

Concepts Guide

HP Vertica Analytic Database

Software Version: 7.0.x



Document Release Date: 2/24/2014



Legal Notices

Warranty

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

The information contained herein is subject to change without notice.

Restricted Rights Legend

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

Copyright Notice

© Copyright 2006 - 2014 Hewlett-Packard Development Company, L.P.

Trademark Notices

Adobe® is a trademark of Adobe Systems Incorporated.

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

UNIX® is a registered trademark of The Open Group.

Contents

Contents	3
The HP Vertica Approach	7
Column Storage	7
Compression	7
Clustering	8
Continuous Performance	8
HP Vertica Components	11
Column Store Architecture with FlexStore	11
How FlexStore™ Enhances Your Column-Based Architecture	11
Architecture of the HP Vertica Cluster	12
Terminology	12
Data Encoding and Compression	13
Encoding	13
Compression	13
K-Safety	13
K-Safety Example	14
K-Safety Requirements	16
Determining K-Safety	16
Monitoring K-Safety	17
Finding Critical Nodes	17
High Availability and Recovery	17
K-Safety Requirements	17
Determining K-Safety	18
Monitoring K-Safety	18
Loss of K-Safety	18
High Availability With Projections	20
Replication (Unsegmented Projections)	20
Buddy Projections (Segmented Projections)	20
High Availability With Fault Groups	21

Automatic fault groups	22
User-defined fault groups	22
Example cluster topology	22
How to create fault groups	23
Hybrid Storage Model	23
Logical Schema	25
Physical Schema	25
How Projections Are Created	26
Anatomy of a Projection	26
Column List and Encoding	27
Base Query	27
Sort Order	27
Segmentation	28
Projection Concepts	28
Projection Performance	28
Encoding and Compression	28
Sort Order	28
Segmentation	29
Projection Segmentation	29
Hash Segmentation	29
Projection Naming	29
Database Setup	30
Prepare SQL Scripts and Data Files	31
Create the Database	31
Test the Empty Database	31
Test the Partially-Loaded Database	31
Complete the Fact Table Load	32
Set up Security	32
Set up Incremental Loads	32
Database Connections	32
The Administration Tools	33

Running the Administration Tools	33
First Time Only	34
Between Dialogs	35
Management Console	36
What You Can Do with Management Console	36
How to Get MC	36
What You Need to Know	37
Management Console Architecture	37
MC Components	37
Application/web Server	38
MC Agents	38
Management Console Security	39
OAuth and SSL	39
User Authentication and Access	39
Management Console Home Page	40
Database Designer	40
Database Security	41
Data Loading and Modification	42
Workload Management	42
SQL Overview	45
HP Vertica Support for ANSI SQL Standards	45
Support for Historical Queries	45
Joins	45
Transactions	45
About Query Execution	47
Snapshot Isolation Mode	47
Historical Queries	47
Transactions	49
Implementation Details	49
Automatic Rollback	49
Savepoints	50

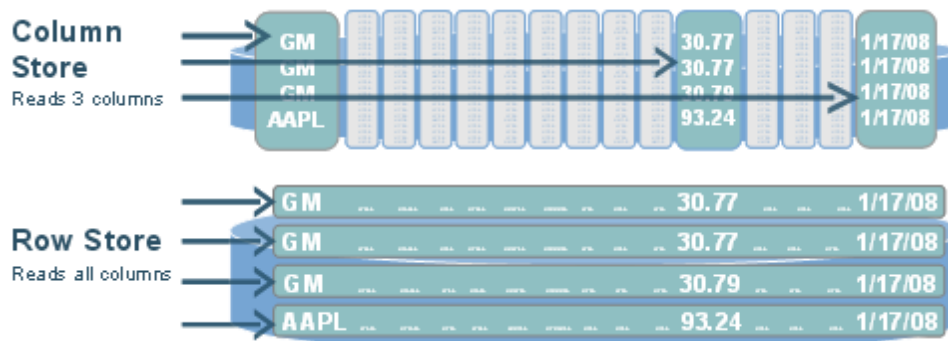
READ COMMITTED Isolation	50
SERIALIZABLE Isolation	54
International Languages and Character Sets	57
Unicode Character Encoding	57
Locales	57
String Functions	57
Character String Literals	58
Extending HP Vertica	59
User-Defined SQL Functions	59
User Defined Extensions and User Defined Functions	59
Get Started	61
We appreciate your feedback!	63

The HP Vertica Approach

HP Vertica is built from the Ground Up on the 4 C's:

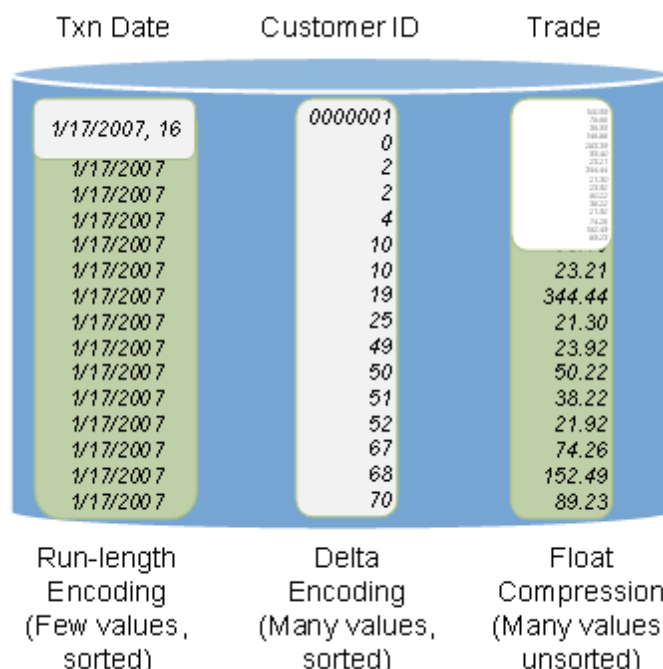
Column Storage

Stores data the way it is typically queried for best performance. Column storage is ideal for read-intensive workloads because it can dramatically reduce disk I/O.



Compression

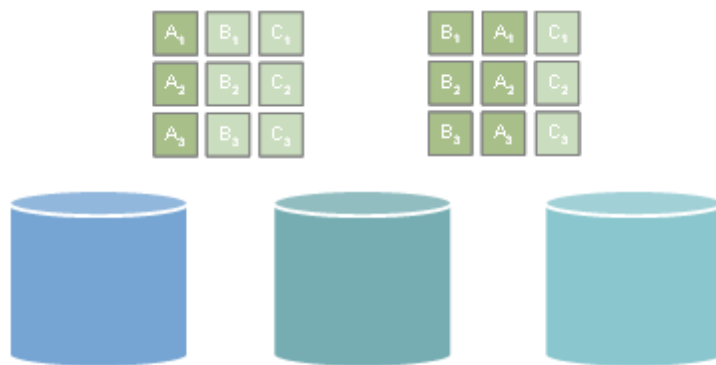
Stores more data, provides more views, and uses less hardware, which lets you keep much more historical data in physical storage.



- When similar data is grouped, you have even more compression options
- HP Vertica applies over twelve compression schemas
 - Dependent on data
 - System chooses which to apply
 - NULLs take virtually no space
- Typically see 50% - 90% compression
- HP Vertica queries data in encoded form

Clustering

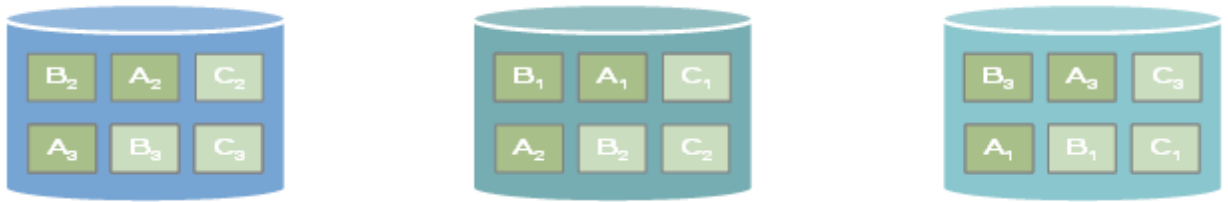
Lets you scale out your database cluster easily by adding more hardware.



- Columns are duplicated across cluster nodes. If one machine goes down, you still have a copy:
 - Data warehouse log based recovery is impractical
 - Instead, store enough projections for **K-safety**
- New cluster node queries existing nodes for the data it needs
 - Rebuilds missing objects from other nodes
 - Another benefit of multiple sort orders

Continuous Performance

Queries and loads data 24x7 with virtually no database administration.



- Concurrent loading and querying means that you get real-time views and eliminate nightly *load windows*.
- On-the-fly schema changes mean that you can add columns and projections without database downtime.
- Automatic data replication, failover, and recovery provides for *active* redundancy, which increases performance. Nodes recover automatically by querying the system.

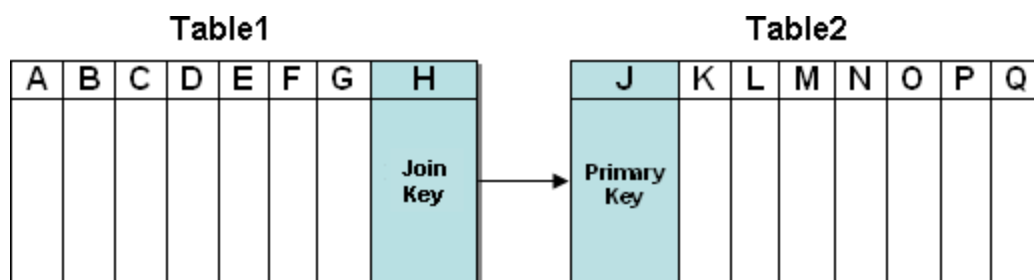
HP Vertica Components

This section describes the unique components that make up HP Vertica.

Column Store Architecture with FlexStore

Traditionally databases were designed for **OLTP** and used a row-store architecture. 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 might be or how many columns are actually needed. Often, analytic queries access only two or three columns from tables containing up to several hundred columns, resulting in a lot of unnecessary data retrieval.

Unlike other **RDBMS**, HP Vertica reads the columns from database objects called **projections**, which are described in the [Physical Schema](#) section of this guide. No resources are wasted 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 run this query:

```
SELECT A, C, N
FROM Table1 JOIN Table2
ON H = J;
```

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

How FlexStore™ Enhances Your Column-Based Architecture

FlexStore™ is a combination of physical design, database storage, and query execution techniques that HP Vertica applies to the database to optimize it for the analytic workload supports at the time. These techniques include:

- **Column grouping.** Refers to a technique for storing column data together to optimize I/O during query processing. Such groupings can be advantageous for correlated columns and for columns that are always accessed together for projecting, but not for filtering or joining. Grouped

columns also benefit from special compression and retrieval techniques. An example might be bid and ask prices in a TickStore database. Column grouping is described in the [CREATE PROJECTION](#) statement's GROUPED clause.

- **Intelligent disk use.** Allows optimizing performance to place frequently-needed disk resources onto faster media. This includes mixing solid-state and rotating "disk" storage in the database nodes. You can prioritize disk use for:
 - data versus temporary storage
 - storage for columns in a projection

See [Working With Storage Locations](#) in the Administrator's Guide for details.

- **Fast deletes.** Refers to projection design techniques to speed up delete processing, together with the function `EVALUATE_DELETE_PERFORMANCE()` to help identify potential delete problems. See [Optimizing Deletes and Updates for Performance](#) in the Administrator's Guide for details.

Architecture of the HP Vertica Cluster

Terminology

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

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

Host —

A computer system with a 32-bit (non-production use only) 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 HP Vertica consists of the running HP Vertica process and disk storage (catalog and data) on a host. Only one instance of HP Vertica can be running on a host at any time.

Node —

A host configured to run an instance of HP Vertica. It is a member of a database cluster. For a database to have the ability to recover from the failure of a node requires at least three nodes. HP recommends that you use a minimum of four nodes.

Cluster —

Refers a collection of hosts (nodes) bound to a database. A cluster is not part of a database definition and does not have a name.

Database —

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

Data Encoding and Compression

Encoding

The process of converting data into a standard format. In HP Vertica, encoded data can be processed directly, while compressed data cannot. HP Vertica uses a number of different encoding strategies, depending on column data type, table cardinality, and sort order.

The query executor in HP 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

The process of transforming data into a compact format. Compressed data cannot be directly processed; it must first be decompressed. HP Vertica uses integer packing for unencoded integers and LZ0 for compressible data. Although compression is generally considered to be a form of encoding, the terms have different meanings in HP 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.

HP 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.

K-Safety

K-safety is a measure of fault tolerance in the database cluster. The value K represents the number of replicas of the data in the database that exist in the database cluster. These replicas allow other nodes to take over for failed nodes, allowing the database to continue running while still ensuring data integrity. If more than K nodes in the database fail, some of the data in the database may become unavailable. In that case, the database is considered unsafe and automatically shuts down.

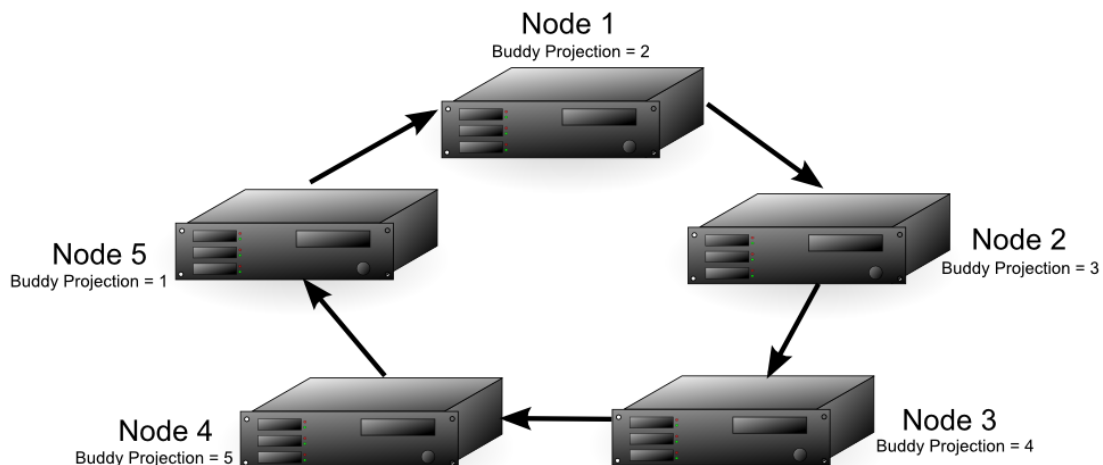
It is possible for an HP Vertica database to have more than K nodes fail and still continue running safely, because the database continues to run as long as every data segment is available on at least one functioning cluster node. Potentially, up to half the nodes in a database with a K-safety

level of 1 could fail without causing the database to shut down. As long as the data on each failed node is available from another active node, the database continues to run.

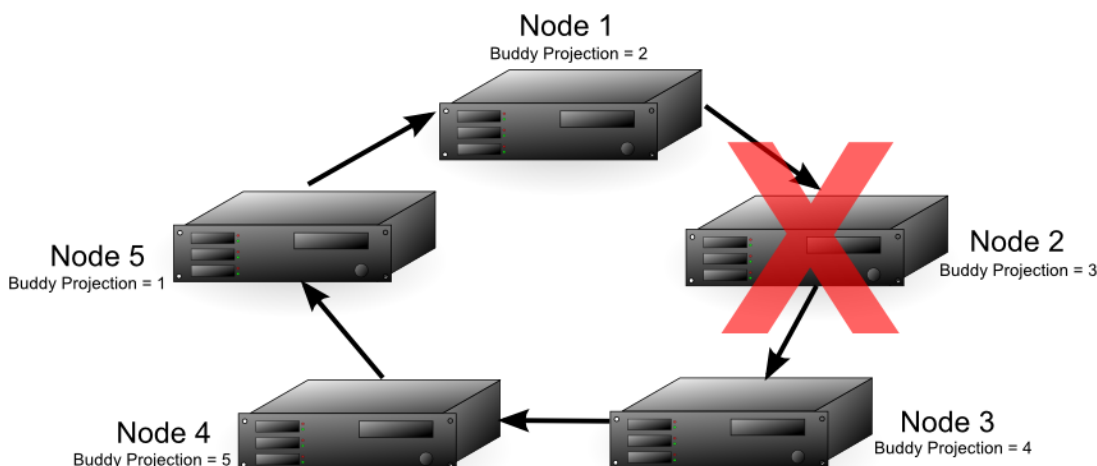
Note: If half or more of the nodes in the database cluster fail, the database will automatically shut down even if all of the data in the database is technically available from replicas. This behavior prevents issues due to network partitioning.

In HP 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, HP recommends using the **Database Designer**.

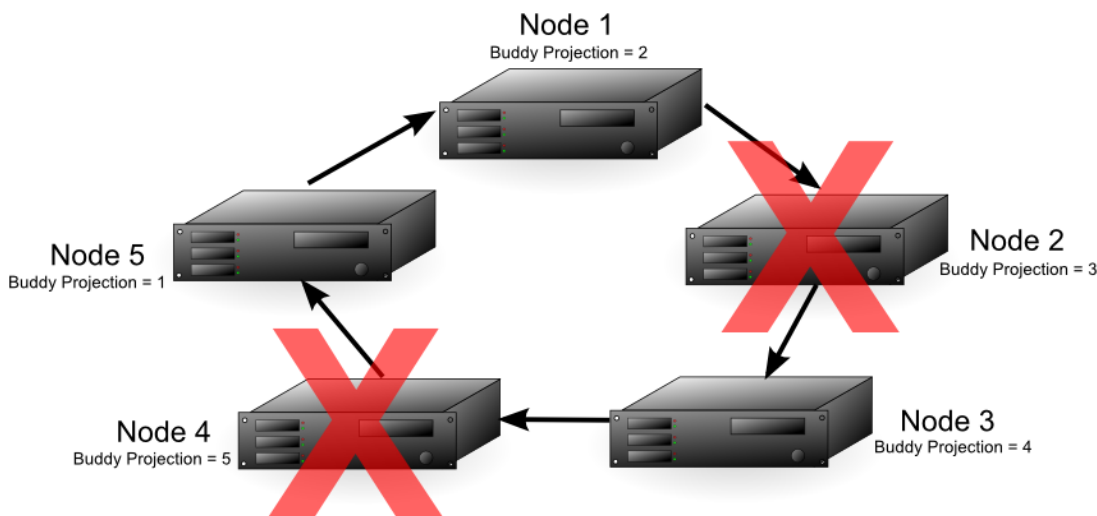
K-Safety Example



The diagram above shows a 5-node cluster that has a K-safety level of 1. Each of the nodes contains buddy projections for the data stored in the next higher node (node 1 has buddy projections for node 2, node 2 has buddy projections for node 3, etc.). Any of the nodes in the cluster could fail, and the database would still be able to continue running (although with lower performance, since one of the nodes has to handle its own workload and the workload of the failed node).

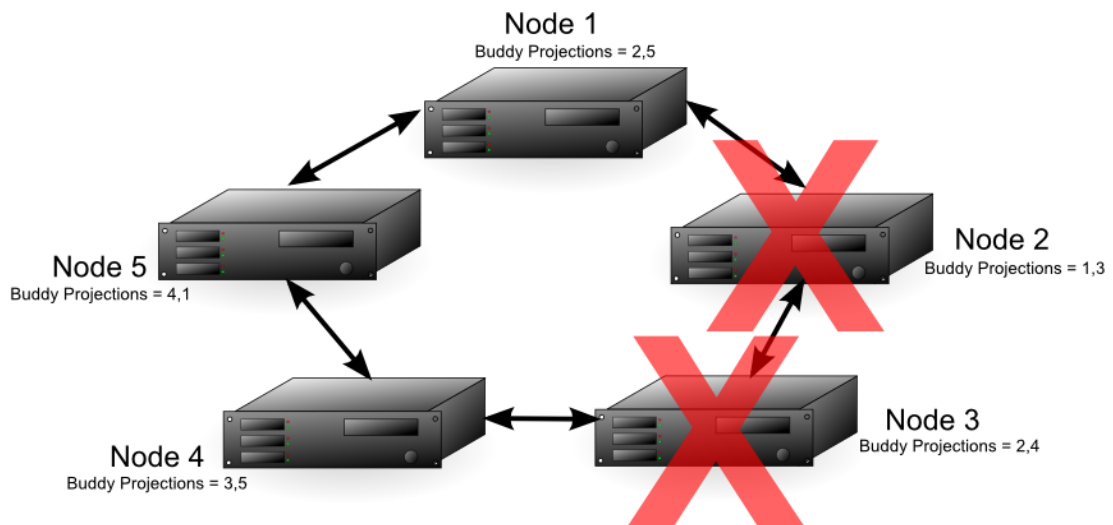


If node 2 fails, node 1 handles requests on its behalf using its replica of node 2's data, in addition to performing its own role in processing requests. The fault tolerance of the database will fall from 1 to 0, since a single node could cause the database to become unsafe. In this example, if either node 1 or node 3 fail, the database would become unsafe since not all of its data would be available. If node 1 fails, then node 2's data will no longer be available. If node 3 fails, its data will no longer be available, since node 2 is also down and could not fill in for it. In this case, nodes 1 and 3 are considered **critical nodes**. In a database with a K-safety level of 1, the node that contains the buddy projection of a failed node and the node whose buddy projections were on the failed node will always become critical nodes.



With node 2 down, either node 4 or 5 in the cluster could fail and the database would still have all of its data available. For example, if node 4 fails, node 3 is able to use its buddy projections to fill in for it. In this situation, any further loss of nodes would result in a database shutdown, since all of the nodes in the cluster are now critical nodes. (In addition, if one more node were to fail, half or more of

the nodes would be down, requiring HP Vertica to automatically shut down, no matter if all of the data were available or not.)



In a database with a K-safety level of 2, any node in the cluster could fail after node 2 and the database would be able to continue running. For example, if in the 5-node cluster each node contained buddy projections for both its neighbors (for example, node 1 contained buddy projections for both node 5 and node 2), then nodes 2 and 3 could fail and the database could continue running. Node 1 could fill in for node 2, and node 4 could fill in for node 3. Due to the requirement that half or more nodes in the cluster be available in order for the database to continue running, the cluster could not continue running if node 5 were to fail as well, even though nodes 1 and 4 both have buddy projections for its data.

K-Safety Requirements

When creating projections with the Database Designer, projection definitions that meet K-Safe design requirements are recommended and marked with the K-safety level. Note the output from running the optimized design script generated by the Database Designer in the following example:

```
=> \i VMart_Schema_design_opt_1.sql

CREATE PROJECTION
CREATE PROJECTION
mark_design_ksafe
-----
Marked design 1-safe
(1 row)
```

Determining K-Safety

To determine the K-safety state of a running database, execute the following SQL command:

```
=> SELECT current_fault_tolerance FROM system;
```



```
current_fault_tolerance
-----
1
(1 row)
```

Monitoring K-Safety

Monitoring tables can be accessed programmatically to enable external actions, such as alerts. You monitor the K-safety level by polling the `SYSTEM` table column and checking the value. See [SYSTEM](#) in the SQL Reference Manual.

Finding Critical Nodes

You can view a list of critical nodes in your database by querying the `v_monitor.critical_nodes` table:

```
=> SELECT * FROM v_monitor.critical_nodes;
node_name
-----
v_exampleDB_node0001
v_exampleDB_node0003
(2 rows)
```

High Availability and Recovery

HP Vertica's unique approach to failure recovery is based on the distributed nature of a database. An HP Vertica database is said to be **K-safe** if any node can fail at any given time without causing the database to shut down. When the lost node comes back online and rejoins the database, it recovers its lost objects by querying the other nodes. See [Managing Nodes](#) and [Monitoring Recovery](#) in the Administrator's Guide.

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

K-Safety Requirements

Your database must have a minimum number of nodes to be able to have a K-safety level greater than zero, as shown in the following table.:

K-level	Number of Nodes Required
0	1+
1	3+
2	5+
K	$(K+1)/2$

Note: HP Vertica does not officially support values of K higher than 2.

The value of K can be 1 or 2 only when the physical schema design meets certain redundancy requirements. See [Physical Schema](#). To create designs that are K-safe, HP recommends that you use the **Database Designer**.

By default, HP Vertica creates K-safe superprojections when the database has a K-safety greater than 0 ($K > 0$). When creating projections with Database Designer, projection definitions that meet K-safe design requirements are recommended and marked with the K-safety level. Database Designer creates a script that uses the [MARK_DESIGN_KSAFE](#) function to set the K-safety of the physical schema to 1:

```
=> SELECT MARK_DESIGN_KSAFE (1);
      MARK_DESIGN_KSAFE
-----
Marked design 1-safe
(1 row)
```

Determining K-Safety

To determine the K-safety state of a running database, run the following SQL command:

```
-> SELECT current_fault_tolerance FROM system;
      current_fault_tolerance
-----
                           1
(1 row)
```

Monitoring K-Safety

Monitoring tables can be accessed programmatically to enable external actions, such as alerts. You monitor the K-safety level by polling the SYSTEM table and checking the value. See [SYSTEM](#) in the SQL Reference Manual.

Loss of K-Safety

When K nodes in your cluster fail, your database continues to run, although performance is affected. Further node failures could potentially cause the database to shut down if the failed node's

data is not available from another functioning node in the cluster.

High Availability With Projections

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

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

Replication (Unsegmented Projections)

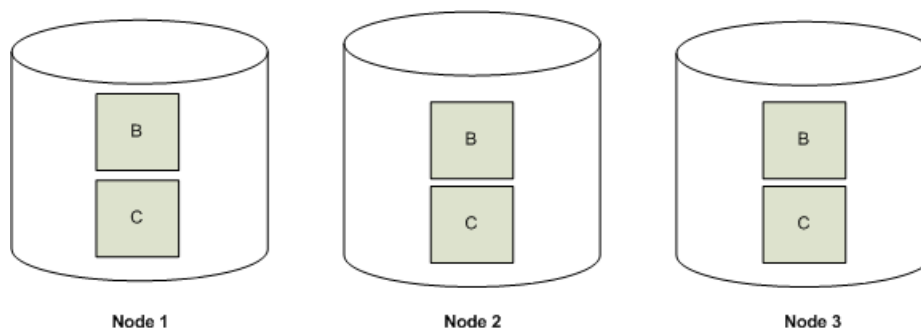
When it creates projections, Database Designer does not segment projections for small tables; rather it replicates them, creating and storing duplicates of these projections on all nodes within the database.

Replication ensures:

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

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

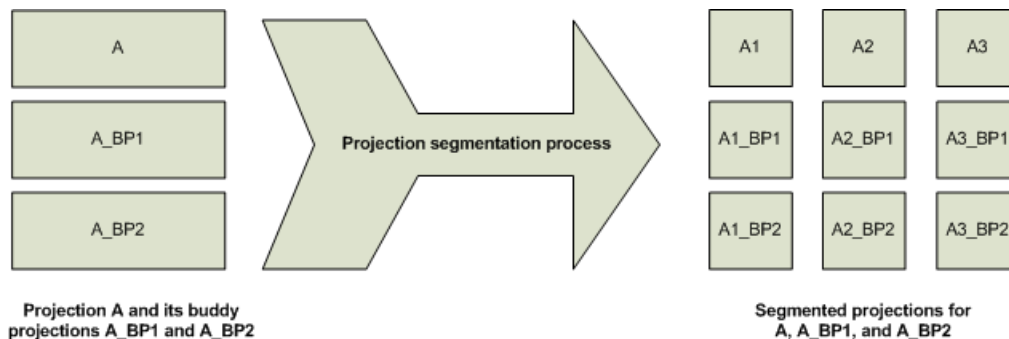
The following illustration shows two projections, B and C, replicated across a three node cluster.



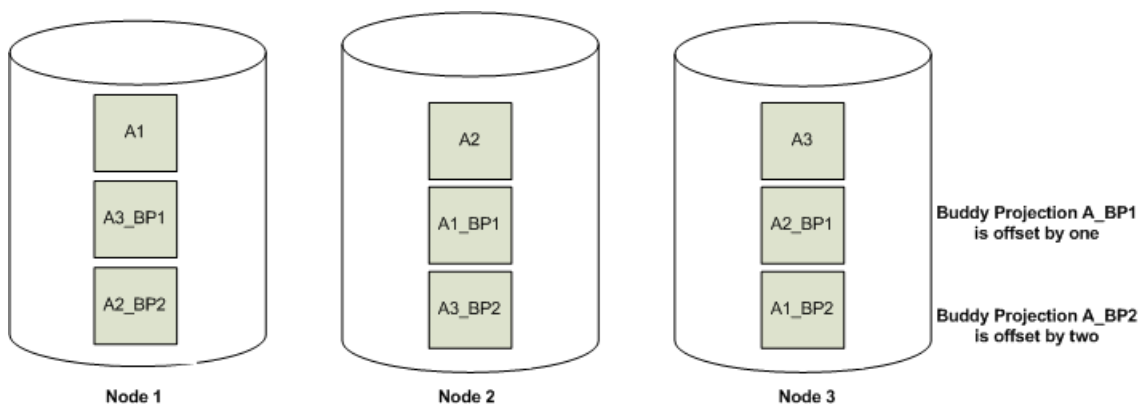
Buddy Projections (Segmented Projections)

HP Vertica creates *buddy projections*, which are copies of segmented projections that are distributed across database nodes. (See [Projection Segmentation](#).) HP Vertica ensures that segments that contain the same data are distributed to different nodes. This ensures that if a node goes down, all the data is available on the remaining nodes. HP Vertica distributes segments to different nodes by using offsets. For example, segments that comprise the first buddy projection (A_BP1) would be offset from projection A by one node and segments from the second buddy projection (A_BP2) would be offset from projection A by two nodes.

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



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



This example illustrates how one projection and its buddies are segmented across nodes. However, each node can store a collection of segments from various projections.

High Availability With Fault Groups

Fault groups let you configure HP Vertica for your physical cluster layout in order to reduce the risk of correlated failures inherent in your environment. Correlated failures occur when two or more nodes fail as a result of a single failure event. These failures often occur due to shared resources, such as power, networking, or storage.

Although correlated failure scenarios cannot always be avoided, HP Vertica helps you minimize the risk of failure by letting you define fault groups on your cluster. HP Vertica then uses your fault groups definitions to distribute data segments across the cluster, so the database stays up if a single failure event occurs.

The following list describes some of the causes of correlated failures:

- Servers in the same data center machine rack:
 - Power loss to a machine rack could cause all nodes on those servers to fail
 - User error during maintenance could affect an entire machine rack
- Multiple virtual machines that reside on a single VM host server
- Nodes that use other shared infrastructure that could cause correlated failures of a subset of nodes

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

HP Vertica supports complex, hierarchical fault groups of different shapes and sizes. Fault groups are integrated with [elastic cluster](#) and [large cluster](#) arrangements to provide added cluster flexibility and reliability.

Automatic fault groups

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

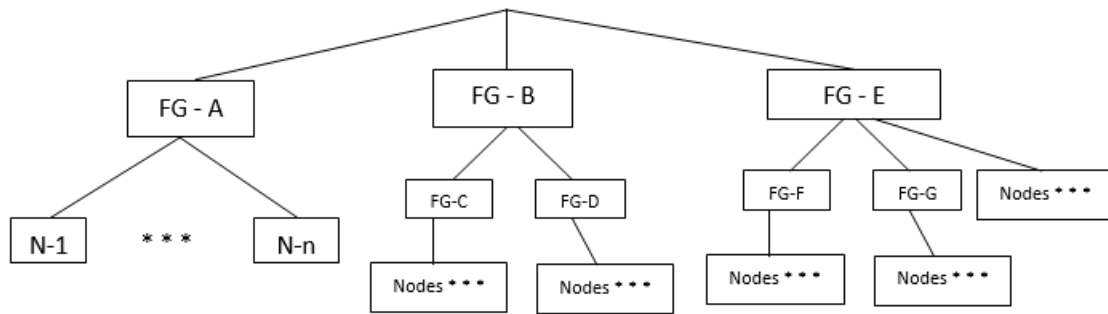
User-defined fault groups

If your cluster layout has the potential for correlated failures, or if you want to influence which cluster hosts manage control messaging, you should define your own fault groups.

Example cluster topology

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

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



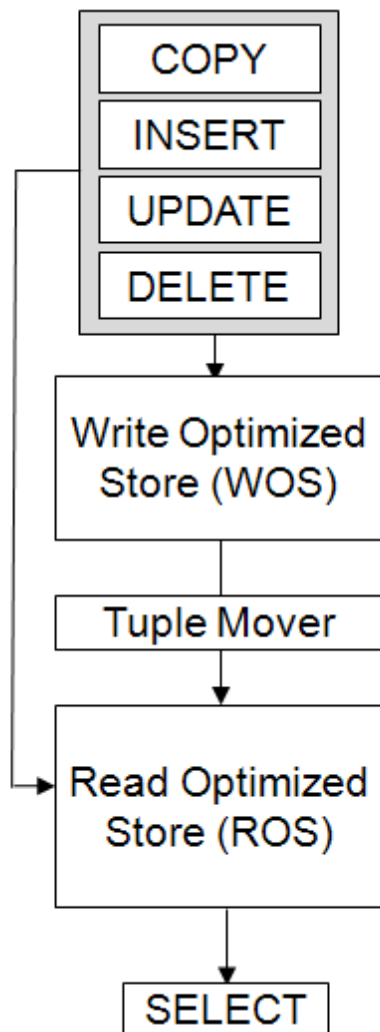
How to create fault groups

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

The simplest way to define fault groups is to create an input file of your cluster arrangement. You pass the file to an HP Vertica-supplied script, which returns to the console the SQL statements you need to run. See [Fault Groups](#) in the Administrator's Guide for details.

Hybrid Storage Model

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



Write Optimized Store (WOS) is a memory-resident data structure for storing INSERT, UPDATE, DELETE, and COPY (without DIRECT hint) actions. Like the Read Optimized Store (ROS), the WOS is arranged by projection. To support very fast data load speeds, the WOS stores records without data compression or indexing. A projection in the WOS is sorted only when it is queried. It remains sorted as long as no further data is inserted into it. The WOS organizes data by epoch and holds both committed and uncommitted transaction data.

The Tuple Mover (TM) is the HP Vertica database optimizer component that moves data from memory (WOS) to disk (ROS). The TM also combines small ROS containers into larger ones, and purges deleted data. During moveout operations, the TM is also responsible for adhering to any storage policies that are in effect for the storage location. The Tuple Mover runs in the background, performing some tasks automatically (ATM) at time intervals determined by its configuration parameters. For information about changing the TM configuration parameters, see [Tuple Mover Parameters](#) in the Administrator's Guide for further information.

The Read Optimized Store (ROS) is a highly optimized, read-oriented, disk storage structure, organized by projection. The ROS 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.

Note: HP Vertica allows optional spaces before and after the plus sign in direct hints (between the `/*` and the `+`).

A grouped ROS is a highly-optimized, read-oriented physical storage structure organized by projection. A grouped ROS makes heavy use of compression and indexing. Unlike a ROS, however, a grouped ROS stores data for two or more grouped columns in one disk file.

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

Logical Schema

Designing a logical schema for an HP 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. HP Vertica supports any relational schema design of your choice.

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

Physical Schema

In traditional database architectures, data is primarily stored in tables. Additionally, secondary tuning structures such as index and materialized view structures are created for improved query performance. In contrast, tables do not occupy any physical storage at all in HP Vertica. Instead, physical storage consists of collections of table columns called projections.

Projections store data in a format that optimizes query execution. They are similar to materialized views in that they store result sets on disk rather than compute them each time they are used in a query. The result sets are automatically refreshed whenever data values are inserted or loaded.

Using projections provides the following benefits:

- Projections compress and encode data to greatly reduce the space required for storing data. Additionally, HP Vertica operates on the encoded data representation whenever possible to avoid the cost of decoding. This combination of compression and encoding optimizes disk space while maximizing query performance. See [Projection Performance](#).
- Projections can be segmented or replicated across database nodes depending on their size. For instance, projections for large tables can be segmented and distributed across all nodes. Unsegmented projections for small tables can be replicated across all nodes in the database. See [Projection Performance](#).
- Projections are transparent to end-users of SQL. The HP Vertica query optimizer automatically picks the best projections to use for any query.

- Projections also provide high availability and recovery. To ensure high availability and recovery, HP Vertica duplicates table columns on at least $K+1$ nodes within the cluster. Thus, if one machine fails in a **K-Safe** environment, the database continues to operate normally using duplicate data on the remaining nodes. Once the node resumes its normal operation, it automatically recovers its data and lost objects by querying other nodes. See [High Availability and Recovery](#) for an overview of this feature and [High Availability With Projections](#) for an explanation of how HP Vertica uses projections to ensure high availability and recovery.

How Projections Are Created

For each table in the database, HP Vertica requires a minimum of one projection, called a **superprojection**. A superprojection is a projection for a single table that contains all the columns in the table.

To get your database up and running quickly, HP Vertica automatically creates a superprojection when you load or insert data into an existing table created using the [CREATE TABLE](#) or [CREATE TEMPORARY TABLE](#) statement.

By creating a superprojection for each table in the database, HP Vertica ensures that all SQL queries can be answered. Default superprojections do not exploit the full power of HP Vertica. Therefore, Vertica recommends loading a sample of your data and then running the Database Designer to create optimized projections. Database Designer creates new projections that optimize your database based on its data statistics and the queries you use. The Database Designer:

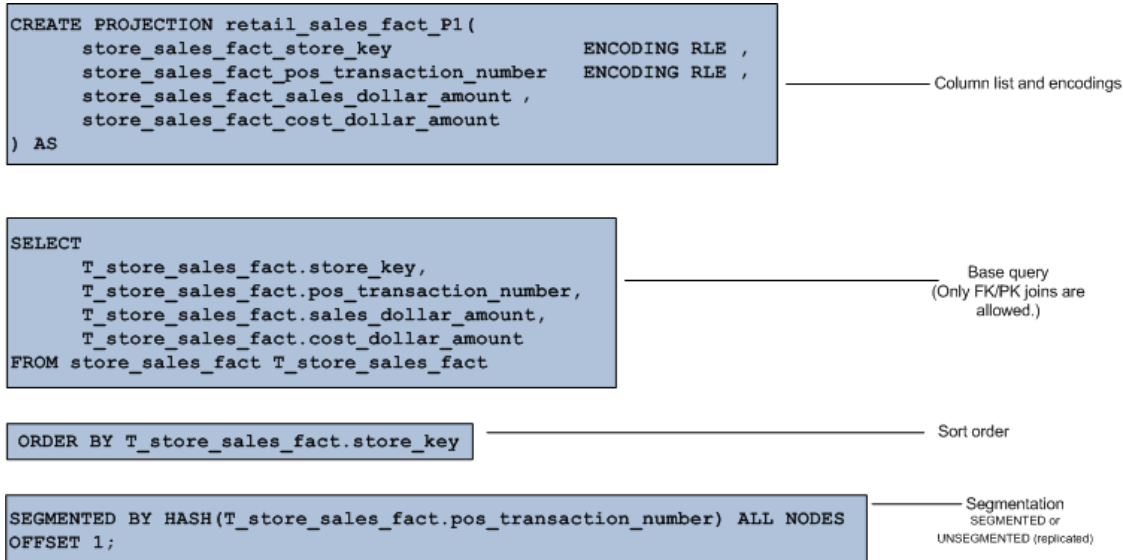
1. Analyzes your **logical schema**, sample data, and sample queries (optional)
2. Creates a **physical schema** design (projections) in the form of a SQL script that can be deployed automatically or manually

In most cases, the designs created by the Database Designer provide excellent query performance within physical constraints. The Database Designer uses sophisticated strategies to provide excellent ad-hoc query performance while using disk space efficiently. If you prefer, you can design [custom projections](#).

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

Anatomy of a Projection

The [CREATE PROJECTION](#) statement defines the individual elements of a projection, as the following graphic shows.



The previous example contains the following significant elements:

Column List and Encoding

Lists every column in the projection and defines the encoding for each column. Unlike traditional database architectures, HP Vertica operates on encoded data representations. Therefore, HP recommends that you 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 large table projections can contain PK/FK joins to smaller tables.

Sort Order

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 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, HP Vertica uses the order in which columns are specified in the column definition as the projection's 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. For maximum performance, do not sort projections on LONG VARBINARY and LONG VARCHAR columns.

Segmentation

The segmentation clause determines whether a projection is segmented across nodes within the database. Segmentation distributes contiguous pieces of projections, called *segments*, for large and medium tables across database nodes. Segmentation maximizes database performance by distributing the load. Use SEGMENTED BY HASH to segment large table projections.

For small tables, use the UNSEGMENTED keyword to direct HP Vertica to replicate these tables, rather than segment them. Replication creates and stores identical copies of projections for small tables across all nodes in the cluster. Replication ensures high availability and recovery.

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

Projection Concepts

For each table in the database, HP Vertica requires a projection, called a **superprojection**. A superprojection is a projection for a single table that contains all the columns in the table. By creating a superprojection for each table in the database, HP Vertica ensures that all SQL queries can be answered.

In addition to superprojections, you can optimize your queries by creating one or more projections that contain only the subset of table columns required to process the query. These projections are called **query-specific projections**.

Projections can contain joins between tables that are connected by PK/FK constraints. These projections are called **pre-join projections**. Pre-join projections can have only inner joins between tables on their primary and foreign key columns. Outer joins are not allowed. Pre-join projections provide a significant performance advantage over joining tables at query run-time.

Projection Performance

HP Vertica provides the following methods for maximizing the performance of all projections:

Encoding and Compression

HP Vertica operates on encoded data representations. Therefore, HP encourages you to use data encoding whenever possible because it results in less disk I/O and requires less disk space. For a description of the available encoding types, see [encoding-type](#) in the SQL Reference Manual.

Sort Order

The sort order optimizes for a specific query or commonalities in a class of queries based on the query predicate. For example, if the WHERE clause of a query is ($x=1$ AND $y=2$) and a projection is sorted on (x , y), the query runs almost instantaneously. It is also useful for sorting a projection to optimize a group by query. Simply match the sort order for the projection to the query group by clause.

Segmentation

Segmentation distributes contiguous pieces of projections, called segments, for large tables across database nodes. This maximizes database performance by distributing the load. See [Projection Segmentation](#).

In many cases, the performance gain for **superprojections** provided through these methods is sufficient enough that creating additional query-specific projections is unnecessary.

Projection Segmentation

Projection segmentation splits individual projections into chunks of data of similar size, called segments. One segment is created for and stored on each node. Projection segmentation provides high availability and recovery and optimizes query execution. Specifically, it:

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

HP Vertica segments large tables, to spread the query execution workload across multiple nodes. HP Vertica does not segment small tables; instead, HP Vertica replicates small projections, creating a duplicate of each unsegmented projection on each node.

Hash Segmentation

HP Vertica uses hash segmentation to segment large projections. 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. 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.

Projection Naming

HP Vertica uses a standard naming convention for projections. The first part of the projection name is the name of the associated table, followed by characters that HP Vertica appends to the table name; this string is called the projection's *base name*. All buddy projections have the same base name so they can be identified as a group.

HP Vertica then appends a suffix that indicates the projection type. The projection type suffix, described in the following table, can be:

- `_super`
- `_<node_name>`

- `_b<offset>`

Projection Type	Suffix	Examples
Unsegmented or segmented (when only one auto projection was created with the table)	<code>_super</code>	Example: <code>customer_dimension_vmart_super</code> Unique name example: <code>customer_dimension_vmart_super_v1</code>
Replicated (unsegmented) on all nodes	<code>_<i><node_name></i></code>	Example: <code>customer_dimension_vmart_node01</code> <code>customer_dimension_vmart_node02</code> <code>customer_dimension_vmart_node03</code> Unique name example: <code>customer_dimension_vmart_v1_node01</code> <code>customer_dimension_vmart_v1_node02</code> <code>customer_dimension_vmart_v1_node03</code>
Segmented (when multiple buddy projections were created with the table)	<code>_b<offset></code>	Example: <code>customer_dimension_vmart_b0</code> <code>customer_dimension_vmart_b1</code> Unique name example: <code>customer_dimension_vmart_v1_b0</code> <code>customer_dimension_vmart_v2_b1</code>

If the projection-naming convention will result in a duplicate name, HP Vertica automatically appends v1 or v2 to the projection name. HP Vertica uses this naming convention for projections created by the CREATE TABLE statement or by the Database Designer.

Note: If the projection name exceeds the maximum length, HP Vertica truncates the projection name.

Database Setup

The process of setting up an HP Vertica database is described in detail in the [Administrator's Guide](#). It involves the following tasks:

Prepare SQL Scripts and Data Files

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

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

Create the Database

This part requires that HP Vertica be installed on at least one host. The following tasks are not in sequential order.

- Use the **Administration Tools** to:
 - Create a database
 - Connect to the database
- Use the **Database Designer** to design the physical schema.
- Use the **vsql** interactive interface to run 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

Set up Security

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

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

Set up Incremental Loads

Set up periodic ("trickle") loads.

Database Connections

You can connect to an HP Vertica database in the following ways:

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

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

- Programmatically using the **JDBC** driver provided by HP Vertica, as described in [Programming JDBC Client Applications](#) in the Programmer's Guide.

An abbreviation for Java Database Connectivity, **JDBC** is a call-level application programming interface (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 HP Vertica, as described in [Programming](#)

[ODBC Client Applications](#) in the Programmer's Guide.

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

- Programmatically using the **ADO.NET** driver provided by HP Vertica, as described in [Programming ADO.NET Applications](#) in the Programmer's Guide.
The HP Vertica driver for ADO.NET allows applications written in C# and Visual Studio to read data from, update, and load data into HP Vertica databases. It provides a data adapter that facilitates reading data from a database into a data set, and then writing changed data from the data set back to the database. It also provides a data reader ([VerticaDataReader](#)) for reading data and [autocommit](#) functionality for committing transactions automatically.
- Programmatically using **Perl** and the DBI driver, as described in [Programming Perl Client Applications](#) in the Programmer's Guide.

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

- Programmatically using **Python** and the pyodbc driver, as described in [Programming Python Client Applications](#) in the Programmer's Guide.

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

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

The Administration Tools

HP Vertica provides a set of tools that allows you to perform administrative tasks quickly and easily. Most of the database administration tasks in HP Vertica can be done using the Administration Tools.

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

A man page is available for admintools. If you are running as the dbadmin user, simply type: `man admintools`. If you are running as a different user, type: `man -M /opt/vertica/man admintools`.

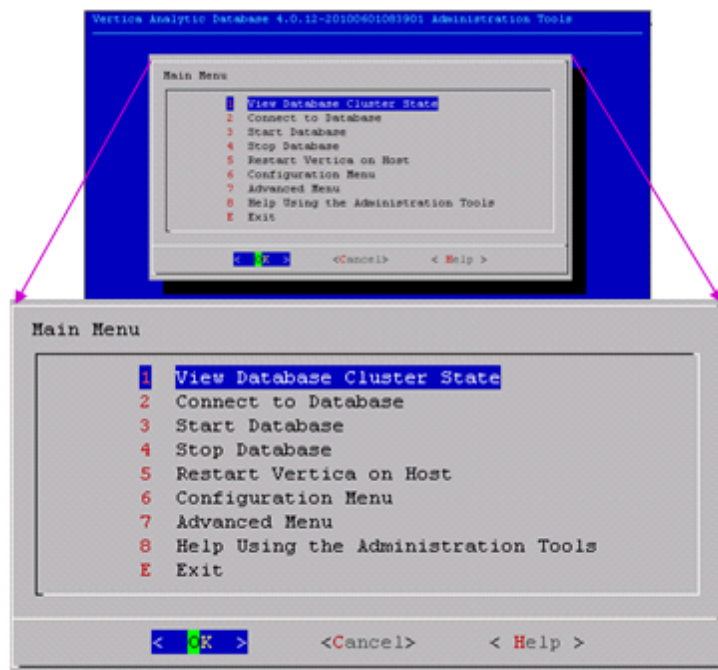
Running the Administration Tools

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 subcommands and options as described in Writing Administration Tools Scripts .

If you omit toolname and options parameters, 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 interface, read [Using the Administration Tools 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.

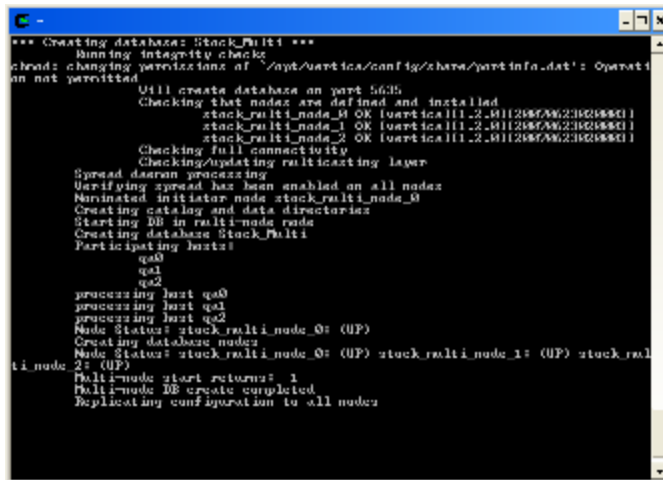
1. In the EULA (end-user license agreement) window, type **accept** to proceed.

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

2. Type the absolute path to your license key (for example, `/tmp/vlicense.dat`) 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_multi ***
Running integrity checks
chroot: changing permissions of '/opt/vertica/config/share/partinfo.dat': Operation
not permitted
Will create database on port 5435
Checking that nodes are defined and installed
stock_multi_node_0 OK (vertica111.2.011200700/21020000)
stock_multi_node_1 OK (vertica111.2.011200700/21020000)
stock_multi_node_2 OK (vertica111.2.011200700/21020000)
Checking full connectivity
Checking/updating replicating layer
Spread daemon processing
Verifying spread has been enabled on all nodes
Nominated initiator node: stock_multi_node_0
Creating catalog and data directories
Starting DB in multi-node mode
Creating database: stock_multi
Participating hosts:
  qa0
  qa1
  qa2
processing host qa0
processing host qa1
processing host qa2
Node Status: stock_multi_node_0: (UP)
Creating database nodes
Node Status: stock_multi_node_0: (UP) stock_multi_node_1: (UP) stock_multi_node_2: (UP)
Multi-node start returns: 1
Multi-node DB create completed
Replicating configuration to all nodes
```

Management Console

Management Console (MC) is a database management tool that provides a unified view of your HP Vertica cluster. Through a single point of access—a browser connection—you can create, import, manage, and monitor multiple databases on one or more clusters. You can also create and manage MC users that you map to an HP Vertica database and then manage on the MC interface.

What You Can Do with Management Console

- Create a database cluster on hosts that do not have HP Vertica installed
- Create, import, and monitor multiple HP Vertica databases on one or more clusters from a single point of control
- Create MC users and grant them access to MC and MC-managed databases
- Manage user information and monitor their activity on MC
- Configure database parameters and user settings dynamically
- Access a single message box of alerts for all managed databases
- Export all database messages or log/query details to a file
- View license usage and conformance
- Diagnose and resolve MC-related issues through a browser
- Access a quick link to recent databases and clusters
- View dynamic metrics about your database cluster

Management Console provides some, but not all of the functionality that the **Administration Tools** provides. In addition, MC provides extended functionality not available in the Administration Tools, such as a graphical view of your HP Vertica database and detailed monitoring charts and graphs, described in [Monitoring HP Vertica Using MC](#). See [Administration Tools and Management Console](#) in the Administrator's Guide.

How to Get MC

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

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

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

What You Need to Know

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

If you want to ...	See ...
Create a new, empty HP Vertica database	Create a Database on a Cluster
Import into MC an existing HP Vertica database cluster	Managing Database Clusters on MC
Understand how MC users are different from database users	About MC Users
Read about the MC privilege model	About MC privileges and roles
Create new MC users	Creating an MC user
Grant MC users privileges on one or more MC-managed HP Vertica databases	Granting database access to MC users
Use HP Vertica functionality through the MC interface	Using Management Console
Monitor MC and MC-managed HP Vertica databases	Monitoring HP Vertica Using Management Console

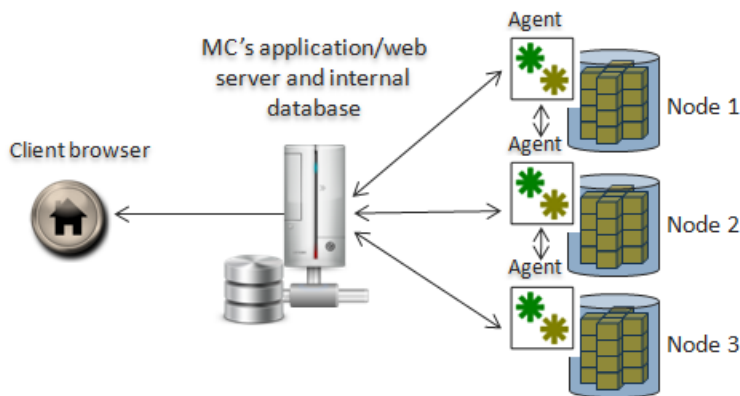
Management Console Architecture

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

MC Components

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

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



Application/web Server

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

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

MC Agents

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

- Provide local access, command, and control over database instances on a given node, using functionality similar to **Administration Tools**
- Report log-level data from the Administration Tools and Vertica log files
- Cache details from long-running jobs—such as create/start/stop database operations—that you can view through your browser
- Track changes to data-collection and monitoring utilities and communicate updates to MC
- Communicate between all cluster nodes and MC through a **webhook** subscription, which automates information sharing and reports on cluster-specific issues like node state, alerts, events, and so on

See Also

-

Management Console Security

Through a single point of control, the Management Console (MC) platform is designed to manage multiple HP Vertica clusters, all which might have differing levels and types of security, such as user names and passwords and LDAP authentication. You can also manage MC users who have varying levels of access across these components.

OAuth and SSL

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

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

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

- [SSL Prerequisites](#)
- [Implementing SSL](#)
- [Generating certifications and keys for MC](#)
- [Importing a new certificate to MC](#)

User Authentication and Access

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

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

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

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

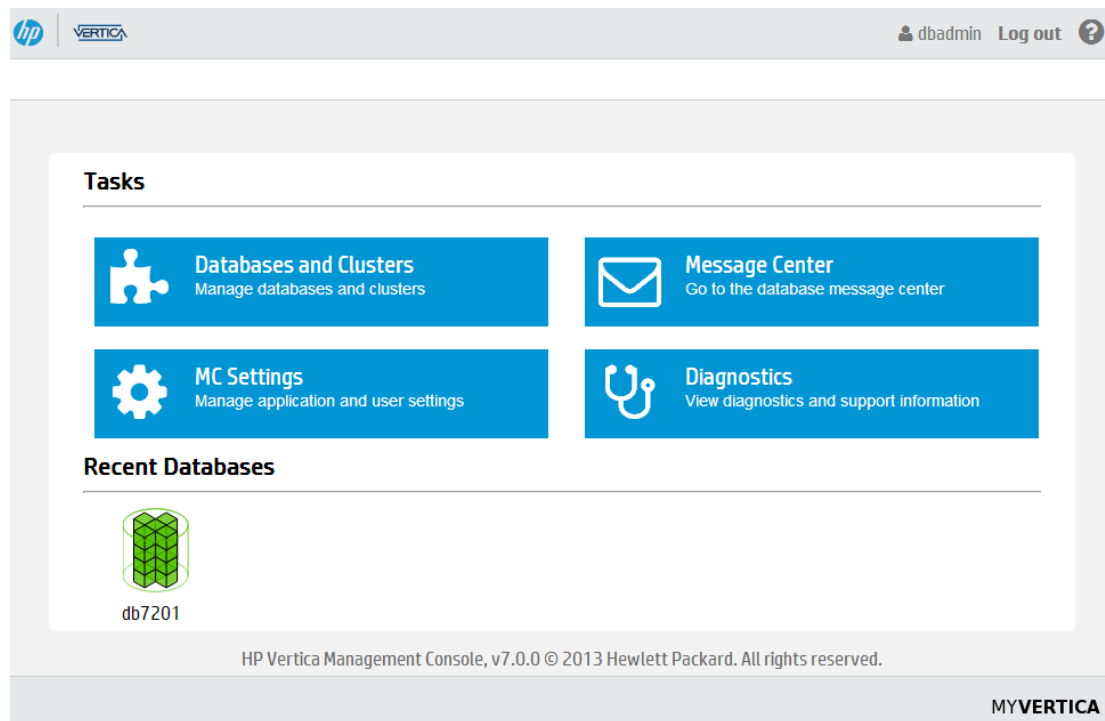
Management Console also has rules for what users can see when they sign in to MC from a client browser. These rules are governed by access levels, each of which is made up of a set of roles.

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

- [About MC users](#)
- [About MC privileges and roles](#)
- [Creating an MC user](#)

Management Console Home Page

The MC Home page is the entry point to all MC-managed HP Vertica database clusters and MC users. Information on this page, as well as throughout the MC interface, will appear or be hidden, based on the signed-on user's permissions ([access levels](#)). Layout and navigation are described in [Using Management Console](#).



Database Designer

HP Vertica's **Database Designer** is a tool that:

- Analyzes your logical schema, sample data, and, optionally, your sample queries.
- Creates a physical schema design (a set of projections) that can be deployed automatically (or manually).

- Can be used by anyone without specialized database knowledge (even business users can run Database Designer).
- Can be run and re-run any time for additional optimization without stopping the database.

There are three ways to run Database Designer:

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

There are two types of designs you can create with Database Designer:

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

Some of the benefits that Database Designer provides:

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

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

See Also

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

Database Security

HP Vertica secures access to the database and its resources by enabling you to control who has access to the database and what they are authorized to do with database resources once they have gained access. See [Implementing Security](#).

Data Loading and Modification

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

In HP Vertica, the **COPY** statement is designed for bulk loading data into the database. COPY reads data from text files or data pipes and inserts it into **WOS** (memory) or directly into the **ROS** (disk). COPY automatically commits itself and any current transaction but is not atomic; some rows could be rejected. Note that COPY does not automatically commit when copying data into temporary tables.

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

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

Workload Management

HP Vertica provides a sophisticated resource management scheme that allows diverse, concurrent workloads to run efficiently on the database. For basic operations, the built-in **GENERAL pool** is pre-configured based on RAM and machine cores, but you can customized this pool to handle specific concurrency requirements.

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

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

- A large batch job takes up all server resources, leaving small jobs that update a web page to starve, which can degrade user experience.

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

- A certain application has lower priority than other applications, and you would like to limit the amount of memory and number of concurrent users for the low-priority application.

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

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

SQL Overview

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

Note: In HP Vertica, use a semicolon to end a statement or to combine multiple statements on one line.

HP Vertica Support for ANSI SQL Standards

HP Vertica SQL supports a subset of ANSI SQL-99.

See [BNF Grammar for SQL-99](#)

Support for Historical Queries

Unlike most databases, the [DELETE](#) command in HP Vertica does not delete data; it marks records as deleted. The [UPDATE](#) command performs an INSERT and a DELETE. This behavior is necessary for **historical queries**. See [Historical \(Snapshot\) Queries](#) in the Programmer's Guide.

Joins

HP Vertica supports typical data warehousing query joins. For details, see [Joins](#) in the Programmer's Guide.

HP Vertica also provides the [INTERPOLATE](#) predicate, which allows for a special type of join. The event series join is an HP Vertica SQL extension that lets you analyze two **event series** when their measurement intervals don't align precisely—such as when timestamps don't match. These joins provide a natural and efficient way to query misaligned event data directly, rather than having to normalize the series to the same measurement interval. See [Event Series Joins](#) in the Programmer's Guide for details.

Transactions

Session-scoped isolation levels determine transaction characteristics for **transactions** within a specific user **session**. You set them through the [SET SESSION CHARACTERISTICS](#) command. Specifically, they determine what data a transaction can access when other transactions are running concurrently. See [Transactions](#) in the Concepts Guide.

About Query Execution

When you submit a query, the **initiator** chooses the projections to use, optimizes and plans the query execution, and logs the SQL statement to its log. Planning and optimization are quick, requiring at most a few milliseconds.

Based on the tables and projections chosen, the query plan that the optimizer produces is decomposed into “mini-plans.” These mini-plans are distributed to the other nodes, known as **executors**, to handle, for example, other segments of a segmented fact table. (The initiator node typically does executor work as well.) The nodes process the mini-plans in parallel, interspersed with data movement operations.

The query execution proceeds in data-flow style, with intermediate result sets (rows) flowing through network connections between the nodes as needed. Some, but not all, of the tasks associated with a query are recorded in the executors' log files.

In the final stages of executing a query plan, some wrapup work is done at the initiator, such as:

- Combining results in a grouping operation
- Merging multiple sorted partial result sets from all the executors
- Formatting the results to return to the client

The initiator has a little more work to do than the other nodes, but if the projections are well designed for the workload, the nodes of the cluster share most of the work of executing expensive queries.

Some small queries, for example, queries on replicated dimension tables, can be executed locally. In these types of queries, the query planning avoids unnecessary network communication.

For detailed information about writing and executing queries, see [Writing Queries](#) in the Programmer's Guide.

Snapshot Isolation Mode

HP 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 a 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, which could cause the query to miss rows loaded by other users up to (but no more than) a specific number of minutes before execution.

Historical Queries

HP Vertica can run 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. The specified `TIMESTAMP` value must be greater than or equal to the Ancient History Mark epoch.

The [DELETE](#) command in HP Vertica does not actually delete data; it marks records as deleted. (The [UPDATE](#) command is actually a combined INSERT and a DELETE.) Thus,

You can control how much deleted data is stored on disk. For more information, see [Managing Disk Space](#) in the Administrator's Guide.

Transactions

Session-scoped isolation levels determine transaction characteristics for **transactions** within a specific user **session**. You set them through the `SET SESSION CHARACTERISTICS` command. Specifically, they determine what data a transaction can access when other transactions are running concurrently.

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

Although the HP Vertica query parser understands all four standard SQL isolation levels (`READ UNCOMMITTED`, `READ COMMITTED`, `REPEATABLE READ`, and `SERIALIZABLE`) for a user session, internally HP Vertica uses only `READ COMMITTED` and `SERIALIZABLE`. HP Vertica automatically translates `READ UNCOMMITTED` to `READ COMMITTED` and `REPEATABLE READ` to `SERIALIZABLE`. Therefore, the isolation level HP Vertica uses could be more strict than the isolation level you choose.

By default, HP Vertica uses the `READ COMMITTED` isolation level. However, you can change the isolation level for the database or individual transactions. See [Change Transaction Isolation Levels](#).

The following table highlights the behaviors of transaction isolation. For specific information see, [SERIALIZABLE Isolation](#) and [READ COMMITTED Isolation](#).

Isolation Level	Dirty Read	Non Repeatable Read	Phantom Read
<code>READ COMMITTED</code>	Not Possible	Possible	Possible
<code>SERIALIZABLE</code>	Not Possible	Not Possible	Not Possible

Implementation Details

HP Vertica supports conventional SQL transactions with standard **ACID** properties:

- ANSI SQL 92 style-implicit transactions. You do not need to run a `BEGIN` or `START TRANSACTION` command.
- No redo/undo log or two-phase commits.
- The `COPY` command automatically commits itself and any current transaction (except when loading temporary tables). HP recommends that you `COMMIT` or `ROLLBACK` the current transaction before you use `COPY`.

Automatic Rollback

Reverts data in a database to an earlier state by discarding any changes to the database state that have been performed by a transaction's statements. In addition, it releases any locks that the

transaction might have held. A rollback can be done automatically in response to an error or through an explicit `ROLLBACK` transaction.

HP Vertica supports transaction-level and statement-level rollbacks. A transaction-level rollback discards all modifications made by a transaction. A statement-level rollback reverses just the effects made by a particular statement. Most errors caused by a statement result in a statement-level rollback to undo the effects of the erroneous statement. HP Vertica uses `ERROR` messages to indicate this type of error. DDL errors, systemic failures, dead locks, and resource constraints result in transaction-level rollback. HP Vertica uses `ROLLBACK` messages to indicate this type of error.

To implement automatic, statement-level rollbacks in response to errors, HP Vertica automatically inserts an implicit savepoint after each successful statement one at a time. This marker allows the next statement, and only the next statement, to be rolled back if it results in an error. If the statement is successful, the marker automatically rolls forward. Implicit savepoints are available to HP Vertica only and cannot be referenced directly.

To explicitly roll back an entire transaction, use the `ROLLBACK` statement. To explicitly roll back individual statements, use explicit savepoints.

Savepoints

A savepoint is a special mark inside a transaction that allows all commands run after the savepoint was established to be rolled back, restoring the transaction to its former state in which the savepoint was established.

Savepoints are useful when creating nested transactions. For example, a savepoint could be created at the beginning of a subroutine. That way, the result of the subroutine could be rolled back, if necessary.

HP Vertica supports using savepoints.

Use the `SAVEPOINT` statement to establish a savepoint, the `RELEASE SAVEPOINT` statement to destroy it, or the `ROLLBACK TO SAVEPOINT` statement to roll back all operations that occur within the transaction after the savepoint was established.

READ COMMITTED Isolation

A `SELECT` query sees a snapshot of the committed data at the start of the transaction. It also sees the results of updates run within its transaction, even if they have not been committed. This is standard ANSI SQL semantics for **ACID** transactions. Any `SELECT` query within a transaction should see the transactions's own changes regardless of isolation level.

DML statements acquire write locks to prevent other `READ COMMITTED` transactions from modifying the same data. `SELECT` statements do not acquire locks, which prevents read and write statements from conflicting.

`READ COMMITTED` is the default isolation level used by HP Vertica. For most general querying purposes, the `READ COMMITTED` isolation effectively balances database consistency and concurrency. However, data can be changed by other transactions between individual statements within the current transaction. This can result in **nonrepeatable** and **phantom reads**. Applications

that require complex queries and updates might need a more consistent view of the database. If this is the case, use `SERIALIZABLE` isolation.

The following example illustrates reads and writes using `READ COMMITTED` isolation.

Session A	Session B	Description
<pre>SELECT C1 FROM T1; C1 ---- (0 rows)</pre>		The <code>SELECT</code> statement in Session A reads committed data from T1.
<pre>COMMIT;</pre>		
<pre>INSERT INTO T1 (C1) VALUES (1); OUTPUT ----- 1 (1 row)</pre>		Session A inserts a row, but does not yet commit.
<pre>SELECT C1 FROM T1; C1 ---- 1 (1 rows)</pre>	<pre>SELECT C1 FROM T1; C1 ---- (0 rows)</pre>	Session A reads the inserted row because it was inserted during the same transaction. However, Session B does not see the inserted value because it can only read committed data.
<pre>COMMIT;</pre>		Session A commits the <code>INSERT</code> and ends the transaction.

	<pre> SELECT C1 FROM T1; C1 ---- 1 (1 row) </pre>	<p>The SELECT statement in Session B now observes the insert that session A committed.</p> <p>This is an example of a non-repeatable read.</p>
<pre> SELECT C1 FROM T1; C1 ---- 1 (1 row) </pre>		<p>The SELECT statement in Session A begins a new transaction. It sees the previous inserted value because it was committed.</p>
COMMIT;		

READ COMMITTED isolation uses exclusive (X) write locks that are maintained until the end of the transaction. The following example illustrates this.

Session A	Session B	Description
<pre> INSERT INTO T1 (C1) VALUES (2); OUTPUT ----- 1 (1 row) </pre>		<p>The transaction in session A acquires an insert (I) lock to insert row 2 into table T1.</p>

	DELETE FROM T1 WHERE C1 >1;	The DELETE statement in Session B is blocked because the transaction cannot acquire an exclusive lock (X lock) until the entire transaction in Session A is completed and the insert (I) lock is released.
INSERT INTO T1 (C1) VALUES (3); OUTPUT ----- 1 (1 rows)		The transaction in session A inserts row 3. (It already has an insert (I) lock.)
COMMIT;		The COMMIT statement ends the transaction in Session A and releases its insert (I) lock.
	(2 rows)	The transaction in Session B obtains its X lock and deletes rows 2 and 3.

See Also

- [LOCKS](#)
- [SET SESSION CHARACTERISTICS](#)
- [Configuration Parameters](#)

SERIALIZABLE Isolation

SERIALIZABLE is the strictest level of SQL transaction isolation. Although this isolation level permits transactions to run concurrently, it creates the effect that transactions are running in serial order. It acquires locks for both read and write operations, which ensures that successive **SELECT** commands within a single transaction always produce the same results. **SERIALIZABLE** isolation establishes the following locks:

- Table-level read locks are acquired on selected tables and released at the end of the transaction. This prevents a transaction from modifying rows that are currently being read by another transaction.
- Table-level write locks are acquired on update and are released at the end of the transaction. This prevents a transaction from reading uncommitted changes to rows made within another transaction.

A **SELECT** sees, in effect, a snapshot of the committed data at the *start of the transaction*. It also sees the results of updates run within its transaction, even if they have not been committed.

The following example illustrates locking within concurrent transactions running with **SERIALIZABLE** isolation.

Session A	Session B	Description
<pre>SELECT C1 FROM T1; C1 ---- (0 rows)</pre>	<pre>SELECT C1 FROM T1; C1 ---- (0 rows)</pre>	Transactions in sessions A and B acquire shared locks. Both transactions can read from table T1.
	<pre>COMMIT;</pre>	The COMMIT statement in Session 2 ends the transaction and releases its read lock.
<pre>INSERT INTO T1 (C1) VALUES (1);</pre>		The transaction in Session A acquires an exclusive lock (X lock) and inserts a new row.
<pre>COMMIT;</pre>		The COMMIT statement in Session 1 ends the transaction and releases its X lock.

<pre>SELECT C1 FROM T1;C1 -- 1 (1 rows)</pre>	<pre>SELECT C1 FROM T1;C1 -- 1 (1 rows)</pre>	New transactions in Sessions A and B use a <code>SELECT</code> statement to see the new row previously created in Session A. They acquire shared locks.
<pre>INSERT INTO T1 (C1) VALUES (2);</pre>		<p>The transaction in Session A is blocked from inserting</p> <p>a row because it cannot upgrade to an X lock.</p>

The advantage of `SERIALIZABLE` isolation is that it provides a consistent view. This is useful for applications that require complex queries and updates. However, it reduces concurrency. For example, it is not possible to perform queries during a bulk load.

Additionally, applications using `SERIALIZABLE` must be prepared to retry transactions due to serialization failures. Serialization failures can occur due to deadlocks. When a deadlock occurs, the transaction that is waiting for the lock automatically times out after five (5) minutes. The following example illustrates a condition that can create a deadlock.

Session A	Session B	Description
<pre>SELECT C1 FROM T1; C1 ---- (0 rows)</pre>	<pre>SELECT C1 FROM T1; C1 ---- (0 rows)</pre>	Transactions in sessions A and B acquire shared locks on table T1. Both transactions can read from T1.
<pre>INSERT INTO T1 (C1) VALUES (1); OUTPUT ----- 1 (1 row)</pre>		The transaction in Session A is blocked because it cannot upgrade to an exclusive lock (X lock) on T1 unless the transaction in Session B releases its lock on T1.
	<pre>INSERT INTO T1 (C1) VALUES (2); OUTPUT ----- 1 (1 row)</pre>	The transaction in Session B is blocked because it cannot acquire an exclusive lock (X lock) unless the transaction in Session A releases its lock on T1. Neither session can proceed because each one is waiting for the other.

	<i>ROLLBACK</i>	HP Vertica automatically breaks the deadlock by rolling back the transaction in Session B and releasing the locks.
(1 rows)COMMIT;		The transaction in session A is able to upgrade to an X lock on T1 and insert the row.

Note: SERIALIZABLE isolation does not acquire locks on temporary tables, which are isolated by their transaction scope.

International Languages and Character Sets

This section describes how HP Vertica handles internationalization and character sets.

Unicode Character Encoding

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 HP Vertica is in UTF-8. The ODBC API operates on data in UCS-2 on Windows systems, and normally UTF-8 on Linux systems. (A UTF-16 ODBC driver is available for use with the DataDirect ODBC manager.) JDBC and ADO.NET APIs operate on data in UTF-16. The client drivers automatically convert data to and from UTF-8 when sending to and receiving data from HP Vertica using API calls. The drivers do not transform data loaded by executing a [COPY](#) or [COPY LOCAL](#) statement.

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

Locales

The locale is a parameter that defines the user's language, country, and any special variant preferences, such as collation. HP 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 for details:

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

String Functions

HP Vertica provides 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.

See [String Functions](#) in the SQL Reference Manual for details.

Character String Literals

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

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

See [Character String Literals](#) in the SQL Reference Manual for details.

Extending HP Vertica

HP Vertica lets you extend its capabilities through several different features:

- [User-Defined SQL Functions](#) let you define a function using HP Vertica SQL statements.
- [User Defined Extensions and User Defined Functions](#) are high-performance extensions to HP Vertica's capabilities you develop using the HP Vertica Software Development Kit (SDK).
- [External Procedures](#) let you pipe data from HP Vertica through external programs or shell scripts to perform some form of processing on it.

User-Defined SQL Functions

User-Defined SQL Functions let you define and store commonly-used SQL expressions as a function. User-Defined SQL Functions are useful for executing complex queries and combining HP Vertica built-in functions. You simply call the function name you assigned in your query.

A User-Defined SQL Function can be used anywhere in a query where an ordinary SQL expression can be used, except in the table partition clause or the projection segmentation clause.

User Defined Extensions and User Defined Functions

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

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

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

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

Get Started

To get started using HP Vertica, follow the steps presented in the [Getting Started Guide](#). The tutorial requires that you install HP Vertica on one or more hosts as described in the [Installation Guide](#).

We appreciate your feedback!

If you have comments about this document, you can [contact the documentation team](#) by email. If an email client is configured on this system, click the link above and an email window opens with the following information in the subject line:

Feedback on Concepts Guide (Vertica Analytic Database 7.0.x)

Just add your feedback to the email and click send.

If no email client is available, copy the information above to a new message in a web mail client, and send your feedback to vertica-docfeedback@hp.com.