Vertica Database 2.0.5-0

Database Administrator's Guide (Advanced)

Date of Publication: 1/8/2008

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

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

Preface

This document describes features that should be used only when advised to do so by Technical Support. If used incorrectly, these features can corrupt data and/or damage a database.

The features described here may not be available in future releases of Vertica.

Overview

This document describes:

- How to define custom projections to augment or replace those produced by the Database Designer.
- How to manually control the movement of data from the WOS to the ROS in order to handle special load requirements.
- How to write Administration Tools scripts.

Defining Custom Projections

Vertica strongly recommends that you use the physical schema design produced by the Database Designer, which provides K-Safety, excellent ad-hoc query performance and efficient use of storage space. However, if you find that the projections produced by the Database Designer are not satisfactory, contact Vertica Technical Support and provide your:

- logical schema definition
- sample data files
- sample queries
- parameters used to run the Database Designer
- projection script created by the Database Designer
- description of your reasons for wanting to write custom projections

Technical Support may be able to provide a solution that does not involve custom projections, or that involves adding custom query-specific projections to the physical schema design.

Should you decide to create your own physical schema design:

- Refer to the CREATE PROJECTION command in the SQL Reference Manual.
- You must follow the ***Design Requirements*** (page 13) described in this document.
- **Vertica is not responsible for the results. Use this option at your own risk.**

Design Requirements

Minimal Design Requirements

These are the minimal projection requirements for a functioning database with no K-Safety (K=0).

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

K-Safe Design Requirements

Because of the complexity involved, Vertica does not define specific requirements for a K-Safe physical schema design. However, the Database Designer provides a way to make a custom physical schema design K-Safe, as described in ***Writing Administration Tools Scripts*** (page 27).

Hash Segmentation

Hash segmentation is the preferred method of segmentation in Vertica 2.0 and later. Refer to the CREATE PROJECTION command in the SQL Reference Manual for detailed information about using hash segmentation in a projection.

Hash segmentation allows you to segment a projection based on a built-in hash function that provides even distribution of data across some or all of the nodes in a cluster, resulting in optimal query execution. To use hash segmentation, simply choose one or more column values to use as input parameters to the built-in Hash Function (see HASH in the SQL Reference Manual for more information).

Hash segmentation's `ALL NODES ... OFFSET` syntax provides an easy way to create the buddy projections that form part of a K-Safe design. For example:

```
CREATE PROJECTION ... SEGMENTED BY HASH(C1,C2,...) ALL NODES;
CREATE PROJECTION ... SEGMENTED BY HASH(C1,C2,...) ALL NODES OFFSET 1;
```

produces two hash-segmented buddy projections. The projections can use different sort orders.

Range Segmentation

Unlike hash segmentation, range segmentation does not automatically distribute data evenly across some or all nodes in a cluster. Instead, you specify a list of nodes, each of which stores a specific range of data values, except for the MAXVALUE node, which has no upper limit. Refer to the CREATE PROJECTION command in the SQL Reference Manual for detailed information about using range segmentation in a projection.

Use range segmentation only when your projection includes a column that is known to contain data that is suitable for use as as a segmentation expression. In other words, avoid using columns that distribute data in a way that causes skewed distribution and execution (some nodes consistently storing more data and working harder than others). This includes data that does not yet exist but can cause skew if loaded in the future.

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

Range segmentation provides very little advantage over hash segmentation:

- It avoids the overhead of using the hash function when loading data, which may be insignificant.
- There are some theoretical advantages in query performance optimization that have not yet been researched.

You can force the Database Designer to use range segmentation by ***Using an Artificial Segmentation Column*** (page 17).

Automatic Replication of Unsegmented Projections

Refer to the CREATE PROJECTION command in the SQL Reference Manual for detailed information about using unsegmented projections.

Vertica requires an exact, non-segmented copy of each dimension table superprojection on each node. You can accomplish this using a single CREATE PROJECTION command for each dimension table. The `UNSEGMENTED ALL NODES` syntax automatically creates a unsegmented projection on each node in the database.

Automatically replicated projections are named:

`projection-name_node-name`

where *projection-name* is the name specified in the CREATE PROJECTION statement and *node-name* is the name of a node in the database. The list of nodes is based on a snapshot of the nodes defined at command execution time. To view a list of the nodes in a database, use the View Database command in the Administration Tools.

For example, if the:

- name of the projection is **ABC**
- names of the nodes are **NODE01**, **NODE02**, and **NODE03**
- command is `CREATE PROJECTION ABC` ... `UNSEGMENTED ALL NODES`

The automatically replicated projections have the names **ABC_NODE01**, **ABC_NODE02**, and **ABC_NODE03**.

Using an Artificial Segmentation Column

The Database Designer recognizes a fact table column named specifically **ARTIFICIAL_SEGMENTATION_COLUMN** and automatically uses that column as the range segmentation column in superprojections and pre-join projections in which that fact table is the anchor table.

The purpose of the artificial segmentation column in Vertica V1.3 was to provide a way to get even data distribution for tables that have no good range segmentation column. Because Vertica V2.0 includes built-in hash segmentation, the use of an artificial segmentation column is supported only for the maintenance of existing databases that use the feature.

In Vertica V2.0 and later, you can use an artificial segmentation column to force the Database Designer to use range segmentation instead of hash segmentation. For example, if your table has an excellent choice of segmentation columns, you can name it ARTIFICIAL_SEGMENTATION_COLUMN in order to to avoid the overhead of using the hash function on that column.

Modifying the Logical Schema

Modifying the logical schema requires nothing more than adding a column to the CREATE TABLE statement that creates the fact table. For example:

```
CREATE TABLE Retail_Sales_Fact (
  Date_Key                   INTEGER NOT NULL,
  Product_Key                INTEGER NOT NULL,
  Store_Key                  INTEGER NOT NULL,
  Promotion_Key              INTEGER NOT NULL,
  POS_Transaction_Number     INTEGER,
  Sales_Quantity             INTEGER,
  Sales_Dollar_Amount        INTEGER,
  Cost_Dollar_Amount         INTEGER,
  Gross_Profit_Dollar_Amount INTEGER,
  ARTIFICIAL_SEGMENTATION_COLUMN INTEGER NOT NULL);
```

Use INTEGER as the data type of the artificial segmentation column (in order to get properly formatted output, should you require it). The column obviously cannot contain nulls. The ordinal position of the column makes no difference.

Preparing the Data

The process of generating segmentation data is quite simple. The goal is to create a **wide range** of data values with **even frequency distribution** within statistical requirements, which need not be stringent. You can use any valid value for the INTEGER data type, whose range is -2^63+1 to 2^63-1. Vertica Systems, Inc. recommends a range consisting of at least **twenty thousand (20K)** unique values.

> Do not use a small range of numbers for this purpose. For example, the number of nodes in the cluster may seem like a good choice but it does not allow for future cluster expansion and is not suited to the Database Designer's algorithms.

Two methods are provided below. If you choose a different method, keep in mind that segmentation column values are used only when writing the data to disk and thus need not be reversible.

1. Random Data

This involves using a random number generator to produce values within a specific range. A random number generator is a function that generates an extremely long sequence of numbers that appear random. The sequence is determined by an initial value called a seed. A computer's real time clock is often used as a seed. High-quality random number generators use physical processes such as atmospheric noise as the seed. The syntax may look something like:

```
data-value = random(ceiling,floor,seed)
```

where the parameters specify the range of the generated numbers and the seed.

2. Hashed Data

> The ***Hash Segmentation*** (page 14) available in Vertica V2.0 and later is similar to this method but does not require any data to be prepared in advance. It also has the advantage of using all column data types equally well.

This involves using a hash function and a modulo function. A hash function is a reproducible method of converting some sort of data into a number that serves as a "fingerprint" of the data. These fingerprints are called hash values. A modulo function causes values to "wrap around" after they reach a certain value (the modulus). The syntax may look something like:

```
data-value = mod(hash(d),n)
```

where:

d is an integer that is derived by manipulating the values of one or more columns in each row of data.

n is an integer where *n*-1 specifies the ceiling (upper limit) of the range of data values. The floor (lower limit) of the range is zero (0).

Adding Custom Projections

You can use the CREATE PROJECTION command to add a projection to a database when one or more of the tables in the projection query already contains data. However, the new projection is out-of-date (not available for query processing) until you start a refresh operation. The START_REFRESH function described in the SQL Reference Manual copies data into the new projection from other projections until it becomes up-to-date.

Before starting a refresh, make sure that the projection has a sufficient number of buddy projections (call MARK_DESIGN_KSAFE) and that all nodes are up .

A refresh runs simultaneously on all nodes. During a refresh, the new projection:

- cannot participate in query execution or INSERT, UPDATE, DELETE, COPY)
- cannot be used as a buddy of another projection

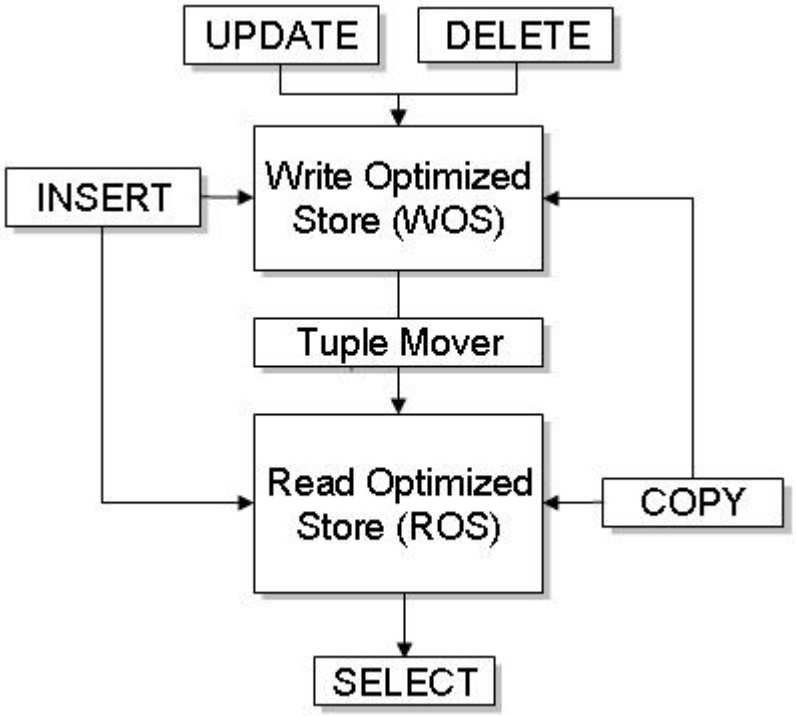
You can view the progress of the refresh by monitoring the log files or calling the following functions:

- GET_PROJECTION_STATUS
- GET_TABLE_PROJECTIONS

When the refresh is complete, the data stored in the projection represents the table columns as of the epoch in which the refresh commits. Prior table history is lost.

Advanced Storage Control

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.



Under ordinary circumstances, the operations performed by the tuple mover are automatic and transparent, and thus of little or no concern to the database administrator. However, when loading data, certain conditions require that you stop the ATM, perform some operations manually, and restart the ATM. This section describes ***what the automatic operations are*** (page 21) and ***how to perform them manually*** (page 25).

Understanding the Automatic Tuple Mover

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 tuple mover actually performs three different operations across all nodes:

- advance epoch

- moveout
- mergeout

Each of these operations occur at different intervals. The most frequent is advance epoch, followed by moveout, and lastly mergeout.

Advance Epoch

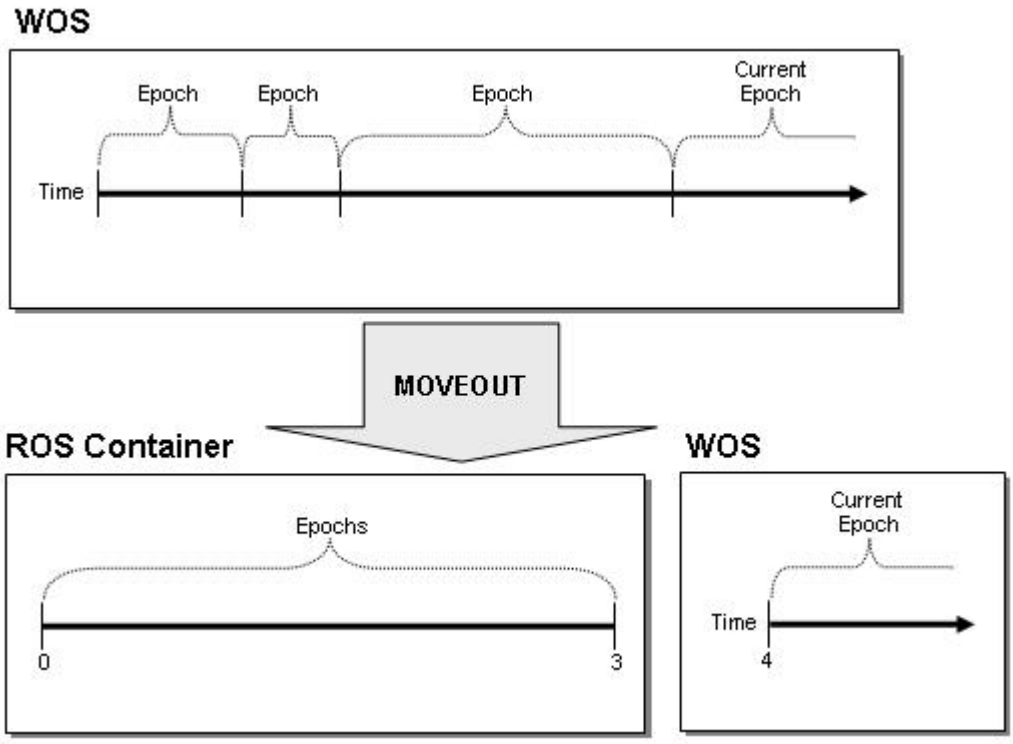
The logical structure of the WOS is a series of epochs. An epoch represents committed changes to the data stored in a database between two specific points in time. In other words, an epoch contains all COPY, INSERT, UPDATE, and DELETE operations that have been executed and committed since the end of the previous epoch.

Advancing the epoch closes the current epoch and opens a new current epoch. The closed epoch contains only committed data; uncommitted data moves into the new current epoch. Epoch numbering begins at zero and increments by one every time an advance epoch occurs. The numbering continues throughout the life of the database. Thus, the epoch number can become quite large.

The automatic tuple mover advances the epoch periodically.

Moveout

Moveout moves all epochs other than the current epoch from the WOS into a new ROS container. It can be thought of as "flushing" all historical data from the WOS to the ROS. The illustration below shows the effect of a moveout of a projection on a single node:



ROS Containers

ROS containers are subsets of the Read Optimized Store (ROS) that are created as the result of changes to the data stored within a projection as a result of bulk loads and DML. The Tuple Mover periodically merges ROS containers in order to maximize performance. A segmented projection can be temporarily stored within several ROS containers on any node at any moment but never fewer than one.

There is not necessarily a one-to-one correspondence between ROS containers and projection segments. For example, consider this projection:

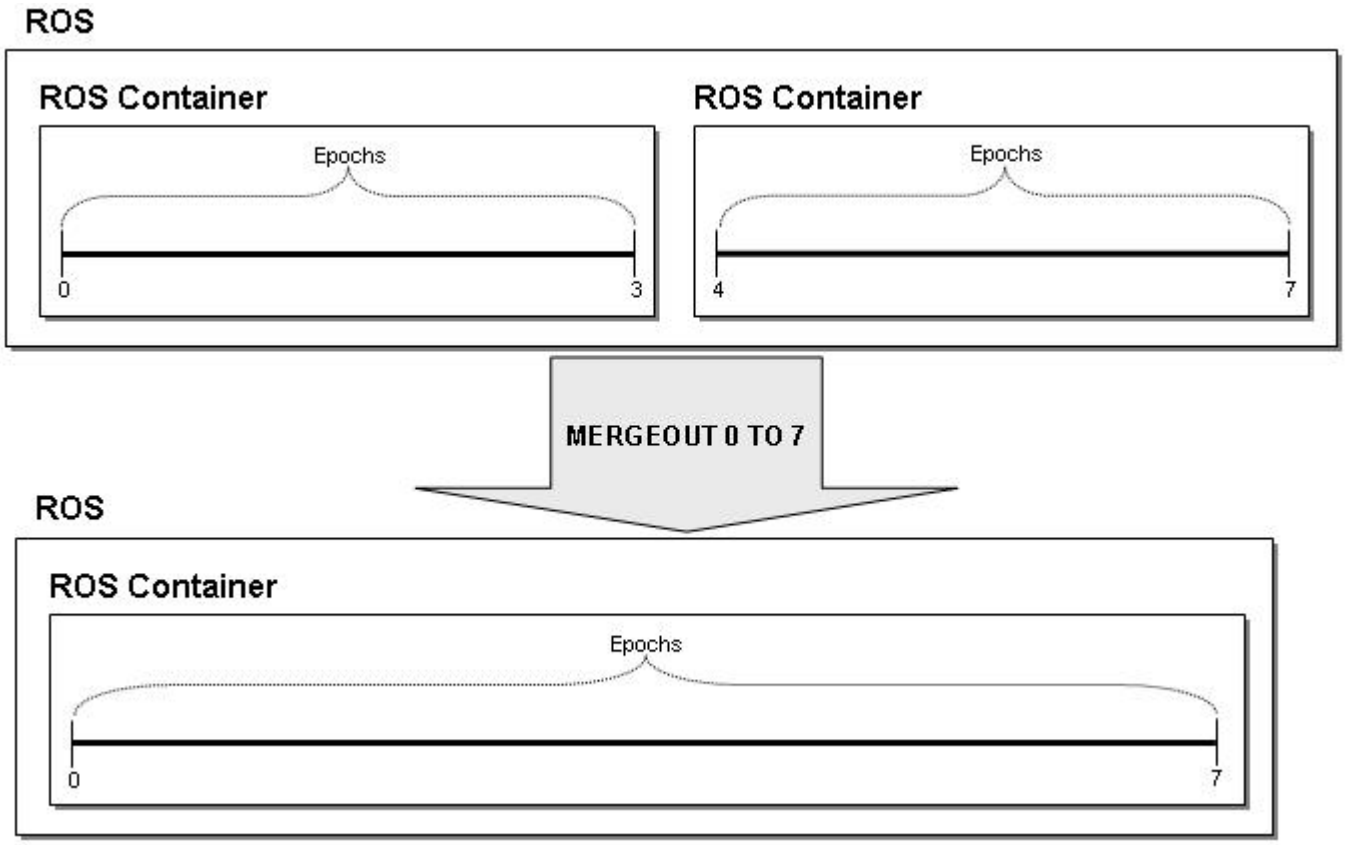
```
CREATE PROJECTION P1 (A, B, C, D) AS
SELECT A, B, C, D
FROM T1
SEGMENTED BY D
NODE S1 VALUES LESS THAN 5
NODE S2 VALUES LESS THAN MAXVALUE;
```

Inserting a tuple with a segmentation column value of 9 creates a new ROS container on node S2 but not on node S1.

Mergeout

Mergeout is the process of consolidating ROS containers. Over time, the number of ROS containers will increase to the point at which it becomes necessary to merge some of them in order to avoid performance degradation. At that point, the tuple mover performs an automatic mergeout, which combines two or more ROS containers into a single container. It can be thought of as "defragmenting" the ROS.

The illustration below shows the effect of a mergeout of a projection on a single node:



Analyze Statistics

The tuple mover collects and aggregates data samples and storage information from all nodes on which a projection is stored, then writes statistics into the catalog so that they can be used by the query optimizer. Without these statistics, the query optimizer would assume uniform distribution of data values and equal storage usage for all projections.

Setting the ATM Mode

The ATM has two modes: **NORMAL** and **BULKLOAD** for writing data to the WOS. BULKLOAD mode does not persist across database startups. In other words, the ATM mode is set to NORMAL at database startup.

> **Note:** Set the ATM mode to BULKLOAD within ten minutes of database startup so that it can take effect before a mergeout begins. A mergeout can take a long time to finish.

NORMAL Use this mode when Trickle Loading.

BULKLOAD Use this mode when Bulk Loading.

To set the automatic tuple mover to bulk load mode:

```
SELECT SET_ATM_MODE('BULKLOAD');
```

To return the automatic tuple mover to normal mode:

```
SELECT SET_ATM_MODE('NORMAL');
```

See SET_ATM_MODE in the SQL Reference Manual for details.

Performing Tuple Mover Operations Manually

Vertica recommends using the automatic tuple mover at all times. However, if you are directed to do so by Technical Support, use the following sequence of operations.

> Note: Running the automatic tuple mover while doing manual moveout/mergeout operations at the same time is not supported.

1. In SQL, stop the automatic tuple mover:
   ```
   SELECT STOP_TUPLE_MOVER();
   ```
2. Restart the database. See Starting and Stopping the Database in the Database Administrator's Guide.
3. Perform manual tuple mover operations as instructed by Technical Support:
   - SELECT ADVANCE_EPOCH
   - ALTER PROJECTION
   - SELECT ANALYZE_STATISTICS

   See the SQL Reference Manual for descriptions of these commands
4. Restart the automatic tuple mover
   ```
   SELECT START_TUPLE_MOVER();
   ```
5. Restart the database.

Writing Administration Tools Scripts

You can invoke any of the Administration Tools from the command line or a shell script.

Syntax

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

For convenience, you can add `/opt/vertica/bin` to your search path.

Semantics

toolname	is one of the tools described in the help output below.	
options	-h --help	shows a brief help message and exits.
	-a --help_all	lists all command line sub-commands and options as shown below.

Tools

```
Usage:
       adminTools [-t | --tool] toolName [options]
Valid tools are:
               kill_node
               run_designer
               list_db
               view_cluster
               uninstall_node
               create_db
               stop_db
               list_node
               install_node
               drop_db
               stop_node
               config_nodes
               drop_node
               restart_db
               restart_node
               check_spread
               start_db
               connect_db
-------------------------------------------------------------------------
usage: kill_node [options]
options:
  -h, --help            show this help message and exit
  -s NODES, --nodes=NODES
                        comma-separated list of nodes on which the vertica
                        prcoess is to be killed
-------------------------------------------------------------------------
usage: runDesigner [options]
options:
  -h, --help            show this help message and exit
  -f PARAMFILE, --file=PARAMFILE
                        Designer parameters file
  -p DBPASSWORD, --password=DBPASSWORD
                        Database password
-------------------------------------------------------------------------
usage: list_db [options]
```

```
options:
  -h, --help            show this help message and exit
  -d DB, --database=DB  Name of database to be listed
-------------------------------------------------------------------------
usage: view_cluster [options]
options:
  -h, --help  show this help message and exit
-------------------------------------------------------------------------
usage: uninstall_node [options]
options:
  -h, --help            show this help message and exit
  -s NODENAME, --node=NODENAME
                        Node name upon which to uninstall
  -p PASSWORD, --password=PASSWORD
                        Name of file containing root password for machines in
                        the list
  -d, --delete          Delete configuration data during uninstall
-------------------------------------------------------------------------
usage: create_db [options]
options:
  -h, --help            show this help message and exit
  -s NODES, --nodes=NODES
                        comma-separated list of nodes to participate in
                        database
  -d DB, --database=DB  Name of database to be created
  -p DBPASSWORD, --password=DBPASSWORD
                        Database password
  -l LICENSEFILE, --license=LICENSEFILE
                        Database license
-------------------------------------------------------------------------
usage: stop_db [options]
options:
  -h, --help            show this help message and exit
  -d DB, --database=DB  Name of database to be stopped
  -p DBPASSWORD, --password=DBPASSWORD
                        Database password
-------------------------------------------------------------------------
usage: list_node [options]
options:
  -h, --help            show this help message and exit
  -s NODENAME, --node=NODENAME
                        Name of the node to be listed
-------------------------------------------------------------------------
usage: install_node [options]
options:
  -h, --help            show this help message and exit
  -s NODENAME, --node=NODENAME
                        Node name upon which to install
  -r RPMNAME, --rpm=RPMNAME
                        Fully qualified file name of the RPM to be used on
                        install
  -p PASSWORD, --password=PASSWORD
                        Name of file containing root password for machines in
                        the list
-------------------------------------------------------------------------
usage: drop_db [options]
options:
  -h, --help            show this help message and exit
  -d DB, --database=DB  Database to be dropped
-------------------------------------------------------------------------
usage: stop_node [options]
options:
  -h, --help            show this help message and exit
  -s NODES, --nodes=NODES
                        comma-separated list of nodes on which the vertica
                        prcoess is to be killed
-------------------------------------------------------------------------
usage: config_nodes [options]
```

```
options:
  -h, --help            show this help message and exit
  -f NODEHOSTFILE, --file=NODEHOSTFILE
                        File containing list of nodes, hostnames, catalog
                        path, and datapath (node<whitespace>host<whitespace>ca
                        talogPath<whitespace>dataPath one per line)
  -i, --install         Attempt to install from RPM on all nodes in the list
  -r RPMNAME, --rpm=RPMNAME
                        Fully qualified file name of the RPM to be used on
                        install
  -p PASSWORD, --password=PASSWORD
                        Name of file containing Root password for machines in
                        the list
  -c, --check           Check all nodes to make sure they can interconnect
  -s SKIPANALYZENODE, --skipanalyzenode=SKIPANALYZENODE
                        skipanalyzenode
------------------------------------------------------------------------
usage: drop_node [options]
options:
  -h, --help            show this help message and exit
  -s NODENAME, --node=NODENAME
                        Name of the node to be dropped
------------------------------------------------------------------------
usage: restart_db [options]
options:
  -h, --help            show this help message and exit
  -d DB, --database=DB  Name of database to be restarted
  -v VERSION, --version=VERSION
                        Catalog Version at which the database is to be
                        restarted. If 'last' is given as argument the db is
                        restarted from the last good catalog version. Either
                        Catalog version or Epoch must be specified.
  -e EPOCH, --epoch=EPOCH
                        Epoch at which the database is to be restarted. If
                        'last' is given as argument the db is restarted from
                        the last good epoch. Either Catalog version or Epoch
                        must be specified.
  -p DBPASSWORD, --password=DBPASSWORD
                        Database password
------------------------------------------------------------------------
usage: restart_node [options]
options:
  -h, --help            show this help message and exit
  -s NODES, --nodes=NODES
                        comma-separated list of nodes to be restarted
  -d DB, --database=DB  Name of database whose node is to be restarted
  -p DBPASSWORD, --password=DBPASSWORD
                        Database password
------------------------------------------------------------------------
usage: check_spread [options]
options:
  -h, --help  show this help message and exit
------------------------------------------------------------------------
usage: start_db [options]
options:
  -h, --help            show this help message and exit
  -d DB, --database=DB  Name of database to be started
  -p DBPASSWORD, --password=DBPASSWORD
                        Database password
------------------------------------------------------------------------
usage: connect_db [options]
options:
  -h, --help            show this help message and exit
  -d DB, --database=DB  Name of database to connect
  -p DBPASSWORD, --password=DBPASSWORD
                        Database password
```

Writing Database Designer Scripts

You can invoke the Database Designer from the command line or a shell script.

Syntax

> **/opt/vertica/bin/adminTools -t run_designer --file *param-file* -p *password***

Semantics

param-file	is the name of a Database Designer parameters file (default is `design_params.txt`)
password	is the database password

Notes

- For convenience, you can add `/opt/vertica/bin` to your search path.

Example

The following is an example of a design_params.txt file:

```
DataDirectory: /scratch/examples/Stock_Schema
SampleRate: 1.0
DatabaseClient: /opt/vertica/bin/vsql
DataNullAndDelimeter: |_\n
FactTableRows: 100000000
WorkingDirectory: /scratch/examples/Stock_Schema/
DatabaseName: Stock_Schema
KSafetyLevel: 2
CatalogDirectory:
/home/dbadmin//Stock_Schema/stock_schema_node1_host01_catalog
DataFileExt: .tbl
QueriesFile: /scratch/examples/Stock_Schema/stock_queries.sql
Port: 5433
TempSchemaFile: catalog_dump.sql
OptimizationStages: 1
Task: MakeSafe
```

Index