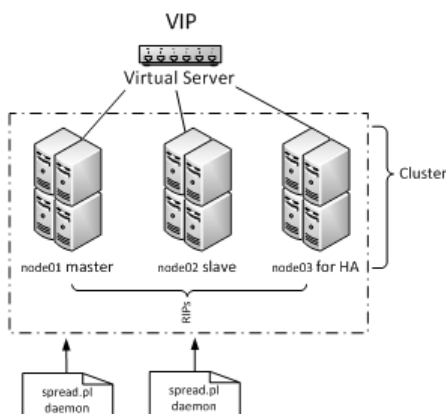
The IP Virtual Server (IPVS) is a network bridge that balances the connection streams. IPVS is made up of the following components:

- The Virtual IP (VIP): The IP address that is accessed by all client connections.
- Real server IPs (RIP): The IP addresses of client network interfaces used for connecting database clients to the database engine.
- Cluster: A cluster of real Vertica servers (nodes).
- Virtual server: The single point of entry (network bridge) that provides access to a cluster.

Client connections made through the Virtual IP (VIP) are managed by a primary (master) director node, which is one of the real server nodes (RIP). The master director handles the routing of requests by determining which node has the fewest connections and sending connections to that node. If the director node fails for any reason, a failover (slave) director takes over request routing until the primary (master) director comes back online.

If a user connects to node03 in a three-node cluster and node03 fails, the current transaction rolls back, the client connection fails, and a connection must be reestablished on another node.

The following graphic illustrates a three-node database cluster where all nodes share a single VIP. The cluster contains a master director (node01), a slave director (node02), and an additional host (node03) that together provide the minimum configuration for high availability (K-safety). In this setup (and in the configuration and examples that follow), node01 and node02 play dual roles as IPVS directors and Vertica nodes.



Notes

- Load balancing on a VIP is supported for Linux Red Hat Enterprise Linux 5, 64-bit.
- Vertica must be installed on each node in the cluster; the database can be installed on any node, but only one database can be running on a Vertica cluster at a time.

- Although a 0 K-safety (two-node) design is supported, Vertica strongly recommends that you create the load-balancing network using a minimum three-node cluster with K-safety set to 1. This way if one node fails, the database stays up. See **Designing for K-Safety** (page 92) for details.
- Subsequent topics in this section describe how to set up two directors (master and slave), but you can set up more than two directors. See the **Keepalived User Guide** <http://www.keepalived.org/pdf/UserGuide.pdf> for details. See also the **Linux Virtual Server Web site** <http://www.linux-vs.org/>.

Configuring Vertica Nodes

This section describes how to configure a Vertica cluster of nodes for load balancing. You'll set up two directors in a master/slave configuration and include a third node for K-safety.

A Vertica cluster designed for load balancing uses the following configuration:

- **Real IP (RIP)** address is the public interface and includes:
 - The master director/node, which handles the routing of requests.
 - The slave director/node, which communicates with the master and takes over routing requests in the event of a master node failure.
 - Unlimited n nodes, such as at least one failover node to provide the minimum configuration for high availability (K-safety).
- **Virtual IP (VIP)** address (generally assigned to eth0 in Linux) is the public network interface over which database clients connect.

Note: The VIP must be public so that clients outside the cluster can contact it.

Once you have set up a Vertica cluster and created a database, you can choose any two nodes to be directors. The instructions in this section use the following node configuration:

Preconfigured IP	Node assignment	Public IPs	Private IPs
VIP	shared among all nodes	10.10.51.180	
RIP master director	node01	10.10.51.55	192.168.51.1
RIP slave director	node02	10.10.51.56	192.168.51.2
RIP failover node	node03	10.10.51.57	192.168.51.3

Notes

- In the above table, the private IPs determine which node to send a request to. They are not the same as the RIPs.
- The VIP must be on the same subnet as the nodes in the Vertica cluster.
- Both the master and slave nodes (node01 and node02 in this section) require additional installation and configuration, as described in **Configuring the Directors** (page 320).
- Use the command `$ cat /etc/hosts` to display a list of all hosts in your cluster

See Also

The following external web sites might be useful. The links worked at the last date of publication, but Vertica does not manage this content. Please report any broken links to **Technical Support** (on page 1).

Linux Virtual Server Web site <http://www.linux-vs.org/>

LVS-HOWTO Page <http://www.austintek.com/LVS/LVS-HOWTO/HOWTO/>

Keepalived.conf(5) man page <http://linux.die.net/man/5/keepalived.conf>

ipvsadm man page

http://at.gnucash.org/vhost/linuxcommand.org/man_pages/ipvsadm8.html

Set Up the Loopback Interface

This procedure sets up the loopback (lo) interface with an alias on each node.

- 1 Log in as root on the master director (node01):

```
$ su - root
```

- 2 Use the text editor of your choice to open `ifcfg-lo`:

```
[root@node01]# vi /etc/sysconfig/network-scripts/ifcfg-lo
```

- 3 Set up the loopback adapter with an alias for the VIP by adding the following block to the end of the file:

```
## vip device
DEVICE=lo:0
IPADDR=10.10.51.180
NETMASK=255.255.255.255
ONBOOT=yes
NAME=loopback
```

Note: When you add the above block to your file, be careful not to overwrite the `127.0.0.1` parameter, which is required for proper system operations.

- 4 Repeat steps 1-3 on each node in the Vertica cluster.

Disable Address Resolution Protocol (ARP)

This procedure disables ARP (Address Resolution Protocol) for the VIP.

- 1 On the master director (node01), log in as root:

```
$ su - root
```

- 2 Use the text editor of your choice to open the `sysctl` configuration file:

```
[root@node01]# vi /etc/sysctl.conf
```

- 3 Add the following block to the end of the file:

```
#LVS
net.ipv4.conf.eth0.arp_ignore = 1
net.ipv4.conf.eth0.arp_announce = 2
# Enables packet forwarding
net.ipv4.ip_forward = 1
```

Note: For additional details, refer to the *LVS-HOWTO Page* <http://www.austintek.com/LVS/LVS-HOWTO/HOWTO/>. You might also refer to the *Linux Virtual Server Wiki page* http://kb.linuxvirtualserver.org/wiki/Using_arp_announce/arp_ignore_to_disable_ARP for information on using `arp_announce/arp_ignore` to disable the Address Resolution Protocol.

4 Use `ifconfig` to verify that the interface is on the same subnet as the VIP:

```
[root@node01]# /sbin/ifconfig
```

In the following output, the `eth0` `inet` `addr` is the VIP, and subnet 51 matches the private RIP under the `eth1` heading:

```
eth0      Link encap:Ethernet  HWaddr 84:2B:2B:55:4B:BE
          inet addr:10.10.51.55  Bcast:10.10.51.255  Mask:255.255.255.0
          inet6 addr: fe80::862b:2bff:fe55:4bbe/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:91694543 errors:0 dropped:0 overruns:0 frame:0
          TX packets:373212 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:49294294011 (45.9 GiB)  TX bytes:66149943 (63.0 MiB)
          Interrupt:15 Memory:da000000-da012800

eth1      Link encap:Ethernet  HWaddr 84:2B:2B:55:4B:BF
          inet addr:192.168.51.55  Bcast:192.168.51.255
          Mask:255.255.255.0
          inet6 addr: fe80::862b:2bff:fe55:4bbf/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:937079543 errors:0 dropped:2780 overruns:0 frame:0
          TX packets:477401433 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:449050544237 (418.2 GiB)  TX bytes:46302821625 (43.1
          GiB)
          Interrupt:14 Memory:dc000000-dc012800

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:6604 errors:0 dropped:0 overruns:0 frame:0
          TX packets:6604 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:21956498 (20.9 MiB)  TX bytes:21956498 (20.9 MiB)

lo:0      Link encap:Local Loopback
          inet addr:10.10.51.180  Mask:255.255.255.255
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
```

5 Use `ifconfig` to verify that the loopback interface is up:

```
[root@node01]# /sbin/ifconfig lo:0
```

You should see output similar to the following:

```
lo:0    Link encap:Local Loopback
        inet addr:10.10.51.180 Mask:255.255.255.255
        UP LOOPBACK RUNNING MTU:16436 Metric:1
```

If you do not see UP LOOPBACK RUNNING, bring up the loopback interface:

```
[root@node01]# /sbin/ifup lo
```

6 Issue the following command to commit changes to the kernel from the configuration file:

```
[root@node01]# /sbin/sysctl -p
```

7 Repeat steps 1-6 on all nodes in the Vertica cluster.

Configuring the Directors

Now you are ready to install the Vertica IPVS Load Balancer package and configure the master (node01) and slave (node02) directors.

Install the Vertica IPVS Load Balancer Package

In this procedure you download and install the Vertica IPVS Load Balancer package.

1 On the master director (node01) log in as root:

```
$ su - root
```

2 Download the `VerticaIPVSLoadBalancer` package from the Vertica **Downloads page** http://www.vertica.com/v-zone/download_vertica.

3 Install (or upgrade) the Load Balancer package:

```
[root@node01]# rpm -Uvh VerticaIPVSLoadBalancer-4.1.x.RHEL5.x86_64.rpm
```

4 Repeat steps 1-3 on the slave director (node02).

Configure the Vertica IPVS Load Balancer

Vertica provides a script called `configure-keepalived.pl` in the IPVS Load Balancer package.

The script is located in `/sbin`, and if you run it with no options it prints a usage summary:

```
--ripips      | Comma separated list of Vertica nodes; public IPs (e.g., 10.10.51.55, etc.)
--riport      | Port on which Vertica runs. Default is 5433
--iface       | Public ethernet interface Vertica is configured to use (e.g., eth0)
--emailto     | Address that should get alerts (e.g., user@server.com)
--emailfrom   | Address that mail should come from (e.g., user@server.com)
--mailserver  | E-mail server IP or hostname (e.g., mail.server.com)
--master      | If this director is the master (default), specify --master
--slave       | If this director is the slave, specify --slave
--authpass    | Password for keepalived
--vip         | Virtual IP address (e.g., 10.10.51.180)
--delayloop   | Seconds keepalived waits between healthchecks. Default is 2
--algo        | Sets the algorithm to use: rr, wrr, lc (default), wlc, lbic, lblcr, dh, sh, sed, nq
--kind        | Sets the routing method to use. Default is DR.
--priority    | By default, master has priority of 100 and the backup has priority of 50
```

For details about each of these parameters, particularly, `--algo` and `--kind`, refer to the ***ipvsadm(8) - Linux man page*** http://at.gnucash.org/vhost/linuxcommand.org/man_pages/ipvsadm8.html.

Set up the Vertica IPVS Load Balancer configuration file

- 1 On the master director (node01) log in as root:

```
$ su - root
```

- 2 Run the Vertica-supplied configuration script with the appropriate switches; for example:

```
[root@node01]# /sbin/configure-keepalived.pl --ripips
192.168.51.55,192.168.51.56,
192.168.51.57 --riport 5433 --iface eth0 --emailto
dbadmin@companyname.com
--emailfrom dbadmin@companyname.com --mailserver mail.server.com
--master
--authpass password --vip 10.10.51.180 --delayloop 2 --algo lc --kind
DR
--priority 100
```

IMPORTANT: The `--authpass` (password) switch must be the same on both the master and slave directors.

- 3 Check the keepalived configuration file to make sure the `real_server` IP address is public. You might need to edit this file to make sure the assignments are correct.

- The `real_server` line contains IP addresses that are RIPv4s (e.g., 10.10.51.55)
- The `MISC_CHECK` line requires that the RIPv4s be private (e.g., 192.168.51.55). Thus, the `--ripips` parameter should be the private interface (Vertica-supplied IPs or names that resolve to these IPs) and not the public interface.

They should be IP addresses, not node names because otherwise the `spread.pl` script would fail.

```
[root@p6 keepalived]# cat /etc/keepalived/keepalived.conf
! Configuration File for keepalived
global_defs {
    notification_email {
        dbadmin@companyname.com !
    }
    notification_email_from dbadmin@companyname.com !Email comes from
this address
    smtp_server mail.server.com !Your email server
    smtp_connect_timeout 30
    router_id node1
}
vrrp_instance VI_1 {
    state MASTER
    interface eth0
    smtp_alert
    virtual_router_id 1
    priority 100
    advert_int 10
    authentication {
        auth_type PASS
        auth_pass password
    }
}
```

```

    virtual_ipaddress {
        10.10.51.180
    }
    # Invoked to master transition
    notify_master "/etc/keepalived/bypass_ipvs.pl del 10.10.51.180"
    # Invoked to slave transition
    notify_backup "/etc/keepalived/bypass_ipvs.pl add 10.10.51.180"
    # Invoked to fault transition
    notify_fault "/etc/keepalived/bypass_ipvs.pl add 10.10.51.180"
}
virtual_server 10.10.51.180 5433
{
    delay_loop 2
    lb_algo lc
    lb_kind DR
    protocol TCP
    real_server 10.10.51.55 5433 {
        MISC_CHECK {
            misc_path "/etc/keepalived/check.pl 192.168.51.55"
        }
    }
    real_server 10.10.51.56 5433 {
        MISC_CHECK {
            misc_path "/etc/keepalived/check.pl 192.168.51.56"
        }
    }
    real_server 10.10.51.57 5433 {
        MISC_CHECK {
            misc_path "/etc/keepalived/check.pl 192.168.51.57"
        }
    }
}
}

```

4 Start spread:

```
[root@node01]# /etc/init.d/spread.pl start
```

The `spread.pl` script writes to the `check.txt` file, which is rewritten to include only the remaining nodes in the event of a node failure. Thus, the virtual server knows to stop sending vsql requests to the failed node.

5 Start keepalived on node01:

```
[root@node01]# /etc/init.d/keepalived start
```

6 If not already started, start sendmail to allow mail messages to be sent by the directors:

```
[root@node01]# /etc/init.d/sendmail start
```

7 Repeat steps 1-5 on the slave director (node02), using the same switches, except (IMPORTANT**) replace the `--master` switch with the `--slave` switch. For example:**

```
[root@node01]# /sbin/configure-keepalived.pl --ripips
    192.168.51.55,192.168.51.56,
192.168.51.57 --ripport 5433 --iface eth0 --emailto
dbadmin@companyname.com
```



```
--emailfrom dbadmin@companyname.com --mailserver mail.server.com
--slave
--authpass password --vip 10.10.51.180 --delayloop 2 --algo lc --kind
DR
--priority 100
```

See Also

Keepalived.conf(5) -Linux man page <http://linux.die.net/man/5/keepalived.conf>

Connecting to the Virtual IP (VIP)

To connect to the Virtual IP address using vsql, issue a command similar to the following. The IP address, which could also be a DNS address, is the VIP that is shared among all nodes in the Vertica cluster.

```
$ /opt/vertica/bin/vsql -h 10.10.51.180 -U dbadmin
```

To verify connection distribution over multiple nodes, repeat the following statement multiple times and observe connection distribution in an `lc` (least amount of connections) fashion.

```
$ vsql -h <VIP> -c "SELECT node_name FROM sessions"
```

Replace `<VIP>` in the above script with the IP address of your virtual server; for example:

```
$ vsql -h 10.10.51.180 -c "SELECT node_name FROM sessions"
 node_name
-----
v_ipvs_node01
v_ipvs_node02
v_ipvs_node03
(3 rows)
```

Monitoring Which Nodes Are Connected

If you want to monitor which nodes are sharing connections, view the `check.txt` file by issuing the following command at a shell prompt:

```
# watch cat /etc/keepalived/check.txt
Every 2.0s: cat /etc/keepalived/check.txt          Wed Nov  3 10:02:20 2010
N192168051057
N192168051056
N192168051055
```

The `check.txt` is a file located in the `/etc/keepalived/` directory, and it gets updated when you submit changes to the kernel using `sysctl -p`, described in **[Disable the Address Resolution Protocol \(ARP\)](#)** (page 115). For example, the `spread.pl` script (see **[Configuring the Directors](#)** (page 320)), writes to the `check.txt` file, which is then modified to include only the remaining nodes in the event of a node failure. Thus, the virtual server knows to stop sending vsql requests to the failed node.

You can also look for messages by issuing the following command at a shell prompt:

```
# tail -f /var/log/messages
```

```
Nov 3 09:21:00 p6 Keepalived: Starting Keepalived v1.1.17 (05/17,2010)
Nov 3 09:21:00 p6 Keepalived: Starting Healthcheck child process, pid=32468
Nov 3 09:21:00 p6 Keepalived: Starting VRRP child process, pid=32469
Nov 3 09:21:00 p6 Keepalived_healthcheckers: Using LinkWatch kernel netlink reflector...
Nov 3 09:21:00 p6 Keepalived_vrrp: Using LinkWatch kernel netlink reflector...
Nov 3 09:21:00 p6 Keepalived_healthcheckers: Netlink reflector reports IP 10.10.51.55 added
Nov 3 09:21:00 p6 Keepalived_vrrp: Netlink reflector reports IP 10.10.51.55 added
Nov 3 09:21:00 p6 Keepalived_healthcheckers: Netlink reflector reports IP 192.168.51.55 added
Nov 3 09:21:00 p6 Keepalived_vrrp: Netlink reflector reports IP 192.168.51.55 added
Nov 3 09:21:00 p6 Keepalived_vrrp: Registering Kernel netlink reflector
Nov 3 09:21:00 p6 Keepalived_healthcheckers: Registering Kernel netlink reflector
Nov 3 09:21:00 p6 Keepalived_vrrp: Registering Kernel netlink command channel
Nov 3 09:21:00 p6 Keepalived_vrrp: Registering gratuitous ARP shared channel
Nov 3 09:21:00 p6 Keepalived_healthcheckers: Registering Kernel netlink command channel
Nov 3 09:21:00 p6 Keepalived_vrrp: Opening file '/etc/keepalived/keepalived.conf'.
Nov 3 09:21:00 p6 Keepalived_healthcheckers: Opening file '/etc/keepalived/keepalived.conf'.
Nov 3 09:21:00 p6 Keepalived_vrrp: Configuration is using : 63730 Bytes
Nov 3 09:21:00 p6 Keepalived_healthcheckers: Configuration is using : 16211 Bytes
Nov 3 09:21:00 p6 Keepalived_healthcheckers: Activating healthchecker for service [10.10.51.55:5433]
Nov 3 09:21:00 p6 Keepalived_healthcheckers: Activating healthchecker for service [10.10.51.56:5433]
Nov 3 09:21:00 p6 Keepalived_healthcheckers: Activating healthchecker for service [10.10.51.57:5433]
Nov 3 09:21:00 p6 Keepalived_vrrp: VRRP sockpool: [ifindex(2), proto(112), fd(10,11)]
Nov 3 09:21:01 p6 Keepalived_healthcheckers: Misc check to [10.10.51.56] for
[/etc/keepalived/check.pl 192.168.51.56] failed.
Nov 3 09:21:01 p6 Keepalived_healthcheckers: Removing service [10.10.51.56:5433] from VS
[10.10.51.180:5433]
Nov 3 09:21:01 p6 Keepalived_healthcheckers: Remote SMTP server [127.0.0.1:25] connected.
Nov 3 09:21:01 p6 Keepalived_healthcheckers: Misc check to [10.10.51.55] for
[/etc/keepalived/check.pl 192.168.51.55] failed.
Nov 3 09:21:01 p6 Keepalived_healthcheckers: Removing service [10.10.51.55:5433] from VS
[10.10.51.180:5433]
Nov 3 09:21:01 p6 Keepalived_healthcheckers: Remote SMTP server [127.0.0.1:25] connected.
Nov 3 09:21:01 p6 Keepalived_healthcheckers: Misc check to [10.10.51.57] for
[/etc/keepalived/check.pl 192.168.51.57] failed.
Nov 3 09:21:01 p6 Keepalived_healthcheckers: Removing service [10.10.51.57:5433] from VS
[10.10.51.180:5433]
Nov 3 09:21:01 p6 Keepalived_healthcheckers: Remote SMTP server [127.0.0.1:25] connected.
Nov 3 09:21:01 p6 Keepalived_healthcheckers: SMTP alert successfully sent.
Nov 3 09:21:10 p6 Keepalived_vrrp: VRRP_Instance(VI_1) Transition to MASTER STATE
Nov 3 09:21:20 p6 Keepalived_vrrp: VRRP_Instance(VI_1) Entering MASTER STATE
Nov 3 09:21:20 p6 Keepalived_vrrp: VRRP_Instance(VI_1) setting protocol VIPs.
Nov 3 09:21:20 p6 Keepalived_vrrp: VRRP_Instance(VI_1) Sending gratuitous ARPs on eth0 for
10.10.51.180
Nov 3 09:21:20 p6 Keepalived_healthcheckers: Netlink reflector reports IP 10.10.51.180 added
Nov 3 09:21:20 p6 Keepalived_vrrp: Remote SMTP server [127.0.0.1:25] connected.
Nov 3 09:21:20 p6 Keepalived_vrrp: Netlink reflector reports IP 10.10.51.180 added
Nov 3 09:21:20 p6 Keepalived_vrrp: SMTP alert successfully sent.
Nov 3 09:21:25 p6 Keepalived_vrrp: VRRP_Instance(VI_1) Sending gratuitous ARPs on eth0 for 10.10.51.1
```

Determining Where Connections Are Going

Ipvsadm is the user code interface to the IP Virtual Server. It is used to set up, maintain, or inspect the virtual server table in the Linux kernel.

If you want to identify where user connections are going, install ipvsadm.

1 Log in to the master director (node01) as root:

```
$ su - root
```

2 Install ipvsadm:

```
[root@node01]# yum install ipvsadm
Loading "installonlyn" plugin
Setting up Install Process
Setting up repositories
```

```

Reading repository metadata in from local files
Parsing package install arguments
Resolving Dependencies
--> Populating transaction set with selected packages. Please wait.
---> Downloading header for ipvsadm to pack into transaction set.
ipvsadm-1.24-10.x86_64.rp 100% |=====| 6.6 kB
    00:00
---> Package ipvsadm.x86_64 0:1.24-10 set to be updated
--> Running transaction check

```

Dependencies Resolved

Package Size	Arch	Version	Repository
=====			
Installing:			
ipvsadm 32 k	x86_64	1.24-10	base

Transaction Summary

```

=====
Install      1 Package(s)
Update       0 Package(s)
Remove       0 Package(s)

```

```

Total download size: 32 k
Is this ok [y/N]: y
Downloading Packages:
(1/1): ipvsadm-1.24-10.x8 100% |=====| 32 kB
    00:00

```

```

Running Transaction Test
Finished Transaction Test
Transaction Test Succeeded
Running Transaction

```

```

    Installing: ipvsadm                               #####
    [1/1]

```

```

Installed: ipvsadm.x86_64 0:1.24-10
Complete!

```

3 Run ipvsadm:

```

[root@node01 ~]# ipvsadm
IP Virtual Server version 1.2.1 (size=4096)
Prot LocalAddress:Port Scheduler Flags
  -> RemoteAddress:Port          Forward Weight ActiveConn InActConn
TCP  vs-wks1.verticacorp.com:pyrr lc
  -> node03.verticacorp.com:pyr Route      1      1      8
  -> node02.verticacorp.com:pyr Route      1      0      8

```

```
-> node01.verticacorp.com:pyr Local 1 0 8
```

See Also

ipvsadm man page

http://at.gnucash.org/vhost/linuxcommand.org/man_pages/ipvsadm8.html

Virtual IP Connection Problems

Issue

Users cannot connect to the database through the Virtual IP (VIP) address.

Resolution

- 1 Check if spread is running:

```
$ ps ax | grep spread
```

```
11895 ?          S<s      4:30 /opt/vertica/spread/sbin/spread -n  
      N192168051055 -c  
                               /opt/vertica/config/vspread.conf  
29617 pts/3      S+       0:00 grep spread
```

1. If spread is not running, start spread as root or using sudo:

```
[root@node01]# /etc/init.d/spreadd start
```

2. If spread is running, restart spread as root or using sudo:

```
[root@node01]# /etc/init.d/spreadd restart
```

3. Check the spread status as root or using sudo:

```
[root@node01]# /etc/init.d/spreadd status
```

4. Issue the `ifconfig` command to check the current IP addresses of the hosts, and verify that those IP addresses are listed in `/opt/vertica/config/vspread.conf`.

```
[root@node01]# ifconfig
```

If spread fails to start, examine the following files for problems:

```
/tmp/spread*.log
```

```
/var/log/spreadd.log
```

Permission problems and syntax problems are identified in the log files.

- 2 Check if keepalived is running:

```
$ ps ax | grep keepalived
```

```
29622 pts/3      S+       0:00 grep keepalived
```

1. If keepalived is not running, start keepalived as root or using sudo:

```
# /etc/init.d/keepalived start
```

2. If keepalived is running, restart keepalived as root or using sudo:

```
# /etc/init.d/keepalived restart
```

Issue

Users cannot connect to the database.

Resolution

Try to telnet to the VIP and port:

```
# telnet 10.10.51.180 5433
```

If telnet reports no route to host, recheck your `/etc/keepalived/keepalived.conf` file to make sure you entered the correct VIP and RIPs.

Errors and informational messages from the keepalived daemon are written to the `/var/log/messages` file, so check the messages file first:

```
# tail -f /var/log/messages
```

```
May 18 09:04:32 dell02 Keepalived_vrrp: VRRP_Instance(VI_1) Sending gratuitous
ARPs on eth0 for 10.10.10.100
May 18 09:04:32 dell02 avahi-daemon[3191]: Registering new address record for
10.10.10.100 on eth0.
May 18 09:04:32 dell02 Keepalived_healthcheckers: Netlink reflector reports IP
10.10.10.100 added
```

Expected e-mail messages from the keepalived daemon

- Upon startup:
Subject: [node01] VRRP Instance VI_1 - Entering MASTER state
=> VRRP Instance is now owning VRRP VIPs <=
- When a node fails:
Subject: [node01] Realserver 10.10.10.1:5433 - DOWN
=> MISC CHECK failed on service <=
- When a node comes back up:
Subject: [node02] Realserver 10.10.10.1:5433 - UP
=> MISC CHECK succeed on service <=

Troubleshooting Keepalived Issues

If there are connection or other issues related to the Virtual IP server and Keepalived, try some of the following tips:

- Set `KEEPALIVED_OPTIONS="-D -d"` in the `/etc/sysconfig/keepalived` file to enable both debug mode and dump configuration.
- Monitor the system log in `/var/log/messages`. If `keepalived.conf` is incorrect, the only indication is in the messages log file. For example:

```
$ tail /var/log/messages
```

Errors and informational messages from the keepalived daemon are also written to the `/var/log/messages` files.

- Type `ip addr list` and see the configured VIP addresses for `eth0`. For example:

```
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 16436 qdisc noqueue
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
    inet 10.10.51.180/32 brd 127.255.255.255 scope global lo:0
    inet6 ::1/128 scope host
        valid_lft forever preferred_lft forever
```

```
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast qlen
1000
   link/ether 84:2b:2b:55:4b:be brd ff:ff:ff:ff:ff:ff
   inet 10.10.51.55/24 brd 10.10.51.255 scope global eth0
   inet6 fe80::862b:2bff:fe55:4bbe/64 scope link
       valid_lft forever preferred_lft forever
3: eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast
qlen 1000
   link/ether 84:2b:2b:55:4b:bf brd ff:ff:ff:ff:ff:ff
   inet 192.168.51.55/24 brd 192.168.51.255 scope global eth1
   inet6 fe80::862b:2bff:fe55:4bbf/64 scope link
       valid_lft forever preferred_lft forever
4: sit0: <NOARP> mtu 1480 qdisc noop
   link/sit 0.0.0.0 brd 0.0.0.0
```

- Check `iptables` and notice the PREROUTING rule on the BACKUP (slave) director. Even though `ipvsadm` has a complete list of real servers to manage, it does not route anything as the prerouting rule redirects packets to the loopback interface.

```
# /sbin/iptables -t nat -n -L
Chain PREROUTING (policy ACCEPT)
target     prot opt source                destination
Chain POSTROUTING (policy ACCEPT)
target     prot opt source                destination
Chain OUTPUT (policy ACCEPT)
target     prot opt source                destination
```

Note: On some kernels, the `nat` tables does not show by default without the `-t` parameter, and `-n` is used to avoid long DNS lookups. See the *iptables(8) - Linux man page* <http://linux.die.net/man/8/iptables> for details.

- During failover, it is normal to expect delay in new connection establishment until the slave node takes control. The delay could be several minutes depending on the load on the cluster. If you cannot connect to the database, try to telnet to the VIP and port:
telnet 10.10.51.180 5433
If telnet reports no route to host, recheck the `keepalived` configuration file (`/etc/keepalived/keepalived.conf`) to make sure you entered the correct VIP and RIPv.

Using the Administration Tools

Vertica provides a set of tools that allows you to perform administrative tasks quickly and easily. Most of the database administration tasks in Vertica can be done using the Administration Tools.

Note: Always run the Administration Tools using the Database Administrator account on the Administration Host if possible. Make sure that no other Administration Tools processes are running.

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

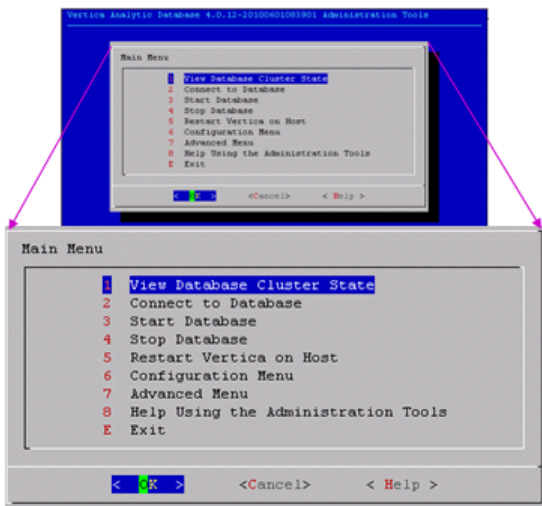
Running the Administration Tools

At the Linux command line:

```
$ /opt/vertica/bin/admintools [ -t | --tool ] toolname [ options ]
```

toolname	Is one of the tools described in the <i>Administration Tools Reference</i> (page 336).	
options	-h --help	Shows a brief help message and exits.
	-a --help_all	Lists all command-line subcommands and options as described in <i>Writing Administration Tools Scripts</i> (page 351).

If you omit the toolname and options, the Main Menu dialog box appears inside your console or terminal window with a dark blue background and a title on top. The screen captures used in this documentation set are cropped down to the dialog box itself, as shown below.



If you are unfamiliar with this type of graphical user interface, read *Using the Graphical User Interface* (page 330) before you do anything else.

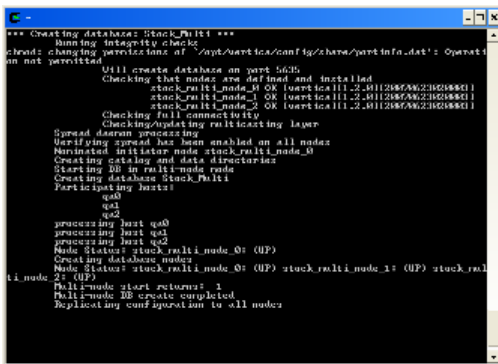
First Time Only

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

- 1 In the EULA (license agreement) window, type **accept** to proceed.
A window displays, requesting the location of the license key file you downloaded from the Vertica Web site. The default path is `/tmp/vlicense.key`.
- 2 Type the absolute path to your license key (for example, `/tmp/vlicense.txt`) and click **OK**.

Between Dialogs

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



```
*** Creating database: Stock_Pull1 ***
Running integrity check
chmod: changing permissions of /opt/vertica/config/zhare/partinfo.dat: Operation
not permitted
!! create database on port 5455
Checking that nodes are defined and installed
stock_multi_node_0 OK (vertica111.2.0112007002:10200001)
stock_multi_node_1 OK (vertica111.2.0112007002:10200001)
stock_multi_node_2 OK (vertica111.2.0112007002:10200001)
Checking full connectivity
Checking/updating routing layer
Spread layer processing
Verifying spread has been enabled on all nodes
Minimized initiator node stock_multi_node_0
Creating catalog and data directories
Starting DB in multi-node mode
Creating database Stock_Pull1
Participating hosts:
  qa0
  qa1
  qa2
processing host qa0
processing host qa1
processing host qa2
Made status: stock_multi_node_0: (UP)
Creating database nodes
Made status: stock_multi_node_0: (UP) stock_multi_node_1: (UP) stock_multi_node_2: (UP)
!! node 2: (UP)
Full-mode start returns: 1
Full-mode DB create completed
Replicating configuration to all nodes
```

Using the Graphical User Interface

The Vertica Administration Tools are implemented using Dialog, a graphical user interface that works in terminal (character-cell) windows. The interface responds to mouse clicks in some terminal windows, particularly local Linux windows, but you might find that it responds only to keystrokes. Thus, this section describes how to use the Administration Tools using only keystrokes.

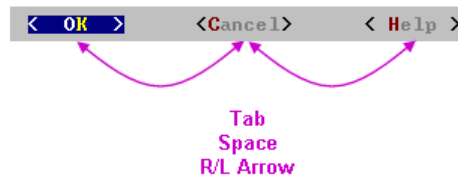
Note: This section does not describe every possible combination of keystrokes that can be used to accomplish a particular task. Feel free to experiment and to use whatever keystrokes you prefer.

Enter [Return]

In all dialogs, when you are ready to run a command, select a file, or cancel the dialog, press the **Enter** key. The command descriptions in this section do not explicitly instruct you to press Enter.

OK - Cancel - Help

The OK, Cancel, and Help buttons are present on virtually all dialogs. Use the tab, space bar, or right and left arrow keys to select an option and then press Enter. The same keystrokes apply to dialogs that present a choice of Yes or No.



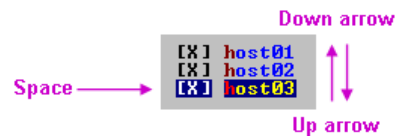
Menu Dialogs

Some dialogs require that you choose one command from a menu. Type the alphanumeric character shown or use the up and down arrow keys to select a command and then press Enter.



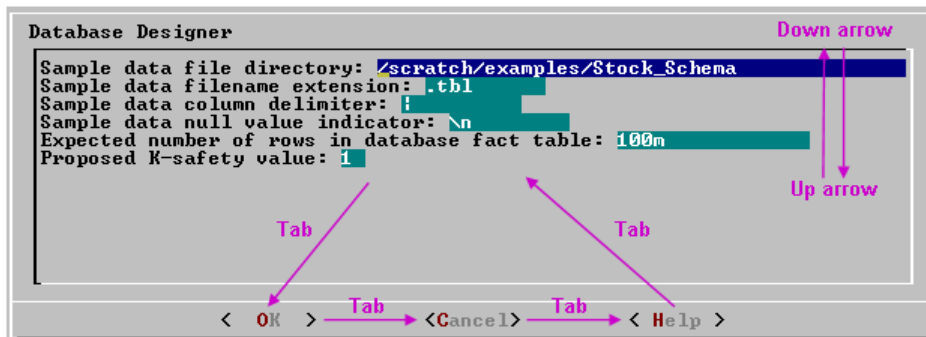
List Dialogs

In a list dialog, use the up and down arrow keys to highlight items, then use the space bar to select the items (which marks them with an X). Some list dialogs allow you to select multiple items. When you have finished selecting items, press Enter.



Form Dialogs

In a form dialog (also referred to as a dialog box), use the tab key to cycle between **OK**, **Cancel**, **Help**, and the form field area. Once the cursor is in the form field area, use the up and down arrow keys to select an individual field (highlighted) and enter information. When you have finished entering information in all fields, press Enter.



Help Buttons

Online help is provided in the form of text dialogs. If you have trouble viewing the help, see **Notes for Remote Terminal Users** (page 333) in this document.

K-Safety Support in Administration Tools

The Administration Tools allow certain operations on a K-Safe database, even if up to *K* nodes are down.

Note: The database must have been marked as K-Safe using the MARK_DESIGN_KSAFE function.

The following management functions within the Administration Tools are operational when up to *K* nodes are down:

- Start Database (including Manual Recovery)
- Shutdown Database
- Connect to database
- Replace Node (assuming node that is down is the one being replaced)
- View database cluster state
- View database parameters
- Upgrade license key

The following operations work with nodes down; however, you might have to repeat the operation on the failed nodes after they are back in operation:

- Edit Authentication
- Distribute Config Files
- Install External Procedure
- (Setting) Database Parameters

The following management functions within the Administration Tools require that all nodes be UP in order to be operational:

- Create Database
- Run Database Designer
- Drop database
- Set Restart Policy
- RollBack Database To Last Good Epoch

Notes for Remote Terminal Users

The appearance of the graphical interface depends on the color and font settings used by your terminal window. The screen captures in this document were made using the default color and font settings in a PuTTY terminal application running on Windows XP.

Note: If you are using a remote terminal application, such as PuTTY or a Cygwin bash shell, make sure your window is at least 81 characters wide and 23 characters high

If you are using PuTTY, you can make the Administration Tools look like the screen captures in this document:

- 1 In a PuTTY window, right click the title area and select Change Settings.
- 2 Create or load a saved session.
- 3 In the Category dialog, click Window > Appearance.
- 4 In the Font settings, click the Change... button.
- 5 Select Font: Courier New: Regular Size: 10
- 6 Click Apply.

Repeat these steps for each existing session that you use to run the Administration Tools.

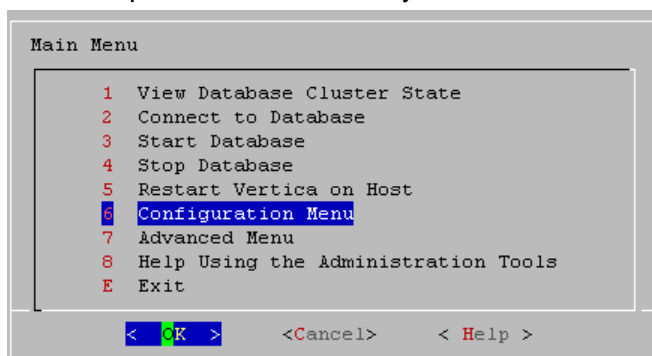
You can also change the translation to support UTF-8:

- 1 In a PuTTY window, right click the title area and select Change Settings.
- 2 Create or load a saved session.
- 3 In the Category dialog, click Window > Translation.
- 4 In the "Received data assumed to be in which character set" drop-down menu, select UTF-8.
- 5 Click Apply.

Using the Online Help

In a Menu Dialog

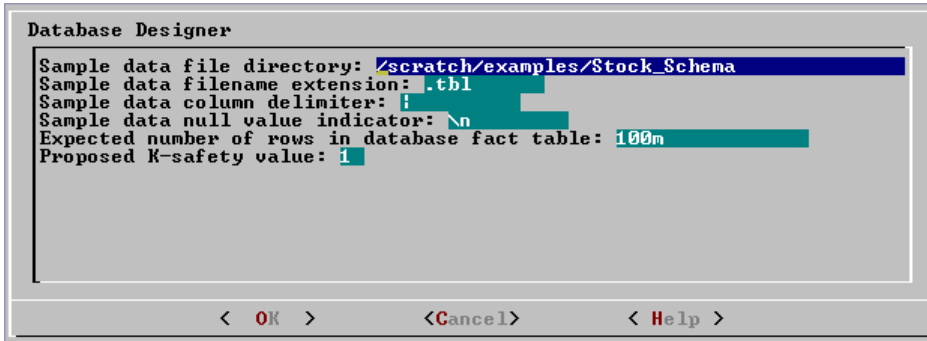
- 1 Use the up and down arrow keys to choose the command for which you want help.



- 2 Use the Tab key to move the cursor to the Help button.
- 3 Press Enter (Return).

In a Dialog Box

- 1 Use the up and down arrow keys to choose the field on which you want help.



- 2 Use the Tab key to move the cursor to the Help button.
- 3 Press Enter (Return).

Scrolling

Some help files are too long for a single screen. Use the up and down arrow keys to scroll through the text.

Password Authentication

When you create a new user with the CREATE USER command, you can configure the password or leave it empty. You cannot bypass the password if the user was created with a password configured. Change passwords using the ALTER USER command.

See *Implementing Security* (page 112) for more information about controlling database authorization through passwords.

Tip: Unless the database is used solely for evaluation purposes, Vertica recommends that all database users have encrypted passwords.

Distributing Changes Made to the Administration Tools Metadata

Metadata (specific to the Administration Tools) for a failed node falls out of synchronization with the other nodes in the cluster if you make the following changes:

- Modify the restart policy.
- Add one or more nodes.
- Drop one or more nodes.

When the node is restored, you can use the Administration Tools to update the node with the latest Administration Tools metadata:

- 1 Log on to a host that contains the metadata you want to transfer and start the Administration Tools. (See *Using the Administration Tools* (page 329).)
- 2 On the **Main Menu** in the Administration Tools, select **Configuration Menu** and click **OK**.

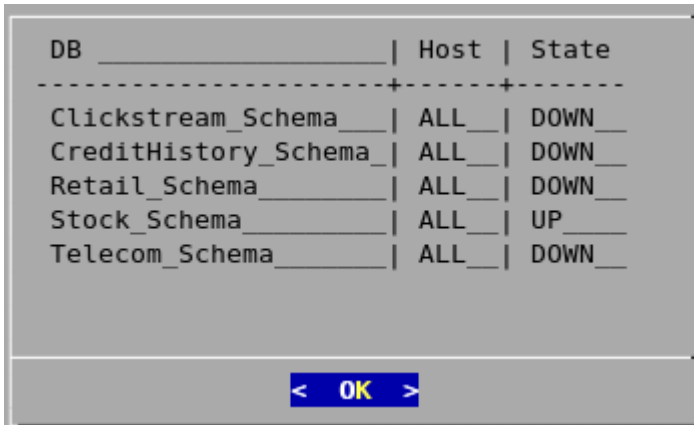
- 3 On the **Configuration Menu**, select **Distribute Config Files** and click **OK**.
- 4 Select **AdminTools Meta-Data**.
The Administration Tools metadata is distributed to every host in the cluster.
- 5 ***Restart the database*** (page 142).

Administration Tools Reference

Viewing Database Cluster State

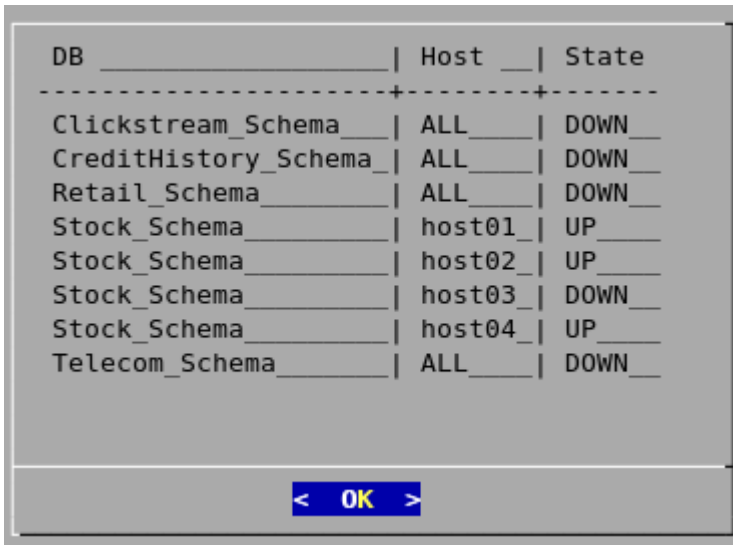
This tool shows the current state of the nodes in the database.

- 1 On the Main Menu, select **View Database Cluster State**, and click **OK**.
The normal state of a running database is ALL UP. The normal state of a stopped database is ALL DOWN.



DB	Host	State
Clickstream_Schema	ALL	DOWN
CreditHistory_Schema	ALL	DOWN
Retail_Schema	ALL	DOWN
Stock_Schema	ALL	UP
Telecom_Schema	ALL	DOWN

- 2 If some hosts are UP and some DOWN, restart the specific host that is down using **Restart Vertica on Host** from the Administration Tools, or you can start the database as described in **Starting and Stopping the Database** (page 142) (unless you have a known node failure and want to continue in that state.)



DB	Host	State
Clickstream_Schema	ALL	DOWN
CreditHistory_Schema	ALL	DOWN
Retail_Schema	ALL	DOWN
Stock_Schema	host01	UP
Stock_Schema	host02	UP
Stock_Schema	host03	DOWN
Stock_Schema	host04	UP
Telecom_Schema	ALL	DOWN

Nodes that are INITIALIZING or RECOVERING indicate that **failure recovery** (page 235) is in progress.

Nodes in other states (such as NEEDS_CATCHUP) are transitional and can be ignored unless they persist. In that case, contact **Technical Support** (on page 1).

See Also

- **Advanced Menu Options** (page 346)
- **Startup Problems** (page 241)
- **Shutdown Problems** (page 237)

Connecting to the Database

This tool connects to a running database with vsql. You can use the Administration Tools to connect to a database from any node within the database while logged into any user account with access privileges. You cannot use the Administration Tools to connect from a host that is not a database node. To connect from other hosts, run vsql as described in **Connecting From the Command Line** (page 360) in the Programmer's Guide.

- 1 On the Main Menu, click Connect to Database, and then click OK.
- 2 Supply the database password if asked:

Password:

When creating a new user via CREATE USER you can configure the password or leave it empty. There is no way to bypass the password if the user was created with a password configured. Passwords can be changed via the ALTER USER command.

- 3 The Administration Tools connect to the database and transfer control to vsql.

```
Welcome to the vsql, Vertica_Database v4.1.x interactive terminal.
Type:  \h for help with SQL commands
        \? for help with vsql commands
        \g or terminate with semicolon to execute query
        \q to quit
vmartdb=>
```

See **Using vsql** (page 358) for more information.

Note: After entering your password, you may be prompted to change your password if it has expired. See **Implementing Client Authentication** (page 114) for details of password security.

Starting a Database

This tool starts an existing database.

- 1 Use **View Database Cluster State** (page 336) to make sure that all nodes are down and that no other database is running. If not all nodes are down, see **Shutdown Problems** (page 237).
- 2 On the Main Menu, select Start Database, and then select OK.
- 3 Select the database to start, and click OK.

Note: Vertica Systems, Inc. strongly recommends that you start only one database at a time. If you start more than one database at any time, the results are unpredictable. Users could encounter resource conflicts or perform operations in the wrong database.

- 4 Enter the database password, and click OK.

When you create a new user with the CREATE USER command, you can configure the password or leave it empty. You cannot bypass the password if the user was created with a password configured. Change passwords using the ALTER USER command.

- 5 A message confirms that the database started successfully. Click **OK**.
- 6 Check the log files to make sure that no startup problems occurred, as described in *Monitoring the Database* (page 197).

Notes

If the database does not start successfully, see *Startup Problems* (page 241).

Stopping a Database

This tool stops a running database.

- 1 Use *View Database Cluster State* (page 336) to make sure that all nodes are up. If not all nodes are up, see *Restarting Vertica on Host* (page 341).
- 2 On the Main Menu, select Stop Database, and click **OK**.
- 3 Select the database you want to stop, and click **OK**.
- 4 Enter the password if asked, and click **OK**.
- 5 A message confirms that the database has been successfully stopped. Click **OK**.

Error

If users are connected during shutdown operations, the Administration Tools displays a message similar to the following:

```
Database Stock_Schema did not appear to stop in the allotted time.  
NOTICE: Cannot shut down while users are connected  
shutdown
```

```
-----  
Shutdown: aborting shutdown  
(1 row)
```

```
If you need to force a database shutdown, use the  
'Stop Vertica on Node' command in the Advanced menu,  
selecting the appropriate nodes to stop.
```

Description

The message indicates that there are active user connections (sessions). See *Managing Sessions* (page 313) in the Administrator's Guide for more information.

Resolution

The following examples were taken from a different database.

- 1 To see which users are connected, connect to the database and query the `SESSIONS` system table described in the SQL Reference Manual. For example:

```
=> \pset expanded  
Expanded display is on.  
=> SELECT * FROM SESSIONS;
```

```
-[ RECORD 1 ]
```



```

node_name          | site01
user_name          | dbadmin
client_hostname    | 127.0.0.1:57141
login_timestamp    | 2009-06-07 14:41:26
session_id         | rhel4-1-30361:0xd7e3e:994462853
transaction_start  | 2009-06-07 14:48:54
transaction_id     | 45035996273741092
transaction_description | user dbadmin (select * from session;)
statement_start    | 2009-06-07 14:53:31
statement_id       | 0
last_statement_duration | 1
current_statement  | select * from sessions;
ssl_state          | None
authentication_method | Trust
-[ RECORD 2 ]
node_name          | site01
user_name          | dbadmin
client_hostname    | 127.0.0.1:57142
login_timestamp    | 2009-06-07 14:52:55
session_id         | rhel4-1-30361:0xd83ac:1017578618
transaction_start  | 2009-06-07 14:53:26
transaction_id     | 45035996273741096
transaction_description | user dbadmin (COPY ClickStream_Fact FROM
'/data/clickstream/lg/ClickStream_Fact.tbl' DELIMITER '|' NULL '\\n' DIRECT;);
statement_start    | 2009-06-07 14:53:26
statement_id       | 17179869528
last_statement_duration | 0
current_statement  | COPY ClickStream_Fact FROM '/data/clickstream/lg/ClickStream_Fact.tbl'
DELIMITER '|' NULL '\\n' DIRECT;
ssl_state          | None
authentication_method | Trust

```

The current statement column of Record 1 shows that session is the one you are using to query the system table. Record 2 shows the session that must end before the database can be shut down.

- 2 If a statement is running in a session, that session must be closed. Use the function `CLOSE_SESSION` or `CLOSE_ALL_SESSIONS` described in the SQL Reference Manual.

Note: `CLOSE_ALL_SESSIONS` is the more common command because it forcefully disconnects all user sessions.

```
=> SELECT * FROM SESSIONS;
```

```

-[ RECORD 1 ]
node_name          | site01
user_name          | dbadmin
client_hostname    | 127.0.0.1:57141
client_pid         | 17838
login_timestamp    | 2009-06-07 14:41:26
session_id         | rhel4-1-30361:0xd7e3e:994462853
client_label       |
transaction_start  | 2009-06-07 14:48:54
transaction_id     | 45035996273741092
transaction_description | user dbadmin (select * from sessions;)
statement_start    | 2009-06-07 14:53:31
statement_id       | 0
last_statement_duration_us | 1
current_statement  | select * from sessions;
ssl_state          | None
authentication_method | Trust
-[ RECORD 2 ]
node_name          | site01
user_name          | dbadmin
client_hostname    | 127.0.0.1:57142
client_pid         | 17839

```

```

login_timestamp      | 2009-06-07 14:52:55
session_id           | rhel4-1-30361:0xd83ac:1017578618
client_label        |
transaction_start   | 2009-06-07 14:53:26
transaction_id       | 45035996273741096
transaction_description | user dbadmin (COPY ClickStream_Fact FROM
                    | '/data/clickstream/lg/ClickStream_Fact.tbl'
                    | DELIMITER '|' NULL '\n' DIRECT;)
statement_start      | 2009-06-07 14:53:26
statement_id         | 17179869528
last_statement_duration_us | 0
current_statement    | COPY ClickStream_Fact FROM
                    | '/data/clickstream/lg/ClickStream_Fact.tbl'
                    | DELIMITER '|' NULL '\n' DIRECT;
ssl_state            | None
authentication_method | Trust

```

```

=> SELECT CLOSE_SESSION('rhel4-1-30361:0xd83ac:1017578618');
-[ RECORD 1 ]
close_session | Session close command sent. Check sessions for progress.
=> SELECT * FROM SESSIONS;

```

```

-[ RECORD 1 ]
node_name      | site01
user_name      | dbadmin
client_hostname | 127.0.0.1:57141
client_pid     | 17838
login_timestamp | 2009-06-07 14:41:26
session_id     | rhel4-1-30361:0xd7e3e:994462853
client_label   |
transaction_start | 2009-06-07 14:48:54
transaction_id   | 45035996273741092
transaction_description | user dbadmin (select * from sessions;)
statement_start | 2009-06-07 14:54:11
statement_id    | 0
last_statement_duration_us | 98
current_statement | select * from sessions;
ssl_state       | None
authentication_method | Trust

```

3 Query the SESSIONS table again. For example, two columns have changed:

- stmtid is now 0, indicating that no statement is in progress.
- stmt_duration now indicates how long the statement ran in milliseconds before being interrupted.

The SELECT statements that call these functions return when the interrupt or close message has been delivered to all nodes, not after the interrupt or close has completed.

4 Query the SESSIONS table again. When the session no longer appears in the SESSION table, disconnect and run the **Stop Database** (page 338) command.

Controlling Sessions

The database administrator must be able to disallow new incoming connections in order to shut down the database. On a busy system, database shutdown is prevented if new sessions connect after the CLOSE_SESSION or CLOSE_ALL_SESSIONS() command is invoked — and before the database actually shuts down.

One option is for the administrator to issue the SHUTDOWN('true') command, which forces the database to shut down and disallow new connections. See SHUTDOWN in the SQL Reference Manual.

Another option is to modify the `MaxClientSessions` parameter from its original value to 0, in order to prevent new non-dbadmin users from connecting to the database.

- 1 Determine the original value for the `MaxClientSessions` parameter by querying the `V_MONITOR.CONFIGURATIONS_PARAMETERS` system table:

```
=> SELECT CURRENT_VALUE FROM CONFIGURATION_PARAMETERS WHERE
      parameter_name='MaxClientSessions';
      CURRENT_VALUE
-----
          50
(1 row)
```

- 2 Set the `MaxClientSessions` parameter to 0 to prevent new non-dbadmin connections:

```
=> SELECT SET_CONFIG_PARAMETER('MaxClientSessions', 0);
```

Note: The previous command allows up to five administrators to log in.

- 3 Issue the `CLOSE_ALL_SESSIONS()` command to remove existing sessions:

```
=> SELECT CLOSE_ALL_SESSIONS();
```

- 4 Query the `SESSIONS` table:

```
=> SELECT * FROM SESSIONS;
```

When the session no longer appears in the `SESSIONS` table, disconnect and run the **Stop Database** (page 338) command.

- 5 Restart the database.

- 6 Restore the `MaxClientSessions` parameter to its original value:

```
=> SELECT SET_CONFIG_PARAMETER('MaxClientSessions', 50);
```

See Also

`CLOSE_SESSION`, `CLOSE_ALL_SESSIONS`, `CONFIGURATION_PARAMETERS`, and `SESSIONS` in the SQL Reference Manual

Managing Sessions (page 313) and **Configuration Parameters** (page 25) in the Administrator's Guide

Notes

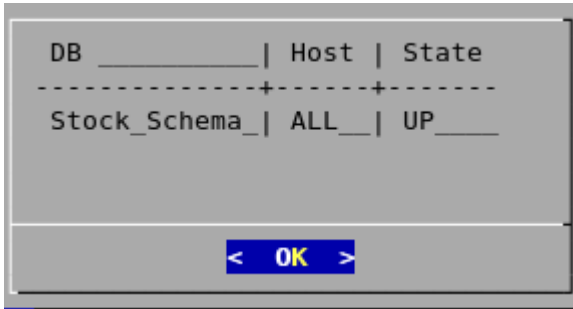
If the database does not stop successfully, see **Shutdown Problems** (page 237).

You cannot stop databases if your password has expired. The Administration Tools displays an error message if you attempt to do so. You need to change your expired password using `vsq` before you can shut down a database.

Restarting Vertica on Host

This tool restarts the Vertica process one or more nodes in a running database. Use this tool when a cluster host reboots while the database is running. The Spread daemon starts automatically but the Vertica process does not, thus the node does not automatically rejoin the cluster.

- 1 On the Main Menu, select **View Database Cluster State**, and click **OK**.
- 2 If one or more nodes are down, select **Restart Vertica on Host**, and click **OK**.
- 3 Select the database that contains the host that you want to restart, and click **OK**.
- 4 Select the Host that you want to restart, and click **OK**.
- 5 Select **View Database Cluster State** again to make sure that all nodes are up.

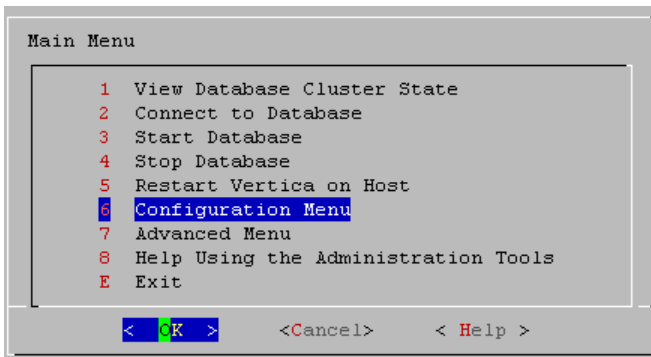


Configuration Menu Item

The main configuration menu allows you to:

- Create, drop, and view databases
- Use the Database Designer to create or modify a physical schema design

- 1 On the Main Menu, click **Configuration**, and then click **OK**.



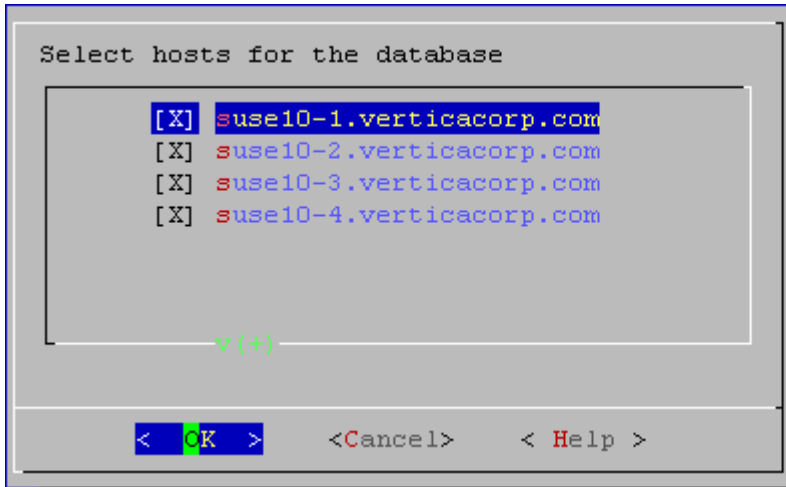
Creating a Database

- 1 On the Configuration menu, click **Create Database** and then click **OK**.
- 2 Enter the name of the database and an optional comment. Click **OK**.
- 3 Enter a password.

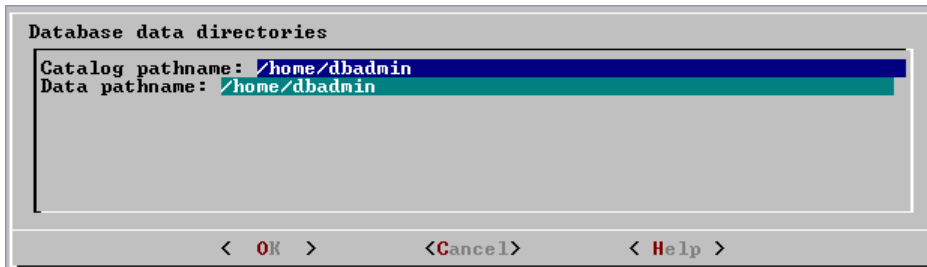
If you do not enter a password, you are prompted to indicate whether you want to enter a password. Click **Yes** to enter a password or **No** to create a database without a superuser password.

Warning: If you do not enter a password at this point, superuser password is set to empty. Unless the database is for evaluation or academic purposes, Vertica Systems, Inc. strongly recommends that you enter a superuser password.

- 4 If you entered a password, enter the password again.
- 5 Select the hosts to include in the database. The hosts in this list are the ones that were specified at installation time (`install_vertica -s`).

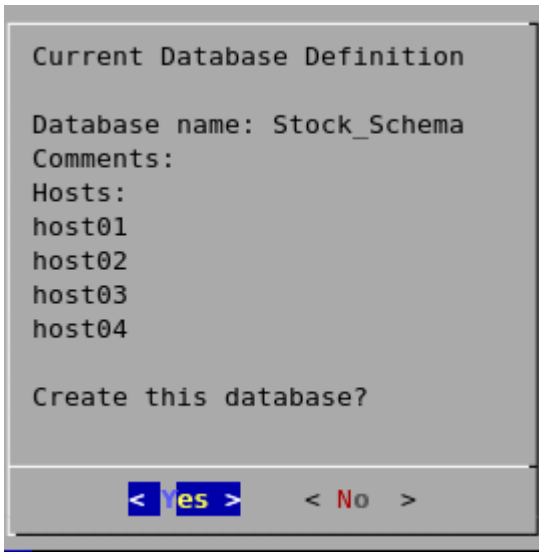


- 6 Specify the directories in which to store the data and catalog files.



Note: Catalog and data paths must contain only alphanumeric characters and cannot have leading space characters. Failure to comply with these restrictions could result in database creation failure.

- 7 Check the current database definition for correctness, and click **Yes** to proceed.



- 8 A message indicates that you have successfully created a database. Click **OK**.

Notes

If you get an error message, see *Startup Problems* (page 241)

Dropping a Database

This tool drops an existing database. Only the Database Administrator is allowed to drop a database.

- 1 Stop the database as described in *Stopping a Database* (page 338).
- 2 On the Configure Database dialog, click **Drop Database** and then click **OK**.
- 3 Select the database to drop and click **OK**.
- 4 Click **Yes** to confirm that you want to drop the database.
- 5 Type **yes** and click **OK** to reconfirm that you really want to drop the database.
- 6 A message indicates that you have successfully dropped the database. Click **OK**.

Notes

In addition to dropping the database, Vertica automatically drops the node definitions that refer to the database unless:

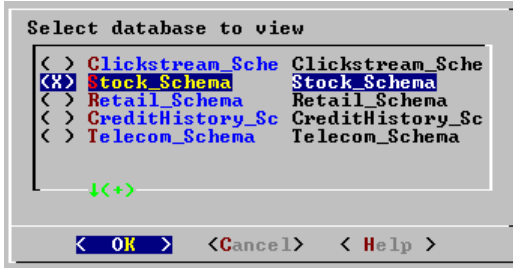
- Another database uses a node definition. If another database refers to any of these node definitions, none of the node definitions are dropped.
- A node definition is the only node defined for the host. (Vertica uses node definitions to locate hosts that are available for database creation, so removing the only node defined for a host would make the host unavailable for new databases.)

Viewing a Database

This tool displays the characteristics of an existing database.

- 1 On the Configure Database dialog, select **View Database** and click **OK**.

- 2 Select the database to view.



- 3 Vertica displays the following information about the database:

- The name of the database.
- The name and location of the log file for the database.
- The hosts within the database cluster.
- The value of the Restart Policy setting. This setting determines whether or not nodes within a K-Safe database are restarted when they are rebooted. See Set Restart Policy.
- The database port.

Setting the Restart Policy

The Restart Policy enables you to determine whether or not nodes in a K-Safe database are automatically restarted when they are rebooted. Since this feature does not automatically restart nodes if the entire database is DOWN, it is not useful for databases that are not K-Safe.

To set the Restart Policy for a database:

- 1 Open the Administration Tools.
- 2 On the Main Menu, select **Configuration Menu**, and click **OK**.
- 3 In the Configuration Menu, select **Set Restart Policy**, and click **OK**.
- 4 Select the database for which you want to set the Restart Policy, and click **OK**.
- 5 Select one of the following policies for the database:
 - **Never** — Nodes are never restarted automatically.
 - **K-Safe** — Nodes are automatically restarted if the database cluster is still UP. This is the default setting.
 - **Always** - Node on a single node database is restarted automatically
- 6 Click **OK**.

Best Practice for Restoring Failed Hardware

Following this procedure will prevent Vertica from misdiagnosing missing disk or bad mounts as data corruptions, which would result in a time-consuming, full-node recovery.

If a server fails due to hardware issues, for example a bad disk or a failed controller, upon repairing the hardware:

- 1 Reboot the machine into runlevel 1, which is a root and console-only mode.
Runlevel 1 prevents network connectivity and keeps Vertica from attempting to reconnect to the cluster.

- 2 In runlevel 1, validate that the hardware has been repaired, the controllers are online, and any RAID recover is able to proceed.

Note: You do not need to initialize RAID recover in runlevel 1; simply validate that it can recover.

- 3 Once the hardware is confirmed consistent, only then reboot to runlevel 3 or higher.

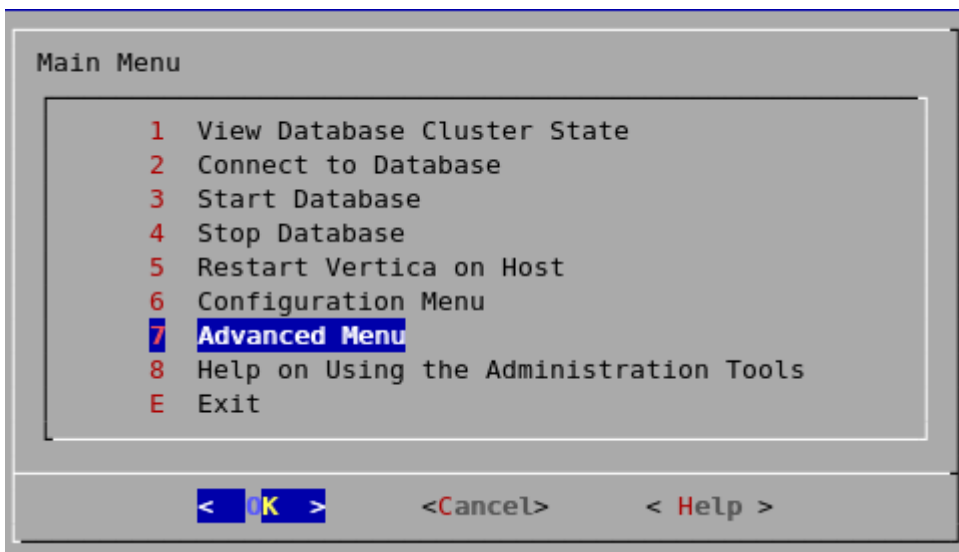
At this point, the network activates, and Vertica rejoins the cluster and automatically recovers any missing data.

Advanced Menu Options

This Advanced Menu provides interactive recovery and repair commands.

Note: Use these commands only when instructed to do so by **Technical Support** (on page 1).

- 1 On the Main Menu, click **Advanced Menu** and then **OK**.



Rolling Back Database to the Last Good Epoch

IMPORTANT! Use this command only when instructed to do so by **Technical Support** (on page 1).

Vertica provides the ability to roll the entire database back to a specific epoch primarily to assist in the correction of human errors during data loads or other accidental corruptions. For example, suppose that you have been performing a bulk load and the cluster went down during a particular COPY command. You might want to discard all epochs back to the point at which the previous COPY command committed and run the one that did not finish again. You can determine that point by examining the log files (see **Monitoring the Log Files** (page 197)).

- 1 On the Advanced menu, select **Roll Back Database to Last Good Epoch**.
- 2 Select the database to roll back. The database must be stopped.
- 3 Accept the suggested restart epoch or specify a different one.
- 4 Confirm that you want to discard the changes after the specified epoch.
The database restarts successfully.

Important note:

In Vertica 4.1, the default for the `HistoryRetentionTime` configuration parameter changed to 0, which means that Vertica only keeps historical data when nodes are down. This new setting effectively prevents the use of the Administration Tools 'Roll Back Database to Last Good Epoch' option because the AHM remains close to the current epoch and a rollback is not permitted to an epoch prior to the AHM. If you rely on the Roll Back option to remove recently loaded data, consider setting a day-wide window for removing loaded data; for example:

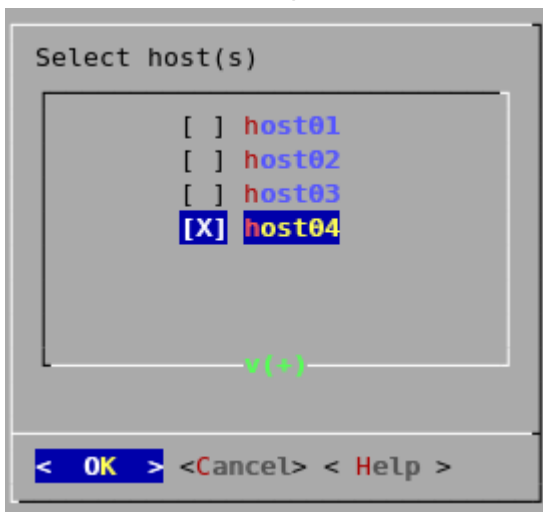
```
=> SELECT SET_CONFIG_PARAMETER ('HistoryRetentionTime', '86400');
```

Stopping Vertica on Host

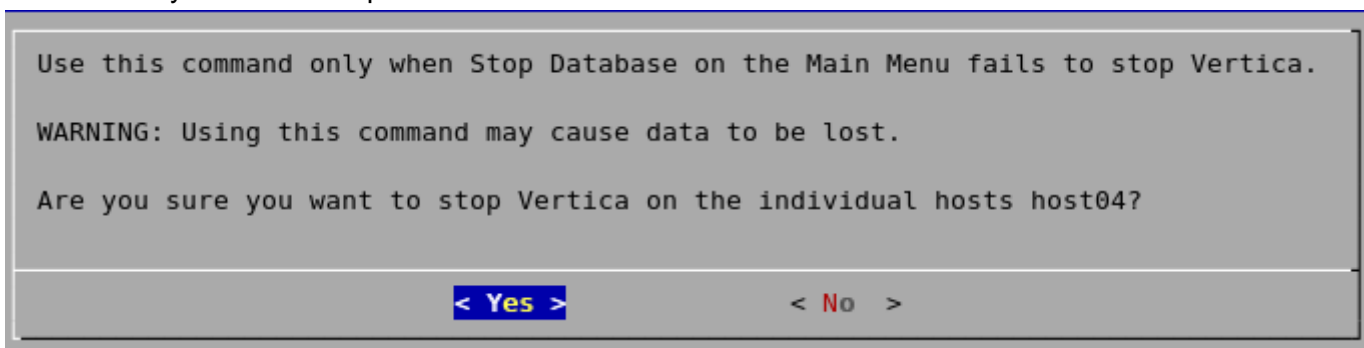
This command attempts to gracefully shut down the Vertica process on a single node.

Caution: Do not use this command if you are intending to shut down the entire cluster. Use **Stop Database** (page 338) instead, which performs a clean shutdown to minimize data loss.

- 1 On the Advanced menu, select **Stop Vertica on Host** and click **OK**.
- 2 Select the hosts to stop.



- 3 Confirm that you want to stop the hosts.



If the command succeeds **View Database Cluster State** (page 336) shows that the selected hosts are DOWN.

DB	Host	State
Stock_Schema_	host01_	UP
Stock_Schema_	host02_	UP
Stock_Schema_	host03_	UP
Stock_Schema_	host04_	DOWN

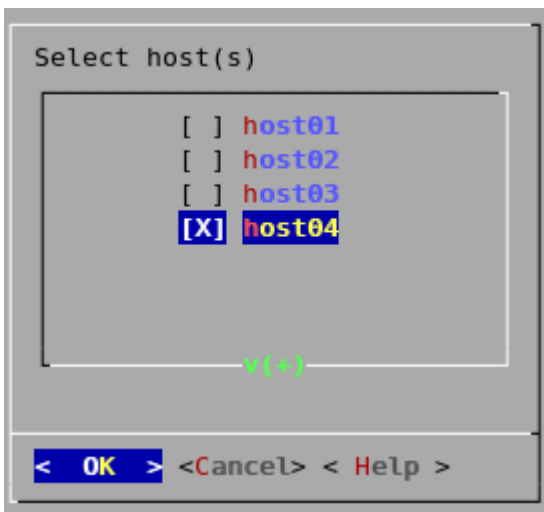
If the command fails to stop any selected nodes, proceed to **Killing Vertica Process on** (page 348)Host.

Killing a Vertica Process on Host

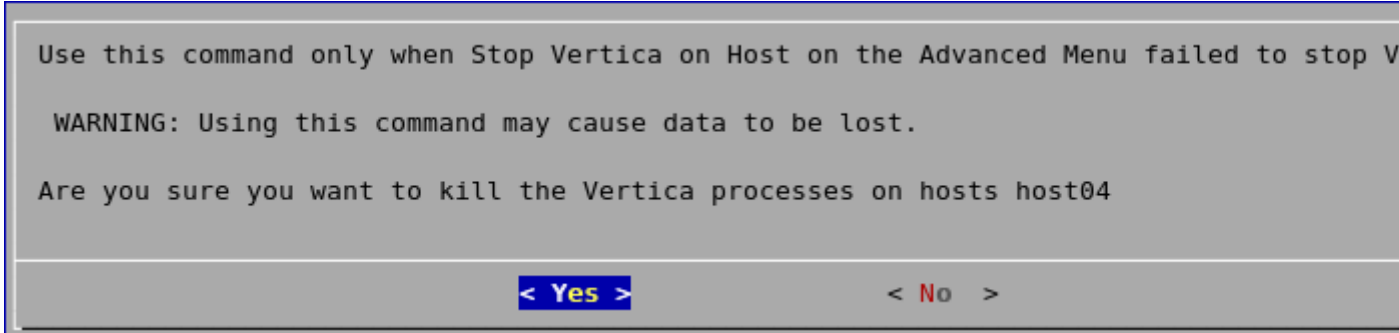
This command sends a kill signal to the Vertica process on a node.

Caution: Do not use this command unless you have already tried **Stop Database** (page 338) and **Stop Vertica on Node** (page 347) and both were unsuccessful.

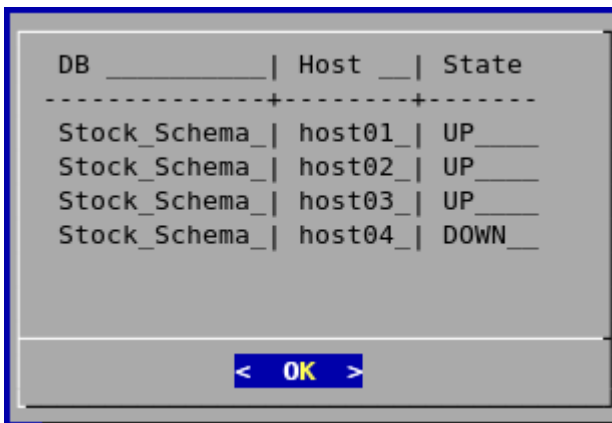
- 1 On the Advanced menu, select **Kill Vertica Process on Host** and click **OK**.
- 2 Select the hosts on which to kills the Vertica process.



- 3 Confirm that you want to stop the processes.



- 4 If the command succeeds **View Database Cluster State** (page 336) shows that the selected hosts are DOWN.



- 5 If the command fails to stop any selected processes, see **Shutdown Problems** (page 237).

Upgrading the License Key

This command copies a license key file into the database. See **Managing Your License Key** (page 140) for more information.

- 1 On the Advanced menu select **Upgrade License Key** and click **OK**.
- 2 Select the database for which to upgrade the license key.
- 3 Enter the absolute pathname of your downloaded license key file (for example, /tmp/vlicense.txt) and click **OK**.
- 4 Click OK when you see a message that indicates that the upgrade succeeded.

Managing Clusters

Cluster Management lets you add, replace, or remove hosts from a database cluster. These processes are usually part of a larger process of **adding** (page 264), **removing** (page 271), or **replacing** (page 275) a database node.

Before Using Cluster Management: View the database state to verify that it is running. See **View Database Cluster State** (page 336). If the database isn't running, restart it. See **Starting a Database** (page 337).

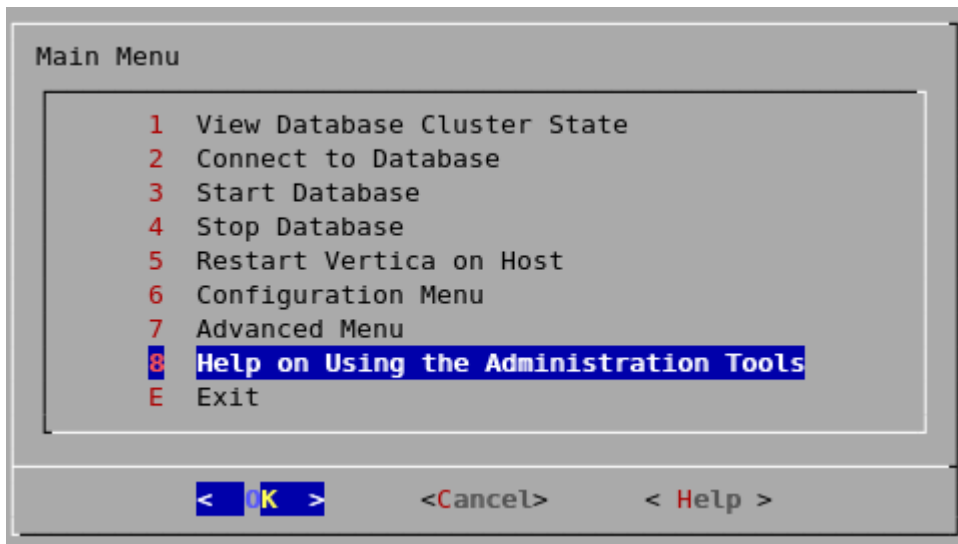
Using Cluster Management

To use Cluster Management:

- 1 From the **Main Menu**, select **Advanced**, and then click **OK**.
- 2 In the **Advanced Menu**, select **Cluster Management**, and then click **OK**.
- 3 Select one of the following, and then click **OK**.
 - **Add Hosts to Database:** See *Adding Hosts to a Database* (page 267).
 - **Replace Host:** See *Replacing Hosts* (page 277).
 - **Remove Host from Database:** See *Removing Hosts from a Database* (page 272).

Using the Administration Tools

The **Help on Using the Administration Tools** command displays a help screen about using the Administration Tools.



Most of the online help in the Administration Tools is context-sensitive. For example, if you use the up/down arrows to select a command, press tab to move to the Help button, and press return, you get help on the selected command.

Writing Administration Tools Scripts

You can invoke any of the Administration Tools from the command line or a shell script.

Syntax

```
> /opt/vertica/bin/admintools [ -t | --tool ] toolname [ options ]
```

Note: For convenience, you can add `/opt/vertica/bin` to your search path.

Parameters

[-t] [--tool]	Instructs the Administration Tools to run the specified tool. Note: If you use the <code>--no-log</code> option to run the Administration Tools silently, <code>--no-log</code> must appear <i>before</i> the <code>-t</code> option.	
toolname	Is one of the tools described in the help output below.	
[options]	-h --help	Shows a brief help message and exits.
	-a --help_all	Lists all command-line subcommands and options as shown below.

Tools

To return a description of all the tools, issue the following command at a vsql prompt:

```
$ adminTools -a
```

Usage:

```
adminTools [-t | --tool] toolName [options]
```

Valid tools are:

```
host_to_node
rebalance_data
stop_host
kill_node
run_designer
logrotate
list_db
node_map
db_replace_node
view_cluster
upgrade_license_key
edit_auth
uninstall_node
create_db
stop_db
db_status
install_procedure
show_active_db
return_epoch
list_node
install_node
list_allnodes
drop_db
db_add_node
stop_node
kill_host
set_restart_policy
config_nodes
```

```
drop_node
restart_db
database_parameters
restart_node
check_spread
list_host
start_db
command_host
connect_db
db_remove_node
```

usage: host_to_node [options]

options:

```
-h, --help          show this help message and exit
-s HOST, --host=HOST comma separated list of hostnames which is to be
                    converted into its corresponding nodenames
-d DB, --database=DB show only node/host mapping for this database.
```

usage: rebalance_data [options]

options:

```
-h, --help          show this help message and exit
-d DBNAME, --dbname=DBNAME
                    database name
-k KSAFETY, --ksafety=KSAFETY
                    specify the new k value to use
-p PASSWORD, --password=PASSWORD
--script            Don't re-balance the data, just provide a script for later use.
```

usage: stop_host [options]

options:

```
-h, --help          show this help message and exit
-s HOSTS, --hosts=HOSTS
                    comma-separated list of hosts on which the vertica
                    process is to be killed
--compat21          Use Vertica 2.1 method using node names instead of
                    hostnames
```

usage: kill_node [options]

options:

```
-h, --help          show this help message and exit
-s HOSTS, --hosts=HOSTS
                    comma-separated list of hosts on which the vertica
                    process is to be killed
--compat21          Use Vertica 2.1 method using node names instead of
                    hostnames
```

usage: logrotateconfig [options]

options:

```
-h, --help          show this help message and exit
-d DBNAME, --dbname=DBNAME
                    database name
-r ROTATION, --rotation=ROTATION
                    set how often the log is rotated. [
                    daily|weekly|monthly ]
-s MAXLOGSZ, --maxsize=MAXLOGSZ
                    set maximum log size before rotation is forced.
-k KEEP, --keep=KEEP set # of old logs to keep
```

usage: list_db [options]

options:

```
-h, --help          show this help message and exit
-d DB, --database=DB Name of database to be listed
```

usage: node_map [options]

options:

```
-h, --help          show this help message and exit
-d DB, --database=DB List only data for this database.
```

```

usage: db_replace_node [options]
options:
  -h, --help            show this help message and exit
  -d DB, --database=DB  Name of database to be restarted
  -o ORIGINAL, --original=ORIGINAL
                        Name of host you wish to replace
  -n NEWHOST, --new=NEWHOST
                        Name of the replacement host
  -p DBPASSWORD, --password=DBPASSWORD
                        Database password in single quotes
  -i, --noprompts      do not stop and wait for user input(default false)
-----
usage: view_cluster [options]
options:
  -h, --help            show this help message and exit
  -x, --xexpand        show the full cluster state, node by node
  -d DB, --database=DB  filter the output for a single database
-----
usage: upgrade_license_key [options]
options:
  -h, --help            show this help message and exit
  -d DB, --database=DB  Name of database[optional]
  -l LICENSE, --license=LICENSE
                        Database license
  -i INSTALL, --install=INSTALL
                        argument '-i install' to Install license else without
                        '-i install' Upgrade license
  -p PASSWORD, --password=PASSWORD
                        Database password[optional]
-----
usage: edit_auth [options]
options:
  -h, --help            show this help message and exit
  -d DATABASE, --database=DATABASE
                        database to edit authentication parameters for
-----
usage: uninstall_node [options]
options:
  -h, --help            show this help message and exit
  -s HOSTNAME, --host=HOSTNAME
                        Comma separated list of hostnames upon which to
                        uninstall
  -p PASSWORD, --password=PASSWORD
                        Name of file containing root password for machines in
                        the list
  -d, --delete          Delete configuration data during uninstall
  --compat21            Use Vertica 2.1 method using node names instead of
                        hostnames
-----
usage: create_db [options]
options:
  -h, --help            show this help message and exit
  -s NODES, --hosts=NODES
                        comma-separated list of hosts to participate in
                        database
  -d DB, --database=DB  Name of database to be created
  -c CATALOG, --catalog_path=CATALOG
                        Path of catalog directory[optional] if not using
                        compat21
  -D DATA, --data_path=DATA
                        Path of data directory[optional] if not using compat21
  -p DBPASSWORD, --password=DBPASSWORD
                        Database password in single quotes [optional]
  -l LICENSEFILE, --license=LICENSEFILE
                        Database license [optional]
  -P POLICY, --policy=POLICY
                        Database restart policy [optional]
  --compat21            Use Vertica 2.1 method using node names instead of

```

Administrator's Guide

```
hostnames
-----
usage: stop_db [options]
options:
  -h, --help          show this help message and exit
  -d DB, --database=DB  Name of database to be stopped
  -p DBPASSWORD, --password=DBPASSWORD
                      Database password in single quotes
  -i, --noprompts     do not stop and wait for user input(default false)
-----
usage: db_status [options]
options:
  -h, --help          show this help message and exit
  -s STATUS, --status=STATUS
                      Database status UP,DOWN or ALL(list running dbs -
                      UP,list down dbs - DOWN list all dbs - ALL)
-----
usage: install_procedure [options]
options:
  -h, --help          show this help message and exit
  -d DBNAME, --database=DBNAME
                      Name of database for installed procedure
  -f PROCPATH, --file=PROCPATH
                      Path of procedure file to install
  -p OWNERPASSWORD, --password=OWNERPASSWORD
                      Password of procedure file onwer
-----
usage: show_active_db [options]
options:
  -h, --help          show this help message and exit
-----
usage: return_epoch [options]
options:
  -h, --help          show this help message and exit
  -d DB, --database=DB  Name of database
-----
usage: list_node [options]
options:
  -h, --help          show this help message and exit
  -n NODENAME, --node=NODENAME
                      Name of the node to be listed
-----
usage: install_node [options]
options:
  -h, --help          show this help message and exit
  -s HOSTNAME, --host=HOSTNAME
                      Comma separated list of hostnames upon which to
                      install
  -r RPMNAME, --rpm=RPMNAME
                      Fully qualified file name of the RPM to be used on
                      install
  -p PASSWORD, --password=PASSWORD
                      Name of file containing root password for machines in
                      the list
  --compat21          Use Vertica 2.1 method using node names instead of
                      hostnames
-----
usage: list_allnodes [options]
options:
  -h, --help          show this help message and exit
-----
usage: drop_db [options]
options:
  -h, --help          show this help message and exit
  -d DB, --database=DB  Database to be dropped
-----
usage: db_add_node [options]
options:
```



```

-h, --help          show this help message and exit
-d DB, --database=DB  Name of database to be restarted
-s HOSTS, --hosts=HOSTS
                    Comma separated list of hosts to add to database
-p DBPASSWORD, --password=DBPASSWORD
                    Database password in single quotes
-a AHOSTS, --add=AHOSTS
                    Comma separated list of hosts to add to database
-i, --noprompts     do not stop and wait for user input(default false)
--compat21          Use Vertica 2.1 method using node names instead of
                    hostnames
-----
usage: stop_node [options]
options:
-h, --help          show this help message and exit
-s HOSTS, --hosts=HOSTS
                    comma-separated list of hosts on which the vertica
                    process is to be killed
--compat21          Use Vertica 2.1 method using node names instead of
                    hostnames
-----
usage: kill_host [options]
options:
-h, --help          show this help message and exit
-s HOSTS, --hosts=HOSTS
                    comma-separated list of hosts on which the vertica
                    process is to be killed
--compat21          Use Vertica 2.1 method using node names instead of
                    hostnames
-----
usage: set_restart_policy [options]
options:
-h, --help          show this help message and exit
-d DB, --database=DB  Name of database for which to set policy
-p POLICY, --policy=POLICY
                    Restart policy: ('never', 'ksafe', 'always')
-----
usage: config_nodes [options]
options:
-h, --help          show this help message and exit
-f NODEHOSTFILE, --file=NODEHOSTFILE
                    File containing list of nodes, hostnames, catalog
                    path, and datapath (node<whitespace>host<whitespace>ca
                    talogPath<whitespace>dataPath one per line)
-i, --install       Attempt to install from RPM on all nodes in the list
-r RPMNAME, --rpm=RPMNAME
                    Fully qualified file name of the RPM to be used on
                    install
-p PASSWORD, --password=PASSWORD
                    Name of file containing Root password for machines in
                    the list
-c, --check         Check all nodes to make sure they can interconnect
-s SKIPANALYZENODE, --skipanalyzenode=SKIPANALYZENODE
                    skipanalyzenode
-----
usage: drop_node [options]
options:
-h, --help          show this help message and exit
-n NODENAME, --node=NODENAME
                    Name of the node to be dropped
--force            Force a node to be dropped if its the last reference to the host.
-----
usage: restart_db [options]
options:
-h, --help          show this help message and exit
-d DB, --database=DB  Name of database to be restarted
-e EPOCH, --epoch=EPOCH
                    Epoch at which the database is to be restarted. If

```

Administrator's Guide

```

        'last' is given as argument the db is restarted from
        the last good epoch.
-p DBPASSWORD, --password=DBPASSWORD
        Database password in single quotes
-i, --noprompts
        do not stop and wait for user input(default false)
-----
usage: database_parameters [options]
options:
-h, --help
        show this help message and exit
-d DB, --database=DB
        Name of database
-P PARAMETER, --parameter=PARAMETER
        Database parameter
-c COMPONENT, --component=COMPONENT
        Component[optional]
-s SUBCOMPONENT, --subcomponent=SUBCOMPONENT
        Sub Component[optional]
-p PASSWORD, --password=PASSWORD
        Database password[optional]
-----
usage: restart_node [options]
options:
-h, --help
        show this help message and exit
-s NODES, --hosts=NODES
        comma-separated list of hosts to be restarted
-d DB, --database=DB
        Name of database whose node is to be restarted
-p DBPASSWORD, --password=DBPASSWORD
        Database password in single quotes
-i, --noprompts
        do not stop and wait for user input(default false)
--compat21
        Use Vertica 2.1 method using node names instead of
        hostnames
-----
usage: check_spread [options]
options:
-h, --help
        show this help message and exit
-----
usage: list_host [options]
options:
-h, --help
        show this help message and exit
-----
usage: start_db [options]
options:
-h, --help
        show this help message and exit
-d DB, --database=DB
        Name of database to be started
-p DBPASSWORD, --password=DBPASSWORD
        Database password in single quotes
-i, --noprompts
        do not stop and wait for user input(default false)
-----
usage: command_host [options]
options:
-h, --help
        show this help message and exit
-c CMD, --command=CMD
        Command to run
-----
usage: connect_db [options]
options:
-h, --help
        show this help message and exit
-d DB, --database=DB
        Name of database to connect
-p DBPASSWORD, --password=DBPASSWORD
        Database password in single quotes
-----
usage: db_remove_node [options]
options:
-h, --help
        show this help message and exit
-d DB, --database=DB
        Name of database to be modified
-s HOSTS, --hosts=HOSTS
        Name of the host to remove from the db
-p DBPASSWORD, --password=DBPASSWORD
        Database password in single quotes
```

```
-i, --noprompts      do not stop and wait for user input (default false)
--compat21          Use Vertica 2.1 method using node names instead of
                    hostnames
```

To run the help command:

```
$ admintools -h
```

options:

```
-h, --help          Display this help. Can be combined with -t <tool> for help on a specific tool
-a, --help_all      List all command line sub-commands and switches
```

Available tools:

```
check_spread          command_host
config_nodes          connect_db
create_db             database_parameters
db_add_node           db_remove_node
db_replace_node       db_status
drop_db              drop_node
edit_auth            host_to_node
install_node
install_procedure     kill_host
kill_node            list_allnodes
list_db              list_host
list_node            logrotate
node_map             rebalance_data
restart_db           restart_node
return_epoch         set_restart_policy
show_active_db
start_db             stop_db
stop_host            stop_node
uninstall_node       upgrade_license_key
view_cluster
```

Using vsql

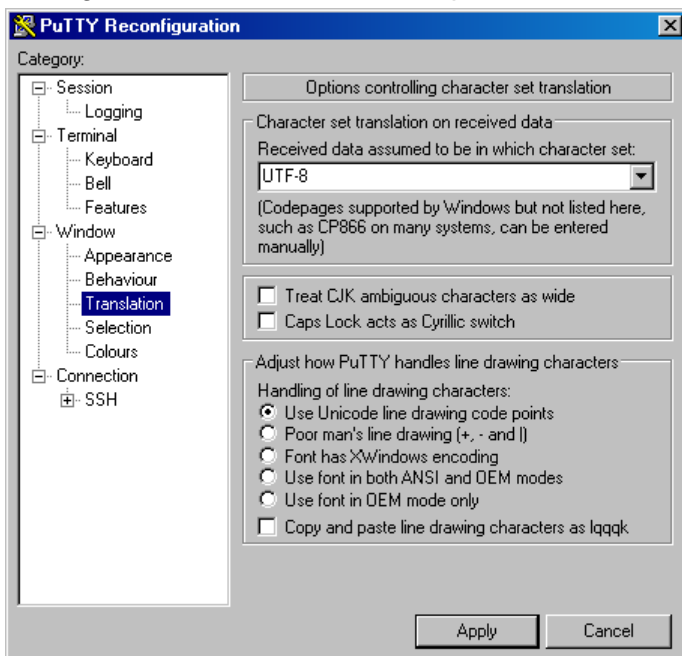
vsql is a character-based, interactive, front-end utility that lets you type SQL statements and see the results. It also provides a number of meta-commands and various shell-like features that facilitate writing scripts and automating a variety of tasks.

You can connect to vsql from the:

- **Administration Tools** (page 359)
- **Linux command line** (page 360)

General Notes

- SQL statements can be spread over several lines for clarity.
- vsql can handles input and output in UTF-8 encoding. Note that the terminal emulator running vsql must be set up to display the UTF-8 characters correctly. Follow the documentation of your terminal emulator. The following example shows the settings in PuTTY from the Change Settings > Window > Translation option:



See also **Best Practices for Working with Locales** (page 21).

- Cancel SQL statements by typing Ctrl+C.
- Traverse command history by typing Ctrl+R.
- When you disconnect a user session, any transactions in progress are automatically rolled back.
- To view wide result sets, use the Linux `less` utility to truncate long lines.
 1. Before connecting to the database, specify that you want to use `less` for query output:

```
$ export PAGER=less
```
 2. Connect to the database.

3. Query a wide table:

```
=> select * from wide_table;
```

4. At the `less` prompt, type:

```
-s
```

- If a shell running `vsq` fails (crashes or freezes), the `vsq` processes continue to run even if you stop the database. In that case, log in as `root` on the machine on which the shell was running and manually kill the `vsq` process. For example:

```
# ps -ef | grep vertica
```

```

:
fred  2401      1  0 06:02 pts/1    00:00:00 /opt/vertica/bin/vsqr -p
      5433 -h test01_site01 quick_start_single
:

```

```
# kill -9 2401
```

Connecting From the Administration Tools

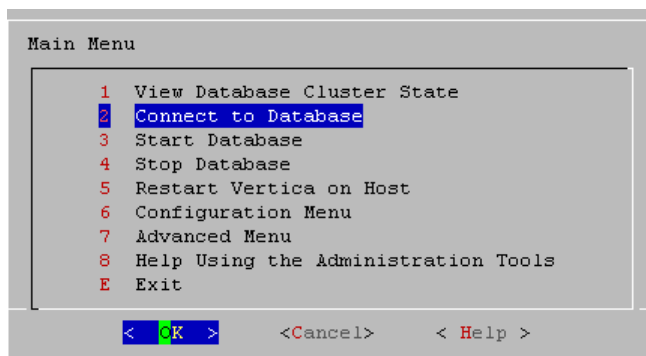
You can use the Administration Tools to connect to a database using `vsq` on any node in the cluster.

- 1 Log in as any user that does not have root privileges. (Vertica does not allow users with root privileges to connect to a database for security reasons).

- 2 Run the Administration Tools.

```
/opt/vertica/bin/admintools
```

- 3 On the Main Menu, select Connect to Database.



- 4 Supply the database password if asked:

```
Password:
```

- 5 The Administration Tools connect to the database and transfer control to `vsq`.

```
Welcome to the vsqr, Vertica_Database v4.1.x interactive terminal.
```

```
Type:  \h for help with SQL commands
        \? for help with vsqr commands
        \g or terminate with semicolon to execute query
        \q to quit
```

```
vmartdb=>
```

Note: See *Meta-Commands* (page 366) for the various commands you can run while connected to the database through the Administration Tools.

Connecting from the Command Line

You can use `vsq` from the command line to connect to a database from any Linux machine, including those not part of the cluster. Copy `/opt/vertica/bin/vsqr` to your machine.

Syntax

```
/opt/vertica/bin/vsqr [ option... ] [ dbname [ username ] ]
```

Parameters

<i>option</i>	One or more of the vsqr command line options (on page 360)
<i>dbname</i>	The name of the target database
<i>username</i>	The name of the user to connect as

Notes

- If the database is password protected, you must specify the `-w` (see "**w password**" on page 364) or `--password` command line option.
- The default `dbname` and `username` is your Linux user name.
- If the connection cannot be made for any reason (for example, insufficient privileges, server is not running on the targeted host, etc.), `vsqr` returns an error and terminates.
- `vsqr` returns the following informational messages:
 - 0 to the shell if it finished normally
 - 1 if a fatal error of its own (out of memory, file not found) occurs
 - 2 if the connection to the server went bad and the session was not interactive
 - 3 if an error occurred in a script and the variable `ON_ERROR_STOP` was set
- Unrecognized words in the command line might be interpreted as database or user names.

Example

The following example redirects `vsqr` output and error messages into an output file called `retail_queries.out` and captures any error messages:

```
$ vsqr --echo-all < retail_queries.sql > retail_queries.out 2>&1
```

Command Line Options

This section contains the command-line options.

? --help

`-? --help` displays help about `vsqr` command line arguments and exits.

a --echo-all

-a `--echo-all` prints all input lines to standard output as they are read. This is more useful for script processing than interactive mode. It is equivalent to setting the variable `ECHO` (page 384) to `all`.

A --no-align

-A `--no-align` switches to unaligned output mode. (The default output mode is aligned.)

c command --command command

-c `command` `--command command` runs one command and exits. This is useful in shell scripts. The command must be either a command string that is completely parsable by the server (it contains no vsql specific features), or a single meta-command. In other words, you cannot mix SQL and vsql meta-commands. To achieve that, you can pipe the string into vsql like this:

```
echo "\\timing\\\\"select * from t" | ../Linux64/bin/vsql
Timing is on.
 i | c | v
---+---+---
(0 rows)
```

Note: If you use double quotes with echo, you must double the backslashes.

d dbname --dbname dbname

-d `dbname` `--dbname dbname` specifies the name of the database to connect to. This is equivalent to specifying `dbname` as the first non-option argument on the command line.

e --echo-queries

-e `--echo-queries` copies all SQL commands sent to the server to standard output as well. This is equivalent to setting the variable `ECHO` (page 384) to `queries`.

E

-E displays queries generated by internal commands.

f filename --file filename

-f `filename` `--file filename` uses the file `filename` as the source of commands instead of reading commands interactively. After the file is processed, vsql terminates. This is in many ways equivalent to the internal command `\i` (see "`i FILE`" on page 377).

If `filename` is `-` (hyphen), the standard input is read.

Using this option is subtly different from writing `vsq1 < filename`. In general, both do what you expect, but using `-f` enables some nice features such as error messages with line numbers. There is also a slight chance that using this option reduces the start-up overhead. On the other hand, the variant using the shell's input redirection is (in theory) guaranteed to yield exactly the same output that you would have gotten had you entered everything by hand.

Using `f` filename to Read Data Piped into `vsq1`

To read data piped into `vsq1` from a data file:

1 Create the following:

- A named pipe.

For example, to create a named pipe called `pipe1`:

```
mkfifo pipe1
```

- A data file. The data file in this example is called *data_file*.
- The command file that selects the table into which you want to copy data, copies the data from the pipe file (`pipe1`), and removes the pipe file. The command file in this example is called *command_line*.

2 From the command line, run a command that pipes the data file (`data_file`) into the appropriate table through `vsq1`. The following example pipes the data file into `public.shipping_dimension` in the VMart database.

```
cat data_file > pipe1 | vsq1 -f 'command_line'
```

Example `data_file`:

```
110|EXPRESS|SEA|FEDEX
111|EXPRESS|HAND CARRY|MSC
112|OVERNIGHT|COURIER|USPS
```

Example `command_line` file:

```
SELECT * FROM public.shipping_dimension;
\set dir `pwd`/
\set file ''':dir'pipe1''

COPY public.shipping_dimension FROM :file delimiter '|';
SELECT * FROM public.shipping_dimension;
--Remove the pipe1
\! rm pipe1
```

F separator `--field-separator separator`

`-F separator --field-separator separator` specifies the field separator for unaligned output (default: "|") (`-P fieldsep=`). (See `-A --no-align` (page 361).) This is equivalent to `\pset` (page 379) `fieldsep` or `\f` (see "`f [string]`" on page 376).

h hostname `--host hostname`

`-h hostname --host hostname` specifies the host name of the machine on which the server is running.

Notes:

- If you are using client authentication with a connection method of either "gss" or "krb5" (Kerberos), you are required to specify `-h hostname`.
- If you are using client authentication with a "local" connection type specified, avoid using `-h hostname` if you want to match the client authentication entry.

H --html

`-H --html` turns on HTML tabular output. This is equivalent to `\pset` (page 379) `format html` or the `\H` (see "H" on page 376) command.

I --list

`-I --list` returns all available databases, then exits. Other non-connection options are ignored. This command is similar to the internal command `\list`.

n

`-n` disables command line editing.

o filename --output filename

`-o filename --output filename` writes all query output into file *filename*. This is equivalent to the command `\o` (page 378).

p port --port port

`-p port --port port` specifies the TCP port or the local socket file extension on which the server is listening for connections. Defaults to port 5433.

P assignment --pset assignment

`-P assignment --pset assignment` lets you specify printing options in the style of `\pset` (page 379) on the command line. Note that you have to separate name and value with an equal sign instead of a space. Thus to set the output format to LaTeX, you could write `-P format=latex`.

q --quiet

`-q --quiet` specifies that vsql do its work quietly. By default, it prints welcome messages and various informational output. If this option is used, none of this appears. This is useful with the `-c` (page 361) option. Within vsql you can also set the `QUIET` (page 386) variable to achieve the same effect.

R separator --record-separator separator

`-R separator --record-separator separator` uses *separator* as the record separator. This is equivalent to the `\pset` (page 379) `recordsep` command.

s --single-step

`-s --single-step` runs in single-step mode for debugging scripts. Forces `vsq` to prompt before each statement is sent to the database and allows you to cancel execution.

S --single-line

`-S --single-line` runs in single-line mode where a newline terminates a SQL command, like the semicolon does.

Note: This mode is provided for those who insist on it, but you are not necessarily encouraged to use it, particularly if you mix SQL and meta-commands on a line. The order of execution might not always be clear to the inexperienced user.

t --tuples-only

`-t --tuples-only` disables printing of column names, result row count footers, and so on. This is equivalent to the `\t` (see "t" on page 381) command.

T table_options --table-attr table_options

`-T table_options --table-attr table_options` allows you to specify options to be placed within the HTML `table` tag. See `\pset` (page 379) for details.

U username --username username

`-U username --username username` connects to the database as the user *username* instead of the default.

v assignment --set assignment --variable assignment

`-v assignment --set assignment --variable assignment` performs a variable assignment, like the `\set` (see "set [NAME [VALUE [...]]]" on page 381) internal command.

Note: You must separate name and value, if any, by an equal sign on the command line.

To unset a variable, omit the equal sign. To set a variable without a value, use the equal sign but omit the value. These assignments are done during a very early stage of start-up, so variables reserved for internal purposes can get overwritten later.

V --version

`-V --version` prints the `vsq` version and exits.

w password

`-w password` specifies the password for a database user.

Note: Using this command line option displays the database password in plain text on the screen. Use it with care, particularly if you are connecting as the database administrator.

W --password

`-W --password` forces vsql to prompt for a password before connecting to a database.

The password is not displayed on the screen. This option remains set for the entire session, even if you change the database connection with the meta-command `\connect` (see "`c` (or `\connect`) [dbname [username]]" on page 368).

x --expanded

`-x --expanded` enables extended table formatting mode. This is equivalent to the command `\ x` (see "`x`" on page 382).

X, --no-vsqlrc

`-X, --no-vsqlrc` prevents the start-up file from being read (the system-wide `vsqlrc` file or the user's `~/.vsqlrc` file).

Connecting From a Non-Cluster Host

You can use the Vertica vsql executable image on a non-cluster Linux host to connect to a Vertica database.

- On Red Hat 5.0 64-bit and SUSE 10/11 64-bit, you can install the client driver RPM, which includes the vsql executable. See *Installing the Client RPM on Red Hat 5 64-bit, and SUSE 64-bit* for details.
- If the non-cluster host is running the same version of Linux as the cluster, copy the image file to the remote system. For example:

```
$ scp host01:/opt/vertica/bin/vsql .
$ ./vsql
```

- If the non-cluster host is running a different version of Linux than your cluster hosts, and that operating system is not Red Hat version 5 64-bit or SUSE 10/11 64-bit, you must install the Vertica server RPM in order to get vsql. Download the appropriate rpm package from the Vertica **Download Website** http://www.vertica.com/v-zone/download_vertica then log into the non-cluster host as root and install the rpm package using the command:

```
# rpm -Uvh filename
```

In the above command, *filename* is package you downloaded. Note that you do not have to run the `install_Vertica` script on the non-cluster host in order to use vsql.

Notes

- Use the same **command line options** (on page 360) that you would on a cluster host.
- You cannot run vsql on a Cygwin bash shell (Windows). Use ssh to connect to a cluster host, then run vsql.

Meta-Commands

Anything you enter in vsql that begins with an unquoted backslash is a vsql meta-command that is processed by vsql itself. These commands help make vsql more useful for administration or scripting. Meta-commands are more commonly called slash or backslash commands.

The format of a vsql command is the backslash, followed immediately by a command verb, then any arguments. The arguments are separated from the command verb and each other by any number of whitespace characters.

To include whitespace into an argument you can quote it with a single quote. To include a single quote into such an argument, precede it by a backslash. Anything contained in single quotes is furthermore subject to C-like substitutions for `\n` (new line), `\t` (tab), `\digits`, `\0digits`, and `\0xdigits` (the character with the given decimal, octal, or hexadecimal code).

If an unquoted argument begins with a colon (:), it is taken as a vsql variable and the value of the variable is used as the argument instead.

Arguments that are enclosed in backquotes (`) are taken as a command line that is passed to the shell. The output of the command (with any trailing newline removed) is taken as the argument value. The above escape sequences also apply in backquotes.

Some commands take a SQL identifier (such as a table name) as argument. These arguments follow the syntax rules of SQL: Unquoted letters are forced to lowercase, while double quotes (") protect letters from case conversion and allow incorporation of whitespace into the identifier. Within double quotes, paired double quotes reduce to a single double quote in the resulting name. For example, `FOO"BAR"BAZ` is interpreted as `fooBARbaz`, and `"A weird" name"` becomes `A weird" name`.

Parsing for arguments stops when another unquoted backslash occurs. This is taken as the beginning of a new meta-command. The special sequence `\\` (two backslashes) marks the end of arguments and continues parsing SQL commands, if any. That way SQL and vsql commands can be freely mixed on a line. But in any case, the arguments of a meta-command cannot continue beyond the end of the line.

! [COMMAND]

`\! [COMMAND]` executes a command in a Linux shell (passing arguments as entered) or starts an interactive shell.

?

`\?` displays help information about the meta-commands.

```
=> \?
```

```
General
```

```

\c[onnect] [DBNAME|- [USER]]
    connect to new database (currently "vmartdb")
\cd [DIR]
    change the current working directory
\q
    quit vsql
\set [NAME [VALUE]]
    set internal variable, or list all if no parameters
\timing
    toggle timing of commands (currently off)
\unset NAME
    unset (delete) internal variable
\! [COMMAND]
    execute command in shell or start interactive shell
\password [USER]
    change user's password

Query Buffer
\e [FILE]
    edit the query buffer (or file) with external editor
\g
    send query buffer to server
\g FILE
    send query buffer to server and results to file
\g | COMMAND
    send query buffer to server and pipe results to command
\p
    show the contents of the query buffer
\r
    reset (clear) the query buffer
\s [FILE]
    display history or save it to file
\w FILE
    write query buffer to file

Input/Output
\echo [STRING]
    write string to standard output
\i FILE
    execute commands from file
\o FILE
    send all query results to file
\o | COMMAND
    pipe all query results to command
\o
    close query-results file or pipe
\qecho [STRING]
    write string to query output stream (see \o)

Informational
\d [PATTERN]
    describe tables (list tables if no argument is supplied)
\df [PATTERN]
    list functions
\dj [PATTERN]
    list projections
\dn [PATTERN]
    list schemas
\dp [PATTERN]
    list table access privileges
\ds [PATTERN]
    list sequences
\dS [PATTERN]
    list system tables
\dt [PATTERN]
    list tables
\dtv [PATTERN]
    list tables and views
\dT [PATTERN]
    list data types
\du [PATTERN]
    list users
\dv [PATTERN]
    list views
\l
    list all databases
\z [PATTERN]
    list table access privileges (same as \dp)

Formatting
\a
    toggle between unaligned and aligned output mode
\b
    toggle beep on command completion
\C [STRING]
    set table title, or unset if none
\f [STRING]
    show or set field separator for unaligned query output
\H
    toggle HTML output mode (currently off)
\pset NAME [VALUE]
    set table output option
    (NAME := {format|border|expanded|fieldsep|footer|null|

```

```
recordsep|tuples_only|title|tableattr|pager})
\l          show only rows (currently off)
\T [STRING] set HTML <table> tag attributes, or unset if none
\X          toggle expanded output (currently off)
```

a

`\a` toggles output format alignment. This command is kept for backwards compatibility. See `\pset` (page 379) for a more general solution.

`\a` is similar to the command line option **-A --no-align** (page 361), which only disables alignment.

b

`\b` toggles beep on command completion.

c (or \connect) [dbname [username]]

`\c` (or `\connect`) [*dbname* [*username*]] establishes a connection to a new database and/or under a user name. The previous connection is closed. If *dbname* is - the current database name is assumed.

If *username* is omitted the current user name is assumed.

As a special rule, `\connect` without any arguments connects to the default database as the default user (as you would have gotten by starting `vsq` without any arguments).

If the connection attempt fails (wrong user name, access denied, etc.), the previous connection is kept if and only if `vsq` is in interactive mode. When executing a non-interactive script, processing immediately stops with an error. This distinction that avoids typos and a prevent scripts from accidentally acting on the wrong database.

C [STRING]

`\C` [*STRING*] sets the title of any tables being printed as the result of a query or unsets any such title. This command is equivalent to `\pset` (page 379) `title` *title*. (The name of this command derives from "caption", as it was previously only used to set the caption in an HTML table.)

cd [DIR]

`\cd` [*DIR*] changes the current working directory to *directory*. Without argument, changes to the current user's home directory.

To print your current working directory, use `V` (see "**! [COMMAND]**" on page 366)`pwd`. For example:

```
=> \!pwd
/home/dbadmin
```

The \d [PATTERN] meta-commands

This section describes the various `\d` meta-commands

All `\d` meta-commands take an optional pattern (asterisk [*] or question mark [?]) and return only the records that match that pattern.

The `?` argument is useful if you can't remember if a table name uses an underscore or a dash:

```
=> \dn v?internal
      List of schemas
      Name      | Owner
      -----+-----
      v_internal | dbadmin
(1 row)
```

The output from the `\d` metacommands places double quotes around non-alphanumeric table names and table names that are keywords, such as in the following example.

```
=> CREATE TABLE my_keywords.precision(x numeric (4,2));
CREATE TABLE
=> \d
```

```
      List of tables
      Schema      | Name          | Kind  | Owner
      -----+-----+-----+-----
      my_keywords | "precision"  | table | dbadmin
```

Double quotes are optional when you use a `\d` command with pattern matching.

d [PATTERN]

The `\d [PATTERN]` meta-command lists all tables in the database and returns their schema, table name, kind (e.g., table), and owner. For example, the following is the result of `\d` in the `vmart` schema.

```
vmartdb=> \d
      List of tables
      Schema      | Name          | Kind  | Owner
      -----+-----+-----+-----
      online_sales | call_center_dimension | table | dbadmin
      online_sales | online_page_dimension | table | dbadmin
      online_sales | online_sales_fact     | table | dbadmin
      public       | customer_dimension    | table | dbadmin
      public       | date_dimension        | table | dbadmin
      public       | employee_dimension    | table | dbadmin
      public       | inventory_fact        | table | dbadmin
      public       | product_dimension     | table | dbadmin
      public       | promotion_dimension   | table | dbadmin
      public       | shipping_dimension    | table | dbadmin
      public       | vendor_dimension      | table | dbadmin
      public       | warehouse_dimension   | table | dbadmin
      store        | store_dimension      | table | dbadmin
      store        | store_orders_fact     | table | dbadmin
      store        | store_sales_fact      | table | dbadmin
(15 rows)
```

If you provide the table name as an argument, the result shows the schema name, table name, column name, column data type, data type size, default value, whether it is Nullable or has a NOT NULL constraint, and whether there is a primary key or foreign key constraint.

vmartdb=> \d inventory_fact

List of Fields by Tables								
Schema	Table	Column	Type	Size	Default	Not Null	Primary Key	Foreign Key
public	inventory_fact	date_key	int	8		t	f	
public.date_dimension(date_key)								
public	inventory_fact	product_key	int	8		t	f	
public.product_dimension(product_key)								
public	inventory_fact	product_version	int	8		t	f	
public.product_dimension(product_version)								
public	inventory_fact	warehouse_key	int	8		t	f	
public.warehouse_dimension(warehouse_key)								
public	inventory_fact	qty_in_stock	int	8		f	f	

(5 rows)

You can also use the question mark [?] argument to replace a single character. For example, the ? argument replaces the last character in the SubQ1 and SubQ2 tables, so the command returns information about both:

=> \d SubQ?

List of Fields by Tables								
Schema	Table	Column	Type	Size	Default	Not Null	Primary Key	Foreign Key
public	SubQ1	a	int	8		f	f	
public	SubQ1	b	int	8		f	f	
public	SubQ1	c	int	8		f	f	
public	SubQ2	x	int	8		f	f	
public	SubQ2	y	int	8		f	f	
public	SubQ2	z	int	8		f	f	

(6 rows)

d \d <table> \df \dj \dn \dp \ds \dS \dt \dT \dtv \du \dv

The \df [PATTERN] meta-command returns all function names, the function return data type, and the function argument data type. Also returns the procedure names and arguments for all procedures that are available to the user.

vmartdb=> \df

List of functions		
procedure_name	procedure_return_type	procedure_argument_types
abs	Float	Float
abs	Integer	Integer
abs	Interval	Interval
abs	Interval	Interval
abs	Numeric	Numeric
acos	Float	Float
add_location	Varchar	Varchar
add_location	Varchar	Varchar, Varchar, Varchar
...		
width_bucket	Integer	Float, Float, Float, Integer
width_bucket	Integer	Interval, Interval, Interval, Integer


```

width_bucket      | Integer          | Interval, Interval, Interval, Integer
width_bucket      | Integer          | Timestamp, Timestamp, Timestamp,
Integer
...

```

The following example uses the wildcard character to search for all functions that begin with as:

```
vmartdb=> \df as*
```

```

                          List of functions
procedure_name | procedure_return_type | procedure_argument_types
-----+-----+-----
ascii          | Integer               | Varchar
asin           | Float                 | Float
(2 rows)

```

dj [PATTERN]

The `\dj [PATTERN]` meta-command returns all projections showing the schema, projection name, owner, and node:

```
vmartdb=> \dj
```

```

                          List of projections
Schema      | Name                                     | Owner | Node
-----+-----+-----+-----
public      | product_dimension_node0001             | dbadmin | v_wmartdb_node0001
public      | product_dimension_node0002             | dbadmin | v_wmartdb_node0002
public      | product_dimension_node0003             | dbadmin | v_wmartdb_node0003
online_sales | call_center_dimension_node0001         | dbadmin | v_wmartdb_node0001
online_sales | call_center_dimension_node0002         | dbadmin | v_wmartdb_node0002
online_sales | call_center_dimension_node0003         | dbadmin | v_wmartdb_node0003
...

```

If you supply a projection name as an argument, the system returns fewer records:

```
vmartdb=> \dj call_center_dimension_n*
```

```

                          List of projections
Schema      | Name                                     | Owner | Node
-----+-----+-----+-----
online_sales | call_center_dimension_node0001         | dbadmin | v_wmartdb_node0001
online_sales | call_center_dimension_node0002         | dbadmin | v_wmartdb_node0002
online_sales | call_center_dimension_node0003         | dbadmin | v_wmartdb_node0003
(3 rows)

```

dn [PATTERN]

The `\dn [PATTERN]` meta-command returns the schema names and schema owner.

```
vmartdb=> \dn
```

```

                          List of schemas
Name      | Owner
-----+-----
v_internal | dbadmin
v_catalog  | dbadmin
v_monitor  | dbadmin
public     | dbadmin
store      | dbadmin
online_sales | dbadmin

```

(6 rows)

The following command returns all schemas that begin with the letter v:

```
=> \dn v*
List of schemas
Name      | Owner
-----+-----
v_internal | dbadmin
v_catalog | dbadmin
v_monitor | dbadmin
(3 rows)
```

dp [PATTERN]

The `\dp [PATTERN]` meta-command returns the grantee, grantor, privileges, schema, and name for all table access privileges in each schema:

```
vmartdb=> \dp
Access privileges for database "vmartdb"
Grantee | Grantor | Privileges | Schema | Name
-----+-----+-----+-----+-----
        | dbadmin | USAGE     |        | public
        | dbadmin | USAGE     |        | v_internal
        | dbadmin | USAGE     |        | v_catalog
        | dbadmin | USAGE     |        | v_monitor
(4 rows)
```

Note: `\dp` is the same as `\z` (see "z" on page 382).

ds [PATTERN]

The `\ds [PATTERN]` meta-command (lowercase s) returns a list of sequences and their parameters.

The following series of commands creates a sequence called `my_seq` and uses the `vsq` command to display its parameters:

```
=> CREATE SEQUENCE my_seq MAXVALUE 5000 START 150;
CREATE SEQUENCE
=> \ds
List of Sequences
Schema | Sequence | CurrentValue | IncrementBy | Minimum | Maximum | AllowCycle
-----+-----+-----+-----+-----+-----+-----
public | my_seq   | 149          | 1           | 1       | 5000    | f
(1 row)
```

Note: You can return additional information about sequences by issuing `SELECT * FROM V_CATALOG_SEQUENCES`, as described in the SQL Reference Manual.

dS [PATTERN]

The `\dS [PATTERN]` meta-command (uppercase S) returns all system table (monitoring API) names. You can get identical results issuing `SELECT * FROM system_tables;`

```
vmartdb=> \ds
```

Schema	Name	Kind	Description
v_catalog	columns	system	Table column information
v_catalog	dual	system	Oracle(TM) compatibility DUAL table
v_catalog	foreign_keys	system	Foreign key information
v_catalog	grants	system	Grant information
v_catalog	passwords	system	User password history and password reuse policy
v_catalog	primary_keys	system	Primary key information
v_catalog	profile_parameters	system	Profile Parameters information
v_catalog	profiles	system	Profile information
v_catalog	projection_columns	system	Projection columns information
v_catalog	projections	system	Projection information
...			
v_monitor	host_resources	system	Per host profiling information
v_monitor	load_streams	system	Load metrics for each load stream on each node
v_monitor	locks	system	Lock grants and requests for all nodes
v_monitor	node_resources	system	Per node profiling information
...			

dt [PATTERN]

The `\dt [PATTERN]` meta-command (lowercase t) is identical to `\d` and returns all tables in the database—unless a table name is specified—in which case the command lists only the schema, name, kind and owner for the specified table (or tables if wildcards used).

```
vmartdb=> \dt inventory_fact
          List of tables
 Schema | Name          | Kind | Owner
-----+-----+-----+-----
 public | inventory_fact | table | dbadmin
(1 row)
```

The following command returns all table names that begin with "st":

```
vmartdb=> \dt st*
          List of tables
 Schema | Name              | Kind | Owner
-----+-----+-----+-----
 store  | store_dimension   | table | dbadmin
 store  | store_orders_fact | table | dbadmin
 store  | store_sales_fact  | table | dbadmin
(3 rows)
```

dT [PATTERN]

The `\dT [PATTERN]` meta-command (uppercase T) lists all supported data types.

```
vmartdb=> \dT
List of data types
```

```

type_name
-----
Binary
Boolean
Char
Date
Float
Integer
Interval
Numeric
Time
TimeTz
Timestamp
TimestampTz
Varbinary
Varchar
(14 rows)

```

dtv [PATTERN]

The `\dtv [PATTERN]` meta-command lists all tables and views, returning the schema, table or view name, kind (table of view), and owner.

```

vmartdb=> \dtv

```

Schema	Name	Kind	Owner
online_sales	call_center_dimension	table	release
online_sales	online_page_dimension	table	release
online_sales	online_sales_fact	table	release
public	customer_dimension	table	release
public	date_dimension	table	release
public	employee_dimension	table	release
public	inventory_fact	table	release
public	my_seqview	view	release
public	product_dimension	table	release
public	promotion_dimension	table	release
public	shipping_dimension	table	release
public	vendor_dimension	table	release
public	warehouse_dimension	table	release
store	store_dimension	table	release
store	store_orders_fact	table	release
store	store_sales_fact	table	release

```

(16 rows)

```

du [PATTERN]

The `\du [PATTERN]` meta-command returns all database users and attributes, such as if user is a superuser.

```

vmartdb=> \du

```

User name	Is Superuser
-----------	--------------

```
-----+-----
dbadmin  | t
(1 row)
```

dv [PATTERN]

The `\dv [PATTERN]` meta-command returns the schema name, view name, and view owner.

The following example defines a view using the SEQUENCES system table:

```
vmartdb=> CREATE VIEW my_seqview AS (SELECT * FROM sequences);
CREATE VIEW
vmartdb=> \dv
      List of views
 Schema |   Name   | Owner
-----+-----+-----
 public | my_seqview | dbadmin
(1 row)
```

If a view name is provided as an argument, the result shows the schema, view name, and the following for all columns within the view's result set: schema name, view name, column name, column data type, and data type size.

```
vmartdb=> \dv my_seqview
      List of View Fields
 Schema |   View   |   Column   |   Type   | Size
-----+-----+-----+-----+-----
 public | my_seqview | sequence_schema | varchar(128) | 128
 public | my_seqview | sequence_name   | varchar(128) | 128
 public | my_seqview | owner_name      | varchar(128) | 128
 public | my_seqview | identity_table_name | varchar(128) | 128
 public | my_seqview | session_cache_count | int          | 8
 public | my_seqview | allow_cycle     | boolean      | 1
 public | my_seqview | output_ordered  | boolean      | 1
 public | my_seqview | increment_by    | int          | 8
 public | my_seqview | minimum         | int          | 8
 public | my_seqview | maximum         | int          | 8
 public | my_seqview | current_value   | int          | 8
 public | my_seqview | sequence_schema_id | int          | 8
 public | my_seqview | sequence_id     | int          | 8
 public | my_seqview | owner_id        | int          | 8
 public | my_seqview | identity_table_id | int          | 8
(15 rows)
```

e \edit [FILE]

`\e \edit [FILE]` edits the query buffer (or specified file) with an external editor. When the editor exits, its content is copied back to the query buffer. If no argument is given, the current query buffer is copied to a temporary file which is then edited in the same fashion.

The new query buffer is then re-parsed according to the normal rules of vsql, where the whole buffer up to the first semicolon is treated as a single line. (Thus you cannot make scripts this way. Use `\i` (see "`i FILE`" on page 377) for that.) If there is no semicolon, vsql waits for one to be entered (it does not execute the query buffer).

Tip: vsql searches the environment variables VSQL_EDITOR, EDITOR, and VISUAL (in that order) for an editor to use. If all of them are unset, vi is used on Linux systems, notepad.exe on Windows systems.

echo [STRING]

`\echo [STRING]` writes the string to standard output

Tip: If you use the `\o` (page 378) command to redirect your query output you might want to use `\qecho` (page 380) instead of this command.

f [string]

`\f [string]` sets the field separator for unaligned query output. The default is the vertical bar (`|`). See also `\pset` (page 379) for a generic way of setting output options.

g

The `\g` meta-command sends the query in the input buffer (see `\p` (see "`p`" on page 378)) to the server. With no arguments, it displays the results in the standard way.

`\g FILE` sends the query input buffer to the server, and writes the results to FILE.

`\g | COMMAND` sends the query buffer to the server, and pipes the results to a shell COMMAND.

See Also

`\o` meta-command (see "`o`" on page 378)

H

`\H` toggles HTML query output format. This command is for compatibility and convenience, but see `\pset` (page 379) about setting other output options.

h \help [command]

`\h \help [command]` gives syntax help on the specified SQL command. If *command* is not specified, vsql lists all the commands for which syntax help is available. If *command* is an asterisk (*), syntax help on all SQL commands is shown.

Note: To simplify typing, commands that consists of several words do not have to be quoted. For example:
`\help alter table.`

i FILE

`\i filename` command reads input from the file *filename* and executes it as though it had been typed on the keyboard.

Note: To see the lines on the screen as they are read, set the variable **ECHO** (page 384) to all.

l

`\l` provides a list of databases and their owners.

```
vmartdb=> \l
  List of databases
  name      | user_name
-----+-----
 vmartdb   | dbadmin
(1 row)
```

locale

The vsql `\locale` command displays the current locale setting or lets you set a new locale for the session.

This command does not alter the default locale for all database sessions. To change the default for all sessions, set the `DefaultSessionLocale` **configuration parameter** (page 25).

Viewing the Current Locale Setting

To view the current locale setting, use the vsql command `\locale`, as follows:

```
vmartdb=> \locale
en_US@collation=binary
```

Overriding the Default Local for a Session

To override the default local for a specific session, use the vsql command `\locale <ICU-locale-identifier>`. The session locale setting applies to any subsequent commands issued in the session.

For example:

```
\locale en_GB
INFO:  Locale: 'en_GB'
INFO:   English (United Kingdom)
INFO:  Short form: 'LEN'
```

You can also use the **short form** (page 403) of an ICU locale identifier:

```
\locale LEN
INFO:  Locale: 'en'
INFO:   English
INFO:  Short form: 'LEN'
```

Notes

The server locale settings impact only the collation behavior for server-side query processing. The client application is responsible for ensuring that the correct locale is set in order to display the characters correctly. Below are the best practices recommended by Vertica to ensure predictable results:

- The locale setting in the terminal emulator for vsql (POSIX) should be set to be equivalent to session locale setting on server side (ICU) so data is collated correctly on the server and displayed correctly on the client.
- The vsql locale should be set using the POSIX LANG environment variable in terminal emulator. Refer to the documentation of your terminal emulator for how to set locale.
- Server session locale should be set using the set as described in ***Specify the Default Locale for the Database*** (page 20).
- Note that all input data for vsql should be in UTF-8 and all output data is encoded in UTF-8
- Non UTF-8 encodings and associated locale values are not supported.

O

The `\o` meta-command is used to control where vsql directs its query output. The output can be written to a file, piped to a shell command, or sent to the standard output.

`\o FILE` sends all subsequent query output to FILE.

`\o | COMMAND` pipes all subsequent query output to a shell COMMAND.

`\o` with no argument closes any open file or pipe, and switches back to normal query result output.

Notes

- Query results includes all tables, command responses, and notices obtained from the database server.
- To intersperse text output with query results, use ***!qecho*** (page 380).

See Also

!q meta-command (page 376)

p

`\p` prints the current query buffer to the standard output. For example:

```
=> \p
CREATE VIEW my_seqview AS (SELECT * FROM sequences);
```

password [USER]

`\password` starts the password change process. Users can only change their own passwords. They are prompted for their old password, their new password, and then their new password again to confirm.

The superuser can change the password of another user by supplying the username. The superuser is not prompted for the old password, either when changing his or her own password, or when changing another user's password.

Note: If you want to cancel the password change process, press ENTER until you return to the vsql prompt.

pset NAME [VALUE]

`\pset NAME [VALUE]` sets options affecting the output of query result tables. NAME describes which option to set, as illustrated in the following table. The parameters of VALUE depend thereon.

It is an error to call `\pset` without arguments

Adjustable printing options are:

<code>format</code>	Sets the output format to one of <code>unaligned</code> , <code>aligned</code> , <code>html</code> , or <code>latex</code> . Unique abbreviations are allowed. (That would mean one letter is enough.) "Unaligned" writes all columns of a row on a line, separated by the currently active field separator. This is intended to create output that might be intended to be read in by other programs (tab-separated, comma-separated). "Aligned" mode is the standard, human-readable, nicely formatted text output that is default. The "HTML" and "LaTeX" modes put out tables that are intended to be included in documents using the respective mark-up language. They are not complete documents! (This might not be so dramatic in HTML, but in LaTeX you must have a complete document wrapper.)
<code>border</code>	The second argument must be a number. In general, the higher the number the more borders and lines the tables have, but this depends on the particular format. In HTML mode, this translates directly into the <code>border=...</code> attribute, in the others only values 0 (no border), 1 (internal dividing lines), and 2 (table frame) make sense.
<code>expanded</code>	Toggles between regular and expanded format. When expanded format is enabled, all output has two columns with the column name on the left and the data on the right. This mode is useful if the data wouldn't fit on the screen in the normal "horizontal" mode. Expanded mode is supported by all four output formats. <code>\x</code> is the same as <code>\pset expanded</code> .
<code>fieldsep</code>	Specifies the field separator to be used in unaligned output mode. That way one can create, for example, tab- or comma-separated output, which other programs might prefer. To set a tab as field separator, type <code>\pset fieldsep '\t'</code> . The default field separator is <code>' '</code> (a vertical bar).
<code>footer</code>	Toggles the display of the default footer (<code>x rows</code>).
<code>null</code>	The second argument is a string that is printed whenever a column is null. The default is not to print anything, which can easily be mistaken for, say, an empty string. Thus, one might choose to write <code>\pset null '(null)'</code> .
<code>recordsep</code>	Specifies the record (line) separator to use in unaligned output mode. The

	default is a newline character.
<code>tuples_only</code> (or <code>t</code>)	Toggles between tuples only and full display. Full display might show extra information such as column headers, titles, and various footers. In tuples only mode, only actual table data is shown.
<code>title</code> [<code>text</code>]	Sets the table title for any subsequently printed tables. This can be used to give your output descriptive tags. If no argument is given, the title is unset.
<code>tableattr</code> (or <code>T</code>) [<code>text</code>]	Allows you to specify any attributes to be placed inside the HTML <code>table</code> tag. This could for example be <code>cellpadding</code> or <code>bgcolor</code> . Note that you probably don't want to specify <code>border</code> here, as that is already taken care of by <code>\pset border</code> .
<code>pager</code>	Controls use of a pager for query and <code>vsql</code> help output. If the environment variable <code>PAGER</code> is set, the output is piped to the specified program. Otherwise a platform-dependent default (such as <code>more</code>) is used. When the pager is off, the pager is not used. When the pager is on, the pager is used only when appropriate; that is, the output is to a terminal and does not fit on the screen. (<code>vsql</code> does not do a perfect job of estimating when to use the pager.) <code>\pset pager</code> turns the pager on and off. Pager can also be set to <code>always</code> , which causes the pager to be always used.

See illustrations on how these different formats look in the **Examples** (page 393) section.

Tip: There are various shortcut commands for `\pset`. See **`\a`** (see "**a**" on page 368), **`\C`** (**`C`** [**`STRING`**] on page 368), **`\H`** (see "**H**" on page 376), **`\t`** (see "**t**" on page 381), **`\T`** (**`T`** [**`STRING`**] on page 381), and **`\x`** (see "**x**" on page 382).

q

`\q` quits the `vsql` program.

qecho [**STRING**]

`\qecho` [`STRING`] is identical to `\echo` (see "`echo` [`STRING`]" on page 376) except that the output is written to the query output stream, as set by **`\o`** (see "**o**" on page 378).

r

`\r` resets (clears) the query buffer.

For example, run the **`\p`** (see "**p**" on page 378) meta-command to see what is in the query buffer:

```
=> \p
CREATE VIEW my_seqview AS (SELECT * FROM sequences);
```

Now reset the query buffer:

```
=> \r
Query buffer reset (cleared).
```

If you reissue the command to see what's in the query buffer, you can see it is now empty:

```
=> \p
Query buffer is empty.
```

s [FILE]

`\s [FILE]` prints or saves the command line history to *filename*. If a filename is not specified, `\s` writes the history to the standard output. This option is only available if vsql is configured to use the GNU Readline library.

set [NAME [VALUE [...]]]

`\set [name [value [...]]]` sets the internal variable *name* to *value* or, if more than one value is given, to the concatenation of all of values. If no second argument is given, the variable is set with no value.

If no argument is provided, `\set` lists all internal variables; for example:

```
vmartdb=> \set
VERSION = 'Vertica Analytic Database v4.1.6-0'
AUTOCOMMIT = 'off'
VERBOSITY = 'default'
PROMPT1 = '%/%R%# '
PROMPT2 = '%/%R%# '
PROMPT3 = '>> '
ROWS_AT_A_TIME = '1000'
DBNAME = 'vmartdb'
USER = 'dbadmin'
PORT = '5433'
LOCALE = 'en_US@collation=binary'
HISTSIZE = '500'
```

Notes

- Valid variable names are case sensitive and can contain characters, digits, and underscores. vsql treats several variables as special, which are described in **Variables** (page 382).
- The `\set` parameter `ROWS_AT_A_TIME` defaults to 1000. It retrieves results as blocks of rows of that size. The column formatting for the first block is used for all blocks, so in later blocks some entries could overflow. See **timing** (page 382) for examples.
- To unset a variable, use the **unset** (page 382) command.

t

`\t` toggles the display of output column name headings and row count footer. This command is equivalent to `\pset` (page 379) `tuples_only` and is provided for convenience.

T [STRING]

`\T [STRING]` specifies attributes to be placed within the `table` tag in HTML tabular output mode. This command is equivalent to `\pset` (page 379) `tableattr` *table_options*.

timing

`\timing` toggles toggles the timing of commands (currently off). The meta-command displays how long each SQL statement takes, in milliseconds, and reports both the time required to fetch the first block of rows from the server and the total until the last block is formatted.

Example

```
=> \o /dev/null
=> SELECT * FROM fact LIMIT 100000;
Time: First fetch (1000 rows): 22.054 ms. All rows formatted: 235.056 ms
```

Note that the database retrieved the first 1000 rows in 22 ms and completed retrieving and formatting all rows in 235 ms.

```
=> \unset ROWS_AT_A_TIME
=> select * from fact limit 100000;
Time: First fetch (100000 rows): 220.286 ms. All rows formatted: 231.778 ms
```

In this case, the database retrieved all 100000 rows in 220 ms and spent 11 ms formatting them.

Note: Use `\unset` (page 382) with the `ROWS_AT_A_TIME` (page 381) parameter to get results comparable to Vertica 2.5.

See Also

`lset` (page 381)

unset [NAME]

`\unset [NAME]` unsets (deletes) the internal variable *name* that was set using the `lset` (page 381) meta-command.

w [FILE]

`\w [FILE]` outputs the current query buffer to the file *filename*.

x

`\x` toggles extended table formatting mode. Is equivalent to `\pset` (page 379) expanded.

Note: There is no space between the backslash and the x.

z

`\z` lists table access privileges (grantee, grantor, privilege, and name) for all table access privileges in each schema. Is the same as `ldp` (see "`dp [PATTERN]`" on page 372)

Variables

`vsq` provides variable substitution features similar to common Linux command shells. Variables are simply name/value pairs, where the value can be any string of any length. To set variables, use the `vsq` meta-command `\set` (see "`set [NAME [VALUE [...]]]`" on page 381):

```
testdb=> \set fact dim
```

sets the variable `fact` to the value `dim`. To retrieve the content of the variable, precede the name with a colon and use it as the argument of any slash command:

```
testdb=> \echo :fact
dim
```

Note: The arguments of `\set` are subject to the same substitution rules as with other commands. For example, `\set dim :fact` is a valid way to copy a variable.

If you call `\set` without a second argument, the variable is set, with an empty string as value. To unset (or delete) a variable, use the command `\unset` (see "`unset [NAME]`" on page 382).

vsq's internal variable names can consist of letters, numbers, and underscores in any order and any number. Some of these variables are treated specially by vsq. They indicate certain option settings that can be changed at run time by altering the value of the variable or represent some state of the application. Although you can use these variables for any other purpose, this is not recommended. By convention, all specially treated variables consist of all upper-case letters (and possibly numbers and underscores). To ensure maximum compatibility in the future, avoid using such variable names for your own purposes.

SQL Interpolation

An additional useful feature of vsq variables is that you can substitute ("interpolate") them into regular SQL statements. The syntax for this is again to prepend the variable name with a colon (:).

```
testdb=> \set fact 'my_table'
testdb=> SELECT * FROM :fact;
```

would then query the table `my_table`. The value of the variable is copied literally, so it can even contain unbalanced quotes or backslash commands. Make sure that it makes sense where you put it. Variable interpolation is not performed into quoted SQL entities.

AUTOCOMMIT

When AUTOCOMMIT is set 'on', each SQL command is automatically committed upon successful completion; for example:

```
\set (see "set [ NAME [ VALUE [ ... ] ]" on page 381) AUTOCOMMIT on
```

To postpone COMMIT in this mode, set the value as off.

```
\set AUTOCOMMIT off
```

If AUTOCOMMIT is empty or defined as off, SQL commands are not committed unless you explicitly issue COMMIT.

Notes

- AUTOCOMMIT is off by default.
- AUTOCOMMIT must be in uppercase, but the values, on or off, are case insensitive.
- In autocommit-off mode, you must explicitly abandon any failed transaction by entering ABORT or ROLLBACK.
- If you exit the session without committing, your work is rolled back.
- Validation on vsq variables is done when they are run, not when they are set.

- The COPY statement, by default, commits on completion, so it does not matter which AUTOCOMMIT mode you use, unless you issue COPY NO COMMIT.
- To tell if AUTOCOMMIT is on or off, issue the set command:

```
$ \set
...
AUTOCOMMIT = 'off'
...
```

- AUTOCOMMIT is off if a SELECT * FROM LOCKS shows locks from the statement you just ran.

```
$ \set AUTOCOMMIT off
$ \set
...
AUTOCOMMIT = 'off'
...
SELECT COUNT(*) FROM customer_dimension;
  count
-----
 50000
(1 row)
SELECT node_names, object_name, lock_mode, lock_scope
FROM LOCKS;
 node_names |          object_name          | lock_mode | lock_scope
-----+-----+-----+-----
  site01    | Table:customer_dimension      | S         | TRANSACTION
(1 row)
```

DBNAME

The name of the database to which you are currently connected. DBNAME is set every time you connect to a database (including program startup), but it can be unset.

ECHO

If set to `all`, all lines entered from the keyboard or from a script are written to the standard output before they are parsed or run.

To select this behavior on program start-up, use the switch `-a` (see "`a --echo-all`" on page 361). If set to `queries`, `vsq` merely prints all queries as they are sent to the server. The switch for this is `-e` (see "`e --echo-queries`" on page 361).

ECHO_HIDDEN

When this variable is set and a backslash command queries the database, the query is first shown. This way you can study the Vertica internals and provide similar functionality in your own programs. (To select this behavior on program start-up, use the switch `-E` (see "`E`" on page 361).)

If you set the variable to the value `noexec`, the queries are just shown but are not actually sent to the server and run.

ENCODING

The current client character set encoding.

HISTCONTROL

If this variable is set to `ignorespace`, lines that begin with a space are not entered into the history list. If set to a value of `ignoredups`, lines matching the previous history line are not entered. A value of `ignoreboth` combines the two options. If unset, or if set to any other value than those previously mentioned, all lines read in interactive mode are saved on the history list.

Source: Bash.

HISTSIZE

The number of commands to store in the command history. The default value is 500.

Source: Bash.

HOST

The database server host you are currently connected to. This is set every time you connect to a database (including program startup), but can be unset.

IGNOREEOF

If unset, sending an EOF character (usually Control+D) to an interactive session of vsql terminates the application. If set to a numeric value, that many EOF characters are ignored before the application terminates. If the variable is set but has no numeric value, the default is 10.

Source: Bash.

ON_ERROR_STOP

By default, if non-interactive scripts encounter an error, such as a malformed SQL command or internal meta-command, processing continues. This has been the traditional behavior of vsql but it is sometimes not desirable. If this variable is set, script processing immediately terminates. If the script was called from another script it terminates in the same manner. If the outermost script was not called from an interactive vsql session but rather using the `-f` (see "`f filename --file filename`" on page 361) option, vsql returns error code 3, to distinguish this case from fatal error conditions (error code 1).

PORT

The database server port to which you are currently connected. This is set every time you connect to a database (including program start-up), but can be unset.

PROMPT1 PROMPT2 PROMPT3

These specify what the prompts vsql issues look like. See *Prompting* (page 387) below.

QUIET

This variable is equivalent to the command line option `-q` (see "q" on page 380). It is probably not too useful in interactive mode.

SINGLELINE

This variable is equivalent to the command line option `-S` (see "S --single-line" on page 364).

SINGLESTEP

This variable is equivalent to the command line option `-s` (page 364).

USER

The database user you are currently connected as. This is set every time you connect to a database (including program startup), but can be unset.

VERBOSITY

This variable can be set to the values `default`, `verbose`, or `terse` to control the verbosity of error reports.

VSQL_HOME

By default, the `vsq` program reads configuration files from the user's home directory. In cases where this is not desirable, the configuration file location can be overridden by setting the `VSQL_HOME` environment variable in a way that does not require modifying a shared resource.

In the following example, `vsq` reads configuration information out of `/tmp/jsmith` rather than out of `~`.

```
# Make an alternate configuration file in /tmp/jsmith
mkdir -p /tmp/jsmith
echo "\\echo Using VSQ_LRC in tmp/jsmith" > /tmp/jsmith/.vsq_lrc
# Note that nothing is echoed when invoked normally
vsq
# Note that the .vsq_lrc is read and the following is
# displayed before the vsq prompt
#
# Using VSQ_LRC in tmp/jsmith
VSQ_LHOME=/tmp/jsmith vsq
```


Prompting

The prompts vsql issues can be customized to your preference. The three variables `PROMPT1`, `PROMPT2`, and `PROMPT3` contain strings and special escape sequences that describe the appearance of the prompt. Prompt 1 is the normal prompt that is issued when vsql requests a new command. Prompt 2 is issued when more input is expected during command input because the command was not terminated with a semicolon or a quote was not closed. Prompt 3 is issued when you run a SQL `COPY` command and you are expected to type in the row values on the terminal.

The value of the selected prompt variable is printed literally, except where a percent sign (%) is encountered. Depending on the next character, certain other text is substituted instead. Defined substitutions are:

<code>%M</code>	The full host name (with domain name) of the database server, or [local] if the connection is over a socket, or [local:/dir/name], if the socket is not at the compiled in default location.
<code>%m</code>	The host name of the database server, truncated at the first dot, or [local].
<code>%></code>	The port number at which the database server is listening.
<code>%n</code>	The database session user name.
<code>%/</code>	The name of the current database.
<code>%~</code>	Like %/, but the output is ~ (tilde) if the database is your default database.
<code>%#</code>	If the session user is a database superuser, then a #, otherwise a >. (The expansion of this value might change during a database session as the result of the command <code>SET SESSION AUTHORIZATION</code> .)
<code>%R</code>	In prompt 1 normally =, but ^ if in single-line mode, and ! if the session is disconnected from the database (which can happen if <code>\connect</code> fails). In prompt 2 the sequence is replaced by -, *, a single quote, a double quote, or a dollar sign, depending on whether vsql expects more input because the command wasn't terminated yet, because you are inside a <code>/ * ... */</code> comment, or because you are inside a quoted or dollar-escaped string. In prompt 3 the sequence doesn't produce anything.
<code>%x</code>	Transaction status: an empty string when not in a transaction block, or * when in a transaction block, or ! when in a failed transaction block, or ? when the transaction state is indeterminate (for example, because there is no connection).
<code>%digits</code>	The character with the indicated numeric code is substituted. If digits starts with 0x the rest of the characters are interpreted as hexadecimal; otherwise if the first digit is 0 the digits are interpreted as octal; otherwise the digits are read as a decimal number.
<code>%:name:</code>	The value of the vsql variable name. See the section Variables for details.
<code>%`command`</code>	The output of command, similar to ordinary "back-tick" substitution.
<code>%[... %]</code>	Prompts may contain terminal control characters which, for example, change the color, background, or style of the prompt text, or change the title of the terminal window. In order for the line editing features of Readline to work properly, these non-printing control characters must be designated as invisible by surrounding them

with %[and %]. Multiple pairs of these may occur within the prompt. The following example results in a boldfaced (1;) yellow-on-black (33;40) prompt on VT100-compatible, color-capable terminals:

```
testdb=> \set PROMPT1 '%[%033[1;33;40m%] %n@%/%R%[%033[0m%##] '
```

To insert a percent sign into your prompt, write %%. The default prompts are '%/%R%#' for prompts 1 and 2, and '>>' for prompt 3.

Note: This feature was adapted from tcsh.

Command Line Editing

vsq! supports the tecla library for convenient line editing and retrieval.

The command history is automatically saved when vsq! exits and is reloaded when vsq! starts up. Tab-completion is also supported, although the completion logic makes no claim to be a SQL parser. If for some reason you do not like the tab completion, you can turn it off by putting this in a file named `.teclarc` in your home directory:

```
bind ^I
```

Read the tecla documentation for further details.

Notes

The vsq! implementation of the tecla library deviates from the tecla documentation as follows:

- **Recalling Previously Typed Lines**
Under pure tecla, all new lines are appended to a list of historical input lines maintained within the GetLine resource object. In vsq!, only different, non-empty lines are appended to the list of historical input lines.
- **History Files**
tecla has no standard name for the history file. In vsq!, the file name is called `~/vsq!_hist`.
- **International Character Sets (Meta keys and locales)**
In vsq!, 8-bit meta characters are no longer supported. Make sure that meta characters send an escape by setting their EightBitInput X resource to False. You can do this in one of the following ways:
 - Edit the `~/Xdefaults` file by adding the following line:

```
XTerm*EightBitInput: False
```
 - Start an xterm with an `-xrm '*EightBitInput: False'` command-line argument.
- **Key Bindings:**
- The following key bindings are specific to vsq!:
 - *Insert* switches between insert mode (the default) and overwrite mode.
 - *Delete* deletes the character to the right of the cursor.
 - *Home* moves the cursor to the front of the line.
 - *End* moves the cursor to the end of the line.
 - `^R` Performs a history backwards search.

Environment

PAGER

If the query results do not fit on the screen, they are piped through this command. Typical values are `more` or `less`. The default is platform-dependent. The use of the pager can be disabled by using the `\pset` (see "`pset NAME [VALUE]`" on page 379) command.

PGDATABASE

Default connection database

PGHOST

PGPORT

PGUSER

Default connection parameters

VSQL_EDITOR

EDITOR

VISUAL

Editor used by the `\e` command. The variables are examined in the order listed; the first that is set is used.

SHELL

Command run by the `\!` (see "`! [COMMAND]`" on page 366) command.

TMPDIR

Directory for storing temporary files. The default is `/tmp`.

Locales

The default terminal emulator under Linux is `gnome-terminal`, although `xterm` can also be used.

Vertica recommends that you use `gnome-terminal` with `vsql` in UTF-8 mode, which is its default.

To change settings on Linux

- 1 From the tabs at the top of the `vsql` screen, select **Terminal**.
- 2 Click **Set Character Encoding**.
- 3 Select **Unicode (UTF-8)**.

Note: This works well for standard keyboards. `xterm` has a similar UTF-8 option.

To change settings on Windows using PuTTY

- 1 Right click the `vsql` screen title bar and select **Change Settings**.
- 2 Click **Window** and click **Translation**.
- 3 Select **UTF-8** in the drop-down menu on the right.

Notes

- vsql has no way of knowing how you have set your terminal emulator options.
- The tecla library is prepared to do POSIX-type translations from a local encoding to UTF-8 on interactive input, using the POSIX LANG, etc., environment variables. This could be useful to international users who have a non-UTF-8 keyboard. See the tecla documentation for details.

Vertica recommends the following (or whatever other .UTF-8 locale setting you find appropriate):

```
export LANG=en_US.UTF-8
```

- The vsql **Vocale** (see "**locale**" on page 377) command invokes and tracks the server SET LOCALE TO command, described in the SQL Reference Manual. vsql itself currently does nothing with this locale setting, but rather treats its input (from files or from tecla), all its output, and all its interactions with the server as UTF-8. vsql ignores the POSIX locale variables, except for any "automatic" uses in `printf`, and so on.

Files

Before starting up, vsql attempts to read and execute commands from the system-wide `vsqllrc` file and the user's `~/.vsqllrc` file. The command-line history is stored in the file `~/.vsql_history`.

Tip: If you want to save your old history file, open another terminal window and save a copy to a different file name.

Exporting Data Using vsql

You can use vsql for simple data exports tasks by changing its output format options so the output is suitable for importing into other systems (tab delimited or comma-separated files, for example). These options can be set either from within an interactive vsql session, or through command-line arguments to the vsql command (making the export process suitable for automation through scripting). After you have set vsql's options so it outputs the data in a format your target system can read, you run a query and capture the result in a text file.

The following table lists the meta-commands and command-line options that are useful for changing the format of vsql's output.

Description	Meta-command	Command-line Option
Disable padding used to align output.	la (page 368)	-A (page 361) or <code>--no-align</code>
Show only tuples, disabling column headings and row counts.	lt (page 381)	-t (page 364) or <code>--tuples-only</code>
Set the field separator character.	lpset (page 379)	-F (page 362) or

	fieldsep	--field-separator
Send output to a file.	o (page 378)	-o (page 363) or --output
Specify a SQL statement to execute.	N/A	-c (page 361) or --command

The following example demonstrates disabling padding and column headers in the output, and setting a field separator to dump a table to a tab-separated text file within an interactive session.

```
=> SELECT * FROM my_table;
 a |   b   | c
---+-----+---
 a | one   | 1
 b | two   | 2
 c | three | 3
 d | four  | 4
 e | five  | 5
(5 rows)
```

```
=> \a
Output format is unaligned.
=> \t
Showing only tuples.
=> \pset fieldsep '\t'
Field separator is "    ".
=> \o dumpfile.txt
=> select * from my_table;
=> \o
=> \! cat dumpfile.txt
a      one      1
b      two      2
c      three    3
d      four     4
e      five     5
```

Note: You could encounter issues with empty strings being converted to NULLs or the reverse using this technique. You can prevent any confusion by explicitly setting null values to output a unique string such as NULLNULLNULL (for example, `\pset null 'NULLNULLNULL'`). Then, on the import end, convert the unique string back to a null value. For example, if you are copying the file back into a Vertica database, you would give the argument `NULL 'NULLNULLNULL'` to the COPY statement.

When logged into one of the database nodes, you can create the same output file directly from the command line by passing the right parameters to vsql:

```
> vsql -U username -F '$\t' -At -o dumpfile.txt -c "SELECT * FROM my_table;"
Password:
> cat dumpfile.txt
a      one      1
b      two      2
c      three    3
d      four     4
e      five     5
```

Note: `$'...'` is a BASH-specific string format that interprets backslash escapes, so it will pass a literal tab character to the `vsql` command as the argument for the `-F` parameter. Shells other than BASH may have other string literal syntax.

If you want to convert null values to a unique string as mentioned earlier, you can add the argument `-P null='NULLNULLNULL'` (or whatever unique string you choose).

By adding the `-w` `vsql` command-line option to the example command line, you could use the command within a batch script to automate the data export. However, the script would contain the database password as plain text. If you take this approach, you should prevent unauthorized access to the batch script, and also have the script use a database user account that has limited access.

Copying Data Using `vsql`

You can use `vsql` to copy data between two Vertica databases. This technique is similar to the technique explained in *Exporting Data via `vsql`* (page 390), except instead of having `vsql` save data to a file for export, you pipe one `vsql`'s output to the input of another `vsql` command that runs a `COPY` statement from `STDIN`. This technique can also work for other databases or applications that accept data from an input stream.

The easiest way to copy using `vsql` is to log into a node of the target database, then issue a `vsql` command that connects to the source Vertica database to dump the data you want. For example, the following command copies the `store.store_sales_fact` table from the `vmart` database on node `testdb01` to the `vmart` database on the node you are logged into:

```
vsql -U username -w passwd -h testdb01 -d vmart -At -c "SELECT * from store.store_sales_fact" \
| vsql -U username -w passwd -d vmart -c "COPY store.store_sales_fact FROM STDIN DELIMITER '|';"
```

Note: The above example copies the data only, not the table design. The target table for the data copy must already exist in the target database. You can export the design of the table using `EXPORT_OBJECTS` or `EXPORT_CATALOG`.

Monitoring Progress (optional)

You may want some way of monitoring progress when copying large amounts of data between Vertica databases. One way of monitoring the progress of the copy operation is to use a utility such as *Pipe Viewer* (<http://www.ivarch.com/programs/pv.shtml>) that pipes its input directly to its output while displaying the amount and speed of data it passes along. Pipe Viewer can even display a progress bar if you give it the total number of bytes or lines you expect to be processed. You can get the number of lines to be processed by running a separate `vsql` command that executes a `SELECT COUNT` query.

Note: Pipe Viewer isn't a standard Linux or Solaris command, so you will need download and install it yourself. See the *Pipe Viewer* (<http://www.ivarch.com/programs/pv.shtml>) page for download packages and instructions. Vertica Systems, Inc. does not support Pipe Viewer. Install and use it at your own risk.

The following command demonstrates how you can use Pipe Viewer to monitor the progress of the copy shown in the prior example. The command is complicated by the need to get the number of rows that will be copied, which is done using a separate vsql command within a BASH backquote string, which executes the strings contents and inserts the output of the command into the command line. This vsql command just counts the number of rows in the store.store_sales_fact table.

```
vsql -U username -w passwd -h testdb01 -d vmart -At -c "SELECT * from store.store_sales_fact" \
| pv -lpetr -s `vsql -U username -w passwd -h testdb01 -d vmart -At -c "SELECT COUNT (*) FROM
store.store_sales_fact;"` \
| vsql -U username -w passwd -d vmart -c "COPY store.store_sales_fact FROM STDIN DELIMITER '|';"
```

While running, the above command displays a progress bar that looks like this:

```
0:00:39 [12.6M/s] [=====] ] 50% ETA 00:00:40
```

Notes for Windows Users

vsql is built as a "console application." The Windows console windows use a different encoding than the rest of the system, so take care when you use 8-bit characters within vsql. If vsql detects a problematic console code page, it warns you at startup. To change the console code page, two things are necessary:

- Set the code page by entering `cmd.exe /c chcp 1252`.
1252 is a code page that is appropriate for German; replace it with your value.
Note: If you use Cygwin, you can put this command in `/etc/profile`.
- Set the console font to "Lucida Console", because the raster font does not work with the ANSI code page.

Output Formatting Examples

The first example shows how to spread a command over several lines of input. Notice the changing prompt:

```
testdb=> CREATE TABLE my_table (
testdb(> first integer not null default 0,
testdb(> second text) testdb-> ;
CREATE TABLE
```

Assume you have filled the table with data and want to take a look at it:

```
testdb=> SELECT * FROM my_table;
 first | second
-----+-----
      1 | one
      2 | two
      3 | three
      4 | four
(4 rows)
```

You can display tables in different ways by using the `\pset` command:

```
testdb=> \pset border 2
Border style is 2.
```

```
testdb=> SELECT * FROM my_table;
+-----+-----+
| first | second |
+-----+-----+
|      1 | one    |
|      2 | two    |
|      3 | three  |
|      4 | four   |
+-----+-----+
(4 rows)
testdb=> \pset border 0
Border style is 0.
testdb=> SELECT * FROM my_table;
first second
-----
      1 one
      2 two
      3 three
      4 four
(4 rows)
testdb=>
\pset border 1
Border style is 1.
testdb=> \pset format unaligned
Output format is unaligned.
testdb=> \pset fieldsep ","
Field separator is ",".
testdb=> \pset tuples_only
Showing only tuples.
testdb=> SELECT second, first FROM my_table; one,1
two,2
three,3
four,4
```

Alternatively, use the short commands:

```
testdb=> \a \t \ x
Output format is aligned.
Tuples only is off.
Expanded display is on.
testdb=> SELECT * FROM my_table;
-[ RECORD 1 ]-
first | 1
second | one
-[ RECORD 2 ]-
first | 2
second | two
-[ RECORD 3 ]-
first | 3
second | three
-[ RECORD 4 ]-
first | 4
second | four
```


Appendix: Locales

Vertica supports the following internationalization features:

Unicode Character Encoding: UTF-8 (8-bit UCS/Unicode Transformation Format)

Vertica 4.1 stores character data in UTF-8 is an abbreviation for Unicode Transformation Format-8 (where 8 equals 8-bit) and is a variable-length character encoding for Unicode created by Ken Thompson and Rob Pike. UTF-8 can represent any universal character in the Unicode standard, yet the initial encoding of byte codes and character assignments for UTF-8 is coincident with ASCII (requiring little or no change for software that handles ASCII but preserves other values).

All input data received by the database server is expected to be in UTF-8, and all data output by Vertica is in UTF-8. The ODBC API operates on data in UCS-2, and JDBC and ADO.NET APIs operate on data in UTF-16. The client drivers automatically convert data to (from) UTF-8 when sending to (receiving from) the database server.

Locales

The locale is a parameter that defines the user's language, country, and any special variant preferences, such as collation. Vertica uses the locale to determine the behavior of various string functions as well for collation for various SQL commands that require ordering and comparison; for example, GROUP BY, ORDER BY, joins, the analytic ORDER BY clause, and so forth.

By default, the locale for the database is `en_US@collation=binary` (English US). You can establish a new default locale that is used for all sessions on the database, as well as override individual sessions with different locales. Additionally the locale can be set through ODBC, JDBC, and ADO.net.

See the following topics in the Administrator's Guide

- **Implement Locales for International Data Sets** (page 19)
- **Supported Locales** (page 404)
- **Appendix** (page 395)

Notes

- Projections are always collated using the `en_US@collation=binary` collation regardless of the session collation. Any locale-specific collation is applied at query time.
- The maximum length parameter for VARCHAR and CHAR data type refers to the number of octets (bytes) that can be stored in that field and not number of characters. When using multi-byte UTF-8 characters, the fields must be sized to accommodate from 1 to 4 bytes per character, depending on the data.
- When the locale is non-binary, the collation function is used to transform the input to a binary string which sorts in the proper order.

This transformation increases the number of bytes required for the input according to this formula:

```
result_column_width = input_octet_width * CollationExpansion + 4
```

CollationExpansion defaults to 5 and should be changed only under the supervision of Vertica **Technical Support** (on page 1).

String Functions

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

- BTRIM removes the longest string consisting only of specified characters from the start and end of a string.
- CHARACTER_LENGTH returns an integer value representing the number of characters or octets in a string.
- GREATEST returns the largest value in a list of expressions.
- GREATESTB returns its greatest argument, using binary ordering, not UTF-8 character ordering.
- INITCAP capitalizes first letter of each alphanumeric word and puts the rest in lowercase.
- INITCAPB capitalizes first letter of each alphanumeric word and puts the rest in lowercase.
- INSTR searches string for substring and returns an integer indicating the position of the character in string that is the first character of this occurrence.
- INSTRB searches string for substring and returns an integer indicating the octet position of the byte in string that is the first byte of this occurrence.
- LEAST returns the smallest value in a list of expressions.
- LEASTB returns its least argument, using binary ordering, not UTF-8 character ordering.
- LEFT returns the specified characters from the left side of a string.
- LENGTH takes one argument as an input and returns returns an integer value representing the number of characters in a string.
- LOWER returns a VARCHAR value containing the argument converted to lowercase letters.
- LOWERB returns a character string with each ASCII character converted to lowercase; multibyte UTF-8 characters are not converted.
- LTRIM returns a VARCHAR value representing a string with leading blanks removed from the left side (beginning).
- OVERLAY returns a VARCHAR value representing a string having had a substring replaced by another string.
- OVERLAYB returns an octet value representing a string having had a substring replaced by another string.
- POSITION returns an integer value representing the character location of a specified substring with a string (counting from one).
- POSITIONB returns an integer value representing the octet location of a specified substring with a string (counting from one).
- REPLACE replaces all occurrences of characters in a string with another set of characters.
- RIGHT returns the *length* right-most characters of string.

- SPLIT_PART splits string on the delimiter and returns the location of the beginning of the given field (counting from one).
- SPLIT_PARTB splits string on the delimiter and returns the octet location of the beginning of the given field (counting from one)
- STRPOS returns an integer value representing the character location of a specified substring within a string (counting from one).
- STRPOSB returns an integer value representing the octet location of a specified substring within a string (counting from one).
- SUBSTR returns a VARCHAR value representing a substring of a specified string.
- SUBSTRB returns a byte value representing a substring of a specified string.
- SUBSTRING given a value, a position, and an optional length, returns a value representing a substring of the specified string at the given position.
- TRANSLATE replaces individual characters in *string_to_replace* with other characters.
- UPPER returns a VARCHAR value containing the argument converted to uppercase letters.
- UPPERB returns a character string with each ASCII character converted to uppercase; multibyte UTF-8 characters are not converted.

See Also

String Literals in the SQL Reference Manual.

Locale Specification

The locale is a parameter that defines the user's language, country, and any special variant preferences, such as collation. Vertica uses the locale to determine the behavior of various string functions as well for collation for various SQL commands that require ordering and comparison; for example, GROUP BY, ORDER BY, joins, the analytic ORDER BY clause, and so forth.

By default, the locale for the database is `en_US@collation=binary` (English US). You can establish a new default locale that is used for all sessions on the database, as well as override individual sessions with different locales. Additionally the locale can be set through ODBC, JDBC, and ADO.net.

Vertica locale specifications follow a subset of the **Unicode LDML** <http://www.unicode.org/reports/tr35> standard as implemented by the ICU library.

Locales are specified using **long** (page 397) or **short** (page 403) forms.

Long Form

The long form uses full keyname pair/value names.

Syntax

```
[language] [_script] [_country] [_variant] [@keyword=type[;keyword=type]...]
```

Note: Only collation-related keywords are supported by Vertica 4.0.

Parameters

language	A two- or three-letter lowercase code for a particular language. For example, Spanish is "es", English is "en" and French is "fr". The two-letter language code uses the ISO-639 standard.
_script	An optional four-letter script code that follows the language code. If specified, it should be a valid script code as listed on the Unicode ISO 15924 Registry.
_country	A specific language convention within a generic language for a specific country or region. For example, French is spoken in many countries, but the currencies are different in each country. To allow for these differences among specific geographical, political, or cultural regions, locales are specified by two-letter, uppercase codes. For example, "FR" represents France and "CA" represents Canada. The two letter country code uses the ISO-3166 standard.
_variant	<p>Differences may also appear in language conventions used within the same country. For example, the Euro currency is used in several European countries while the individual country's currency is still in circulation. To handle variations inside a language and country pair, add a third code, the variant code. The variant code is arbitrary and completely application-specific. ICU adds "_EURO" to its locale designations for locales that support the Euro currency. Variants can have any number of underscored key words. For example, "EURO_WIN" is a variant for the Euro currency on a Windows computer.</p> <p>Another use of the variant code is to designate the Collation (sorting order) of a locale. For instance, the "es__TRADITIONAL" locale uses the traditional sorting order which is different from the default modern sorting of Spanish.</p>

keyword	<p>Use optional keywords and their values to specify collation order and currency instead of variants (as desired). If used, keywords must be unique, but their order is not significant. If a keyword is unknown or unsupported an error is reported. Keywords and values are not case sensitive.</p> <p>Vertica supports the following keywords:</p> <table border="1" data-bbox="407 428 1383 1841"> <thead> <tr> <th data-bbox="407 428 613 464">Keyword</th> <th data-bbox="617 428 743 464">Short form</th> <th data-bbox="747 428 1383 464">Description</th> </tr> </thead> <tbody> <tr> <td data-bbox="407 468 613 499">collation</td> <td data-bbox="617 468 743 499">K</td> <td data-bbox="747 468 1383 1841"> <p>If present, the collation keyword modifies how the collation service searches through the locale data when instantiating a collator. Collation supports the following values:</p> <ul style="list-style-type: none"> ▪ big5han — Pinyin ordering for Latin, big5 charset ordering for CJK characters (used in Chinese). ▪ dict — For a dictionary-style ordering (such as in Sinhala). ▪ direct — Hindi variant. ▪ gb2312/gb2312han — Pinyin ordering for Latin, gb2312han charset ordering for CJK characters (used in Chinese). ▪ phonebook — For a phonebook-style ordering (such as in German). ▪ pinyin — Pinyin ordering for Latin and for CJK characters; that is, an ordering for CJK characters based on a character-by-character transliteration into a pinyin (used in Chinese). ▪ reformed — Reformed collation (such as in Swedish). ▪ standard — The default ordering for each language. For root it is [UCA] order; for each other locale it is the same as UCA (Unicode Collation Algorithm http://unicode.org/reports/tr10/) ordering except for appropriate modifications to certain characters for that language. The following are additional choices for certain locales; they have effect only in certain locales. ▪ stroke — Pinyin ordering for Latin, stroke order for CJK characters (used in Chinese) not supported. ▪ traditional — For a traditional-style ordering (such as in Spanish). ▪ unihan — Pinyin ordering for Latin, Unihan radical-stroke ordering for CJK characters (used in Chinese) not supported. ▪ binary — the Vertica default, providing UTF-8 octet ordering, compatible with Vertica 3.5. </td> </tr> </tbody> </table>	Keyword	Short form	Description	collation	K	<p>If present, the collation keyword modifies how the collation service searches through the locale data when instantiating a collator. Collation supports the following values:</p> <ul style="list-style-type: none"> ▪ big5han — Pinyin ordering for Latin, big5 charset ordering for CJK characters (used in Chinese). ▪ dict — For a dictionary-style ordering (such as in Sinhala). ▪ direct — Hindi variant. ▪ gb2312/gb2312han — Pinyin ordering for Latin, gb2312han charset ordering for CJK characters (used in Chinese). ▪ phonebook — For a phonebook-style ordering (such as in German). ▪ pinyin — Pinyin ordering for Latin and for CJK characters; that is, an ordering for CJK characters based on a character-by-character transliteration into a pinyin (used in Chinese). ▪ reformed — Reformed collation (such as in Swedish). ▪ standard — The default ordering for each language. For root it is [UCA] order; for each other locale it is the same as UCA (Unicode Collation Algorithm http://unicode.org/reports/tr10/) ordering except for appropriate modifications to certain characters for that language. The following are additional choices for certain locales; they have effect only in certain locales. ▪ stroke — Pinyin ordering for Latin, stroke order for CJK characters (used in Chinese) not supported. ▪ traditional — For a traditional-style ordering (such as in Spanish). ▪ unihan — Pinyin ordering for Latin, Unihan radical-stroke ordering for CJK characters (used in Chinese) not supported. ▪ binary — the Vertica default, providing UTF-8 octet ordering, compatible with Vertica 3.5.
Keyword	Short form	Description					
collation	K	<p>If present, the collation keyword modifies how the collation service searches through the locale data when instantiating a collator. Collation supports the following values:</p> <ul style="list-style-type: none"> ▪ big5han — Pinyin ordering for Latin, big5 charset ordering for CJK characters (used in Chinese). ▪ dict — For a dictionary-style ordering (such as in Sinhala). ▪ direct — Hindi variant. ▪ gb2312/gb2312han — Pinyin ordering for Latin, gb2312han charset ordering for CJK characters (used in Chinese). ▪ phonebook — For a phonebook-style ordering (such as in German). ▪ pinyin — Pinyin ordering for Latin and for CJK characters; that is, an ordering for CJK characters based on a character-by-character transliteration into a pinyin (used in Chinese). ▪ reformed — Reformed collation (such as in Swedish). ▪ standard — The default ordering for each language. For root it is [UCA] order; for each other locale it is the same as UCA (Unicode Collation Algorithm http://unicode.org/reports/tr10/) ordering except for appropriate modifications to certain characters for that language. The following are additional choices for certain locales; they have effect only in certain locales. ▪ stroke — Pinyin ordering for Latin, stroke order for CJK characters (used in Chinese) not supported. ▪ traditional — For a traditional-style ordering (such as in Spanish). ▪ unihan — Pinyin ordering for Latin, Unihan radical-stroke ordering for CJK characters (used in Chinese) not supported. ▪ binary — the Vertica default, providing UTF-8 octet ordering, compatible with Vertica 3.5. 					

	<ul style="list-style-type: none">▪
--	---

Collation Keyword Parameters

The following parameters support the collation keyword:

Parameter	Short form	Description
colstrength	S	Sets the default strength for comparison. This feature is locale dependant. Values can be any of the following:

		<ul style="list-style-type: none"> ▪ 1 (primary) — Ignores case and accents. Only primary differences are used during comparison. For example, "a" versus "z". ▪ 2 (secondary) — Ignores case. Only secondary and above differences are considered for comparison. For example, different accented forms of the same base letter ("a" versus "\u00E4"). ▪ 3 (tertiary) — Is the default. Only tertiary and above differences are considered for comparison. Tertiary comparisons are typically used to evaluate case differences. For example "Z" versus "z". ▪ 4 (quarternary) — Used with Hiragana, for example. ▪ 5 (identical) — All differences are considered significant during comparison.
colAlternate	A	<p>Sets alternate handling for variable weights, as described in UCA.</p> <p>Values can be any of the following:</p> <ul style="list-style-type: none"> ▪ Non-ignorable (short form N or D) ▪ Shifted (short form S)
colBackwards	F	<p>For Latin with accents, this parameter determines which accents are sorted. It sets the comparison for the second level to be backwards.</p> <p>Note: colBackwards is automatically set for French accents.</p> <p>If on (short form O), then the normal UCA algorithm is used. If off (short form X), then all strings that are in Fast C or D Normalization Form (FCD http://unicode.org/notes/tn5/#FCD-Test) sort correctly, but others do not necessarily sort correctly. So it should only be set off if the the strings to be compared are in FCD.</p>
colNormalization	N	<p>If on (short form O), then the normal UCA algorithm is used. If off (short form X), all strings that are in [FCD] sort correctly, but others won't necessarily sort correctly. So it should only be set off if the strings to be compared are in FCD.</p>
colCaseLevel	E	<p>If set to on (short form O), a level consisting only of case characteristics is inserted in front of tertiary level. To ignore accents but take cases into account, set strength to primary and case level to on. If set to off (short form X), this level is omitted.</p>
colCaseFirst	C	<p>If set to upper (short form U), causes upper case to sort before lower case. If set to lower (short form L), lower case sorts before upper case. This is useful for locales that have already supported ordering but require different</p>

		order of cases. It affects case and tertiary levels. If set to off (short form X), tertiary weights are not affected.
colHiraganaQuarternary	H	Controls special treatment of Hiragana code points on quaternary level. If turned on (short form O), Hiragana codepoints get lower values than all the other non-variable code points. The strength must be greater or equal than quaternary for this attribute to take effect. If turned off (short form X), Hiragana letters are treated normally.
colNumeric	D	If set to on, any sequence of Decimal Digits (General_Category = Nd in the [UCD]) is sorted at a primary level with its numeric value. For example, "A-21" < "A-123".
variableTop	B	If set to on, any sequence of Decimal Digits (General_Category = Nd in the [UCD]) is sorted at a primary level with its numeric value. For example, "A-21" < "A-123".

Notes

- Locale specification strings are case insensitive. The following are all equivalent: en_us, EN_US, and En_uS.
- You can substitute underscores with hyphens. For example: [-script]
- The ICU library works by adding options, such as S=1 separately after the long-form locale has been accepted. Vertica has extended its long-form processing to accept options as keywords, as suggested by the Unicode Consortium.
- Collations may default to root, the ICU default collation.
- Incorrect locale strings are accepted if the prefix can be resolved to a known locale version. For example, the following works because the language can be resolved:


```
\locale en_XX
INFO:  Locale: 'en_XX'
INFO:   English (XX)
INFO:  Short form: 'LEN'
```
- The following does not work because the language cannot be resolved:


```
\locale xx_XX
xx_XX: invalid locale identifier
```
- Invalid values of the collation keyword and its synonyms do not cause an error. For example, the following does not generate an error. It simply ignores the invalid value:


```
\locale en_GB@collation=xyz
INFO:  Locale: 'en_GB@collation=xyz'
INFO:   English (United Kingdom, collation=xyz)
INFO:  Short form: 'LEN'
```
- POSIX-type locales, such as en_US.UTF-8 work to some extent in that the encoding part "UTF-8" is ignored.

- Vertica 4.0 uses the icu4c-4_2_1 library to support basic locale/collation processing with some extensions as noted here. This does not yet meet the current standard for locale processing, <http://tools.ietf.org/html/rfc5646>.
- To learn more about collation options, consult http://www.unicode.org/reports/tr35/#Locale_Extension_Key_and_Type_Data.

Examples

The following specifies a locale for english as used in the United States:

```
en_US
```

The following specifies a locale for english as used in the United Kingdom:

```
en_GB
```

The following specifies a locale for German as used in Deutschland and uses phonebook-style collation.

```
\locale de_DE@collation=phonebook
INFO:  Locale: 'de_DE@collation=phonebook'
INFO:    German (Germany, collation=Phonebook Sort Order)
INFO:    Deutsch (Deutschland, Sortierung=Telefonbuch-Sortierregeln)
INFO:  Short form: 'KPHONEBOOK_LDE'
```

The following specifies a locale for German as used in Deutschland. It uses phonebook-style collation with a strength of secondary.

```
\locale de_DE@collation=phonebook;colStrength=secondary
INFO:  Locale: 'de_DE@collation=phonebook'
INFO:    German (Germany, collation=Phonebook Sort Order)
INFO:    Deutsch (Deutschland, Sortierung=Telefonbuch-Sortierregeln)
INFO:  Short form: 'KPHONEBOOK_LDE_S2'
```

Short Form

Vertica accepts locales in short form. You can use the short form to specify the locale and keyname pair/value names.

Determining the Short Form of a Locale

To determine the short form for a locale, type in the long form and view the last line of INFO, as follows:

```
\locale fr
INFO:  Locale: 'fr'
INFO:    French
INFO:    franÅ§ais
INFO:  Short form: 'LFR'
```

Specifying a Short Form Locale

The following example specifies the *en* (English) locale:

```
\locale LEN
INFO:  Locale: 'en'
INFO:  English
```

INFO: Short form: 'LEN'

The following example specifies a locale for German as used in Deutschland, and it uses phonebook-style collation.

```
\locale LDE_KPHONEBOOK
INFO: Locale: 'de@collation=phonebook'
INFO: German (collation=Phonebook Sort Order)
INFO: Deutsch (Sortierung=Telefonbuch-Sortierregeln)
INFO: Short form: 'KPHONEBOOK_LDE'
```

The following example specifies a locale for German as used in Deutschland. It uses phonebook-style collation with a strength of `secondary` (see "Collation Keyword Parameters" in **Long Form** (page 397)).

```
\locale LDE_KPHONEBOOK_S2
INFO: Locale: 'de@collation=phonebook'
INFO: German (collation=Phonebook Sort Order)
INFO: Deutsch (Sortierung=Telefonbuch-Sortierregeln)
INFO: Short form: 'KPHONEBOOK_LDE_S2'
```

Supported Locales

The following are the supported locale strings for Vertica. Each locale can optionally have a list of key/value pairs (see **Long Form** (page 397)).

Locale Name	Language or Variant	Region
af	Afrikaans	
af_NA	Afrikaans	Namibian Afrikaans
af_ZA	Afrikaans	South Africa
am	Ethiopic	
am_ET	Ethiopic	Ethiopia
ar	Arabic	
ar_AE	Arabic	United Arab Emirates
ar_BH	Arabic	Bahrain
ar_DZ	Arabic	Algeria
ar_EG	Arabic	Egypt
ar_IQ	Arabic	Iraq
ar_JO	Arabic	Jordan
ar_KW	Arabic	Kuwait
ar_LB	Arabic	Lebanon
ar_LY	Arabic	Libya
ar_MA	Arabic	Morocco

ar_OM	Arabic	Oman
ar_QA	Arabic	Qatar
ar_SA	Arabic	Saudi Arabia
ar_SD	Arabic	Sudan
ar_SY	Arabic	Syria
ar_TN	Arabic	Tunisia
ar_YE	Arabic	Yemen
as	Assamese	
as_IN	Assamese	India
az	Azerbaijani	
az_Cyrl	Azerbaijani	Cyrillic
az_Cyrl_AZ	Azerbaijani	Azerbaijan Cyrillic
az_Latn	Azerbaijani	Latin
az_Latn_AZ	Azerbaijani	Azerbaijan Latin
be	Belarusian	
be_BY	Belarusian	Belarus
bg	Bulgarian	
bg_BG	Bulgarian	Bulgaria
bn	Bengali	
bn_BD	Bengali	Bangladesh
bn_IN	Bengali	India
bo	Tibetan	
bo_CN	Tibetan	PR China
bo_IN	Tibetan	India
ca	Catalan	
ca_ES	Catalan	Spain
cs	Czech	
cs_CZ	Czech	Czech Republic
cy	Welsh	
cy_GB	Welsh	United Kingdom
da	Danish	
da_DK	Danish	Denmark
de	German	
de_AT	German	Austria

de_BE	German	Belgium
de_CH	German	Switzerland
de_DE	German	Germany
de_LI	German	Liechtenstein
de_LU	German	Luxembourg
el	Greek	
el_CY	Greek	Cyprus
el_GR	Greek	Greece
en	English	
en_AU	English	Australia
en_BE	English	Belgium
en_BW	English	Botswana
en_BZ	English	Belize
en_CA	English	Canada
en_GB	English	United Kingdom
en_HK	English	Hong Kong S.A.R. of China
en_IE	English	Ireland
en_IN	English	India
en_JM	English	Jamaica
en_MH	English	Marshall Islands
en_MT	English	Malta
en_NA	English	Namibia
en_NZ	English	New Zealand
en_PH	English	Philippines
en_PK	English	Pakistan
en_SG	English	Singapore
en_TT	English	Trinidad and Tobago
en_US_POSIX	English	United States Posix
en_VI	English	U.S. Virgin Islands
en_ZA	English	Zimbabwe or South Africa
en_ZW	English	Zimbabwe
eo	Esperanto	
es	Spanish	

es_AR	Spanish	Argentina
es_BO	Spanish	Bolivia
es_CL	Spanish	Chile
es_CO	Spanish	Columbia
es_CR	Spanish	Costa Rica
es_DO	Spanish	Dominican Republic
es_EC	Spanish	Ecuador
es_ES	Spanish	Spain
es_GT	Spanish	Guatemala
es_HN	Spanish	Honduras
es_MX	Spanish	Mexico
es_NI	Spanish	Nicaragua
es_PA	Spanish	Panama
es_PE	Spanish	Peru
es_PR	Spanish	Puerto Rico
es_PY	Spanish	Paraguay
es_SV	Spanish	El Salvador
es_US	Spanish	United States
es_UY	Spanish	Uruguay
es_VE	Spanish	Venezuela
et	Estonian	
et_EE	Estonian	Estonia
eu	Basque	Spain
eu_ES	Basque	Spain
fa	Persian	
fa_AF	Persian	Afghanistan
fa_IR	Persian	Iran
fi	Finnish	
fi_FI	Finnish	Finland
fo	Faroese	
fo_FO	Faroese	Faroe Islands
fr	French	
fr_BE	French	Belgium

fr_CA	French	Canada
fr_CH	French	Switzerland
fr_FR	French	France
fr_LU	French	Luxembourg
fr_MC	French	Monaco
fr_SN	French	Senegal
ga	Gaelic	
ga_IE	Gaelic	Ireland
gl	Gallegan	
gl_ES	Gallegan	Spain
gsw	German	
gsw_CH	German	Switzerland
gu	Gujurati	
gu_IN	Gujurati	India
gv	Manx	
gv_GB	Manx	United Kingdom
ha	Hausa	
ha_Latn	Hausa	Latin
ha_Latn_GH	Hausa	Ghana (Latin)
ha_Latn_NE	Hausa	Niger (Latin)
ha_Latn_NG	Hausa	Nigeria (Latin)

haw	Hawaiian	Hawaiian
haw_US	Hawaiian	United States
he	Hebrew	
he_IL	Hebrew	Israel
hi	Hindi	
hi_IN	Hindi	India
hr	Croatian	
hr_HR	Croatian	Croatia
hu	Hungarian	
hu_HU	Hungarian	Hungary
hy	Armenian	
hy_AM	Armenian	Armenia
hy_AM_REVISED	Armenian	Revised Armenia
id	Indonesian	
id_ID	Indonesian	Indonesia
ii	Sichuan	
ii_CN	Sichuan	Yi
is	Icelandic	
is_IS	Icelandic	Iceland
it	Italian	

it_CH	Italian	Switzerland
it_IT	Italian	Italy
ja	Japanese	
ja_JP	Japanese	Japan
ka	Georgian	
ka_GE	Georgian	Georgia
kk	Kazakh	
kk_Cyrl	Kazakh	Cyrillic
kk_Cyrl_KZ	Kazakh	Kazakhstan (Cyrillic)
kl	Kalaallisut	
kl_GL	Kalaallisut	Greenland
km	Khmer	
km_KH	Khmer	Cambodia
kn	Kannada	
kn-IN	Kannada	India
ko	Korean	
ko_KR	Korean	Korea
kok	Konkani	
kok_IN	Konkani	India
kw	Cornish	

kw_GB	Cornish	United Kingdom
lt	Lithuanian	
lt_LT	Lithuanian	Lithuania
lv	Latvian	
lv_LV	Latvian	Latvia
mk	Macedonian	
mk_MK	Macedonian	Macedonia
ml	Malayalam	
ml_IN	Malayalam	India
mr	Marathi	
mr_IN	Marathi	India
ms	Malay	
ms_BN	Malay	Brunei
ms_MY	Malay	Malaysia
mt	Maltese	
mt_MT	Maltese	Malta
nb	Norwegian Bokml	
nb_NO	Norwegian Bokml	Norway
ne	Nepali	
ne_IN	Nepali	India

ne_NP	Nepali	Nepal
nl	Dutch	
nl_BE	Dutch	Belgium
nl_NL	Dutch	Netherlands
nn	Norwegian nynorsk	
nn_NO	Norwegian nynorsk	Norway
om	Oromo	
om_ET	Oromo	Ethiopia
om_KE	Oromo	Kenya
or	Oriya	
or_IN	Oriya	India
pa	Punjabi	
pa_Arab	Punjabi	Arabic
pa_Arab_PK	Punjabi	Pakistan (Arabic)
pa_Guru	Punjabi	Gurmukhi
pa_Guru_IN	Punjabi	India (Gurmukhi)
pl	Polish	
pl_PL	Polish	Poland
ps	Pashto	
ps_AF	Pashto	Afghanistan

pt	Portuguese	
pt_BR	Portuguese	Brazil
pt_PT	Portuguese	Portugal
ro	Romanian	
ro_MD	Romanian	Moldavia
ro_RO	Romanian	Romania
ru	Russian	
ru_RU	Russian	Russia
ru_UA	Russian	Ukraine
si	Sinhala	
si_LK	Sinhala	Sri Lanka
sk	Slovak	
sk_SK	Slovak	Slovakia
sl	Slovenian	
sl_SL	Slovenian	Slovenia
so	Somali	
so_DJ	Somali	Djibouti
so_ET	Somali	Ethiopia
so_KE	Somali	Kenya
so_SO	Somali	Somalia

sq	Albanian	
sq_AL	Albanian	Albania
sr	Serbian	
sr_Cyrl	Serbian	Cyrillic
sr_Cyrl_BA	Serbian	Bosnia and Herzegovina (Cyrillic)
sr_Cyrl_ME	Serbian	Montenegro (Cyrillic)
sr_Cyrl_RS	Serbian	Serbia (Cyrillic)
sr_Latn	Serbian	Latin
sr_Latn_BA	Serbian	Bosnia and Herzegovina (Latin)
sr_Latn_ME	Serbian	Montenegro (Latin)
sr_Latn_RS	Serbian	Serbia (Latin)
sv	Swedish	
sv_FI	Swedish	Finland
sv_SE	Swedish	Sweden
sw	Swahili	
sw_KE	Swahili	Kenya
sw_TZ	Swahili	Tanzania
ta	Tamil	
ta_IN	Tamil	India
te	Telugu	

te_IN	Telugu	India
th	Thai	
th_TH	Thai	Thailand
ti	Tigrinya	
ti_ER	Tigrinya	Eritrea
ti_ET	Tigrinya	Ethiopia
tr	Turkish	
tr_TR	Turkish	Turkey
uk	Ukrainian	
uk_UA	Ukrainian	Ukraine
ur	Urdu	
ur_IN	Urdu	India
ur_PK	Urdu	Pakistan
uz	Uzbek	
uz_Arab	Uzbek	Arabic
uz_Arab_AF	Uzbek	Afghanistan (Arabic)
uz_Cryl	Uzbek	Cyrillic
uz_Cryl_UZ	Uzbek	Uzbekistan (Cyrillic)
uz_Latin	Uzbek	Latin
us_Latin_UZ		Uzbekistan (Latin)

vi	Vietnamese	
vi_VN	Vietnamese	Vietnam
zh	Chinese	
zh_Hans	Chinese	Simplified Han
zh_Hans_CN	Chinese	China (Simplified Han)
zh_Hans_HK	Chinese	Hong Kong SAR China (Simplified Han)
zh_Hans_MO	Chinese	Macao SAR China (Simplified Han)
zh_Hans_SG	Chinese	Singapore (Simplified Han)
zh_Hant	Chinese	Traditional Han
zh_Hant_HK	Chinese	Hong Kong SAR China (Traditional Han)
zh_Hant_MO	Chinese	Macao SAR China (Traditional Han)
zh_Hant_TW	Chinese	Taiwan (Traditional Han)
zu	Zulu	
zu_ZA	Zulu	South Africa

Locale Restrictions and Workarounds

The following list contains the known restrictions for locales for international data sets.

Session related:

- The locale setting is session scoped and applies to queries only (no DML/DDDL) run in that session. You cannot specify a locale for an individual query.
- The default locale for new sessions can be set using a configuration parameter

Query related:

The following restrictions apply when queries are run with locale other than the default `en_US@collation=binary`:

- Multicolumn NOT IN subqueries are not supported when one or more of the left-side NOT IN columns is of CHAR or VARCHAR data type. For example:
=> CREATE TABLE test (x VARCHAR(10), y INT);
=> SELECT ... FROM test WHERE (x,y) NOT IN (SELECT ...);
ERROR: Multi-expression NOT IN subquery is not supported because a left hand expression could be NULL

Note: An error is reported even if columns `test.x` and `test.y` have a "NOT NULL" constraint.

- Correlated HAVING clause subqueries are not supported if the outer query contains a GROUP BY on a CHAR or a VARCHAR column. In the following example, the GROUP BY x in the outer query causes the error:
=> DROP TABLE test CASCADE;
=> CREATE TABLE test (x VARCHAR(10));
=> SELECT COUNT(*) FROM test t GROUP BY x HAVING x IN (SELECT x FROM test WHERE t.x||'a' = test.x||'a');
ERROR: subquery uses ungrouped column "t.x" from outer query
- Subqueries that use analytic functions in the HAVING clause are not supported. For example:
=> DROP TABLE test CASCADE;
=> CREATE TABLE test (x VARCHAR(10));
=> SELECT MAX(x)OVER(PARTITION BY 1 ORDER BY 1)
FROM test GROUP BY x HAVING x IN (
SELECT MAX(x) FROM test);
ERROR: Analytics query with having clause expression that involves aggregates
and subquery is not supported
- The operators LIKE/ILIKE do not currently respect UTF-8 character boundaries. Therefore, expressions such as 'SS' LIKE 'ß' and 'SS' ILIKE 'ß' always return false even in locales where 'SS' = 'ß' return true.

DML/DDDL related:

- SQL identifiers (such as table names, column names, and so on) are restricted to ASCII characters. For example, the following CREATE TABLE statement fails because it uses the non-ASCII ß in the table name:
=> CREATE TABLE straÙe(x int, y int);
ERROR: Non-ASCII characters are not supported in names
- Projection sort orders are made according to the default `en_US@collation=binary` collation. Thus, regardless of the session setting, issuing the following command creates a projection sorted by `coll1` according to the binary collation:
=> CREATE PROJECTION p1 AS SELECT * FROM table1 ORDER BY coll1;
Note that in such cases, `straÙe` and `strasse` would not be near each other on disk.

Sorting by binary collation also means that sort optimizations do not work in locales other than binary. Vertica returns the following warning if you create tables or projections in a non-binary locale:

```
WARNING: Projections are always created and persisted in the default Vertica locale. The current locale is de_DE
```

- When creating pre-join projections, the projection definition query does not respect the locale or collation setting. This means that when you insert data into the fact table of a pre-join projection, referential integrity checks are not locale or collation aware.

For example:

```
\locale LDE_S1      -- German
=> CREATE TABLE dim (coll varchar(20) primary key);
=> CREATE TABLE fact (coll varchar(20) references dim(coll));
=> CREATE PROJECTION pj AS SELECT * FROM fact JOIN dim ON fact.coll =
    dim.coll UNSEGMENTED ALL NODES;
=> INSERT INTO dim VALUES ('ß');
=> COMMIT;
```

The following INSERT statement fails with a "nonexistent FK" error even though 'ß' is in the dim table, and in the German locale 'SS' and 'ß' refer to the same character.

```
=> INSERT INTO fact VALUES ('SS');
    ERROR: Nonexistent foreign key value detected in FK-PK join (fact
    x dim)
    using subquery and dim_node0001; value SS
=> => ROLLBACK;
=> DROP TABLE dim, fact CASCADE;
```

- When the locale is non-binary, the collation function is used to transform the input to a binary string which sorts in the proper order.

This transformation increases the number of bytes required for the input according to this formula:

$$\text{result_column_width} = \text{input_octet_width} * \text{CollationExpansion} + 4$$

CollationExpansion defaults to 5 and should be changed only under the supervision of Vertica **Technical Support** (on page 1).

- CHAR fields are displayed as fixed length, including any trailing spaces. When CHAR fields are processed internally, they are first stripped of trailing spaces. For VARCHAR fields, trailing spaces are usually treated as significant characters; however, trailing spaces are ignored when sorting or comparing either type of character string field using a non-BINARY locale.

Index

!

! [COMMAND] • 375, 378, 399

.

... large moveout is in progress • 241

... users are connected • 242, 322

?

? • 375

? --help • 370

A

a • 377, 390, 400

a --echo-all • 370, 394

A --no-align • 370, 371, 377, 400

About Load Errors • 15, 160

About the Documentation • 2

About the gmond.conf file • 231

Account Locking • 118

Acrobat • 6

Active data partitions • 169

ActivePartitionCount • 26, 169

Add a cron job • 236

Add Node to a Database • 107, 111, 285

Adding Constraints • 49

Adding Disk Space Across the Cluster • 289, 290

Adding Disk Space to a Node • 289

Adding Hosts to a Cluster • 268, 270, 281, 283

Adding Hosts to a Database • 268, 272, 273, 275, 357

Adding Nodes • 267, 268, 281, 357

Adding Storage Locations • 290, 291, 292, 293

Administration Overview • 10

Administration Tools Reference • 11, 239, 270, 336, 343

Administration Tools shows node state as UNKNOWN • 250

Adobe Acrobat • 6

AdvanceAHMInterval • 29

Advanced Formats for Loading Data • 151, 160

Advanced Menu Options • 246, 344, 353

AdvanceEpochInterval • 29

Altering Sequences • 64

Altering Tables • 77

AnalyzeRowCountInterval • 25, 175

Analyzing Constraints (Detecting Constraint Violations) • 51, 55, 56, 149

Anatomy of a Projection • 97

Appendix

Locales • 20, 406

Authenticating Using LDAP or Kerberos • 124

Authentication • 115, 119, 120, 121, 123, 124, 125, 126, 129

Auto Partitioning • 190, 195

AUTOCOMMIT • 393

B

b • 377

Backing Up the Database • 255, 257, 262, 268, 277, 282, 283

Backup • 255, 257, 259

Backup and Restore • 256

Basic Principles for Scalability and Concurrency Tuning • 307

Best Practices for DELETE and UPDATE • 180

Best Practices for Disaster Recovery • 265

Best Practices for Statistics Collection • 176

Best Practices for Working with Locales • 21, 367

Best Practices for Workload Management • 151, 173, 307

Binary data • 153, 160, 161

Binary format file header • 161

Binary format records • 161

Binary native format • 153, 160, 161

Binary varchar format • 153, 160, 161

Bold text • 7

border • 388

Braces • 7

Brackets • 7

Bulk Deleting and Purging Data • 179

Bulk Loading • 15, 146, 177

C

c (or \connect) [dbname [username]] • 374, 377

C [STRING] • 377, 390

c command --command command • 370, 372, 401

Catalog and Data Files • 291

cd [DIR] • 378

Change Transaction Isolation Levels • 24, 26, 150

Character data • 152

check_spread • 359

Choosing Sort-orders for High Cardinality Predicates • 99, 104

Choosing Sort-orders for Low Cardinality Predicates • 99, 103, 181

- CleanQueryRepoInterval • 33
- Client authentication • 115, 119, 120, 121, 123, 125, 126, 129
- Client connections, balancing • 324, 327, 330, 331, 333, 334
- ClientAuthentication • 34, 121, 123, 125
- Collecting Statistics • 25, 175
- Colored bold text • 7
- Command Line Editing • 398
- Command Line Options • 369, 374
- command_host • 359
- Complete the Data Load • 18
- Comprehensive Design • 78
- CompressNetworkData • 25
- config_nodes • 359
- Configuration Menu Item • 349
- Configuration parameters
 - AnalyzeRowCountInterval • 25, 175
 - CompressNetworkData • 25
 - MaxClientSessions • 25, 144, 307, 320
 - RefreshHistoryDuration • 25
- Configuration Parameters • 11, 24, 135, 176, 216, 218, 245, 321, 348, 386
- Configuration parameters, epoch management
 - AdvanceAHMInterval • 29
 - AdvanceEpochInterval • 29
 - DefaultIntervalStyle • 29
 - EpochAdvancementMode • 29
 - EpochMapInterval • 29
 - HistoryRetentionEpochs • 29, 184
 - HistoryRetentionTime • 29, 184
- Configuration parameters, internationalization
 - DefaultSessionLocale • 20, 28, 386
 - EscapeStringWarning • 28
 - StandardConformingStrings • 28
- Configuration parameters, Kerberos authentication
 - KerberosHostname • 35, 125, 126
 - KerberosKeytabFile • 35, 125
 - KerberosRealm • 35, 125, 126
 - KerberosServiceName • 35, 125, 126
- Configuration parameters, monitoring
 - SnmptTrapDestinationsList • 31, 216
 - SnmptTrapEvents • 31, 216
 - SnmptTrapsEnabled • 31, 216
 - SyslogEnabled • 31
 - SyslogEvents • 31
 - SyslogFacility • 31
- Configuration parameters, profiling
 - GlobalEEProfiling • 32
 - GlobalQueryProfiling • 32
 - GlobalSessionProfiling • 32
- Configuration parameters, query repository
 - CleanQueryRepoInterval • 33
 - QueryRepoMemoryLimit • 33
 - QueryRepoRetentionTime • 33
 - QueryRepositoryEnabled • 33
 - SaveQueryRepoInterval • 33
- Configuration parameters, security • 34
 - ClientAuthentication • 34, 121, 123, 125
 - EnableSSL • 34, 135
- Configuration parameters, Tuple Mover
 - ActivePartitionCount • 26, 169
 - EnableStrataBasedMrgOutPolicy • 26, 169
 - MaxMrgOutROSSizeMB • 26, 169
 - MergeOutInterval • 26, 169
 - MergeOutPolicySizeList • 26, 169
 - MoveOutInterval • 26, 169
 - MoveOutMaxAgeTime • 26
 - MoveOutSizePct • 26
- Configuration Procedure • 11, 147
- Configure the Vertica IPVS Load Balancer • 327
- Configuring and Starting lighttpd • 233
- Configuring Authentication Through Kerberos and GSS • 35, 125
- Configuring Disk Usage to Optimize Performance • 12, 13
- Configuring Event Reporting • 210, 214
- Configuring External Authentication Methods in vertica.conf file • 114, 119
- Configuring Ganglia • 227, 228, 229
- Configuring Gmetad on the Monitoring Node • 229, 230
- Configuring Gmond on All Nodes • 229, 231
- Configuring PROJECTION_REFRESHES History • 208
- Configuring Reporting for SNMP • 31, 32, 214, 216, 289
- Configuring Reporting for syslog • 32, 214
- Configuring SSL • 131, 132, 135
- Configuring SSL for JDBC Clients • 133, 135, 136
- Configuring SSL for ODBC Clients • 133, 135, 136
- Configuring the Database • 11
- Configuring the Directors • 324, 327, 330
- Configuring the Vertica Monitoring Package • 233

Configuring Vertica Extension Files • 234
 Configuring Vertica Nodes • 324
 connect_db • 359
 Connecting From a Non-Cluster Host • 374
 Connecting From the Administration Tools • 367, 368
 Connecting from the Command Line • 344, 367, 369
 Connecting to the Database • 344
 Connecting to the Virtual IP (VIP) • 330
 Constraints • 49, 51, 52, 53, 54, 55
 defining multicolumn constraints • 49
 defining single-column constraints • 49
 removing constraints • 54
 COPY • 146, 148, 150, 152, 155, 262, 393
 Copying a Database to Another Cluster • 238, 260, 262, 265
 Copying Data Using vsql • 402
 Copyright Notice • 441
 Create an Empty Database • 16
 Create an Optional Sample Query Script • 16
 CREATE TABLE • 14
 Create the Logical Schema • 14, 17
 create_db • 359
 Creating a Comprehensive Design Using the Database Designer • 80
 Creating a Database • 13, 201, 250, 350
 Creating a Physical Design • 11, 18, 77
 Creating a Query-specific Design Using the Database Designer • 19, 80, 88
 Creating and Configuring Storage Locations • 13, 292
 Creating Base Tables • 45
 Creating Custom Designs • 45, 47, 78, 92, 285, 318
 Creating Native-Format Files to Load Data • 160, 161
 Creating New Projections • 285
 Creating Objects that Span Multiple Schemas • 44
 Creating Records • 34, 120, 123
 Creating Schemas • 42
 Creating Sequences • 62
 Creating Tables • 44
 Creating Temporary Tables • 46
 Creating Views • 75
 CRITICAL • 201
 cron job • 236

D

d [PATTERN] • 378
 d \d <table> \df \dj \dn \dp \ds \dS \dt \dT \dtv \du \dv • 379
 d dbname --dbname dbname • 370
 Data partitions, active • 169
 Data Warehouse Schema Types • 36
 Database did not start cleanly on initiator node! • 247
 Database recovery • 238, 251, 252, 255, 262
 Database startup successful, but it could be incomplete • 246
 database_parameters • 339, 359
 db_add_node • 359
 db_remove_node • 359
 db_replace_node • 359
 db_status • 359
 dbLog • 201, 202, 222, 248
 DBNAME • 394
 DEBUG • 201
 DefaultIntervalStyle • 29
 DefaultSessionLocale • 20, 28, 386
 Defining Partitions • 186, 187, 190
 DELETE and UPDATE best practices • 180
 Deleting data • 146, 180, 181, 183
 DELIMITER • 152
 Deploying Designs • 91
 Deploying Designs Manually • 91
 Design Fundamentals • 92, 96
 Design Requirements • 92, 93
 Design Types • 78
 Design, comprehensive • 78
 Design, query-specific • 79
 Designing a Logical Schema • 11, 14, 36
 Designing for Group By Queries • 99
 Designing for K-Safety • 92, 93, 94, 98, 102, 324
 Designing for Segmentation • 94, 102
 Designing Replicated Projections for K-Safety • 94, 101
 Designing Segmented Projections for K-Safety • 94, 102
 Designing Superprojections • 98
 Determining the Number of Projections to Use • 92, 93
 Determining Where Connections Are Going • 331
 Diagnosing spread problems • 250
 Diagnostics • 1, 359

- Dimension table • 36, 37
- DIRECT • 146
- Director, master • 324, 327, 331
- Director, slave • 324, 327
- DISABLE • 201
- Disable Address Resolution Protocol (ARP) • 325
- Disaster recovery best practices • 265
- Disk Space Requirements for Vertica • 12
- Disk space, managing • 186, 187, 189, 191, 289, 290, 291, 292, 293, 294, 295, 296, 297
- Distributed Sequences • 65
- Distributing Certifications and Keys • 132, 133, 135
- Distributing Changes Made to the Administration Tools Metadata • 341
- Distributing Configuration Files to the New Host • 268, 272, 282, 283, 291
- dj [PATTERN] • 380
- dn [PATTERN] • 381
- Documentation • 6
- dp [PATTERN] • 381, 392
- drop_db • 359
- drop_node • 359
- Dropping a Database • 351
- Dropping Partitions • 187, 189, 296, 297
- Dropping Projections • 91, 97, 267, 268, 275, 277, 285, 287
- Dropping Sequences • 74
- Dropping Storage Locations • 292, 296
- ds [PATTERN] • 381
- dS [PATTERN] • 382
- dt [PATTERN] • 382
- dT [PATTERN] • 383
- dtv [PATTERN] • 383
- du [PATTERN] • 384
- dv [PATTERN] • 384

E

- E • 370, 394
- e --echo-queries • 370, 394
- e \edit [FILE] • 385
- ECHO • 370, 386, 394
- echo [STRING] • 385, 390
- ECHO_HIDDEN • 394
- Edit the gmond.conf file • 232, 233
- Edit the vertica-dashboard.xml file • 234
- Edit the verticadefines.php file [Optional] • 235
- edit_auth • 120, 339, 359
- Ellipses • 7

- EnableSSL • 34, 135
- EnableStrataBasedMrgOutPolicy • 26, 169
- ENCODING • 394
- Enforcing Constraints • 15, 51, 55
- Environment • 399
- Epoch Management Parameters • 29, 184, 185, 240
- EpochAdvancementMode • 29
- EpochMapInterval • 29
- Error starting database, no nodes are up • 246
- EscapeStringWarning • 28
- Event monitoring • 210, 214, 216, 218
- Event Reporting Examples • 216, 218
- events, monitoring • 210, 214, 216, 218
- events, reporting • 214, 216, 218
- Example Records • 123
- Expanded • 388, 392
- Exporting a Catalog • 238, 255
- Exporting Data Using vsql • 400, 402
- External Procedure Privileges • 114, 140

F

- f [string] • 371, 385
- f filename --file filename • 370, 395
- F separator --field-separator separator • 371, 401
- Fact table • 36, 37
 - file command • 152
- Failure Recovery • 17, 30, 144, 211, 238, 239, 260, 268, 343
- fieldsep • 388
- File header, binary format specification • 161
- Files • 400
- Fixing Constraint Violations • 58
- footer • 388
- FOREIGN KEY • 52, 55
- FOREIGN KEY Constraints • 49, 52
- format • 388
- Formatting Rules for Records • 120, 123

G

- g • 385, 388
- Ganglia

- Architecture • 222
- Configuring • 229, 233
- Dependencies • 223, 224
- Installing • 225, 226, 228
- Requirements • 223, 224
- Uninstalling • 237
- Upgrading • 236
- Ganglia Architecture • 222
- Ganglia Prerequisites • 223, 226, 227, 228
- General Parameters • 25, 144, 309
- Generating a Rebalance Script for Later Use • 273, 276
- Generating Certifications and Keys • 133, 137
- GlobalEEProfiling • 32
- GlobalQueryProfiling • 32
- GlobalSessionProfiling • 32
- Gmetad • 222, 225, 226, 228, 230
- Gmond • 222, 225, 226, 228, 230, 231, 232, 233
- gmond configurationfile • 232
- Good epoch logs are available on all nodes • 247
- Guidelines for Setting Pool Parameters • 300, 307, 319

H

- H • 372, 386, 390
- h \help [command] • 386
- h hostname --host hostname • 121, 122, 372
- H --html • 372
- Hash segmentation • 65, 94, 102, 186, 191
- HISTCONTROL • 395
- HistoryRetentionEpochs • 29, 184
- HistoryRetentionTime • 29, 184
- HISTSIZe • 395
- HOST • 395
- host_to_node • 359
- How Statistics are Computed • 176
- HTML • 6

I

- i FILE • 17, 19, 91, 96, 370, 385, 386
- IGNOREEOF • 395
- Ignoring Columns and Fields in the Load File • 159
- Implement Locales for International Data Sets • 11, 19, 406
- Implement Security • 11, 19
- Implementing Client Authentication • 34, 115, 344
- Implementing Database Authorization • 137

- Implementing Security • 19, 113, 341
- Implementing SSL • 34, 114, 115, 116, 131
- Implementing Views • 74
- Importing and Exporting Statistics • 177
- Indentation • 7
- INFO • 201
- INSERT • 146
- Install the Vertica IPVS Load Balancer Package • 327
- install_node • 359
- install_procedure • 359
- Installing Gmond on All Nodes • 229, 230
- Installing the Vertica Monitoring Package • 224, 225, 230, 237
- Internationalization Parameters • 28
- IP Virtual Server • 327, 331, 334
- IP Virtual Server (IPVS) • 324, 327, 330, 331, 333, 334
- IPVS • 327, 331, 334
- Italic text • 7

K

- Kerberos Authentication Parameters • 35
- Kerberos Client Code Example Written in C • 125, 129
- Kerberos Client Code Example Written in Java • 125, 126
- KerberosHostname • 35, 125, 126
- KerberosKeytabFile • 35, 125
- KerberosRealm • 35, 125, 126
- KerberosServiceName • 35, 125, 126
- kill_host • 359
- Killing a Vertica Process on Host • 241, 356
- K-Safety Support in Administration Tools • 339

L

- l • 386
- l --list • 372
- Large Tuple Mover operations, scheduling • 169
- License key • 142, 357
- lighttpd • 222, 233
- Linux resource usage • 220
- list_allnodes • 359
- list_db • 234, 359
- list_host • 230, 359
- list_node • 359
- Load balancing • 324, 327, 330, 331, 333, 334
- Load Balancing • 323
- Loading and Modifying Data • 15, 146, 186

Loading data • 15, 19, 146, 148, 150, 152, 153, 155, 159, 160, 167, 169
Loading Data into Binary Data Types • 153, 160
Loading Data into Character Data Types • 152
Loading Data into Pre-join Projections • 15, 155
Loading Sequences • 74
locale • 20, 386, 400
Locale best practices • 21
Locale Restrictions and Workarounds • 427
Locale Specification • 408
Locales • 399
Log files • 201, 202, 222
Logical schema • 14, 36
logrotate • 202, 359
Long Form • 408, 414

M

Managing Clusters • 357
Managing Disk Space • 210, 289, 319
Managing Load Streams • 322
Managing Nodes • 12, 267
Managing Partitions • 193
Managing Sessions • 25, 241, 242, 245, 319, 320, 345, 348
Managing System Resource Usage • 319
Managing Workloads • 171, 298, 316
Managing Your License Key • 16, 142, 357
Manual Tuper Mover operations • 173
Manually Purging Data • 184, 185
Master director • 324, 327, 331
Master node • 324, 327
MaxClientSessions • 25, 144, 307, 320
Maximizing Projection Performance • 103
MaxMrgOutROSSizeMB • 26, 169
Measuring Location Performance • 292, 293, 294
Mergeout • 167, 169
MergeOutInterval • 26, 169
MergeOutPolicySizeList • 26, 169
Meta-Commands • 369, 375
Metadata Privileges • 114, 138, 140, 320
Modifying Database Designs for Updated Nodes • 275, 277, 285
Modifying Records • 123
Modifying Storage Locations • 292, 293, 295
Monitoring Events • 210, 217
Monitoring Linux Resource Usage • 18, 220, 322
Monitoring Parameters • 31
Monitoring Processes • 209
Monitoring Recovery • 238, 255

Monitoring Resource Pools and Resource Usage by Queries • 303
Monitoring the Database • 201, 255, 345
Monitoring the Log Files • 201, 203, 210, 248, 354
Monitoring Vertica Using Ganglia • 144, 222
Monitoring Which Nodes Are Connected • 330
Monospace text • 7
Moveout • 167, 168
MoveOutInterval • 26, 169
MoveOutMaxAgeTime • 26
MoveOutSizePct • 26
Multicast IP Support • 233
Multiple Schema Examples • 39

N

n • 372
Native (binary) format • 153, 160, 161
Native (Binary) Format • 160
Native varchar format • 160, 161
Native Varchar Format • 161
New K-Safe=2 Database • 107, 108
No good epoch log available on node • 248
No running statement, that session is idle • 245
Node does not recover because of lock timeouts • 249
Nodes stuck in INITIALIZING state • 248
Nodes, managing • 268, 270, 272
Nodes, removing • 277, 278, 279
Nodes, replacing • 281, 284
NOT NULL • 54
NOT NULL Constraints • 49, 54
Notes for Remote Terminal Users • 16, 339, 340
Notes for Windows Users • 403

O

o • 372, 385, 386, 387, 390, 401
o filename --output filename • 372, 401
ON_ERROR_STOP • 395
Operating the Database • 142
Optimize Query Performance • 18
Optimizing Deletes and Updates for Performance • 180, 181
Output Formatting Examples • 390, 403
Override the Default Locale for a Session • 20, 21

P

p • 385, 388, 390

- P assignment --pset assignment • 372
 - p port --port port • 372
 - pager • 388
 - Partition Elimination • 193, 195, 198
 - Partitioning • 186, 187, 189, 191
 - Partitioning and Data Storage • 192
 - Partitioning and Segmenting Data • 187, 191
 - Partitioning best practices • 187
 - Partitioning Tables • 179, 181, 184, 186
 - password [USER] • 116, 388
 - Password Authentication • 114, 116, 330, 341
 - Password Expiration • 117
 - Password Guidelines • 118, 119
 - PDF • 6
 - Perform a Partial Data Load • 15, 17
 - Performance best practices • 176
 - Performance Considerations for Deletes and Updates • 180
 - Performing the Initial Database Load • 146
 - Performing Tuple Mover Operations Manually • 167, 173
 - Planning Your Design • 92, 96
 - PORT • 395
 - Preface • 9
 - Prepare Data Files • 14, 18
 - Prepare Disk Storage Locations • 12, 292, 297
 - Prepare Load Scripts • 15, 18
 - Prepare the Logical Schema Script • 14, 17
 - PRIMARY KEY • 51, 55
 - PRIMARY KEY Constraints • 49, 51
 - Printing Full Books • 4
 - Prioritizing Column Access Speed • 105
 - Privileges • 139, 140
 - Process monitoring • 209
 - Profiles • 116
 - Profiling Parameters • 32
 - Projection Examples • 107
 - Projection Privileges • 114, 138, 139, 140
 - PROMPT1 PROMPT2 PROMPT3 • 395
 - Prompting • 395, 397
 - pset NAME [VALUE] • 371, 372, 373, 377, 385, 386, 388, 391, 392, 399, 401
 - PSTACK • 220
 - Purging Deleted Data • 179, 183, 297
- Q**
- q • 17, 390, 396
 - q --quiet • 372
 - qecho [STRING] • 385, 388, 390
 - Query Queue/Rejection Process • 301
 - Query Repository Parameters • 33
 - Querying Case-sensitive data in System Tables • 209
 - QueryRepoMemoryLimit • 33
 - QueryRepoRetentionTime • 33
 - QueryRepositoryEnabled • 33
 - Query-specific Design • 79
 - QUIET • 372, 396
- R**
- r • 390
 - R separator --record-separator separator • 373
 - Range segmentation • 94, 186
 - Reading the Online Documentation • 2
 - Real IP server • 324, 334
 - Rebalancing Data Across Nodes • 268, 272, 273, 285
 - Rebalancing Data Using the Administration Tools UI • 273, 275
 - Rebuilding a Table • 184, 297
 - Reclaiming Disk Space • 192, 297
 - Record Content • 120, 121, 124
 - Record header, binary format specification • 161
 - recordsep • 388
 - Recovering the Cluster from a Backup • 238, 240, 255
 - Recovering the Database • 238
 - Reducing Run-time of Queries • 318
 - Reenabling error reporting • 60
 - Referencing Objects When Multiple Schemas are Used • 42
 - Refreshing Projections • 91, 97, 177, 267, 275, 285, 286
 - Removing Constraints • 54
 - Removing Hosts from a Cluster • 277, 279, 281, 283
 - Removing Hosts from a Database • 277, 278, 357
 - Removing Nodes • 267, 277, 278, 281, 357
 - Removing Statistics • 178
 - Replacing Failed Disks • 238, 289, 291
 - Replacing Hosts • 283, 284, 357
 - Replacing Nodes • 259, 267, 281, 357
 - Required Packages • 223, 224, 226, 227, 228, 229
 - Resource Manager Impact on Query Execution • 299
 - Resource Pool Architecture • 171, 300, 317
 - Resource pool settings • 169, 298, 305, 307
 - Resource Tracking in a Pool • 301

Resource usage • 220
restart_db • 359
restart_node • 352, 359
Restarting the Database • 238, 240, 252, 272, 291
Restarting Vertica on a Host • 238, 251
Restarting Vertica on Host • 144, 239, 345, 349
Restore • 259, 260
Restoring Retired Storage Locations • 292, 296
Restoring the Database • 259, 260
Restoring the Database from a Backup • 255, 259
Restoring to the Same Cluster • 259
Retiring Storage Locations • 292, 296
return_epoch • 359
RHEL5 • 225, 226
RIP • 327
Rolling Back Database to the Last Good Epoch • 354
Rotating the Log Files • 201, 202
run_designer • 359

S

s [FILE] • 390
S --single-line • 373, 396
s --single-step • 373, 396
SaveQueryRepoInterval • 33
Scenario
 Continuous Load and Query • 316
 Handling Mixed Workloads (Batch vs. Interactive) • 314, 315, 316
 Periodic Batch Loads • 308, 309
 Preventing Run-away Queries • 311, 312
 Restricting Resource Usage of Ad-hoc Query Application • 308, 312, 315, 316
 Restricting Vertica to Take Only 60% of Memory • 317
 Setting a Hard Limit on Concurrency For An Application • 313
 Setting Priorities on Queries Issued by Different Users • 308, 315
 The CEO Query • 310, 314, 316
 Tuning for Recovery • 317
 Tuning for Refresh • 317
 Tuning Tuple Mover Pool Settings • 318
Scheduling large Tuple Mover operations • 169
Schema Privileges • 41, 114, 138, 140
Script • 14, 148, 150
Security Parameters • 34
Segmentation • 92, 94
Sequence Privileges • 61, 140

Servers without Internet Access • 225, 226, 228
set [NAME [VALUE [...]]] • 373, 390, 391, 392, 393
Set Up Incremental (Trickle) Loads • 19
Set Up the Loopback Interface • 325
set_restart_policy • 339, 352, 359
Setting a Purge Policy • 31, 184
Setting Location Performance • 105, 292, 293, 294
Setting Schema Search Paths • 42, 43, 204
Setting the Restart Policy • 352
Shell script • 7
Short Form • 21, 387, 408, 413
show_active_db • 359
Shutdown Problems • 144, 145, 241, 321, 344, 349, 357
SINGLELINE • 396
SINGLESTEP • 396
Slave • 324, 327
Slave director • 324, 327
SNMP • 216
SNMP, configuring reporting • 216
SnmptapDestinationsList • 31, 216
SnmptapEvents • 31, 216
SnmptapsEnabled • 31, 216
Snowflake Schema • 37
Specification of data fields • 161
Specify the Default Locale for the Database • 20, 21, 387
Specifying Default Values for Columns • 158
Specifying Disk Storage at Database Creation Time • 13
Specifying Disk Storage at Installation Time • 12
Spread • 250, 327, 359
Spread is not running • 249, 250
Spread Problems • 249
SSL • 131, 132, 133, 135, 136
SSL Prerequisites • 131, 132, 135
StandardConformingStrings • 28
Star Schema • 36, 146
start_db • 359
Starting a Database • 248, 283, 344, 357
Starting the Database • 120, 135, 144, 260, 342, 343
Startup Problems • 17, 144, 241, 245, 344, 345, 351
Startup successful, but some nodes are recovering • 245
Statistics

- AnalyzeRowCountInterval • 25, 175
 - and the Query Optimizer • 175
 - best practices and guidelines • 175, 176
 - collection • 175, 176
 - computing • 176
 - dropping • 178
 - exporting • 177
 - for manually-deployed designs • 91
 - for projections • 96
 - troubleshooting • 178
 - updating for query-specific designs • 79
 - Statistics Collection Guidelines • 175
 - Statistics Used by the Query Optimizer • 175
 - stop_db • 359
 - stop_host • 359
 - Stopping a Database • 241, 244, 245, 320, 321, 345, 348, 351, 355, 356
 - Stopping the Database • 144, 259
 - Stopping Vertica on Host • 241, 246, 248, 354, 356
 - Storage • 167, 173
 - Suggested Reading Paths • 2, 4
 - Support • 1
 - Supported Locales • 20, 406, 414
 - SuSE SLE 10 and SLE 11 • 225, 227
 - Syntax conventions • 7
 - syslog • 214
 - syslog, configuring reporting • 214
 - SyslogEnabled • 31
 - SyslogEvents • 31
 - SyslogFacility • 31
 - SYSSTAT • 220
 - System tables • 204, 209
- T**
- t • 373, 388, 390, 391, 400
 - T • 388
 - T [STRING] • 390, 391
 - T table_options --table-attr table_options • 373
 - t --tuples-only • 373, 400
 - Table Privileges • 114, 139, 140
 - tableattr • 388, 391
 - Target Memory Determination for Queries in Concurrent Environments • 302, 307, 308, 311
 - Technical Support • 1, 4, 14, 23, 27, 36, 96, 142, 170, 173, 178, 213, 220, 246, 247, 248, 250, 309, 319, 325, 343, 353, 354, 407, 428
 - Test the Database • 18
 - Test the Optimized Database • 18
 - Testing Modified Database Designs • 268, 277, 288
 - The \d [PATTERN] meta-commands • 378
 - The Design Process • 92
 - The GENERAL Pool • 300, 304, 316
 - The Resource Manager • 299
 - Threads, using more • 169
 - TIMEOUT ERROR
 - Could not login with SSH • 247
 - timing • 18, 391
 - TIMING • 201
 - title • 377, 388
 - top utility • 220
 - TRACE • 201
 - Tracking Load Status • 149
 - Transforming Data During Loads • 158, 159
 - Trickle Loading • 19, 148, 150
 - Troubleshooting Issues Using Statistics • 178
 - Troubleshooting Keepalived Issues • 334
 - Tuning the Built-in Pools • 316
 - Tuning the Tuple Mover • 169, 212, 316
 - Tuple Mover • 167, 169, 173
 - Tuple Mover behavior in 3.5 (reverting) • 169
 - Tuple Mover best practices • 169
 - Tuple Mover Parameters • 26
 - tuples_only • 388, 391
 - Typographical Conventions • 7
- U**
- U username --username username • 373
 - Understanding the Tuple Mover • 167
 - uninstall_node • 359
 - Uninstalling the Vertica Monitoring Package • 237
 - UNIQUE Constraints • 49, 53
 - unset [NAME] • 391, 392
 - UPDATE • 146
 - upgrade_license_key • 142, 339, 357, 359
 - Upgrading • 236
 - Upgrading the License Key • 357
 - Upgrading the Vertica Monitoring Package • 224, 236
 - Uppercase text • 7
 - USER • 396
 - User Profiles • 302, 305, 311
 - Using INSERT, UPDATE, and DELETE • 150, 180
 - Using Load Scripts • 15, 149, 150, 160
 - Using more threads • 169

Using Multiple Schemas • 14, 39
Using Parallel Load Streams • 15, 151
Using Sequences • 55, 61, 159
Using Shared Storage with Vertica • 14
Using the Administration Tools • 135, 144, 272, 278, 291, 336, 341, 358
Using the COPY and LCOPY Statements • 148
Using the COPY Command • 150
Using the Database Designer • 78
Using the Graphical User Interface • 11, 336, 337
Using the Online Help • 340
Using the SQL Monitoring API • 204, 210, 289, 319, 322
Using User-defined Pools and User-Profiles for Workload Management • 305, 307, 309
Using Views • 75
Using vsql • 344, 367

V

v assignment --set assignment --variable assignment • 373
V --version • 373
Variables • 391, 392
VERBOSITY • 396
vertica-dashboard.xml • 234
verticadefines.php • 235
Vertical line • 7
View Privileges • 114, 139
view_cluster • 249, 250, 359
Viewing a Database • 101, 201, 257, 260, 264, 352
Viewing Database Cluster State • 144, 343, 344, 345, 355, 357
Views • 74, 75, 139
Virtual IP address • 324, 330, 333
Virtual IP Connection Problems • 333
VSQL_HOME • 396

W

w [FILE] • 392
w password • 369, 374
W --password • 374
WARNING • 201
When to Back Up the Database • 256
Where to Find Additional Information • 6
Where to Find the Vertica Documentation • 2
Workload management best practices • 307
WOS Overflow • 146, 150

Writing Administration Tools Scripts • 202, 336, 359
Writing and Deploying Custom Projections • 92, 96

X

x • 374, 390, 392
x --expanded • 374
X, --no-vsqlirc • 374

Z

z • 381, 392

Copyright Notice

Copyright© 2006-2011 Vertica Systems, Inc., and its licensors. All rights reserved.

Vertica Systems, Inc.
8 Federal Street
Billerica, MA 01821
Phone: (978) 600-1000
Fax: (978) 600-1001
E-Mail: info@vertica.com
Web site: <http://www.vertica.com>
(<http://www.vertica.com>)

The software described in this copyright notice is furnished under a license and may be used or copied only in accordance with the terms of such license. Vertica Systems, Inc. software contains proprietary information, as well as trade secrets of Vertica Systems, Inc., and is protected under international copyright law. Reproduction, adaptation, or translation, in whole or in part, by any means — graphic, electronic or mechanical, including photocopying, recording, taping, or storage in an information retrieval system — of any part of this work covered by copyright is prohibited without prior written permission of the copyright owner, except as allowed under the copyright laws.

This product or products depicted herein may be protected by one or more U.S. or international patents or pending patents.

Trademarks

Vertica™, the Vertica® Analytic Database™, and FlexStore™ are trademarks of Vertica Systems, Inc..

Adobe®, Acrobat®, and Acrobat® Reader® are registered trademarks of Adobe Systems Incorporated.

AMD™ is a trademark of Advanced Micro Devices, Inc., in the United States and other countries.

DataDirect® and DataDirect Connect® are registered trademarks of Progress Software Corporation in the U.S. and other countries.

Fedora™ is a trademark of Red Hat, Inc.

Intel® is a registered trademark of Intel.

Linux® is a registered trademark of Linus Torvalds.

Microsoft® is a registered trademark of Microsoft Corporation.

Novell® is a registered trademark and SUSE™ is a trademark of Novell, Inc., in the United States and other countries.

Oracle® is a registered trademark of Oracle Corporation.

Red Hat® is a registered trademark of Red Hat, Inc.

VMware® is a registered trademark or trademark of VMware, Inc., in the United States and/or other jurisdictions.

Other products mentioned may be trademarks or registered trademarks of their respective companies.

Open Source Software Acknowledgments

Vertica makes no representations or warranties regarding any third party software. All third-party software is provided or recommended by Vertica on an AS IS basis.

This product includes cryptographic software written by Eric Young (eay@cryptsoft.com).

ASMJIT

Copyright (c) 2008-2010, Petr Kobalicek <kobalicek.petr@gmail.com>

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Boost

Boost Software License - Version 1.38 - February 8th, 2009

Permission is hereby granted, free of charge, to any person or organization obtaining a copy of the software and accompanying documentation covered by this license (the "Software") to use, reproduce, display, distribute, execute, and transmit the Software, and to prepare derivative works of the Software, and to permit third-parties to whom the Software is furnished to do so, all subject to the following:

The copyright notices in the Software and this entire statement, including the above license grant, this restriction and the following disclaimer, must be included in all copies of the Software, in whole or in part, and all derivative works of the Software, unless such copies or derivative works are solely in the form of machine-executable object code generated by a source language processor.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT. IN NO EVENT SHALL THE COPYRIGHT HOLDERS OR ANYONE DISTRIBUTING THE SOFTWARE BE LIABLE FOR ANY DAMAGES OR OTHER LIABILITY, WHETHER IN CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

bzip2

This file is a part of bzip2 and/or libbzip2, a program and library for lossless, block-sorting data compression.

Copyright © 1996-2005 Julian R Seward. All rights reserved.

- 1** Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:
- 2** Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- 3** The origin of this software must not be misrepresented; you must not claim that you wrote the original software. If you use this software in a product, an acknowledgment in the product documentation would be appreciated but is not required.
- 4** Altered source versions must be plainly marked as such, and must not be misrepresented as being the original software.
- 5** The name of the author may not be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE AUTHOR ``AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Julian Seward, Cambridge, UK.

jseward@bzip.org <<mailto:jseward@bzip.org>>

bzip2/libbzip2 version 1.0 of 21 March 2000

This program is based on (at least) the work of:

Mike Burrows

David Wheeler

Peter Fenwick

Alistair Moffat

Radioed Neal

Ian H. Witten

Robert Sedgewick

Jon L. Bentley

Daemonize

Copyright © 2003-2007 Brian M. Clapper.

All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- Neither the name of the clapper.org nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Ganglia Open Source License

Copyright © 2001 by Matt Massie and The Regents of the University of California.

All rights reserved.

Permission to use, copy, modify, and distribute this software and its documentation for any purpose, without fee, and without written agreement is hereby granted, provided that the above copyright notice and the following two paragraphs appear in all copies of this software.

IN NO EVENT SHALL THE UNIVERSITY OF CALIFORNIA BE LIABLE TO ANY PARTY FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OF THIS SOFTWARE AND ITS DOCUMENTATION, EVEN IF THE UNIVERSITY OF CALIFORNIA HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

THE UNIVERSITY OF CALIFORNIA SPECIFICALLY DISCLAIMS ANY WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE SOFTWARE PROVIDED HEREUNDER IS ON AN "AS IS" BASIS, AND THE UNIVERSITY OF CALIFORNIA HAS NO OBLIGATION TO PROVIDE MAINTENANCE, SUPPORT, UPDATES, ENHANCEMENTS, OR MODIFICATIONS.

ICU (International Components for Unicode) License - ICU 1.8.1 and later

COPYRIGHT AND PERMISSION NOTICE

Copyright © 1995-2009 International Business Machines Corporation and others

All rights reserved.

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, provided that the above copyright notice(s) and this permission notice appear in all copies of the Software and that both the above copyright notice(s) and this permission notice appear in supporting documentation.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT OF THIRD PARTY RIGHTS. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR HOLDERS INCLUDED IN THIS NOTICE BE LIABLE FOR ANY CLAIM, OR ANY SPECIAL INDIRECT OR CONSEQUENTIAL DAMAGES, OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

Except as contained in this notice, the name of a copyright holder shall not be used in advertising or otherwise to promote the sale, use or other dealings in this Software without prior written authorization of the copyright holder.

All trademarks and registered trademarks mentioned herein are the property of their respective owners.

Keepalived Vertica IPVS (IP Virtual Server) Load Balancer

Copyright © 2007 Free Software Foundation, Inc.

<http://fsf.org/>

The keepalived software contained in the `VerticaIPVSLoadBalancer-4.1.x.RHEL5.x86_64.rpm` software package is licensed under the GNU General Public License ("GPL"). You are entitled to receive the source code for such software. For no less than three years from the date you obtained this software package, you may download a copy of the source code for the software in this package licensed under the GPL at no charge by visiting <http://www.vertica.com/licenses/keepalived-1.1.17.tar.gz> <http://www.vertica.com/licenses/keepalived-1.1.17.tar.gz>. You may download this source code so that it remains separate from other software on your computer system.

jQuery

Copyright © 2009 John Resig, <http://jquery.com/>

All rights reserved.

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Lighttpd Open Source License

Copyright © 2004, Jan Kneschke, incremental
All rights reserved.

- 1** Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:
- 2** Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- 3** Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- 4** Neither the name of the 'incremental' nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

MersenneTwister.h

Copyright © 1997 - 2002, Makoto Matsumoto and Takuji Nishimura,
Copyright © 2000 - 2009, Richard J. Wagner
All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- 1** Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- 2** Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- 3** The names of its contributors may not be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

MIT Kerberos

Copyright © 1985-2007 by the Massachusetts Institute of Technology.

Export of software employing encryption from the United States of America may require a specific license from the United States Government. It is the responsibility of any person or organization contemplating export to obtain such a license before exporting.

WITHIN THAT CONSTRAINT, permission to use, copy, modify, and distribute this software and its documentation for any purpose and without fee is hereby granted, provided that the above copyright notice appear in all copies and that both that copyright notice and this permission notice appear in supporting documentation, and that the name of M.I.T. not be used in advertising or publicity pertaining to distribution of the software without specific, written prior permission. Furthermore if you modify this software you must label your software as modified software and not distribute it in such a fashion that it might be confused with the original MIT software. M.I.T. makes no representations about the suitability of this software for any purpose. It is provided "as is" without express or implied warranty.

Individual source code files are copyright MIT, Cygnus Support, Novell, OpenVision Technologies, Oracle, Red Hat, Sun Microsystems, FundsXpress, and others.

Project Athena, Athena, Athena MUSE, Discuss, Hesiod, Kerberos, Moira, and Zephyr are trademarks of the Massachusetts Institute of Technology (MIT). No commercial use of these trademarks may be made without prior written permission of MIT.

"Commercial use" means use of a name in a product or other for-profit manner. It does NOT prevent a commercial firm from referring to the MIT trademarks in order to convey information (although in doing so, recognition of their trademark status should be given).

Portions of src/lib/crypto have the following copyright:

Copyright © 1998 by the FundsXpress, INC.

All rights reserved.

Export of this software from the United States of America may require a specific license from the United States Government. It is the responsibility of any person or organization contemplating export to obtain such a license before exporting.

WITHIN THAT CONSTRAINT, permission to use, copy, modify, and distribute this software and its documentation for any purpose and without fee is hereby granted, provided that the above copyright notice appear in all copies and that both that copyright notice and this permission notice appear in supporting documentation, and that the name of FundsXpress. not be used in advertising or publicity pertaining to distribution of the software without specific, written prior permission. FundsXpress makes no representations about the suitability of this software for any purpose. It is provided "as is" without express or implied warranty.

THIS SOFTWARE IS PROVIDED "AS IS" AND WITHOUT ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, WITHOUT LIMITATION, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

The implementation of the AES encryption algorithm in src/lib/crypto/aes has the following copyright:

Copyright © 2001, Dr Brian Gladman <brg@gladman.uk.net>, Worcester, UK.
All rights reserved.

LICENSE TERMS

The free distribution and use of this software in both source and binary form is allowed (with or without changes) provided that:

- 1 Distributions of this source code include the above copyright notice, this list of conditions and the following disclaimer.
- 2 Distributions in binary form include the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other associated materials.
- 3 The copyright holder's name is not used to endorse products built using this software without specific written permission.

DISCLAIMER

This software is provided 'as is' with no explicit or implied warranties in respect of any properties, including, but not limited to, correctness and fitness for purpose.

The implementations of GSSAPI mechglue in GSSAPI-SPNEGO in src/lib/gssapi, including the following files:

- lib/gssapi/generic/gssapi_err_generic.et
- lib/gssapi/mechglue/g_accept_sec_context.c
- lib/gssapi/mechglue/g_acquire_cred.c
- lib/gssapi/mechglue/g_canon_name.c
- lib/gssapi/mechglue/g_compare_name.c
- lib/gssapi/mechglue/g_context_time.c
- lib/gssapi/mechglue/g_delete_sec_context.c
- lib/gssapi/mechglue/g_dsp_name.c
- lib/gssapi/mechglue/g_dsp_status.c
- lib/gssapi/mechglue/g_dup_name.c
- lib/gssapi/mechglue/g_exp_sec_context.c
- lib/gssapi/mechglue/g_export_name.c
- lib/gssapi/mechglue/g_glue.c
- lib/gssapi/mechglue/g_imp_name.c

- lib/gssapi/mechglue/g_imp_sec_context.c
- lib/gssapi/mechglue/g_init_sec_context.c
- lib/gssapi/mechglue/g_initialize.c
- lib/gssapi/mechglue/g_inquire_context.c
- lib/gssapi/mechglue/g_inquire_cred.c
- lib/gssapi/mechglue/g_inquire_names.c
- lib/gssapi/mechglue/g_process_context.c
- lib/gssapi/mechglue/g_rel_buffer.c
- lib/gssapi/mechglue/g_rel_cred.c
- lib/gssapi/mechglue/g_rel_name.c
- lib/gssapi/mechglue/g_rel_oid_set.c
- lib/gssapi/mechglue/g_seal.c
- lib/gssapi/mechglue/g_sign.c
- lib/gssapi/mechglue/g_store_cred.c
- lib/gssapi/mechglue/g_unseal.c
- lib/gssapi/mechglue/g_userok.c
- lib/gssapi/mechglue/g_utils.c
- lib/gssapi/mechglue/g_verify.c
- lib/gssapi/mechglue/gssd_pname_to_uid.c
- lib/gssapi/mechglue/mglueP.h
- lib/gssapi/mechglue/oid_ops.c
- lib/gssapi/spnego/gssapiP_spnego.h
- lib/gssapi/spnego/spnego_mech.c

are subject to the following license:

Copyright © 2004 Sun Microsystems, Inc.

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the “Software”), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Npgsql-.Net Data Provider for Postgresql

Copyright © 2002-2008, The Npgsql Development Team

Permission to use, copy, modify, and distribute this software and its documentation for any purpose, without fee, and without a written agreement is hereby granted, provided that the above copyright notice and this paragraph and the following two paragraphs appear in all copies.

IN NO EVENT SHALL THE NPGSQL DEVELOPMENT TEAM BE LIABLE TO ANY PARTY FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES, INCLUDING LOST PROFITS, ARISING OUT OF THE USE OF THIS SOFTWARE AND ITS DOCUMENTATION, EVEN IF THE NPGSQL DEVELOPMENT TEAM HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

THE NPGSQL DEVELOPMENT TEAM SPECIFICALLY DISCLAIMS ANY WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE SOFTWARE PROVIDED HEREUNDER IS ON AN "AS IS" BASIS, AND THE NPGSQL DEVELOPMENT TEAM HAS NO OBLIGATIONS TO PROVIDE MAINTENANCE, SUPPORT, UPDATES, ENHANCEMENTS, OR MODIFICATIONS.

Open LDAP

The OpenLDAP Public License
Version 2.8, 17 August 2003

Redistribution and use of this software and associated documentation ("Software"), with or without modification, are permitted provided that the following conditions are met:

- 1 Redistributions in source form must retain copyright statements and notices,
- 2 Redistributions in binary form must reproduce applicable copyright statements and notices, this list of conditions, and the following disclaimer in the documentation and/or other materials provided with the distribution, and
- 3 Redistributions must contain a verbatim copy of this document.

The OpenLDAP Foundation may revise this license from time to time. Each revision is distinguished by a version number. You may use this Software under terms of this license revision or under the terms of any subsequent revision of the license.

THIS SOFTWARE IS PROVIDED BY THE OPENLDAP FOUNDATION AND ITS CONTRIBUTORS ``AS IS" AND ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE OPENLDAP FOUNDATION, ITS CONTRIBUTORS, OR THE AUTHOR(S) OR OWNER(S) OF THE SOFTWARE BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

The names of the authors and copyright holders must not be used in advertising or otherwise to promote the sale, use or other dealing in this Software without specific, written prior permission. Title to copyright in this Software shall at all times remain with copyright holders.

OpenLDAP is a registered trademark of the OpenLDAP Foundation.

Copyright 1999-2003 The OpenLDAP Foundation, Redwood City, California, USA. All Rights Reserved. Permission to copy and distribute verbatim copies of this document is granted.

Open SSL

OpenSSL License

Copyright © 1998-2008 The OpenSSL Project. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- 1 Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- 2 Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- 3 All advertising materials mentioning features or use of this software must display the following acknowledgment: "This product includes software developed by the OpenSSL Project for use in the OpenSSL Toolkit. (<http://www.openssl.org/>)"
- 4 The names "OpenSSL Toolkit" and "OpenSSL Project" must not be used to endorse or promote products derived from this software without prior written permission. For written permission, please contact openssl-core@openssl.org.
- 5 Products derived from this software may not be called "OpenSSL" nor may "OpenSSL" appear in their names without prior written permission of the OpenSSL Project.
- 6 Redistributions of any form whatsoever must retain the following acknowledgment: "This product includes software developed by the OpenSSL Project for use in the OpenSSL Toolkit (<http://www.openssl.org/>)"

THIS SOFTWARE IS PROVIDED BY THE OpenSSL PROJECT ``AS IS'' AND ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE OpenSSL PROJECT OR ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

PCRE LICENCE

PCRE is a library of functions to support regular expressions whose syntax and semantics are as close as possible to those of the Perl 5 language.

Release 8 of PCRE is distributed under the terms of the "BSD" licence, as specified below. The documentation for PCRE, supplied in the "doc" directory, is distributed under the same terms as the software itself.

The basic library functions are written in C and are freestanding. Also included in the distribution is a set of C++ wrapper functions.

THE BASIC LIBRARY FUNCTIONS

Written by: Philip Hazel
Email local part: ph10
Email domain: cam.ac.uk
University of Cambridge Computing Service,
Cambridge, England.
Copyright (c) 1997-2010 University of Cambridge
All rights reserved.

THE C++ WRAPPER FUNCTIONS

Contributed by: Google Inc.
Copyright (c) 2007-2010, Google Inc.
All rights reserved.

THE "BSD" LICENCE

- Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:
- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

Neither the name of the University of Cambridge nor the name of Google Inc. nor the names of their contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

End

Perl Artistic License

Copyright © August 15, 1997

Preamble

The intent of this document is to state the conditions under which a Package may be copied, such that the Copyright Holder maintains some semblance of artistic control over the development of the package, while giving the users of the package the right to use and distribute the Package in a more-or-less customary fashion, plus the right to make reasonable modifications.

Definitions

"Package" refers to the collection of files distributed by the Copyright Holder, and derivatives of that collection of files created through textual modification.

"Standard Version" refers to such a Package if it has not been modified, or has been modified in accordance with the wishes of the Copyright Holder as specified below.

"Copyright Holder" is whoever is named in the copyright or copyrights for the package.

"You" is you, if you're thinking about copying or distributing this Package.

"Reasonable copying fee" is whatever you can justify on the basis of media cost, duplication charges, time of people involved, and so on. (You will not be required to justify it to the Copyright Holder, but only to the computing community at large as a market that must bear the fee.)

"Freely Available" means that no fee is charged for the item itself, though there may be fees involved in handling the item. It also means that recipients of the item may redistribute it under the same conditions they received it.

- 1 You may make and give away verbatim copies of the source form of the Standard Version of this Package without restriction, provided that you duplicate all of the original copyright notices and associated disclaimers.
- 2 You may apply bug fixes, portability fixes and other modifications derived from the Public Domain or from the Copyright Holder. A Package modified in such a way shall still be considered the Standard Version.
- 3 You may otherwise modify your copy of this Package in any way, provided that you insert a prominent notice in each changed file stating how and when you changed that file, and provided that you do at least ONE of the following:
- 4 place your modifications in the Public Domain or otherwise make them Freely Available, such as by posting said modifications to Usenet or an equivalent medium, or placing the modifications on a major archive site such as uunet.uu.net, or by allowing the Copyright Holder to include your modifications in the Standard Version of the Package.
 1. use the modified Package only within your corporation or organization.
 2. rename any non-standard executables so the names do not conflict with standard executables, which must also be provided, and provide a separate manual page for each non-standard executable that clearly documents how it differs from the Standard Version.
 3. make other distribution arrangements with the Copyright Holder.
- 5 You may distribute the programs of this Package in object code or executable form, provided that you do at least ONE of the following:
 1. distribute a Standard Version of the executables and library files, together with instructions (in the manual page or equivalent) on where to get the Standard Version.
 2. accompany the distribution with the machine-readable source of the Package with your modifications.

3. give non-standard executables non-standard names, and clearly document the differences in manual pages (or equivalent), together with instructions on where to get the Standard Version.
4. make other distribution arrangements with the Copyright Holder.
- 6 You may charge a reasonable copying fee for any distribution of this Package. You may charge any fee you choose for support of this Package. You may not charge a fee for this Package itself. However, you may distribute this Package in aggregate with other (possibly commercial) programs as part of a larger (possibly commercial) software distribution provided that you do not advertise this Package as a product of your own. You may embed this Package's interpreter within an executable of yours (by linking); this shall be construed as a mere form of aggregation, provided that the complete Standard Version of the interpreter is so embedded.
- 7 The scripts and library files supplied as input to or produced as output from the programs of this Package do not automatically fall under the copyright of this Package, but belong to whomever generated them, and may be sold commercially, and may be aggregated with this Package. If such scripts or library files are aggregated with this Package via the so-called "undump" or "unexec" methods of producing a binary executable image, then distribution of such an image shall neither be construed as a distribution of this Package nor shall it fall under the restrictions of Paragraphs 3 and 4, provided that you do not represent such an executable image as a Standard Version of this Package.
- 8 C subroutines (or comparably compiled subroutines in other languages) supplied by you and linked into this Package in order to emulate subroutines and variables of the language defined by this Package shall not be considered part of this Package, but are the equivalent of input as in Paragraph 6, provided these subroutines do not change the language in any way that would cause it to fail the regression tests for the language.
- 9 Aggregation of this Package with a commercial distribution is always permitted provided that the use of this Package is embedded; that is, when no overt attempt is made to make this Package's interfaces visible to the end user of the commercial distribution. Such use shall not be construed as a distribution of this Package.
- 10 The name of the Copyright Holder may not be used to endorse or promote products derived from this software without specific prior written permission.

THIS PACKAGE IS PROVIDED "AS IS" AND WITHOUT ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, WITHOUT LIMITATION, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

The End

Pexpect

Copyright © 2010 Noah Spurrier

Credits: Noah Spurrier, Richard Holden, Marco Molteni, Kimberley Burchett, Robert Stone, Hartmut Goebel, Chad Schroeder, Erick Tryzelaar, Dave Kirby, Ids vander Molen, George Todd, Noel Taylor, Nicolas D. Cesar, Alexander Gattin, Geoffrey Marshall, Francisco Lourenco, Glen Mabey, Karthik Gurusamy, Fernando Perez, Corey Minyard, Jon Cohen, Guillaume Chazarain, Andrew Ryan, Nick Craig-Wood, Andrew Stone, Jorgen Grahn (Let me know if I forgot anyone.)

Free, open source, and all that good stuff.

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

PHP License

The PHP License, version 3.01

Copyright © 1999 - 2009 The PHP Group. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, is permitted provided that the following conditions are met:

- 1 Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- 2 Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- 3 The name "PHP" must not be used to endorse or promote products derived from this software without prior written permission. For written permission, please contact group@php.net.
- 4 Products derived from this software may not be called "PHP", nor may "PHP" appear in their name, without prior written permission from group@php.net. You may indicate that your software works in conjunction with PHP by saying "Foo for PHP" instead of calling it "PHP Foo" or "phpfoo"
- 5 The PHP Group may publish revised and/or new versions of the license from time to time. Each version will be given a distinguishing version number.
 - Once covered code has been published under a particular version of the license, you may always continue to use it under the terms of that version. You may also choose to use such covered code under the terms of any subsequent version of the license published by the PHP Group. No one other than the PHP Group has the right to modify the terms applicable to covered code created under this License.
- 6 Redistributions of any form whatsoever must retain the following acknowledgment:
"This product includes PHP software, freely available from [<http://www.php.net/software/>](http://www.php.net/software/)".

THIS SOFTWARE IS PROVIDED BY THE PHP DEVELOPMENT TEAM ``AS IS" AND ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE PHP DEVELOPMENT TEAM OR ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

This software consists of voluntary contributions made by many individuals on behalf of the PHP Group.

The PHP Group can be contacted via Email at group@php.net.

For more information on the PHP Group and the PHP project, please see <<http://www.php.net>>.

PHP includes the Zend Engine, freely available at <<http://www.zend.com>>.

PostgreSQL

This product uses the PostgreSQL Database Management System(formerly known as Postgres, then as Postgres95)

Portions Copyright © 1996-2005, The PostgreSQL Global Development Group

Portions Copyright © 1994, The Regents of the University of California

Permission to use, copy, modify, and distribute this software and its documentation for any purpose, without fee, and without a written agreement is hereby granted, provided that the above copyright notice and this paragraph and the following two paragraphs appear in all copies.

IN NO EVENT SHALL THE UNIVERSITY OF CALIFORNIA BE LIABLE TO ANY PARTY FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES, INCLUDING LOST PROFITS, ARISING OUT OF THE USE OF THIS SOFTWARE AND ITS DOCUMENTATION, EVEN IF THE UNIVERSITY OF CALIFORNIA HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

THE UNIVERSITY OF CALIFORNIA SPECIFICALLY DISCLAIMS ANY WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE SOFTWARE PROVIDED HEREUNDER IS ON AN "AS IS" BASIS, AND THE UNIVERSITY OF CALIFORNIA HAS NO OBLIGATIONS TO PROVIDE MAINTENANCE, SUPPORT, UPDATES, ENHANCEMENTS, OR MODIFICATIONS.

Python Dialog

The Administration Tools part of this product uses Python Dialog, a Python module for doing console-mode user interaction.

Upstream Author:

Peter Astrand <peter@cendio.se>

Robb Shecter <robb@acm.org>

Sultanbek Tezadov <<http://sultan.da.ru>>

Florent Rougon <flo@via.ecp.fr>

Copyright © 2000 Robb Shecter, Sultanbek Tezadov

Copyright © 2002, 2003, 2004 Florent Rougon

License:

This package is free software; you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This package is distributed in the hope that it is useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details.

You should have received a copy of the GNU Lesser General Public License along with this package; if not, write to the Free Software Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA

The complete source code of the Python dialog package and complete text of the GNU Lesser General Public License can be found on the Vertica Systems Web site at

<http://www.vertica.com/licenses/pythondialog-2.7.tar.bz2>

<http://www.vertica.com/licenses/pythondialog-2.7.tar.bz2>

RRDTool Open Source License

Note: rrdtool is a dependency of using the ganglia-web third-party tool. RRDTool allows the graphs displayed by ganglia-web to be produced.

RRDTOOL - Round Robin Database Tool

A tool for fast logging of numerical data graphical display of this data.

Copyright © 1998-2008 Tobias Oetiker

All rights reserved.

GNU GPL License

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA

FLOSS License Exception

(Adapted from <http://www.mysql.com/company/legal/licensing/foss-exception.html>)

I want specified Free/Libre and Open Source Software ("FLOSS") applications to be able to use specified GPL-licensed RRDtool libraries (the "Program") despite the fact that not all FLOSS licenses are compatible with version 2 of the GNU General Public License (the "GPL").

As a special exception to the terms and conditions of version 2.0 of the GPL:

You are free to distribute a Derivative Work that is formed entirely from the Program and one or more works (each, a "FLOSS Work") licensed under one or more of the licenses listed below, as long as:

- 1** You obey the GPL in all respects for the Program and the Derivative Work, except for identifiable sections of the Derivative Work which are not derived from the Program, and which can reasonably be considered independent and separate works in themselves
- 2** All identifiable sections of the Derivative Work which are not derived from the Program, and which can reasonably be considered independent and separate works in themselves
 - are distributed subject to one of the FLOSS licenses listed below, and
 - the object code or executable form of those sections are accompanied by the complete corresponding machine-readable source code for those sections on the same medium and under the same FLOSS license as the corresponding object code or executable forms of those sections.
- 3** Any works which are aggregated with the Program or with a Derivative Work on a volume of a storage or distribution medium in accordance with the GPL, can reasonably be considered independent and separate works in themselves which are not derivatives of either the Program, a Derivative Work or a FLOSS Work.

If the above conditions are not met, then the Program may only be copied, modified, distributed or used under the terms and conditions of the GPL.

FLOSS License List

License name	Version(s)/Copyright Date
Academic Free License	2.0
Apache Software License	1.0/1.1/2.0
Apple Public Source License	2.0
Artistic license	From Perl 5.8.0
BSD license	"July 22 1999"
Common Public License	1.0
GNU Library or "Lesser" General Public License (LGPL)	2.0/2.1
IBM Public License, Version	1.0
Jabber Open Source License	1.0
MIT License (As listed in file MIT-License.txt)	-
Mozilla Public License (MPL)	1.0/1.1
Open Software License	2.0
OpenSSL license (with original SSLeay license)	"2003" ("1998")
PHP License	3.0
Python license (CNRI Python License)	-
Python Software Foundation License	2.1.1
Sleepycat License	"1999"

W3C License	"2001"
X11 License	"2001"
Zlib/libpng License	-
Zope Public License	2.0/2.1

Spread

This product uses software developed by Spread Concepts LLC for use in the Spread toolkit. For more information about Spread see <http://www.spread.org> (<http://www.spread.org>).

Copyright © 1993-2006 Spread Concepts LLC.

All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- 1 Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer and request.
- 2 Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer and request in the documentation and/or other materials provided with the distribution.
- 3 All advertising materials (including web pages) mentioning features or use of this software, or software that uses this software, must display the following acknowledgment: "This product uses software developed by Spread Concepts LLC for use in the Spread toolkit. For more information about Spread see <http://www.spread.org>"
- 4 The names "Spread" or "Spread toolkit" must not be used to endorse or promote products derived from this software without prior written permission.
- 5 Redistributions of any form whatsoever must retain the following acknowledgment:
- 6 "This product uses software developed by Spread Concepts LLC for use in the Spread toolkit. For more information about Spread, see <http://www.spread.org>"
- 7 This license shall be governed by and construed and enforced in accordance with the laws of the State of Maryland, without reference to its conflicts of law provisions. The exclusive jurisdiction and venue for all legal actions relating to this license shall be in courts of competent subject matter jurisdiction located in the State of Maryland.

TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, SPREAD IS PROVIDED UNDER THIS LICENSE ON AN AS IS BASIS, WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, WITHOUT LIMITATION, WARRANTIES THAT SPREAD IS FREE OF DEFECTS, MERCHANTABILITY, FIT FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT. ALL WARRANTIES ARE DISCLAIMED AND THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE CODE IS WITH YOU. SHOULD ANY CODE PROVE DEFECTIVE IN ANY RESPECT, YOU (NOT THE COPYRIGHT HOLDER OR ANY OTHER CONTRIBUTOR) ASSUME THE COST OF ANY NECESSARY SERVICING, REPAIR OR CORRECTION. THIS DISCLAIMER OF WARRANTY CONSTITUTES AN ESSENTIAL PART OF THIS LICENSE. NO USE OF ANY CODE IS AUTHORIZED HEREUNDER EXCEPT UNDER THIS DISCLAIMER.

TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, IN NO EVENT SHALL THE COPYRIGHT HOLDER OR ANY OTHER CONTRIBUTOR BE LIABLE FOR ANY SPECIAL, INCIDENTAL, INDIRECT, OR CONSEQUENTIAL DAMAGES FOR LOSS OF PROFITS, REVENUE, OR FOR LOSS OF INFORMATION OR ANY OTHER LOSS.

YOU EXPRESSLY AGREE TO FOREVER INDEMNIFY, DEFEND AND HOLD HARMLESS THE COPYRIGHT HOLDERS AND CONTRIBUTORS OF SPREAD AGAINST ALL CLAIMS, DEMANDS, SUITS OR OTHER ACTIONS ARISING DIRECTLY OR INDIRECTLY FROM YOUR ACCEPTANCE AND USE OF SPREAD.

Although NOT REQUIRED, we at Spread Concepts would appreciate it if active users of Spread put a link on their web site to Spread's web site when possible. We also encourage users to let us know who they are, how they are using Spread, and any comments they have through either e-mail (spread@spread.org) or our web site at (<http://www.spread.org/comments>).

SNMP

Various copyrights apply to this package, listed in various separate parts below. Please make sure that you read all the parts. Up until 2001, the project was based at UC Davis, and the first part covers all code written during this time. From 2001 onwards, the project has been based at SourceForge, and Networks Associates Technology, Inc hold the copyright on behalf of the wider Net-SNMP community, covering all derivative work done since then. An additional copyright section has been added as Part 3 below also under a BSD license for the work contributed by Cambridge Broadband Ltd. to the project since 2001. An additional copyright section has been added as Part 4 below also under a BSD license for the work contributed by Sun Microsystems, Inc. to the project since 2003.

Code has been contributed to this project by many people over the years it has been in development, and a full list of contributors can be found in the README file under the THANKS section.

Part 1: CMU/UCD copyright notice: (BSD like)

Copyright © 1989, 1991, 1992 by Carnegie Mellon University

Derivative Work - 1996, 1998-2000

Copyright © 1996, 1998-2000 The Regents of the University of California

All Rights Reserved

Permission to use, copy, modify and distribute this software and its documentation for any purpose and without fee is hereby granted, provided that the above copyright notice appears in all copies and that both that copyright notice and this permission notice appear in supporting documentation, and that the name of CMU and The Regents of the University of California not be used in advertising or publicity pertaining to distribution of the software without specific written permission.

CMU AND THE REGENTS OF THE UNIVERSITY OF CALIFORNIA DISCLAIM ALL WARRANTIES WITH REGARD TO THIS SOFTWARE, INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL CMU OR THE REGENTS OF THE UNIVERSITY OF CALIFORNIA BE LIABLE FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM THE LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

Part 2: Networks Associates Technology, Inc copyright notice (BSD)

Copyright © 2001-2003, Networks Associates Technology, Inc

All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- Neither the name of the Networks Associates Technology, Inc nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDERS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Part 3: Cambridge Broadband Ltd. copyright notice (BSD)

Portions of this code are copyright (c) 2001-2003, Cambridge Broadband Ltd.

All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

- The name of Cambridge Broadband Ltd. may not be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDER ``AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Part 4: Sun Microsystems, Inc. copyright notice (BSD)

Copyright © 2003 Sun Microsystems, Inc., 4150 Network Circle, Santa Clara, California 95054, U.S.A. All rights reserved.

Use is subject to license terms below.

This distribution may include materials developed by third parties.

Sun, Sun Microsystems, the Sun logo and Solaris are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and other countries.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- Neither the name of the Sun Microsystems, Inc. nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS ``AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDERS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Part 5: Sparta, Inc copyright notice (BSD)

Copyright © 2003-2006, Sparta, Inc

All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- Neither the name of Sparta, Inc nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDERS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Part 6: Cisco/BUPTNIC copyright notice (BSD)

Copyright © 2004, Cisco, Inc and Information Network Center of Beijing University of Posts and Telecommunications.

All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- Neither the name of Cisco, Inc, Beijing University of Posts and Telecommunications, nor the names of their contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS ``AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDERS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Part 7: Fabasoft R&D Software GmbH & Co KG copyright notice (BSD)

Copyright © Fabasoft R&D Software GmbH & Co KG, 2003

oss@fabasoft.com

Author: Bernhard Penz

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- The name of Fabasoft R&D Software GmbH & Co KG or any of its subsidiaries, brand or product names may not be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDER ``AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER BE

LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Tecla Command-line Editing

Copyright © 2000 by Martin C. Shepherd.

All rights reserved.

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, provided that the above copyright notice(s) and this permission notice appear in all copies of the Software and that both the above copyright notice(s) and this permission notice appear in supporting documentation.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT OF THIRD PARTY RIGHTS. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR HOLDERS INCLUDED IN THIS NOTICE BE LIABLE FOR ANY CLAIM, OR ANY SPECIAL INDIRECT OR CONSEQUENTIAL DAMAGES, OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

Except as contained in this notice, the name of a copyright holder shall not be used in advertising or otherwise to promote the sale, use or other dealings in this Software without prior written authorization of the copyright holder.

Webmin Open Source License

Copyright © Jamie Cameron

All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- 1** Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- 2** Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- 3** Neither the name of the developer nor the names of contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE DEVELOPER "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE DEVELOPER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

xerces

NOTICE file corresponding to section 4(d) of the Apache License, Version 2.0, in this case for the Apache Xerces distribution.

This product includes software developed by The Apache Software Foundation (<http://www.apache.org/>).

Portions of this software were originally based on the following:

Software copyright © 1999, IBM Corporation., <http://www.ibm.com>.

zlib

This is used by the project to load zipped files directly by COPY command. www.zlib.net/

zlib.h -- interface of the 'zlib' general purpose compression library version 1.2.3, July 18th, 2005

Copyright © 1995-2005 Jean-loup Gailly and Mark Adler

This software is provided 'as-is', without any express or implied warranty. In no event will the authors be held liable for any damages arising from the use of this software.

Permission is granted to anyone to use this software for any purpose, including commercial applications, and to alter it and redistribute it freely, subject to the following restrictions:

- 1** The origin of this software must not be misrepresented; you must not claim that you wrote the original software. If you use this software in a product, an acknowledgment in the product documentation would be appreciated but is not required.
- 2** Altered source versions must be plainly marked as such, and must not be misrepresented as being the original software.
- 3** This notice may not be removed or altered from any source distribution.

Jean-loup Gailly jloup@gzip.org

Mark Adler madler@alumni.caltech.edu