Vertica Database 2.0.5-0

Product Overview

Date of Publication: 1/8/2008

CONFIDENTIAL

Copyright Notice

Vertica Systems, Inc.

Three Dundee Park Drive, Suite 102

Andover, MA 01810-3723

Phone: (978) 475-1070

Fax: (978) 475-6855

E-Mail: info@vertica.com

Web site: http://www.vertica.com
(http://www.vertica.com)

Trademarks

Vertica™ and the Vertica Database™ 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.

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 Acknowledgements

Boost

Boost Software License - Version 1.0 - August 17th, 2003

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.

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>

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 will be 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

On Vertica systems, 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 website at http://www.vertica.com/licenses/pythondialog-2.7.tar.bz2 http://www.vertica.com/licenses/pythondialog-2.7.tar.bz2

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).

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:

"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"

6. 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,

MERCHANTABLE, FIT FOR A PARTICULAR PURPOSE OR NON-INFRINGING. 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).

Contents

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

Technical Support

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

http://www.vertica.com/support (http://www.vertica.com/support)

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

Before reporting a problem, please run the Diagnostics Utility described in the Troubleshooting Guide and attach the resulting .zip file.

Typographical Conventions

It is important to understand the terms and typographical conventions used in this document.

General Convention	Description
colored bold text	introduces new terms defined either in the text, the glossary, or both.
normal italic text	indicates emphasis and the titles of other documents.
UPPERCASE TEXT	indicates the name of an SQL command or keyword.
`monospace text`	indicates literal interactive or programmatic input/output.
`italic monospace text`	indicates user-supplied information in interactive or programmatic input/output.
`bold monospace text`	indicates literal interactive user input
↵	indicates the Return/Enter key; implicit on all user input that includes text

SQL Syntax Convention	Description
indentation	is an attempt to maximize readability; SQL is a free-form language.
backslash \	continuation character used to indicate text that is too long to fit on a single line.
braces { }	indicate required items.
brackets [ ]	indicate optional items.
ellipses ...	indicate an optional sequence of similar items.
vertical ellipses ⋮	indicate an optional sequence of similar items or that part of the text has been omitted for readability.
vertical line |	within braces or brackets, indicates a choice .

Preface

This document presents a high-level overview of Vertica Database, an innovative system that represents a major improvement in performance over products currently available from major DBMS software and appliance vendors. It introduces the basic concepts that you need to be familiar with in order to effectively design, build, operate, and maintain Vertica database.

Prerequisites

This document assumes that you are familiar with the basic concepts and terminology of relational database management systems and SQL.

Audience

This document is intended for all users of Vertica Database.

Introduction

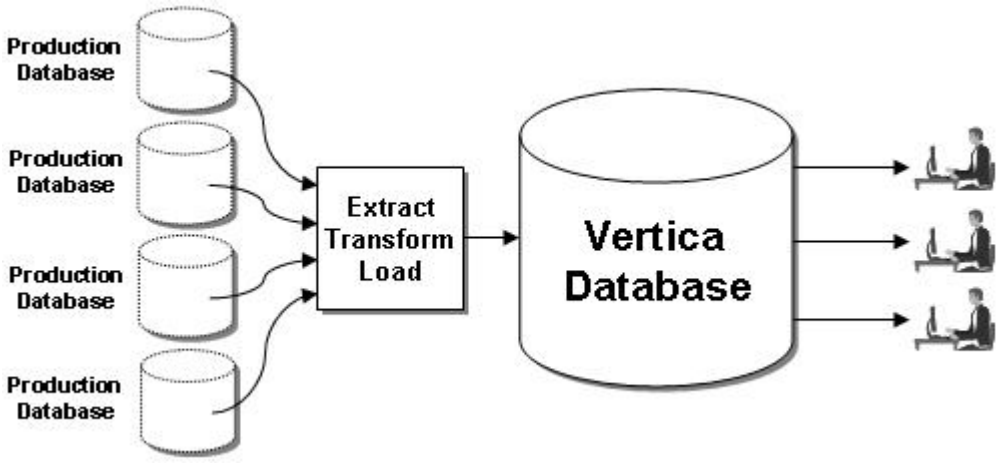
Vertica Database is an innovative, ground-up implementation of a relational database management system that is optimized for **read-intensive workloads**. Vertica provides extremely fast ad hoc SQL query performance, even for very large databases, making it well suited for:

- Data warehousing
- Data marts
- Clickstream analysis
- Fraud detection
- Call detail analysis
- Consumer customer analytics
- Business intelligence
- Other query-intensive applications

Data Warehousing

A data warehouse is a relational database that is designed for query and analysis rather than transaction processing. Data warehouses:

- are often subjected to a heavy load of periodic and ad-hoc queries.
- contain historical information that enables analysis of correlations and trends over long periods of time.
- integrate data from various production (transactional) databases. Extraction, transformation, and loading (ETL) software converts the data to a common format and copies it into a data warehouse at regular intervals.
- typically consist of one or more star or snowflake schemas.

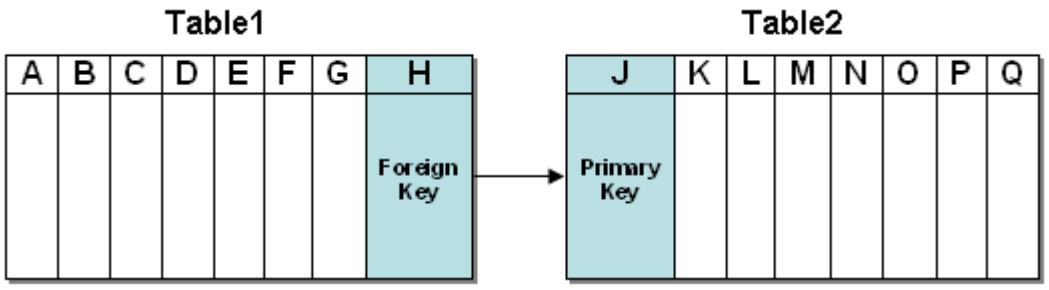

Production Database
Production Database
Production Database
Production Database
Extract Transform Load
Vertica Database

Column Store Architecture

The design of Vertica is based on a **column store architecture** in which the values for each single column (or attribute) are stored contiguously on disk. This approach provides an order of magnitude increase in speed.

To process a query, a row store reads all of the columns in all of the tables named in the query, regardless of how wide the tables may be and how many columns are actually needed. Quite often, queries only use two or three columns from tables of up to several hundred columns, resulting in a great deal of unnecessary data retrieval.

Vertica, however, reads the columns from database objects called Projections, which are described in the ***Projections*** (page 27) section of this document. It does not waste resources by reading large numbers of unused columns. Every byte of data is used by the execution engine. For example, consider this simple two-table schema:



Suppose you want to execute this query:

```
SELECT A, C, N
FROM Table1, Table2
WHERE H = J
```

A row store has to read 16 columns (A through H and J through Q) from physical storage for each tuple in the result set. A column store with a query-specific projection only has to read three columns (A, C, and N).

Data Encoding and Compression

Encoding

Encoding is the process of transforming data from one format into another. In Vertica, encoded data can be processed directly, which distinguishes it from compression. Vertica uses a number of different encoding strategies, depending on column data type, table cardinality, and sort order.

The query executor in Vertica operates on the encoded data representation whenever possible to avoid the cost of decoding. It also passes encoded values to other operations, saving memory bandwidth. In contrast, row stores and most other column stores typically decode data elements before performing any operation.

Compression

Compression is the process of transforming data into a more compact format. Compressed data cannot be directly processed; it must first be decompressed. Vertica uses integer packing for unencoded integers and LZO for compressible data. Although compression is generally considered to be a form of encoding, the terms have different meanings in Vertica.

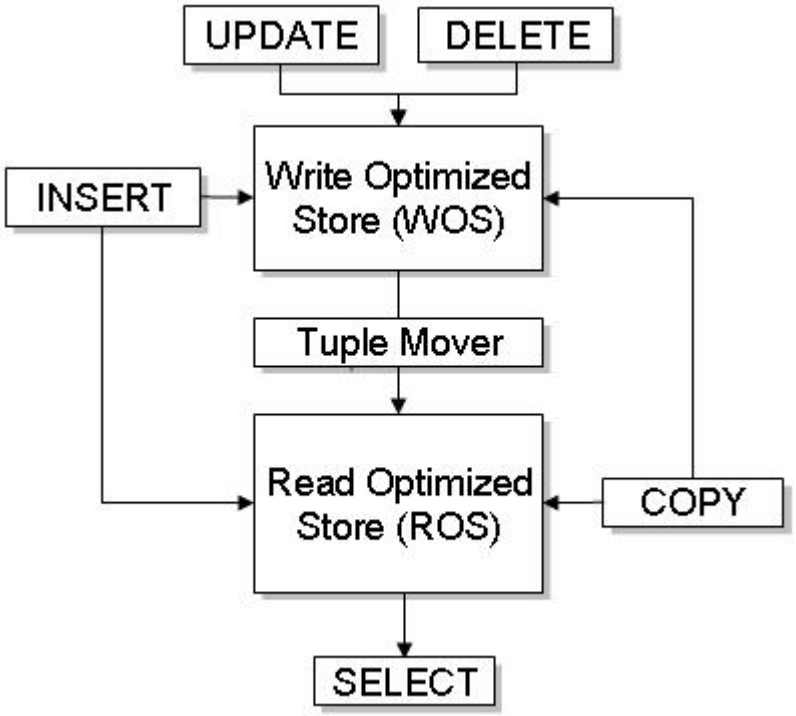
The size of a database is often limited by the availability of storage resources. Typically, when a database exceeds its size limitations, the administrator archives data that is older than a specific historical threshold.

The extensive use of compression allows a column store to occupy substantially less storage than a row store. In a column store, every value stored in a column of a projection has the same data type. This greatly facilitates compression, particularly in sorted columns. In a row store, each value of a row can have a different data type, resulting in a much less effective use of compression.

Vertica's efficient storage allows the database administrator to keep much more historical data in physical storage. In other words, the archiving threshold can be set to a much earlier date than in a less efficient store.

Hybrid Storage Model

To support Data Manipulation Language commands (INSERT, UPDATE, and DELETE) and bulk load operations (COPY) intermixed with queries in a typical data warehouse workload, Vertica implements the storage model shown in the illustration below. This model is the same on each Vertica node.



The WOS (Write Optimized Store) is a memory-resident data structure into which INSERT, UPDATE, DELETE, and COPY (without DIRECT hint) actions are recorded. Like the ROS, the WOS is arranged by projection but it stores tuples without sorting, compression, or indexing and thus supports very fast load speeds. The WOS organizes data by epoch and holds uncommitted transaction data.

The tuple mover is the component of Vertica that moves the contents of the Write Optimized Store (WOS) into the Read Optimized Store (ROS). This data movement is known as a moveout. Normally, the tuple mover runs automatically in the background at preset intervals and is referred to as the ATM.

The ROS (Read Optimized Store) is a highly optimized, read-oriented, physical storage structure that is organized by projection and that makes heavy use of compression and indexing. You can use the COPY...DIRECT and INSERT (with direct hint) statements to load data directly into the ROS.

The COPY command is designed for bulk load operations and can load data into the WOS or the ROS.

Physical Architecture

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

Terminology

The most important terms to understand are host, instance, node, cluster, and database:

Host

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

Instance

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

Node

A node is a host configured to run an instance of Vertica. It is a member of a database cluster (see Node Definition). For a database to have the ability to recover from the failure of a node requires at least three nodes. Vertica Systems, Inc. recommends that you use a minimum of four nodes.

Cluster

A cluster generally refers a collection of hosts or a collection of nodes bound to a database. A cluster is not part of a database definition and thus does not have a name.

Database

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

Logical Schema Design

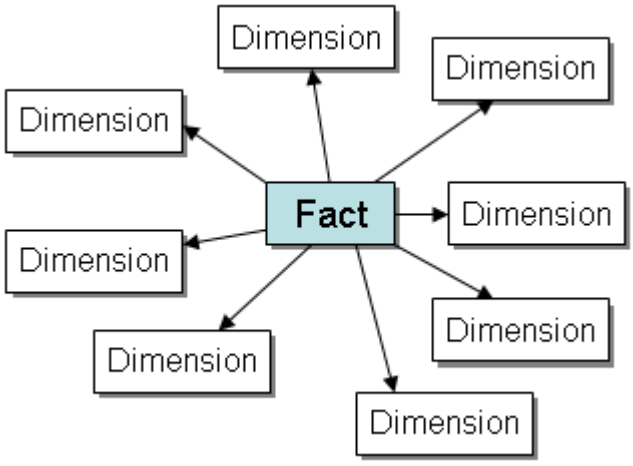
From an SQL user's point of view, Vertica supports the **standard relational data model**, in which a schema consists of a collection of tables, each with a collection of columns (attributes) and rows (tuples), and other SQL objects.

Vertica does not support standard SQL views (stored queries). Instead, it provides projections, which are entirely transparent to SQL users other than the database administrator. These are discussed in the ***Projections*** (page 27) section of this document.

As in most relational systems, specific columns in Vertica tables can be designated as a primary key, or as a foreign key that references a primary key in another table. The tables in a Vertica database are assumed to obey the Dimensional Modeling concepts of a star schema or snowflake schema design. These are typical schema designs used in data warehouses.

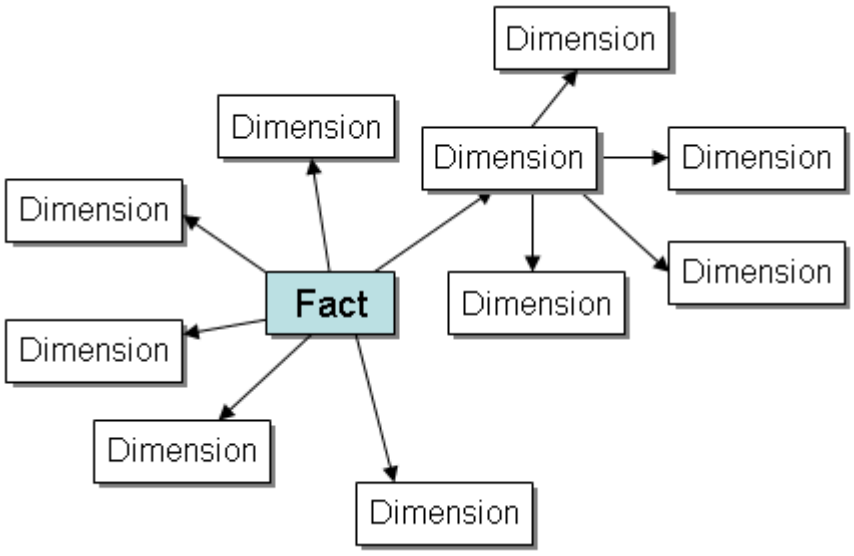
Star Schema

The star schema (sometimes called a star join schema) is the simplest data warehouse schema. In a star schema design there is a central fact table with a large number of tuples, optionally surrounded by a collection of dimension tables, each with a lesser number of tuples. Every dimension table participates in a 1::n join with the fact table. In other words, the primary key of a dimension table is a foreign key in the fact table.



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 other words, the primary key of a dimension table is a foreign key in a fact table or another dimension table.



Projections

Vertica's physical storage consists of collections of table columns called projections. A **projection** is a special case of a materialized view. A materialized view is similar to a standard SQL view with one major exception: the result set is stored on disk rather than computed each time the view is used in a query. The result set is automatically refreshed whenever data values are changed.

Query Performance

Because projections are stored in sort order, a single disk read can fetch many rows at once. Disks read data extremely quickly once positioned, but can take many milliseconds to seek and fetch a single row. For example, if the WHERE clause of a query is (X=1 AND Y=2) and a projection is already sorted on (X,Y), the query runs almost instantaneously.

Another important performance factor is data compression. Sorted data contains repeated sequences that compress very well using, for example, run-length encoding. Compression also makes multiple projections fit within your disk space budget, and thus more extremely fast queries.

Creating Projections

Vertica's Database Designer tool (discussed in a section to follow):

1. analyzes your logical schema, sample data, and sample queries
2. creates a physical schema (projections) in the form of an SQL script that you execute

The query performance of the physical schema produced by the Database Designer depends on the items you provide. For example, if your logical schema includes dimension tables, your sample queries must include at least a minimal set of joins.

Alternatively, you can write your own custom projections as described in the Database Administrator's Guide (Advanced). This option is intended for advanced users only and should be used as advised by ***Technical Support*** (on page 11).

Superprojections

A superprojection is a projection that contains every column of a table in the Logical Schema. A table can have multiple superprojections with different sort orders.

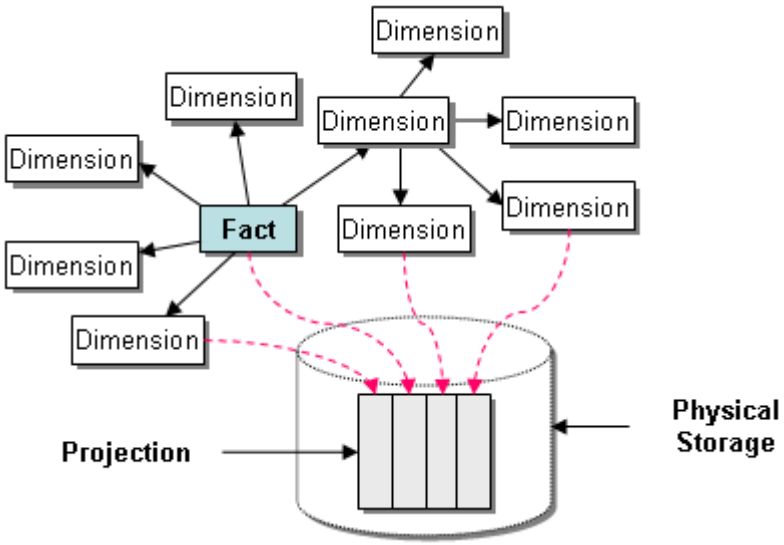
Pre-Join Projections

A pre-join projection stores the result set obtained by joining a single fact table in the logical schema (the anchor table) with one or more dimension tables. The result set is typically sorted for optimal query execution performance. Most query-specific projections are pre-joins.

The anchor table of a join can be:

- the fact table in a star schema
- the central fact table in a snowflake schema
- a dimension table in a snowflake schema that functions as a fact table (with a restriction)

The diagram below shows a pre-join projection consisting of four columns, one from the fact table and three from dimension tables.



Projection Segmentation

To optimize query performance, projections can be segmented (distributed evenly across multiple database nodes). The term **segment** refers to the portion of a projection stored at a particular node. A node can store a collection of segments of various projections.

> Dimension tables are assumed to be much smaller than the fact table and are not segmented; they are replicated on multiple nodes. The system catalogs are also stored redundantly at all nodes; they are small and are updated infrequently.

Segmentation is done using the following methods:

Hash Segmentation

Hash segmentation 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. A hash function is a reproducible method of converting data into a number that can serve as a digital "fingerprint" of the data. 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 may be an excellent choice for hash segmentation.

Hash segmentation is used in segmented projections produced by the Database Designer and is the preferred method of creating custom segmented projections.

Range Segmentation

Range segmentation consists of designating one column of a projection as the **segmentation column**. The range of possible values of a segmentation column is decomposed into disjoint sub-ranges which are allocated to specific nodes.

> Range segmentation based on dates and/or timestamps is not recommended because it results in a skewed data distribution. For example, suppose you segment by year using a DATE column. All rows inserted within the current year would loaded into the same segments on specific nodes. Thus, a query asking for rows inserted during any specific year would be executed only on those nodes. The rest of the nodes would be idle.

Projection Replication

Replication consists of creating duplicates of unsegmented superprojections on multiple nodes (typically all nodes in a database). Replication has two purposes:

- distributed query execution
- K-Safety (a pair of replicated superprojections can be used as buddy projections for the purpose of recovery)

Vertica provides SQL syntax that automatically replicates an unsegmented superprojection across all nodes in a database (or specific nodes). Automatic replication is used by the Database Designer and is the preferred method of creating custom replicated projections.

Database Designer

Vertica's Database Designer tool:

1. analyzes your logical schema, sample data, and sample queries
2. creates a physical schema design (projections) in the form of an SQL script that you execute

In most cases, the designs created by the Database Designer provide excellent query performance within physical constraints. The Database Designer uses very complex strategies to provide excellent ad-hoc query performance while using disk space efficiently.

You can, however, design custom projections as described in the Database Administrator's Guide (Advanced). This option is intended for advanced users only and should be used as advised by ***Technical Support*** (on page 11).

For More Information

- ***Projections*** (page 27)
- The Physical Schema in the Database Administrator's Guide

Setting Up the Database

The process of setting up a Vertica database is described in detail in the Database Administrator's Guide. It involves:

Preparing SQL Scripts and Data Files

The first part of the setup procedure can be done well before Vertica is installed. It consists of preparing the following files:

- logical schema script
- sample query script (training set)
- loadable data files (dimension table and fact tables)
- load scripts

Creating the Database

This part requires that you have installed Vertica on at least one host. The following steps are not in sequential order:

You will use the Administration Tools to:

- create a database
- connect to the database
- run the ***Database Designer*** (page 33)

You will use the vsql interactive interface to execute SQL scripts that:

- create tables and constraints
- create projections

Test the Empty Database

- test for sufficient projections using the sample query script
- test the projections for K-Safety

Test the Partially Loaded Database

- load the dimension tables
- partially load the fact table
- check system resource usage
- check query execution times
- check projection usage

Complete the Fact Table Load

- monitor system usage
- complete the fact table load

Setting Up Security

- set up database users and privileges

Set Up Incremental Loads

- set up periodic or "trickle" loads

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. If your administrative tasks go beyond those provided by the tools, contact ***Technical Support*** (page 11) for information.

> 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.

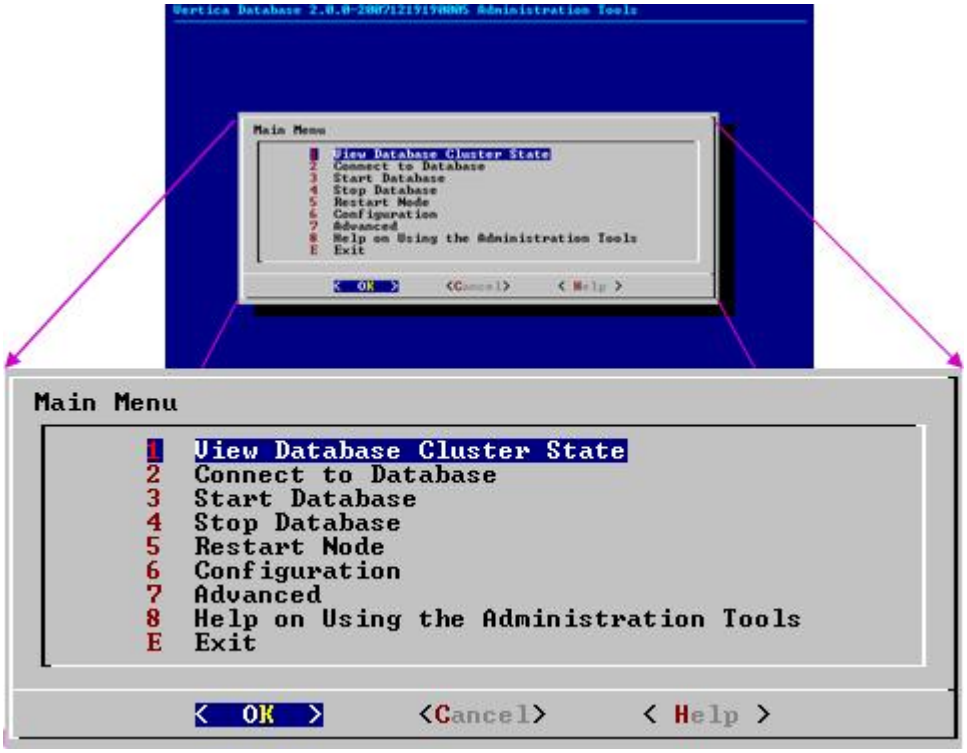
Running the Administration Tools

At the Linux command line:

$ **/opt/vertica/bin/adminTools [-t | --tool]** *toolname* **[** *options* **]**

toolname	is one of the tools described in the Administration Tools Reference.	
options	`-h` `--help`	shows a brief help message and exits.
	`-a` `--help_all`	lists all command line sub-commands and options as described in the Writing Administration Tools Scripts section of the Database Administrator's Guide (Advanced).

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.



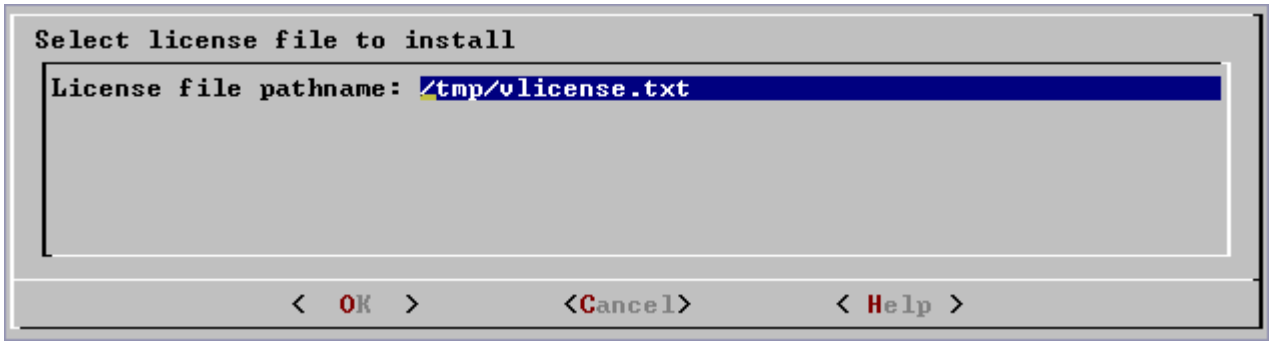
If you are unfamiliar with this type of graphical user interface, read Using the Graphical User Interface before you do anything else.

First Time Only

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

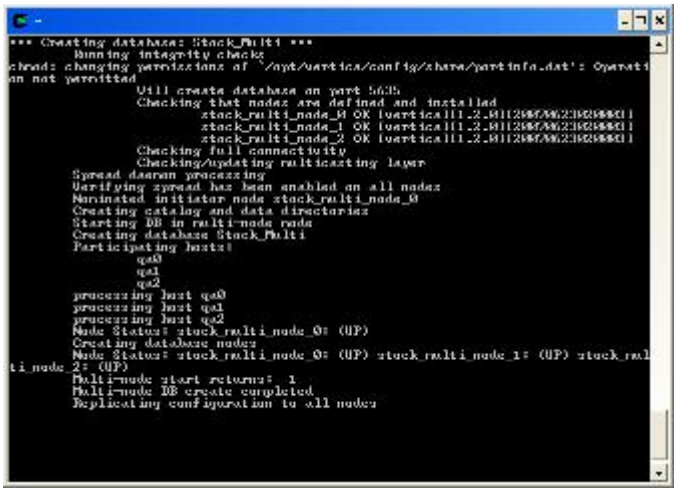
1. the EULA (license agreement). Type **accept** to proceed.

2. a form asking for the pathname to the license key file that you downloaded from the Vertica Web site. The default pathname is `/tmp/vlicense.key`. If this is correct, press OK. Otherwise, enter the **absolute pathname** of the file and press OK.



Between Dialogs

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



Connecting to a Database

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

- interactively using the **vsql** client, as described in Connecting with vsql in the SQL Programmer's Guide.

  vsql is the Vertica implementation of psql, a character-based, interactive, front-end that is part of ***PostgreSQL*** (http://www.postgresql.org/) and used by other database management systems. It allows you to type in SQL statements and see the results. It also provides a number of meta-commands and various shell-like features to facilitate writing scripts and automating a wide variety of tasks.

  You can run psql on any node within a database. To start psql, use the Administration Tools or the shell command described in Connecting Interactively.

- programmatically using the **JDBC** driver provided by Vertica, as described in Connecting With JDBC in the SQL Programmer's Guide.

  JDBC (Java Database Connectivity) is a call-level API that provides connectivity between Java programs and data sources (SQL databases and other non-relational data sources, such as spreadsheets or flat files). JDBC is included in the Java 2 standard and enterprise editions.

- programmatically using the **ODBC** driver provided by Vertica, as described in Connecting With ODBC in the SQL Programmer's Guide.

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

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

Managing Database Security

Authentication and authorization in a Vertica database are based on SQL standard users and privileges except that each database has a superuser who can bypass the authorization mechanism. The superuser has the same name as the Linux user that created the database (the Database Administrator). Creating a database allows the DBA to specify the superuser password or an empty string (no password).

> WARNING: If the DBA does not specify a superuser password at database creation time, the database authentication method is permanently set to "trust," which allows any user to log in without supply a password.

The superuser can:

- create normal database users but not other superusers
- drop normal database users
- grant privileges to normal database users
- revoke privileges granted to normal database users

SQL Overview

SQL (Structured Query Language) is a widely-used, industry standard data definition and data manipulation language for relational databases.

Vertica Support for ANSI SQL Standards

Vertica SQL supports a subset of **ANSI SQL 99**. Over time, Vertica SQL will enlarge and eventually converge with ANSI SQL 99. For information about ANSI SQL 99 see:

- ***BNF Grammar for SQL-99*** (http://savage.net.au/SQL/sql-99.bnf.html)
- ***Index of /education/modules/dbms/SQL99***
  (http://www.ncb.ernet.in/education/modules/dbms/SQL99/)

Vertica Use of PostgreSQL

In addition to using vsql as an interactive front-end, Vertica SQL supports a subset of the PostgreSQL language definition. For information about PostgreSQL see:

- ***PostgreSQL 8.0.12 Documentation*** (http://www.postgresql.org/docs/8.0/interactive/sql-commands.html)

Vertica Major Extensions to SQL

Vertica provides several extensions to SQL that allow you to use the unique aspects of its column store architecture:

- **AT EPOCH LATEST SELECT..**.

  runs a SQL query in snapshot isolation mode in which it does not hold locks or block other processes such as data loads.
- **AT TIME 'timestamp' SELECT...**

  runs historical queries against a snapshot of the database a specific date and time.
- **CONSTRAINT ... CORRELATION (column) REFERENCES (column)**

  captures Functional Dependencies that can be used by the Database Designer to produce more efficient projections.
- **COPY**

  is used for bulk loading data. It reads data from a text file and inserts tuples into the WOS (Write Optimized Store) or directly into the ROS (Read Optimized Store).
- **CREATE/DROP/ALTER PROJECTION**

  are used for manipulating projections as described in the Database Administrator's Guide (Advanced). CREATE PROJECTION commands are generated for you by the Database Designer as described in the Database Administrator's Guide.
- **SELECT Vertica Function**

executes special Vertica functions.

- **SET DATESTYLE**

  chooses the format in which date/time values are displayed.

- **SET SEARCH_PATH**

  specifies the order in which Vertica searches through multiple schemas when a SQL statement contains an unqualified table name.

- **SET TIMEZONE**

  specifies the TIMEZONE run-time parameter for the current session.

- **SHOW**

  displays run-time parameters for the current session.

Support for Historical Queries

Unlike most databases, the DELETE command in Vertica does not actually delete data; it simply marks tuples as deleted. The UPDATE command actually does an INSERT and a DELETE. This behavior is necessary for historical queries. You can control how much historical data is stored on disk.

Non-Standard Syntax and Semantics

In case of non-standard SQL syntax or semantics, Vertica SQL follows Oracle whenever possible. For Oracle SQL documentation, you will need a web account to access the library:

- ***Oracle Database 10g Documentation Library***
  (http://www.oracle.com/pls/db102/homepage)

Joins

Vertica supports only:

- standard inner equi-joins in the WHERE clause

primary-key/foreign-key (1:n) star and snowflake joins Vertica does not support:

- self-joins (use temporary tables for this purpose)
- outer joins
- compound keys

Transactions

Vertica supports conventional SQL transactions with standard ACID properties. Specifically:

- Vertica supports ANSI SQL 92 style implicit transactions. You do not need to execute a BEGIN or START TRANSACTION command.
- Vertica does not use a redo/undo log or two-phase commit.

- The COPY command automatically commits itself and any current transaction. Vertica recommends that you COMMIT or ROLLBACK the current transaction before using COPY.

Session-Scoped Isolation Levels

Vertica supports a **subset of the standard SQL isolation levels and access modes** for a user session as described in SERIALIZABLE Isolation and READ COMMITTED Isolation. These modes determine what data a transaction can access when other transactions are running concurrently. You can change the default isolation level for a user session using the SET SESSION CHARACTERISTICS command.

> Session-scoped isolation levels do not apply to temporary tables, which are scoped to the current transaction. They do not take table locks, are only visible to one user, and their contents are always visible regardless of advancing epochs and COMMITs.

Query-Scoped Isolation Levels

Snapshot Isolation (historical queries) are part of the Vertica query syntax:

```
[ AT EPOCH LATEST ] | [ AT TIME 'timestamp' ] SELECT ...
```

AT EPOCH LATEST allows queries to access all historical data up to but not including the current epoch . It does not hold locks and does not block write operations. This provides a **profound query performance advantage**. It does not apply to temporary tables.

Automatic Rollback

When an error occurs or a user session is disconnected, the current transaction automatically rolls back. This behavior may be different from other databases.

Running Queries

This topic provides a brief overview of query execution in Vertica. For detailed information about writing and executing queries, see the SQL Programmer's Guide.

Snapshot Isolation Mode

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

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

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

Historical Queries

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

```
AT TIME 'timestamp' SELECT...
```

The command queries all data in the database up to and including the epoch representing the specified date and time without holding a lock or blocking write operations.

For this reason, the DELETE command in Vertica does not actually delete data; it simply marks tuples as deleted. The UPDATE command actually does an INSERT and a DELETE.

Parallel Query Execution

The query optimizer chooses the projections to use for any given incoming query and forms an optimized query plan for the ones chosen. The query plan that the optimizer produces is decomposed into “mini-plans” which are run at various nodes in parallel, interspersed with data movement operations. A local executor at each node runs its assigned mini-plan.

Loading and Modifying Data

The SQL data manipulation language (DML) commands INSERT, UPDATE, and DELETE perform the same functions in Vertica that they do in a row-oriented database. They can be intermixed and follow the SQL-92 transaction model.

The COPY command is designed for bulk loading data into a Vertica database. It reads data from a delimited text file and inserts tuples either into the WOS (memory) or directly into the ROS (disk). The COPY command automatically commits itself and any current transaction but is not atomic: some rows may be rejected.

The automatic tuple mover (ATM) has two modes: NORMAL and BULKLOAD. The latter is optimized for loading large volumes of data into the ROS.

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

Failure Recovery

Vertica has a unique approach to failure recovery that is based on the distributed nature of a database. If the physical schema design of a database is K-Safe, the database can lose a node and keep on running as if nothing happened. When the lost node comes back online and rejoins the database, it recovers its lost objects by querying the other nodes. K-Safety is determined by whether or not the ***projections*** (page 27) in the physical schema design meet certain redundancy criteria.

In 2.0.5-0, a database can continue to run if a single node goes down. However, another node failure would cause the database to automatically shut down and can result in loss of data.

Getting Started

To get started using Vertica, follow the tutorial procedures presented in the Quick Start. The tutorials require that you to install Vertica on one or more hosts as described in the Installation Guide.

Index

V

W