



Professional Expertise Distilled

HP Vertica Essentials

Learn how to deploy, administer, and manage HP Vertica,
one of the most robust MPP solutions around

Rishabh Agrawal

[PACKT] enterprise 
PUBLISHING professional expertise distilled

Table of Contents

[HP Vertica Essentials](#)

[Credits](#)

[About the Author](#)

[About the Reviewers](#)

[www.PacktPub.com](#)

[Support files, eBooks, discount offers, and more](#)

[Why subscribe?](#)

[Free access for Packt account holders](#)

[Instant updates on new Packt books](#)

[Preface](#)

[What this book covers](#)

[What you need for this book](#)

[Who this book is for](#)

[Conventions](#)

[Reader feedback](#)

[Customer support](#)

[Errata](#)

[Piracy](#)

[Questions](#)

[1. Installing Vertica](#)

[Understanding the preinstallation steps](#)

[Swap space](#)

[Dynamic CPU frequency scaling](#)

[Understanding disk space requirements](#)

[Steps to install Vertica](#)

[Summary](#)

[2. Cluster Management](#)

[Comprehending the elastic cluster scaling factor](#)

[Enabling and disabling an elastic cluster](#)

[Viewing and setting the scaling factor settings](#)

[Enabling and disabling local segmentation](#)

[Understanding the best practices in cluster management](#)

[Monitoring elastic cluster rebalancing](#)

[Adding nodes in Vertica](#)

[Method](#)

[Using the Management Console to add nodes](#)

[Adding nodes using administration tools](#)

[Removing nodes in Vertica](#)

[Lowering the K-safety level](#)

[Removing nodes using administration tools](#)

[Removing nodes using the Management Console](#)

[Removing hosts from a cluster](#)

[Replacing nodes](#)

[Replacing a node using the same name and IP address](#)

[Replacing a failed node using a different name and IP address](#)

[Redistributing configuration files to nodes](#)

[Using administration tools to replace nodes with different names and IP addresses](#)

[Changing the IP addresses of a Vertica cluster](#)

[Summary](#)

[3. Monitoring Vertica](#)

[Monitoring through the system tables](#)

[Understanding a system table example](#)

[Looking at events](#)

[Looking at events through logfiles](#)

[Looking at events through the ACTIVE_EVENTS system table](#)

[Monitoring Vertica through the Management Console](#)

[Retaining monitoring information](#)

[Enabling and disabling Data Collector](#)

[Viewing the current data retention policy](#)

[Configuring data retention policies](#)

[Monitoring data collection components](#)

[Summary](#)

[4. Backup and Restore](#)

[Requirements for backup hosts](#)

[Generating the vbr.py configuration file](#)

[Miscellaneous settings](#)

[Database access settings](#)

[Data transmission during the backup process](#)

[Mapping](#)

[Creating full and incremental backups](#)

[Understanding the requirements](#)

[Running vbr.py](#)

[Incremental snapshots](#)

[Creating schema and table snapshots](#)

[Restoring full database snapshots](#)

[Restoring from a specific snapshot](#)

[Restoring from the most recent snapshot](#)

[Restoring schema and table snapshots](#)

[Copying a database from one cluster to another](#)

[Copying the database](#)

[Using database snapshot functions](#)

[Creating database snapshots](#)

[Removing snapshots](#)

[Summary](#)

[5. Performance Improvement](#)

[Understanding projections](#)

[Looking into high availability and recovery](#)

[Comprehending unsegmented projections](#)

[Comprehending segmented projections](#)

[Creating projections using Database Designer](#)

[The comprehensive design](#)

- [The query-specific design](#)
 - [Creating projections manually](#)
 - [Column list and encoding](#)
 - [The base query](#)
 - [The sort order](#)
 - [Segmentation](#)
 - [Keeping K-safety \(K-Safe\) in mind](#)
 - [Creating buddy projections](#)
 - [A note on table partitioning](#)
- [Understanding the storage model in Vertica](#)
- [Tuple Mover operations](#)
 - [Moveout](#)
 - [Mergeout](#)
 - [Tuning Tuple Mover](#)
- [Adding storage locations](#)
 - [Adding a new location](#)
 - [Measuring location performance](#)
 - [Setting location performance](#)
 - [Understanding storage location tweaking functions](#)
 - [Altering](#)
 - [Dropping](#)
 - [Retiring storage locations](#)
 - [Restoring retired storage locations](#)
- [Summary](#)
- [6. Bulk Loading](#)
 - [Using the COPY command](#)
 - [Aborting the COPY command](#)
 - [Load methods](#)
 - [Data transformation](#)
 - [Summary](#)
- [Index](#)

HP Vertica Essentials

HP Vertica Essentials

Copyright © 2014 Packt Publishing

All rights reserved. No part of this book may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, without the prior written permission of the publisher, except in the case of brief quotations embedded in critical articles or reviews.

Every effort has been made in the preparation of this book to ensure the accuracy of the information presented. However, the information contained in this book is sold without warranty, either express or implied. Neither the author, nor Packt Publishing, and its dealers and distributors will be held liable for any damages caused or alleged to be caused directly or indirectly by this book.

Packt Publishing has endeavored to provide trademark information about all of the companies and products mentioned in this book by the appropriate use of capitals. However, Packt Publishing cannot guarantee the accuracy of this information.

First published: May 2014

Production Reference: 1080514

Published by Packt Publishing Ltd.

Livery Place

35 Livery Street

Birmingham B3 2PB, UK.

ISBN 978-1-78217-156-0

www.packtpub.com

Cover Image by Paul Steven (<mediakitchenuk@gmail.com>)

Credits

Author

Rishabh Agrawal

Reviewers

Mr. Yagna Narayana Dande

Nishant Garg

Stephan Holler

Pranabesh Sarkar

Commissioning Editor

Kevin Colaco

Acquisition Editor

Kevin Colaco

Content Development Editors

Amey Varangaonkar

Chalini Victor

Technical Editors

Ankita Jha

Dennis John

Neha Mankare

Copy Editors

Karuna Narayanan

Adithi Shetty

Laxmi Subramanian

Project Coordinator

Melita Lobo

Proofreader

Paul Hindle

Graphics

Disha Haria

Indexers

Mehreen Deshmukh

Monica Ajmera Mehta

Priya Subramani

Production Coordinator

Sushma Redkar

Cover Work

Sushma Redkar

About the Author

Rishabh Agrawal is currently working as a senior database research engineer and consultant at Impetus India. He has been tinkering with databases since 2010 and has gained expertise in a variety of NoSQL, massively parallel processing (MPP), and relational databases in the Big Data domain in a short span of time. A MongoDB Certified DBA, he has working knowledge of more than 20 databases, namely Cassandra, Oracle NoSQL, FoundationDB, Riak, Gemfire, Gemfire XD, HBase, Hive, Shark, HP Vertica, Greenplum, SQL Server 2008 R2, and so on. His primary focus areas are research and evaluation of new and cutting-edge database technologies and consulting with clients on the strategic use of diverse database technologies.

When not at work, he revels in photographing vivid subjects, playing badminton, writing poems, and dancing. You can connect with him on LinkedIn at in.linkedin.com/pub/rishabh-agrawal/15/ab4/186.

I would like to acknowledge Impetus (India) that provided me valuable time and resources for creation and completion of this book. I would also like to thank my reviewers and editors (from Packt Publishing) for making sure that this book comes closest to perfect. Last but not least, I will eternally remain indebted to my parents for keeping me inspired and being a pillar of strength in my life.

About the Reviewers

Mr. Yagna Narayana Dande is currently working as a Lead QA Engineer at Collective Media. He has been involved in large-scale testing projects for several years and has exposure to top technologies for both automated and manual testing in functional and non-functional testing. He has worked with both well-established MNCs and startups.

"Software testing is a passion" is the motto that drives his career.

He is mentioned in an acknowledgement in the book *TestNG Beginner's Guide* by Packt Publishing for his great contribution towards reviewing the book.

He has contributed to many fields such as server provisioning, ad serving, and Big Data, including Hadoop and distributed filesystems. He writes interesting test domain articles on his blog —<http://qabypassion.blogspot.com>. You are welcome to contact him for questions regarding testing at <yagna.bitspilani@gmail.com>.

I thank my parents, Venkata Reddiah and Padmavathi, for their constant support.

Nishant Garg has more than 13 years of software architecture and development experience in various technologies, such as Java, Java Enterprise Edition, SOA, Spring, Hibernate, Hadoop, and Hive; NoSQL databases such as MongoDB, CouchDB, Flume, Sqoop, Oozie, Spark, and Shark; and MPP databases such as GreenPlum, Vertica, Kafka, Storm, Mahout, and Solr/Lucene.

He received his MS in Software Systems from Birla Institute of Technology and Science, Pilani, India, and currently works as a Technical Architect in Big Data R&D Group within Impetus Infotech Pvt. Ltd.

Nishant has also previously worked with some of the most recognizable names in IT services and financial industries, employing full software life cycle methodologies such as Agile and Scrum. He has undertaken many speaking engagements on Big Data technologies and is also the author of the book *Apache Kafka, Packt Publishing*.

Stephan Holler currently serves as the Regional Sales Manager for Vertica, HP's answer to enterprise's Big Data. He is responsible for overlooking all sales activities in the DACH region.

Prior to that, he worked as an Enterprise Account Manager (Sales) in HP Networking. His responsibility was to drive business in networking hardware. In fiscal years 2011 and 2012, he successfully served more than 150 commercial accounts in Germany.

In his former role, he acted as a senior business development consultant overseeing all business activities for the Microsoft Solutions Practice in Germany. His duties were generating revenue for Microsoft's services portfolio, starting from SharePoint to Dynamics CRM solutions.

Back in 2007/2008, he was appointed as the leader for the B2C Backend Integration unit within the e-commerce practice EMEA. His responsibilities ranged from team management to generating new business for a large German retailer.

He began his career with EDS (now HP) in 1998 and progressed through multiple assignments ranging from project manager to various roles in delivery and consulting, and has much client-facing experience in retail, telecommunication, and manufacturing, amongst others. This progressed to leadership roles including portfolio management in EMEA, where he was responsible for enabling sales teams to sell a portfolio-driven approach.

Having helped clients in numerous industries, he has a deep understanding of how information technology can be applied to support client business objectives. His unique experience across the sales and delivery life cycle enables the effective creation of solutions to support client demand for better business outcomes.

Pranabesh Sarkar is a technology evangelist on database technologies. He has extensive experience in the IT industry in many facets of data processing and database systems implementation and development, including analysis and design, database administration and development, and performance tuning. He has worked extensively on many database technologies across multiple platforms and possesses expansive knowledge on RDBMS (Oracle, MySQL, and PostgreSQL), MPP databases (Greenplum, Vertica, and Stado), NoSQL database systems (Cassandra, MongoDB, and HBASE), and Big Data technologies including Hadoop/Hive, Shark, Spark, and so on.

In his current assignment, he leads a competency center for emerging database technologies with a primary focus on building expertise on NewSQL/NoSQL databases and MPP database systems. He places

core emphasis on helping enterprises integrate various database solutions for their data processing requirements and provides guidance on their Big Data journey.

www.PacktPub.com

Support files, eBooks, discount offers, and more

You might want to visit www.PacktPub.com for support files and downloads related to your book.

Did you know that Packt offers eBook versions of every book published, with PDF and ePub files available? You can upgrade to the eBook version at www.PacktPub.com and as a print book customer, you are entitled to a discount on the eBook copy. Get in touch with us at [<service@packtpub.com>](mailto:service@packtpub.com) for more details.

At www.PacktPub.com, you can also read a collection of free technical articles, sign up for a range of free newsletters and receive exclusive discounts and offers on Packt books and eBooks.



<http://PacktLib.PacktPub.com>

Do you need instant solutions to your IT questions? PacktLib is Packt's online digital book library. Here, you can access, read and search across Packt's entire library of books.

Why subscribe?

- Fully searchable across every book published by Packt
- Copy and paste, print and bookmark content
- On demand and accessible via web browser

Free access for Packt account holders

If you have an account with Packt at www.PacktPub.com, you can use this to access PacktLib today and view nine entirely free books. Simply use your login credentials for immediate access.

Instant updates on new Packt books

Get notified! Find out when new books are published by following [@PacktEnterprise](#) on Twitter, or the *Packt Enterprise* Facebook page.

Dedicated to maa and papa

--Rishabh Agrawal

Preface

Column-oriented databases have emerged as one of the leading solutions for performing analytics on a huge amount of data. Vertica, recently acquired by Hewlett Packard (HP), is one of the flag bearers in the field of distributed-column-oriented databases. Vertica can be easily classified as a Massively Parallel Processing (MPP) database, which allows you to process queries concurrently on a large number of databases. Vertica's distributed architecture not only allows fast query processing, but also a highly fault tolerant architecture. With innovative features such as projections, a replacement for indexes, and views, Vertica has emerged as one of the most sought after relational analytical database solutions.

With more and more clients pressing for Vertica as a solution, we are glad to present a small admin book on Vertica, which will empower DBAs to learn and perform the most essential administration activities on Vertica.

This book has been especially written for Vertica 6.0 (and later). It is assumed that you are a little familiar with the Vertica database. Our references for this book were our experiences with Vertica, the Vertica administration guide, and the Vertica forums. Since this book is just a snapshot of the day-to-day administration activities of Vertica, it is highly advised to look into the official Vertica guides for more information and other administration tasks.

What this book covers

[Chapter 1](#), *Installing Vertica*, explains how to install Vertica.

[Chapter 2](#), *Cluster Management*, helps you to learn how to manage a running Vertica cluster.

[Chapter 3](#), *Monitoring Vertica*, elucidates different methods of monitoring various aspects of a Vertica cluster.

[Chapter 4](#), *Backup and Restore*, explains how to create backups of a database and restore them.

[Chapter 5](#), *Performance Improvement*, helps you learn some tricks of the trade to achieve good performance in Vertica.

[Chapter 6](#), *Bulk Loading*, illustrates the working of the bulk loading utility of Vertica.

What you need for this book

You just need a running Vertica cluster. If you don't have one, then [Chapter 1, *Installing Vertica*](#), will explain how to create one of your own.

Who this book is for

This book is intended for Vertica users and DBAs who want to perform basic administration and fine tuning. Prior knowledge of Vertica will help in understanding the chapters better, but is not mandatory.

Conventions

In this book, you will find a number of styles of text that distinguish between different kinds of information. Here are some examples of these styles, and an explanation of their meaning.

Code words in text are shown as follows: "Now log in as root or use the `sudo` command."

Any command-line input or output is written as follows:

```
/opt/vertica/bin/adminTools
```

New terms and **important words** are shown in bold. Words that you see on the screen, in menus or dialog boxes for example, appear in the text like this: "Now, to create a database, navigate to **Main Menu | Configuration Menu | Create Database.**"

Note

Warnings or important notes appear in a box like this.

Tip

Tips and tricks appear like this.

Reader feedback

Feedback from our readers is always welcome. Let us know what you think about this book—what you liked or may have disliked. Reader feedback is important for us to develop titles that you really get the most out of.

To send us general feedback, simply send an e-mail to [<feedback@packtpub.com>](mailto:feedback@packtpub.com), and mention the book title via the subject of your message.

If there is a topic that you have expertise in and you are interested in either writing or contributing to a book, see our author guide on www.packtpub.com/authors.

Customer support

Now that you are the proud owner of a Packt book, we have a number of things to help you to get the most from your purchase.

Errata

Although we have taken every care to ensure the accuracy of our content, mistakes do happen. If you find a mistake in one of our books—maybe a mistake in the text or the code—we would be grateful if you would report this to us. By doing so, you can save other readers from frustration and help us improve subsequent versions of this book. If you find any errata, please report them by visiting <http://www.packtpub.com/submit-errata>, selecting your book, clicking on the **errata submission form** link, and entering the details of your errata. Once your errata are verified, your submission will be accepted and the errata will be uploaded on our website, or added to any list of existing errata, under the Errata section of that title. Any existing errata can be viewed by selecting your title from <http://www.packtpub.com/support>.

Piracy

Piracy of copyright material on the Internet is an ongoing problem across all media. At Packt, we take the protection of our copyright and licenses very seriously. If you come across any illegal copies of our works, in any form, on the Internet, please provide us with the location address or website name immediately so that we can pursue a remedy.

Please contact us at [<copyright@packtpub.com>](mailto:copyright@packtpub.com) with a link to the suspected pirated material.

We appreciate your help in protecting our authors, and our ability to bring you valuable content.

Questions

You can contact us at [<questions@packtpub.com>](mailto:questions@packtpub.com) if you are having a problem with any aspect of the book, and we will do our best to address it.

Chapter 1. Installing Vertica

Massively Parallel Processing (MPP) databases are those which partition (and optionally replicate) data into multiple nodes. All meta-information regarding data distribution is stored in master nodes. When a query is issued, it is parsed and a suitable query plan is developed as per the meta-information and executed on relevant nodes (nodes that store related user data). HP offers one such MPP database called Vertica to solve pertinent issues of Big Data analytics.

Vertica differentiates itself from other MPP databases in many ways. The following are some of the key points:

- **Column-oriented architecture:** Unlike traditional databases that store data in a row-oriented format, Vertica stores its data in columnar fashion. This allows a great level of compression on data, thus freeing up a lot of disk space. (More on this is covered in [Chapter 5, Performance Improvement](#).)
- **Design tools:** Vertica offers automated design tools that help in arranging your data more effectively and efficiently. The changes recommended by the tool not only ease pressure on the designer, but also help in achieving seamless performance. (More on this is covered in [Chapter 5, Performance Improvement](#).)
- **Low hardware costs:** Vertica allows you to easily scale up your cluster using just commodity servers, thus reducing hardware-related costs to a certain extent.

This chapter will guide you through the installation and creation of a Vertica cluster. This chapter will also cover the installation of Vertica Management Control, which is shipped with the Vertica Enterprise edition only. It should be noted that it is possible to upgrade Vertica to a higher version but vice versa is not possible.

Before installing Vertica, you should bear in mind the following points:

- Only one database instance can be run per cluster of Vertica. So, if you have a three-node cluster, then all three nodes will be dedicated to one single database.
- Only one instance of Vertica is allowed to run per node/host.
- Each node requires at least 1 GB of RAM.
- Vertica can be deployed on Linux only and has the following requirements:
 - Only the root user or the user with all privileges ([sudo](#)) can run the [install_vertica](#) script. This script is very crucial for installation and will be used at many places.
 - Only [ext3/ext4](#) filesystems are supported by Vertica.

- Verify whether `rsync` is installed.
- The time should be synchronized in all nodes/servers of a Vertica cluster; hence, it is good to check whether NTP daemon is running.

Understanding the preinstallation steps

Vertica has various preinstallation steps that are needed to be performed for the smooth running of Vertica. Some of the important ones are covered here.

Swap space

Swap space is the space on the physical disk that is used when primary memory (RAM) is full. Although swap space is used in sync with RAM, it is not a replacement for RAM. It is suggested to have 2 GB of swap space available for Vertica. Additionally, Vertica performs well when swap-space-related files and Vertica data files are configured to store on different physical disks.

Dynamic CPU frequency scaling

Dynamic CPU frequency scaling, or CPU throttling, is where the system automatically adjusts the frequency of the microprocessor dynamically. The clear advantage of this technique is that it conserves energy and reduces the heat generated. It is believed that CPU frequency scaling reduces the number of instructions a processor can issue. Additional theories state that when frequency scaling is enabled, the CPU doesn't come to full throttle promptly. Hence, it is best that dynamic CPU frequency scaling is disabled. CPU frequency scaling can be disabled from **Basic Input/Output System (BIOS)**. Please note that different hardware might have different settings to disable CPU frequency scaling.

Understanding disk space requirements

It is suggested to keep a buffer of 20-30 percent of disk space per node. Vertica uses buffer space to store temporary data, which is data coming from the merge out operations, hash joins, and sorts, and data arising from managing nodes in the cluster.

Steps to install Vertica

Installing Vertica is fairly simple. With the following steps, we will try to understand a two-node cluster:

1. Download the Vertica installation package from <http://my.vertica.com/> according to the Linux OS that you are going to use.
2. Now log in as root or use the `sudo` command.
3. After downloading the installation package, install the package using the standard command:

- For `.rpm` (CentOS/RedHat) packages, the command will be:

```
rpm -Uvh vertica-x.x.x-x.x.rpm
```

- For `.deb` (Ubuntu) packages, the command will be:

```
dpkg -i vertica-x.x.x-x.x.deb
```

Refer to the following screenshot for more details:

```
[impetus@centos64a setups]$ rpm -Uvh vertica-6.1.3-0.x86_64.RHEL5.rpm
error: can't create transaction lock on /var/lib/rpm/.rpm.lock (Permission denied)
[impetus@centos64a setups]$ sudo rpm -Uvh vertica-6.1.3-0.x86_64.RHEL5.rpm
[sudo] password for impetus:
Preparing...##### [100%]
 1:vertica##### [100%]

Vertica Analytic Database V6.1.3-0 successfully installed on host centos64a

-----
Important Information
-----
If you are upgrading from a previous version, you must backup your database before
continuing with this install. After restarting your database, you will be unable
to revert to a previous version of the software.
-----

To download the latest Vertica documentation in zip or tar format please visit the
myvertica web site.

To complete installation and configuration of the cluster,
run: /opt/vertica/sbin/install_vertica
```

Running the Vertica package

4. In the previous step, we installed the package on only one machine. Note that Vertica is installed under `/opt/vertica`. Now, we will set up Vertica on other nodes as well. For that, run the following command on the same node:

```
/opt/vertica/sbin/install_vertica -s
host_list -r rpm_package -u dba_username
```

Here, `-s` is the hostname/IP of all the nodes of the cluster, including the one on which Vertica is already installed. `-r` is the path of the Vertica package and `-u` is the username that we wish to create for working on Vertica. This user has sudo privileges. If prompted, provide a password for the new user. If we do not specify any username, then Vertica creates `dbadmin` as the user, as shown in the following example:

```
[impetus@centos64a setups]$  
sudo /opt/vertica/sbin/install_vertica -s  
192.168.56.101,192.168.56.101,192.168.56.102  
-  
r  
"/ilabs/setups/vertica-6.1.3-  
0.x86_64.RHEL5.rpm" -u dbadmin
```

```
Vertica Analytic Database 6.1.3-0  
Installation Tool  
Upgrading admintools meta data format..  
scanning /opt/vertica/config/users  
Starting installation tasks...  
Getting system information for cluster (this  
may take a while)....
```

```
Enter password for impetus@192.168.56.102 (2  
attempts left):  
backing up admintools.conf on 192.168.56.101  
Default shell on nodes:  
192.168.56.101 /bin/bash  
192.168.56.102 /bin/bash
```

```
Installing rpm on 1 hosts....  
installing node.... 192.168.56.102
```

```
NTP service not synchronized on the hosts:  
['192.168.56.101', '192.168.56.102']
```

```
Check your NTP configuration for valid NTP  
servers.
```

```
Vertica recommends that you keep the system  
clock synchronized using NTP or some other  
time synchronization mechanism to keep all  
hosts synchronized. Time variances can cause  
(inconsistent) query results when using  
Date/Time Functions. For instructions, see:  
* http://kbase.redhat.com/faq/FAQ\_43\_755.shtm  
* http://kbase.redhat.com/faq/FAQ\_43\_2790.shtm
```


Info: the package 'pstack' is useful during troubleshooting. Vertica recommends this package is installed.

Checking/fixing OS parameters.....
Setting vm.min_free_kbytes to 37872 ...
Info! The maximum number of open file descriptors is less than 65536
Setting open filehandle limit to 65536 ...
Info! The session setting of pam_limits.so is not set in /etc/pam.d/su
Setting session of pam_limits.so in /etc/pam.d/su ...
Detected cpufreq module loaded on 192.168.56.101
Detected cpufreq module loaded on 192.168.56.102
CPU frequency scaling is enabled. This may adversely affect the performance of your database.
Vertica recommends that cpu frequency scaling be turned off or set to 'performance'

Creating/Checking Vertica DBA group

Creating/Checking Vertica DBA user

Password for dbadmin:
Installing/Repairing SSH keys for dbadmin

Creating Vertica Data Directory...

Testing N-way network test. (this may take a while)

All hosts are available

...

Verifying system requirements on cluster.

IP configuration

...

IP configuration

...

Testing hosts (1 of 2)....

Running Consistency Tests

LANG and TZ environment variables

...

Running Network Connectivity and Throughput Tests...

Waiting for 1 of 2 sites...

...

Test of host 192.168.56.101 (ok)

=====

```

    Enough RAM per CPUs (ok)
    -----

    Test of host 192.168.56.102 (ok)
    =====

    Enough RAM per CPUs (FAILED)
    -----

    Vertica requires at least 1 GB per CPU (you
    have 0.71 GB/CPU)
    See the Vertica Installation Guide for more
    information.

    Consistency Test (ok)
    =====

    Info: The $TZ environment variable is not
    set on 192.168.56.101
    Info: The $TZ environment variable is not set
    on 192.168.56.102

    Updating spread configuration...
    Verifying spread configuration on whole
    cluster.
    Creating node node0001 definition for host
    192.168.56.101
    ... Done
    Creating node node0002 definition for host
    192.168.56.102
    ... Done
    Error Monitor 0 errors 4 warnings
    Installation completed with warnings.
    Installation complete.

    To create a database:
    1. Logout and login as dbadmin.**
    2. Run /opt/vertica/bin/adminTools as dbadmin
    3. Select Create Database from the
    Configuration Menu

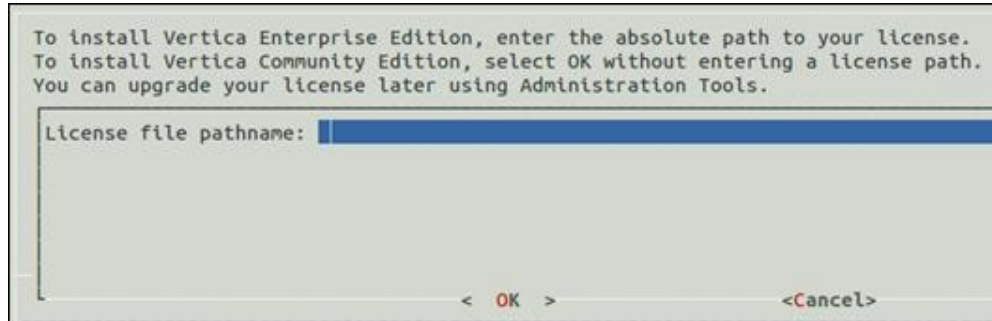
    ** The installation modified the group
    privileges for dbadmin.
    If you used sudo to install vertica as
    dbadmin, you will
    need to logout and login again before the
    privileges are applied.

```

5. After we have installed Vertica on all the desired nodes, it is time to create a database. Log in as a new user ([dbadmin](#) in default scenarios) and connect to the admin panel. For that, we have to run the following command:

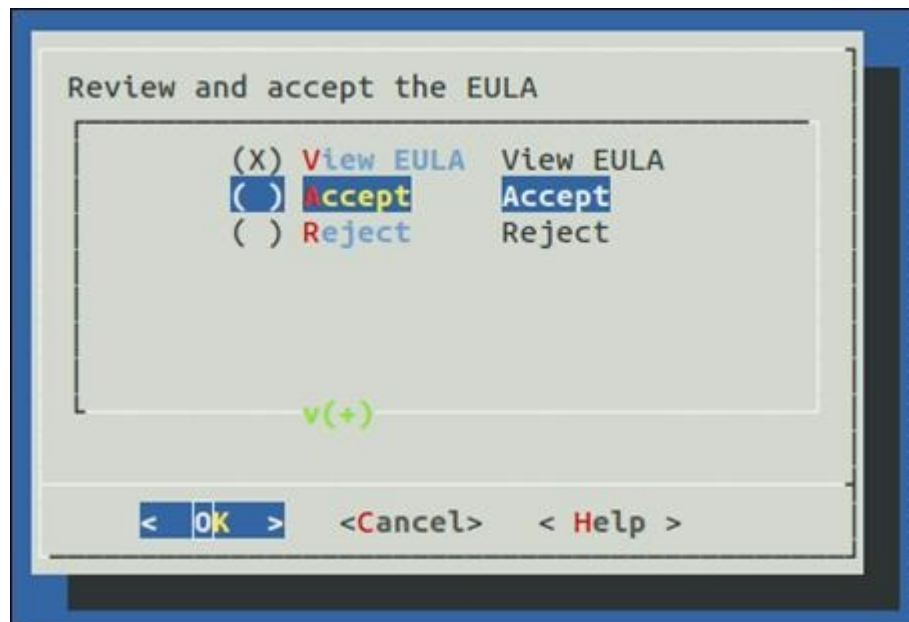
`/opt/vertica/bin/adminTools`

6. If you are connecting to admin tools for the first time, you will be prompted for a license key. If you have the license file, then enter its path; if you want to use the community edition, then just click on **OK**.



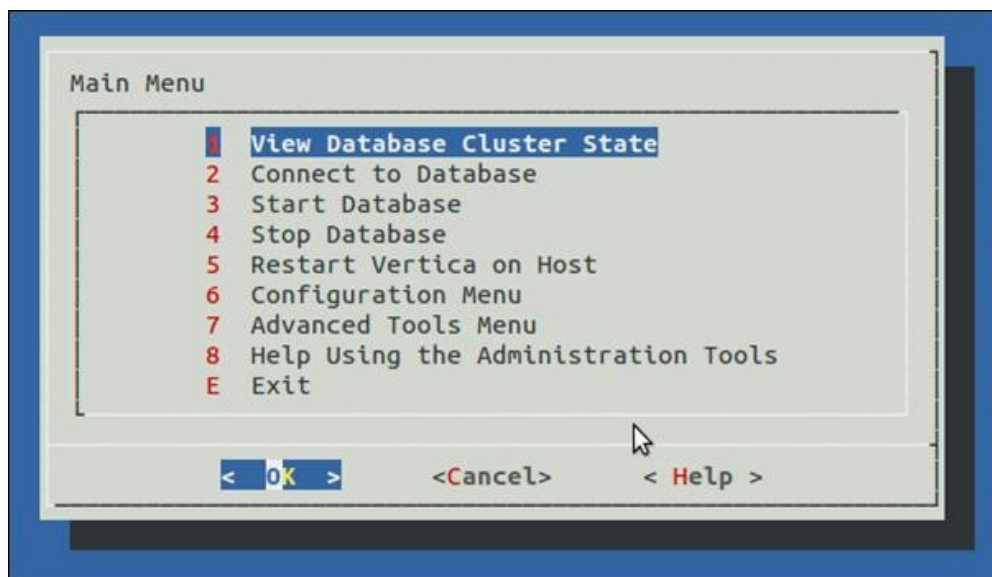
License key prompt

7. After the previous step, you will be asked to review and accept the **End-user License Agreement (EULA)**.



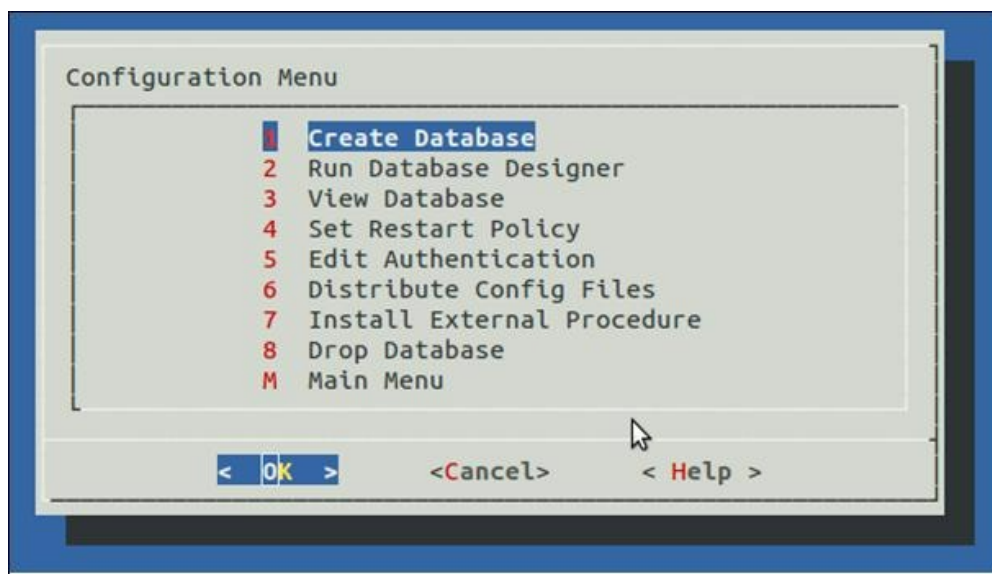
Prompt for EULA

After reviewing and accepting the EULA, you will be presented with the main menu of the admin tools of Vertica.



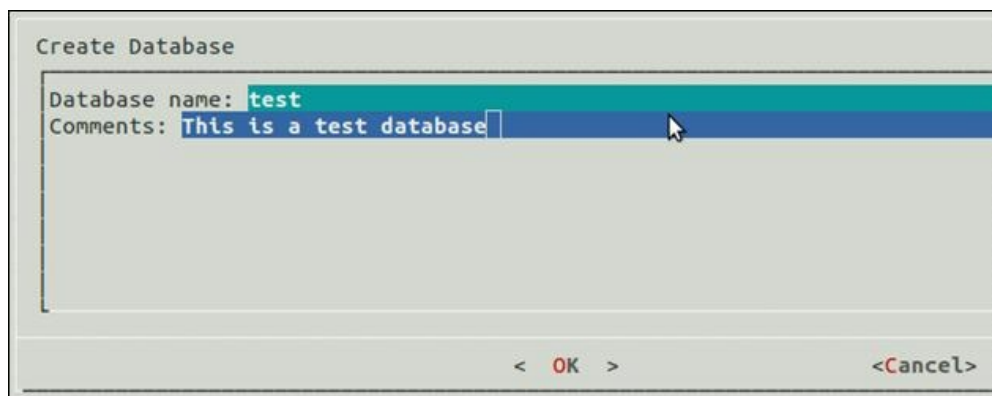
Admin tools main menu

8. Now, to create a database, navigate to **Administration Tools | Configuration Menu | Create Database**.



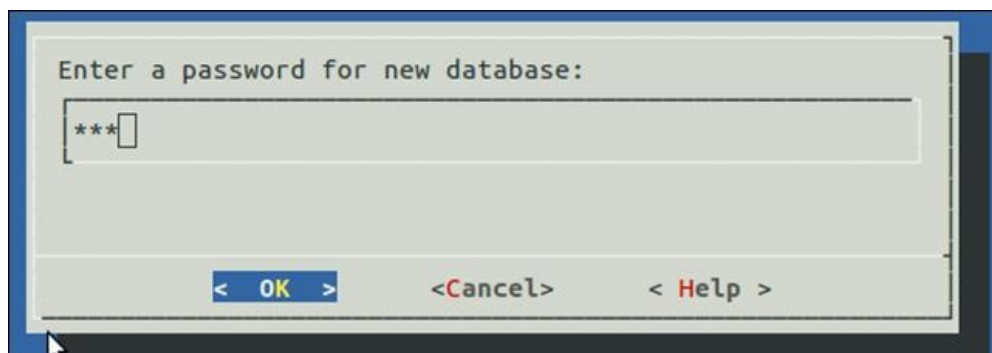
The Create Database option in the configuration menu

9. Now, you will be asked to enter a database name and a comment that you would like to associate with the database.



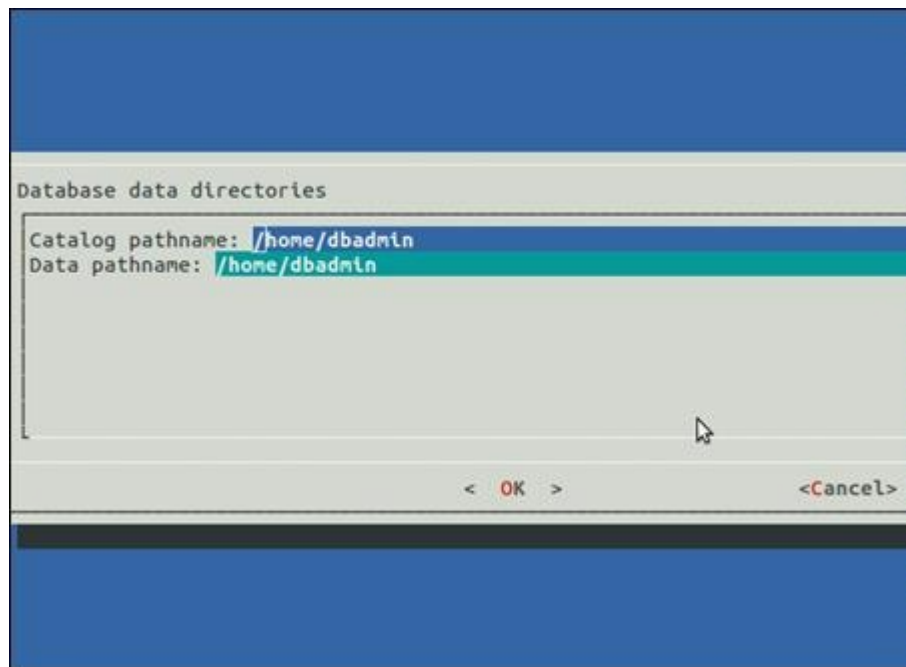
Database name and comments

10. After entering the name and comment, you will be prompted to enter a password for this database.



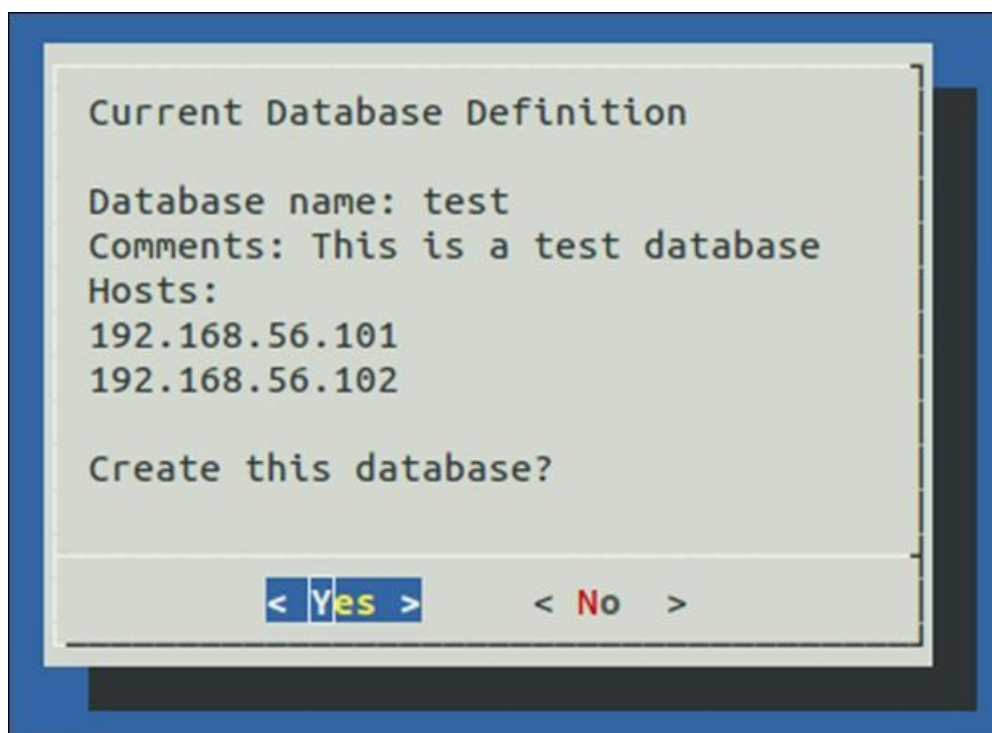
Password for the new database

11. After entering and re-entering (for confirmation) the password, you need to provide pathnames where the files related to user data and catalog data will be stored.



Catalog and data pathnames

After providing all the necessary information related to the database, you will be asked to select hosts on which the database needs to be deployed. Once all the desired hosts are selected, Vertica will ask for one final check.



Final confirmation for database creation

12. Now, Vertica will create and deploy the database.

```
*** Creating database: test ***
      192.168.56.101 OK [vertica][(6, 1, 3)][000][x86_64]
      192.168.56.102 OK [vertica][(6, 1, 3)][000][x86_64]
      Checking full connectivity
```

Database creation

13. Once the database is created, we can connect to it using the VSQL tool or perform admin tasks.

Summary

As you can see, Vertica installation is simple. You can perform further checks by creating sample tables and performing basic CRUD operations.

For a clean installation, it is recommended to serve all the minimum requirements of Vertica. It should be noted that installation of client API(s) and Vertica Management Console needs to be done separately and is not included in the basic package.

In the next chapter, you will learn some tricks relating to cluster management.

Chapter 2. Cluster Management

Vertica provides quite an elastic cluster, which can be scaled both up (adding new nodes) and down (removing nodes) without affecting the processes of the running database. The most important task after alteration in a cluster is the rebalancing of data across new as well as old nodes. This is done to ensure that the data remains K-Safe. Please refer to [Chapter 5, Performance Improvement](#), for more information on K-Safe.

Projections are divided into segments, which are small portions of data. After adding a new node, some segments are given to it, while the other segments are exchanged to ensure proper K-safety. During the process of node removal from a cluster, all of the storage containers, which are residing at the node that is being removed, are moved to other existing nodes in the cluster. This method of partitioning data into movable segments turns a Vertica cluster into an elastic cluster.

Comprehending the elastic cluster scaling factor

Each node in the cluster stores local segments of data. The number of local segments in a node is known as the scaling factor. As discussed earlier, to perform effective rebalancing when nodes are removed or added, local segments from each of the nodes redistribute themselves in the cluster in order to maintain even data distribution across the cluster.

The `MAXIMUM_SKEW_PERCENT` parameter plays a crucial role when the number of segments cannot be evenly divided by the number of nodes in a new cluster. For example, if the scaling factor is 4 and there are initially 4 nodes, there will be 16 (4 x 4) segments in the whole cluster. Suppose one additional node is added to the cluster; then, it is not possible to evenly distribute 16 segments among 5 nodes. Hence, Vertica will assign more segments to some nodes as compared to others. So, one possible combination can be 4 nodes get 3 segments each and 1 node gets 4 segments. This skew is around 33.33 percent. Vertica will make sure that it remains below the set `MAXIMUM_SKEW_PERCENT` parameter. If Vertica is not able to redistribute segments because of the `MAXIMUM_SKEW_PERCENT` limit, the data-rebalancing process will not fail. However, the segmentation space will be evenly distributed among the 5 nodes, and new segments will be created on each node, making the total 20, that is, 4 segments on each node.

Enabling and disabling an elastic cluster

We can query the `ELASTIC_CLUSTER` system table to determine if an elastic cluster is enabled on the database or not, that is, to determine if segmentation is on or off. Run the following query to check if an elastic cluster is enabled or not:

```
=> select is_enabled from ELASTIC_CLUSTER;
is_enabled
-----
t
(1 row)
```

To enable an elastic cluster, we can run the following command:

```
=> SELECT ENABLE_ELASTIC_CLUSTER();
ENABLE_ELASTIC_CLUSTER
-----
ENABLED
(1 row)
```

To disable an elastic cluster, we can run the following command:

```
=> SELECT DISABLE_ELASTIC_CLUSTER();
DISABLE_ELASTIC_CLUSTER
-----
DISABLED
(1 row)
```

Viewing and setting the scaling factor settings

To view the scaling factor, we can query the `ELASTIC_CLUSTER` table (4 is the default). Run the following query to find out the scaling factor:

```
=> SELECT scaling_factor FROM ELASTIC_CLUSTER;
scaling_factor
-----
4
(1 row)
```

We can use the `SET_SCALING_FACTOR` function to change a database's scaling factor. The scaling factor can be any integer between 1 and 32. It should be taken into account that a very high value of the scaling factor may lead to the nodes creating too many small container files, eventually causing too many ROS container errors. These errors are also known as

ROS pushback (refer to [Chapter 5, Performance Improvement](#) to learn more about ROS). The following command is an example of using `SET_SCALING_FACTOR`:

```
=> SELECT SET_SCALING_FACTOR(5) ;
SET_SCALING_FACTOR
-----
SET
(1 row)

=> SELECT scaling_factor FROM ELASTIC_CLUSTER;
scaling_factor
-----
5
(1 row)
```

Enabling and disabling local segmentation

Just setting the scaling factor and enabling the elastic cluster will make Vertica create local segments during the rebalancing stage only. It is advisable for production deployments to keep local segments ready beforehand. For this, we can enable and disable local segmentation, which tells Vertica to always segment its old as well as new data.

To enable local segmentation, we can use the `ENABLE_LOCAL_SEGMENTS` function as follows:

```
=> SELECT ENABLE_LOCAL_SEGMENTS() ;
ENABLE_LOCAL_SEGMENTS
-----
ENABLED
(1 row)
```

To check the status of local segmentation, we can query the `ELASTIC_CLUSTER` system table in the following fashion:

```
=> SELECT is_local_segment_enabled FROM
elastic_cluster;
is_enabled
-----
t
(1 row)
```

To disable local segmentation, we can use the `DISABLE_LOCAL_SEGMENTATION` function as follows:

```
=> SELECT DISABLE_LOCAL_SEGMENTS();
DISABLE_LOCAL_SEGMENTS
-----
DISABLED
(1 row)
```

To check the status of local segmentation, we can query the `elastic_cluster` system table in the following fashion:

```
=> SELECT is_local_segment_enabled FROM
elastic_cluster;
is_enabled
-----
f
(1 row)
```

Understanding the best practices in cluster management

The following are some of the best practices for local segmentation:

- It only makes sense to keep local segments when we are actually planning to scale the cluster up or down
- It is highly recommended that backups of the database(s) are created before the start of the scaling process
- It is not advisable to create segments if our database tables contain a high number of partitions (more than 75)

Monitoring elastic cluster rebalancing

From Vertica 6.0 onwards, system tables can be used to monitor the rebalance status of an elastic cluster. There are two tables that can be employed. They are as follows:

```
REBALANCE_TABLE_STATUS
REBALANCE_PROJECTION_STATUS
```

In each table, `separated_percent` and `transferred_percent` can be used to determine the overall progress.

Adding nodes in Vertica

Before adding nodes, it is very important that we create a full backup of the database, as adding nodes in Vertica is a sensitive process. To add a new node, we will be using the `update_vertica` script. However, before adding a node to an existing cluster, the following are certain restrictions and prerequisites that have to be kept in mind:

- Make sure that the database is running.
- Newly added nodes should be reachable by all the existing nodes in the cluster.
- If we have a single node cluster that is deployed, without specifying the IP address, hostname, or hostname specified as the local host, it is not possible to expand the cluster. We must reinstall Vertica and specify an IP address or hostname.
- Generally, it is not needed to shut down the Vertica database for expansion, but a shutdown is necessary if we are expanding it from a single node cluster.

Method

From any node of the cluster, we can run the `update_vertica` script as follows:

```
# /opt/vertica/sbin/update_vertica -A <hostname1,  
hostname2...> -r <rpm_package>
```

Here, `-A` is used for providing IP(s) or hostname(s), and `-r` is used for providing the location of the `rpm/debian` package.

The `update_vertica` script will install Vertica, verify the installation, and add a node to the cluster. Once we have added one or more hosts to the cluster, we need to empower them as data storing nodes through either of the following:

- The Management Console interface (Enterprise Edition only)
- The administration tools interface

Using the Management Console to add nodes

We can add or remove nodes from a database by going to the **Manage** page. Here, just click on the target node and then click on the **Add node** or **Remove node** button in the node list. When we add a node, the color of the node icon changes from gray (empty) to green.

Adding nodes using administration tools

The following is the process of adding nodes in a Vertica cluster using administration tools:

1. Navigate to **Main Menu | Advanced Menu | Cluster Management | Add Host(s)**.
2. We should now select the database to which we want to add one or more hosts. A list of unused hosts is displayed.
3. Select the host. You will then be prompted for the password. Provide the password for the database.
4. The user will be prompted about the success or failure of the addition of the node.
5. If it is successful, Vertica starts rebalancing the cluster. During database rebalancing, Vertica will ask the user to provide a path to a temporary directory that the database designer will use.
6. Before starting the rebalancing of the cluster, Vertica will prompt the user to provide a new higher K-value, or we can continue with the existing K-value.
7. As a final step, we should select whether Vertica should immediately start rebalancing or whether Vertica should do it at a later time. If we choose to do the rebalancing later, then a script is created and is kept for later execution. It is always advised that we should select the option to automatically start rebalancing. If we choose to automatically rebalance the database, the script will still be created and will be saved for later use and review.

Removing nodes in Vertica

Removing nodes or scaling down the Vertica cluster is a fairly simple process. The procedure of removing nodes comprises the following broad steps:

1. Back up the database.
2. Remember that it is mandatory to lower the K-safety if the cluster is not able to sustain the current level of K-safety after the cluster is scaled down.
3. Remove the host from the database.

Lowering the K-safety level

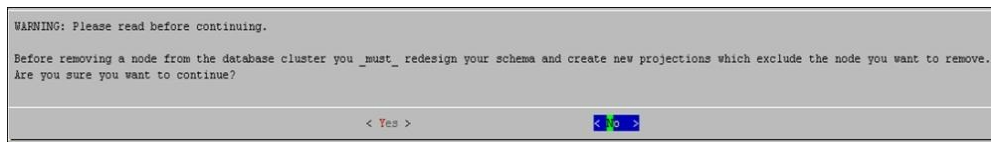
A database with a K-safety level 1 requires at least three nodes to operate, and a database with a K-safety level 2 requires at least five nodes to operate. Vertica doesn't support K-safety of level 3 and above. To lower the K-safety level, we will use the `MARK_DESIGN_KSAFE` function in the `Vsql` console, as shown in the following example:

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

Removing nodes using administration tools

The following are the steps to remove nodes using administration tools:

1. Before starting this step, we must make sure that the number of nodes that will remain after removing the nodes will comply with K-safety.
2. As a precautionary step, create a backup of the database.
3. Make sure that the database is running.
4. Navigate to **Main Menu | Advanced Menu | Cluster Management | Remove Host(s)**.
5. Select the database from which we wish to remove the node and click on **OK**.
6. Select the node that we wish to remove.
7. We will be asked if we are sure about removing the node. If we are, then click on **OK**; otherwise, click on **Cancel**.
8. We will be warned that we must redesign our database and create projections that exclude the hosts we are going to drop; click on **Yes**. The following screenshot shows the warning prompt:



Warning issued during the removal of a node

9. Vertica begins the process of rebalancing the database and removing the node(s). When you are informed that the hosts were successfully removed, click on **OK**.

Removing nodes using the Management Console

We can remove nodes from a database through the **Manage** page. For removing a node, we have to select the node we want to act upon and then click on the **Remove node** button in the node list. We can only remove nodes that are part of the database, that is, nodes that show a state of down or nodes that are not working (represented in red) and are not critical for K-safety. When we remove a node, its color changes from red to clear, and the Management Console updates its state to standby.

Removing hosts from a cluster

Removing a node from a database doesn't result in its removal from the cluster. We can completely remove it from the cluster using the `update_vertica` script, but it must be ensured that the host must not be used by any database. We do not need to shut down the database for this process.

From one of the hosts in the cluster, we need to run `update_vertica` with the `-R` switch, where `-R` specifies a comma-separated list of hosts to be removed from an existing Vertica cluster. Do not confuse `-R` with `-r`, as both have different functionalities. A host can be specified by the hostname or the IP address of the system, as shown in the following example:

```
# /opt/vertica/sbin/update_vertica -R  
192.168.56.103,host04
```


Replacing nodes

If we have a K-Safe database, we can replace nodes, as a copy of the data will be maintained under the nodes. We do not need to shut down the database to replace the nodes.

Replacing a node using the same name and IP address

Sometimes, you will be required to upgrade one or more nodes in the cluster. If the new node has the same IP as the original node, then use the following method for replacement:

1. From a working node in the cluster, run the following `install_vertica` script with the `-s` (used for providing the hostname or IP(s)) and `-r` (the path of the `rpm/deb` package) parameters:

```
# /opt/vertica/sbin/install_vertica -s host  
-r rpm_package
```

2. The installation script will verify the system configuration and the installation of various important components of Vertica, such as Vertica, Spread, and the administration tool's metadata.
3. Now, create catalog and data directories on the new node. Make sure that the path of these directories is the same as that of the original node.
4. Once ready, restart the newly added host, and you will find that it has been added to the cluster.

The new node automatically joins the database cluster and recovers data by querying the other nodes within the database cluster. Data recovery may take some time as huge chunks of data will be moved across the nodes.

Replacing a failed node using a different name and IP address

There may be a time when you will be required to replace a failed node with a node that has a different IP address and hostname. In such cases, proceed with the following steps:

1. As a preventive step, create a backup of the database.
2. After the backup is created, run `update_vertica` with the `-A`, `-R`, `-E`, and `-r` parameters, as shown in the following code, to replace the failed host:

```
/opt/vertica/sbin/update_vertica -A  
NewHostName -R OldHostName -E -r rpm_package
```

Where:

- `NewHostName` is the hostname or IP address of the new node
- `OldHostName` is the hostname or IP address of the node that is being replaced from the cluster
- The `-E` parameter makes sure that the failed node is dropped from the cluster
- `rpm_package` is the name of the rpm package; for example, `-r vertica_6.0.x.x86_64.RHEL5.rpm`

3. Using administration tools, we can replace the original host with the new host. If we are using more than one database, make sure that the old node is replaced by a new node for all databases.
4. Now, distribute configuration files to the new host. The process is discussed in the next section.
5. Then, we have to run `update_vertica` again. However, this time, we will run it with the `-R` parameter, as shown in the following code, to clear all information of the failed node from the administration tool's metadata:

```
# /opt/vertica/sbin/update_vertica -R  
OldHostName
```

`OldHostName` is the hostname or IP address of the system that we removed from the cluster. Do not confuse `-R` with `-r` as both have different functions.

6. Now, start the new node and verify if everything is running properly.

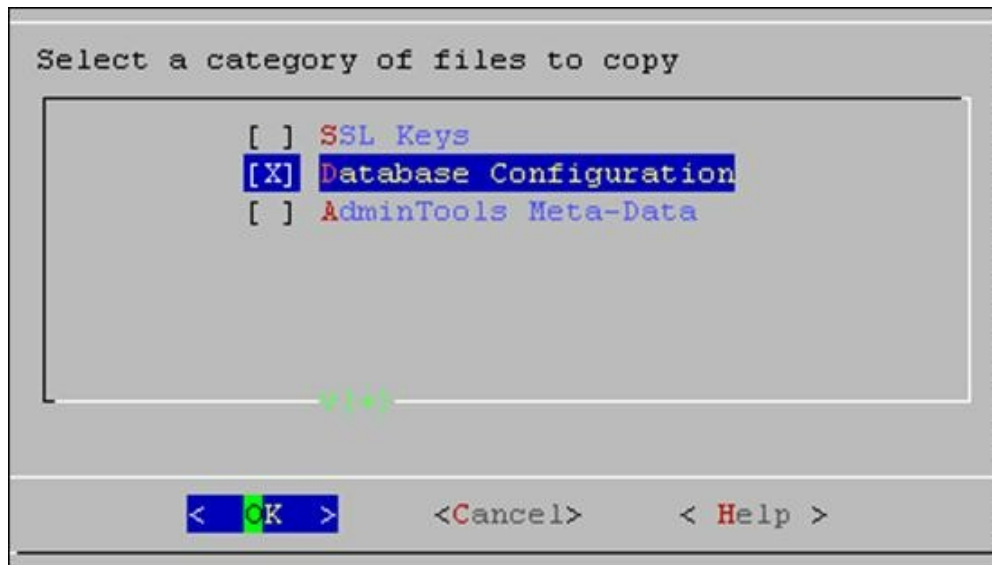
Once you have completed the process, the new node automatically recovers the data that was stored in the original node by querying other nodes in the database cluster.

Redistributing configuration files to nodes

The processes of adding and removing nodes automatically redistribute the Vertica configuration files. To distribute configuration files to a host, log on to a host that contains these files using administration tools. This can be done using the following steps:

1. Navigate to **Main Menu | Configuration Menu | Distribute Config Files**.

2. Click on **Database Configuration** as shown in the following screenshot:



Selecting the redistribution configuration category

3. Then, select the database in which we wish to distribute the files and click on **OK**.

The `vertica.conf` file will be distributed to all the other hosts in the database. If it existed earlier on a host, it is overwritten.

4. We need to follow the same process for **Secure Sockets Layer (SSL)** keys as well as administration tool's metadata. For SSL keys, Vertica may prompt for the location of the server certificate file, server key file, and root certificate file. Provide an appropriate path and click on **OK**.

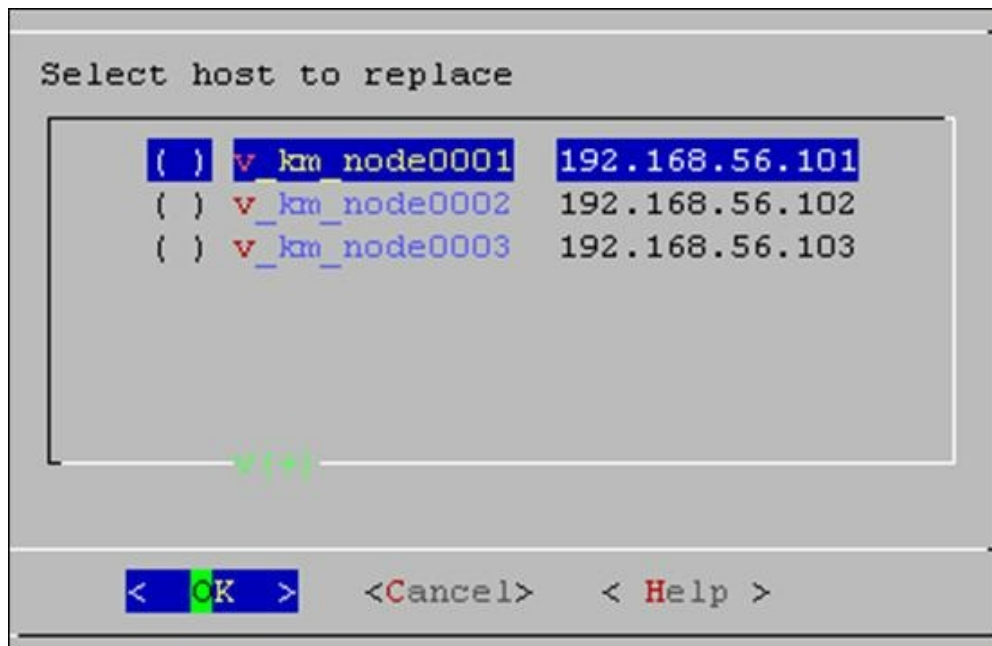
Using administration tools to replace nodes with different names and IP addresses

Using administration tools, you can easily replace a node with a node of a different hostname and IP address. Alternatively, you can also use the Management Console to replace a node.

To replace the original host with a new host using administration tools, proceed with the following steps:

1. As a preventive step, create a backup of the database.
2. You first need to make sure that the database is running.
3. Add the replacement hosts to the cluster using the standard procedure of adding a node.
4. Now, shut down the node that is intended to be replaced.

5. Navigate to **Main Menu | Advanced Menu**.
6. In the **Advanced Menu** option, we need to select **Stop Vertica on Host**.
7. Select the host we wish to replace and then click on **OK** to stop the node.
8. After stopping the node, navigate to **Advanced Menu | Cluster Management | Replace Host**.
9. Select the database that contains the host we wish to replace and then click on **OK**.
10. A list of all the hosts with their internal names and IP addresses will be displayed. We will select the host we wish to replace and then click on **OK**, as shown in the following screenshot:



Selecting nodes to replace

11. Select the host we want to use as the replacement and then click on **OK**.
12. When prompted that the host was successfully replaced, click on **OK**.
13. Restart all the nodes (new nodes may take some time to start as they will be in the recovering state, thus moving data).
14. Verify if the database is running properly.

Changing the IP addresses of a Vertica cluster

Sometimes, during networking-related maintenance chores, the IP(s) of one or more servers running Vertica changes to some other IP. In these cases, IP changes are needed to be done in Vertica as well. However, before making changes in Vertica, we must make sure that changes are made in `/etc/hosts` of all nodes, where the hostname is mapped to the IP address. After making system-level changes, proceed with the following steps to change the IP address of one or more nodes in a cluster:

1. Back up the following three files:

- `/opt/vertica/config/admintools.conf`
- `/opt/vertica/config/vspread.conf`
- `/etc/sysconfig/spreadd`

It is assumed that Vertica is installed on `/opt/vertica`. If Vertica is installed on some other location, then we should take backups from that location.

2. We should stop Vertica on all nodes.
3. As a root user, we need to stop Spread manually on each node. Spread is the messaging system for distributed systems such as Vertica. We can stop Spread using the following command:

```
/etc/init.d/spreadd stop
```

4. On each node, edit `/opt/vertica/config/admintools.conf` and change the IPs as required. The following is the text from `admintools.conf` for one of the nodes (replace the old IP with the new one wherever required):

```
[Configuration]
install_opts = ['-s',
'192.168.56.101,192.168.56.102,192.168.56.103',
'-r', 'vertica-ce-6.0.0-1.x86_64.RHEL5.rpm',
'-u', 'dba']
default_base = /home/dbadmin
show_hostnames = False
format = 3

[Cluster]
hosts =
192.168.56.101,192.168.56.102,192.168.56.103
```

```

spread_hosts =

[Nodes]
node0001 = 192.168.56.101,/home/dba,/home/dba
node0002 = 192.168.56.102,/home/dba,/home/dba
node0003 = 192.168.56.103,/home/dba,/home/dba
v_km_node0001 =
192.168.56.101,/ilabs/data/vertica/catalog,/ilabs/data/vertica/data
v_km_node0002 =
192.168.56.102,/ilabs/data/vertica/catalog,/ilabs/data/vertica/data
v_km_node0003 =
192.168.56.103,/ilabs/data/vertica/catalog,/ilabs/data/vertica/data

[Database:km]
host = 192.168.56.101
restartpolicy = ksafe
port = 5433
path
= /ilabs/data/vertica/catalog/km/v_km_node0001_catalog
nodes =
v_km_node0001,v_km_node0002,v_km_node0003

```

5. On each node, edit `/opt/vertica/config/vspread.conf`. We need to replace the old IP with the new one wherever required. We also need to change the `N` number, where `N` is followed by the IP address without a period (`.`). Following is the text from `vspread.conf` for one of the nodes:

```

Spread_Segment 192.168.56.255:4803 {
  N192168056101    192.168.56.101 {
    192.168.56.101
    127.0.0.1
  }
  N192168056102    192.168.56.102 {
    192.168.56.102
    127.0.0.1
  }
  N192168056103    192.168.56.103 {
    192.168.56.103
    127.0.0.1
  }
}
EventLogFile = /dev/null
EventTimeStamp = "[%a %d %b %Y %H:%M:%S]"
DaemonUser = spread
DaemonGroup = verticadba
DebugFlags = { EXIT }

```

6. Just like the changes we made in `vspread.conf`, we also need to make changes in `etc/sysconfig/spreadd`.
7. After IP-related changes are made in all three configuration files, we should start Spread on each node as a root user:

```
/etc/init.d/spreadd start
```

8. Then, we should start a single Vertica node and connect to our database and run `Vsql`.
9. In `Vsql`, issue the following query to verify that the new IP has been updated:

```
select host_name from host_resources;
```

10. As a final step, we need to modify the database to use the new IPs.
11. In `Vsql`, we have to run the following query to display the current node names that are configured:

```
km=> \x
Expanded display is on.
km=> select node_name, node_address from
nodes;
-[ RECORD 1 ]+-----
node_name      | v_km_node0001
node_address   | 192.168.56.101
-[ RECORD 2 ]+-----
node_name      | v_km_node0002
node_address   | 192.168.56.102
-[ RECORD 3 ]+-----
node_name      | v_km_node0003
node_address   | 192.168.56.103
```

12. For each result, we need to update the IP address by issuing the following command:

```
alter node NODE_NAME is hostname <new IP
address>;
```

In the preceding command, `NODE_NAME` is the internal name of the node, and `<new IP address>` is the new IP of this node. Before altering this data, the node of the IP that has been changed needs to be shut down.

13. Bring up all the nodes in the cluster and test if everything is running properly.

Summary

In this chapter, we saw how easily a Vertica cluster can be scaled up or down. It is worth noting that I have over-emphasized on backing up the database, because it is always better to be safe than sorry. In the next chapter, you will learn how to effectively monitor a Vertica cluster.

Chapter 3. Monitoring Vertica

Monitoring is one of the key database administration tasks. It allows you to find possible bottlenecks in database performance in the most pragmatic fashion. In Vertica, there are several sources of information through which monitoring can be performed. They are as follows:

- System tables
- Logfiles
- Management Console (Enterprise Edition only)

Monitoring through the system tables

Vertica stores most of its information about the various states of the database in system tables. The process of storing such information is completely automatic and runs in the background. By querying these system tables and by relating various types of information fetched from these tables, you can efficiently monitor Vertica. The advantage of using system tables is that we can use them, just like any other user table, to perform aggregation, analytics, and so on. There are more than 100 system tables; we can get the whole list by running the following query:

```
Select * from system_tables;
```

The table shown in the following screenshot shows the output of the query:

```
db> select * from system_tables
db> ;
```

table_schema	table_name	table_description
v_catalog	resource_pools	Information about defined resource pools, both internal and dba-created
v_catalog	projection_columns	Projection columns information
v_catalog	user_procedures	User procedure information
v_catalog	all_tables	A complete listing of all tables and views
v_catalog	projection_checkpoint_epochs	Projection checkpoint epochs
v_catalog	schemata	Schema information
v_catalog	elastic_cluster	Information about cluster elasticity
v_catalog	resource_pool_defaults	Information about default values for resource pools properties, both internal and dba-created
v_catalog	dual	Oracle(TM) compatibility DUAL table
v_catalog	nodes	Node information
v_catalog	foreign_keys	Foreign key information
v_catalog	user_transforms	User Defined Transform Function information
v_catalog	license_audits	Database license compliance computation details
v_catalog	user_audits	User-requested database object size computation details
v_catalog	views	View information
v_catalog	primary_keys	Primary key information
v_catalog	roles	Role information
v_catalog	databases	Database information
v_catalog	constraint_columns	Table column constraint information
v_catalog	projection_delete_concerns	Projections that may have delete performance concerns
v_catalog	sequences	Sequence information
v_catalog	system_columns	System column information
v_catalog	comments	User comments on catalog objects
v_catalog	odbc_columns	An ODBC compliant listing of column metadata
v_catalog	passwords	User password history and password reuse policy
v_catalog	profile_parameters	Profile Parameters information
v_catalog	profiles	Profile information
v_catalog	user_functions	User Defined Function information
v_catalog	tables	Table information
v_catalog	types	Information about supported data types
v_catalog	view_columns	View column information

The system_tables table

On the basis of the type of information a system table stores, system tables can be grouped into the following categories:

- System information
- System resources
- Background processes
- Workload and performance

System tables are clustered into the following schemas (as shown in the preceding screenshot):

- `V_CATALOG`: This schema contains information about persistent objects in the catalog
- `V_MONITOR`: This schema contains information about the temporary system state

Examples of `V_CATALOG` tables include `views`, `roles`, `nodes`, `databases`, `license_audits`, and so on, while examples of `V_MONITOR` tables include `locks`, `disk_storage`, `resource_usage`, and so on.

We do not need to specify a schema (`V_CATALOG` or `V_MONITOR`) for querying system tables as they belong to the default schema. It should be noted that there are certain restrictions imposed on system tables; they are as follows:

1. Vertica provides a SQL-monitoring API that can be used on these system tables, but the database will not connect through this API if the Vertica cluster is in a recovering state.
2. Due to the nature of the data stored in these tables, it is not possible to perform **Data Definition Language (DDL)** and **Data Manipulation Language (DML)** operations on system tables.

Understanding a system table example

For example, we just need to run the following command to retrieve all the information from the tables of a database:

```
select * from tables;
```

The following screenshot shows the output of the command. Please note that in the `table_schema` and `table_name` tables, the columns are case sensitive.

```

km=> \x
Expanded display is on.
km=> select * from tables where table_name like 'ab%';
-[ RECORD 1 ]-----+-----
table_schema_id      | 45035996273704968
table_schema         | public
table_id             | 45035996273821300
table_name           | abc
owner_id             | 45035996273704962
owner_name           | dba
is_temp_table        | f
is_system_table      | f
system_table_creator |
partition_expression |
create_time          | 2013-01-04 16:10:33.227162+05:30
table_definition     |

```

The tables system table

Looking at events

There are various events such as [Emergency](#), [Alerts](#), [Critical](#), [Errors](#), [Warnings](#), [Notices](#), [Information](#), and [Debugs](#) that are logged by Vertica. Vertica logs these events in one of the following ways:

- In the [Vertica.log](#) file
- In the [ACTIVE_EVENTS](#) system table
- Using SNMP
- Using Syslog

This book will cover the usage of the [Vertica.log](#) files and the [ACTIVE_EVENTS](#) system table for logging events.

The following table lists the events that are logged:

Event name	Event type	Description
Low Disk Space	0	This event is logged either when there is not enough space in the physical disk or when there is a hardware issue.
Read Only File System	1	Vertica uses specific directories for storing Catalog or Data information. When Vertica is unable to create or write to files, this event is logged. Changing access rights may solve the issue.
Loss Of K Safety	2	Vertica defines the minimum number of nodes needed to run a database, particularly in the K-Safe mode. For example, for maintaining K-safety of level 2, we need at least five nodes; if the number of active nodes goes below five, this event will be logged. It should be ensured that all nodes are able to reach each other in the cluster.
Current Fault Tolerance	3	As discussed earlier, if a database loses K-safety, it shuts down, and this event is logged.

Event name	Event type	Description
at Critical Level		
Too Many ROS Containers	4	This may occur due to many reasons, primarily if there are a large number of local segments or if Mergeout is not working efficiently. It is suggested that you stop the data load in such events, and the database should be allowed to recover.
Node State Change	6	This event occurs when the node state has changed. For example, a node can change its state from startup to up.
Recovery Failure	7	This event occurs when a database is not able to recover properly from a nonfunctional state.
Recovery Error	8	If the number of recovery errors exceeds the value of Max Tries defined in the vbr.py configuration file, this event is triggered.
Recovery Lock Error	9	This event is logged when a recovering node is not able to secure lock on tables.
Recovery Projection Retrieval Error	10	This event is logged when Vertica is unable to retrieve information about a projection.
Refresh Error	11	This event is logged when a database is not able to refresh itself.

Event name	Event type	Description
Refresh Lock Error	12	This event is logged when a database is not able to obtain the necessary locks during refresh.
Tuple Mover Error	13	This event is logged when a database fails to move data from RAM (WOS) to physical disk (ROS or Read Optimized Store).
Timer Service Task Error	14	This event is logged when an error occurs in an internal scheduled task.
Stale Checkpoint	15	This event occurs when Write Optimized Store (WOS) data is not completely flushed into the Read Optimized Store (ROS) containers before shutdown. This may result in loss of data.

Looking at events through logfiles

During database startup, Vertica writes logs to the `dblog` file. This file resides in the `catalog-path/<database-name>/` directory path. For any information regarding the starting of data, in case of a failure or success, it is advisable to look into the `dblog` file. The following is the sample text of a `dblog` file:

```
Connected to spread at host
192.168.56.101(192.168.56.101) port 4803
[Thu 07 Mar 2013 16:41:49] SP_connect: DEBUG:
Auth list is: NULL
Connected to spread at host
192.168.56.101(192.168.56.101) port 4803
```

After the database is up and running properly, Vertica starts writing logs to `vertica.log`. Just like the `dblog` file, the `vertica.log` file is also maintained on each node of the cluster. It resides in the `catalog-path/<database-name>/<node-name_catalog>` directory path. The following is a sample text:

```

2013-01-
05 04:03:07.225 Main:0x1124df70 [Init] <INFO>
Starting up Vertica Analytic Database v6.0.0-1
2013-01-
05 04:03:07.225 Main:0x1124df70 [Init] <INFO>
Project Codename: BunkerHill
2013-01-
05 04:03:07.225 Main:0x1124df70 [Init] <INFO>
vertica(v6.0.0-
1) built by release@build2.verticacorp.com from
releases/VER_6_0_RELEASE_BUILD_0_1_20120611@95490
on 'Mon Jun 11 10:40:47 2012' $BuildId$
2013-01-
05 04:03:07.225 Main:0x1124df70 [Init] <INFO>
64-bit Optimized Build
2013-01-
05 04:03:07.225 Main:0x1124df70 [Init] <INFO>
Compiler Version: 4.1.2 20080704 (Red Hat
4.1.2-52)
2013-01-
05 04:03:07.225 Main:0x1124df70 <LOG>
@v_km_node0001: 00000/5081: Total swap memory
used: 94208

```

Looking at events through the **ACTIVE_EVENTS** system table

The **ACTIVE_EVENTS** system table can be used to look at various events logged by Vertica. The following table describes the various fields of the **ACTIVE_EVENTS** system table:

Column	Description
<code>node_name</code>	Internal name of the node
<code>event_code</code>	ID of the event type
<code>event_posted_timestamp</code>	Timestamp when the event was initiated
<code>event_expiration</code>	Timestamp when the event expired

Column	Description
<code>event_code_description</code>	Description of the event type
<code>event_problem_description</code>	Actual description of the event
<code>reporting_node</code>	Internal name of the node on which the event occurred

The following screenshot shows the sample output when we query the `ACTIVE_EVENTS` system table:

```

km=> select * from Active_events;
-[ RECORD 1 ]-----+-----
node_name          | v_km_node0001
event_code          | 6
event_id           | 6
event_severity      | Informational
event_posted_timestamp | 2013-03-07 16:55:25.455167+05:30
event_expiration    | 2081-03-25 20:09:32.455167+05:30
event_code_description | Node State Change
event_problem_description | Changing node v_km_node0001 startup state to UP
reporting_node      | v_km_node0001
event_sent_to_channels | Vertica Log
event_posted_count  | 1
-[ RECORD 2 ]-----+-----
node_name          | v_km_node0001
event_code          | 0
event_id           | 0
event_severity      | Warning
event_posted_timestamp | 2013-03-07 16:57:19.003508+05:30
event_expiration    | 2013-03-07 22:59:19.003508+05:30
event_code_description | Low Disk Space
event_problem_description | Warning: Low disk space detected (81% in use)
reporting_node      | v_km_node0001
event_sent_to_channels | Vertica Log
event_posted_count  | 1
-[ RECORD 3 ]-----+-----
node_name          | v_km_node0002
event_code          | 6
event_id           | 6
event_severity      | Informational
event_posted_timestamp | 2013-03-07 16:56:30.047495+05:30
event_expiration    | 2081-03-25 20:10:37.047495+05:30
event_code_description | Node State Change
event_problem_description | Changing node v_km_node0002 startup state to UP
reporting_node      | v_km_node0002
event_sent_to_channels | Vertica Log
event_posted_count  | 1

```

The ACTIVE_EVENTS system table

Monitoring Vertica through the Management Console

Vertica provides a very intuitive web console that can be installed and integrated with Vertica. Vertica Management Console is shipped only with the Enterprise Edition. It can help you perform almost every administration task that is possible through a command-based console.

Management Console runs only on browsers that support HTML5. By default, it is hosted on port 5450, which can be changed. To access it, we can just enter <https://xx.xx.xx.xx:5450/webui>, where `xx.xx.xx.xx` is the IP of the host on which the Management Console is hosted. It will prompt for a username and password; we can supply any credentials of any user who has admin rights. It is not advised to install Management Console on data-hosting nodes.

Retaining monitoring information

As discussed earlier, system tables provide a plethora of information regarding various processes running on a database, including queries. However, sometimes we need to store monitoring information. For this, we can employ **Data Collector**. Information retained by Data Collector is stored on a disk in the `DataCollector` directory under the Vertica catalog path.

Enabling and disabling Data Collector

To enable or disable Data Collector, we can use the `SET_CONFIG_PARAMETER()` function. Data Collector is on by default. To disable Data Collector, use the following command:

```
=> SELECT  
SET_CONFIG_PARAMETER('EnableDataCollector', '0');
```

Use the following command to enable Data Collector:

```
=> SELECT  
SET_CONFIG_PARAMETER('EnableDataCollector', '1');
```

Viewing the current data retention policy

To view the data retention policy, we can use the `GET_DATA_COLLECTOR_POLICY()` function. We can replace the component variable with the actual component name, as shown in the following line of code:

```
GET_DATA_COLLECTOR_POLICY( 'component' );
```

To get the whole list of components, we can query the `V_MONITOR.DATA_COLLECTOR` system table. In addition to the list, we will also get their current retention policies and statistics about how much data is retained.

Configuring data retention policies

We can set Data Collector policies using the `SET_DATA_COLLECTOR_POLICY()` function. Only a superuser can modify policies. This function, shown in the following line of code, allows us to

change how much memory (in kb) and disk space (in kb) to retain for each component on all nodes:

```
SET_DATA_COLLECTOR_POLICY('component',  
    'memoryKB', 'diskKB')
```

Failed nodes are not ignored from the policy, as they will create policy anomalies in the cluster. Hence, when failed nodes rejoin, the latest policy is imposed on them.

Monitoring data collection components

As discussed earlier, when the `DATA_COLLECTOR` system table is queried, we are presented with a list of Data Collector components, their current retention policies, and statistics about how much data is retained and how much has been discarded. `DATA_COLLECTOR` also calculates the approximate collection rate to aid in sizing calculations. The following is a simple query that returns all the columns in this system table:

[illegible]

```

-----
node_name          | v_km_node0001
component          |
AllocationPoolStatisticsBySecond
table_name         |
dc_allocation_pool_statistics_by_second
description        | Information about global
memory pools, which generally cannot be recovered
without restart (historical, by second)
access_restricted  | t
in_db_log          | f
in_vertica_log     | f
memory_buffer_size_kb | 64
disk_size_kb       | 256
record_too_big_errors | 0
lost_buffers       | 0
lost_records       | 0
retired_files      | 292
retired_records    | 29777
current_memory_records | 0
current_disk_records | 343
current_memory_bytes | 0
current_disk_bytes  | 219173
first_time         | 2013-03-08
16:12:41.002868+05:30
last_time          | 2013-03-08
16:15:32.001287+05:30
kb_per_day         | 107977.851408599

```

Summary

Vertica provides several ways to monitor various processes and facilitates the storage of the monitoring information. It is up to the administrator to determine how they can make the best use of it. In the next chapter, we will learn how to create a backup of Vertica databases and restore them.

Chapter 4. Backup and Restore

In this chapter, you will learn how to create database backups and restore backups in Vertica. Vertica provides the `vbr.py` script to back up, restore, and copy a database. We can create both full and incremental database snapshots, as well as snapshots of specific schemas or tables. For creating the most optimum backup, it is suggested that each node have its own dedicated backup host.

Requirements for backup hosts

The `vbr.py` utility lets us back up the database to one or more hosts (called backup hosts) that can be outside the database cluster. The backup hosts must have a password-less SSH access for the database administrator account. Also, backup hosts must have the same versions of Python and rsync as the main nodes. The `vbr.py` utility initially creates a snapshot of the database cluster of which the backup is being created. When the creation of the new snapshot is complete, it is moved to the designated backup location (host and directory). After Vertica copies the snapshot to the backup location, it deletes the snapshot created initially in the main cluster.

Generating the vbr.py configuration file

To invoke `vbr.py` to set up a configuration file, we should use the following command:

```
> vbr.py --setupconfig
```

The script prompts us to answer the following questions for parameters marked with an asterisk, `*`. For setting all the advanced options, we need to answer the last question with a "yes" during initial setup or change the configuration file manually. A configuration file contains various parameters categorized under various headers. The upcoming sections explain the headers and corresponding parameters present in the configuration file.

Miscellaneous settings

The following table shows the parameters present in the `Miscellaneous [Misc]` section of the file:

Parameters	Default	Description
<code>snapshotName*</code>	<code>snapshotName</code>	This value provides the prefix of the directory that will contain the snapshot. The characters in the value of <code>snapshotName</code> can include the following: • Aa-Zz • 0-9 • Period (.), hyphen (-), and underscore (_)
<code>tempDir</code>	<code>/tmp</code>	The <code>vbr.py</code> utility uses this directory path as the path on all nodes to store temporary data/files during the backup process. Since it is the universal

Parameters	Default	Description
		path, it must be present on all nodes in the cluster.
<code>verticaBinDir</code>	<code>/opt/vertica/bin</code>	If Vertica is installed on some other directory than the default location, we need to state where the bin files are present through this parameter.
<code>verticaConfig*</code>	<code>False</code>	Sometimes it is a good idea to backup configuration files. This parameter, when set to <code>true</code> , permits Vertica to back up config files along with data files.
<code>restorePointLimit*</code>	<code>1</code>	If we wish to store incremental snapshots, we need to increase the value of this parameter from 1. The permissible value range is from 1-99.
<code>objects*</code>	<code>None</code>	We can create a partial backup by providing a list of object names in the form of a comma-separated list which is included in a snapshot.
<code>overwrite</code>	<code>True</code>	This parameter is not part of the configuration file and needs to be included manually under the <code>[MISC]</code> header to change its default value. When set to <code>true</code> , a newer object overwrites the existing one in the database whenever OID conflicts occur during restore.

Parameters	Default	Description
<code>retryCount</code>	2	This number indicates the number of attempts or retries of the backup operation after a fatal error occurs.
<code>retryDelay</code>	1	This is the interval time in seconds in between retries for backup after a failure occurs.

Database access settings

The parameter under the `[Database]` header needs to be set for database access. The following table lists the parameters present in this section:

Parameters	Default	Description
<code>dbName</code>	N/A	As the name implies, it tells the utility which database to back up. If not supplied with a name, any live database is chosen as a backup candidate.
<code>dbUser*</code>	The current username	This parameter tells the utility which user is authorized to run <code>vbr.py</code> to back up or restore. The user must have admin rights.
<code>dbPromptForPassword*</code>	True	If this parameter is set to <code>False</code> , the utility will not prompt for a password at runtime. If set otherwise, we must also enter the database administrator password in the <code>dbPassword</code> parameter (discussed next).

Parameters	Default	Description
<code>dbPassword*</code>	None	This identifies the database administrator's password. The <code>vbr.py</code> utility doesn't encrypt the password, and hence precautions must be taken while supplying the password. If <code>dbPromptForPassword</code> is set to <code>true</code> and no password is supplied in the configuration file, we must supply it at runtime (this is more secure).

Data transmission during the backup process

The following table lists the parameters present in the `[Transmission]` section of the file:

Parameters	Default	Description
<code>encrypt</code>	<code>False</code>	When this parameter is set to <code>true</code> , data transmitted during the backup process is encrypted. As a rule of thumb, the utility may engage one whole core on both source and destination nodes to look for encryption. This may create a significant overhead and the performance may suffer.
<code>checksum</code>	<code>False</code>	In order to ensure that there is no data loss during transmission, we can set this parameter to <code>true</code> . When set to <code>true</code> , rsync performs MD5 checksum over the transmitted data to ensure integrity.
<code>port_rsync</code>	<code>50000</code>	This sets the port number on which rsync will work.

Parameters	Default	Description
<code>bwlimit</code>	0	This parameter defines the upper limit of the bandwidth of the network to be used in kbps during backup.
<code>hardLinkedLocal</code>	False	This configuration parameter, when set to <code>true</code> , makes the utility create a backup on a local directory instead of a remote location. A local backup is created by hard links instead of copying data.

Mapping

For each node of the database cluster, a separate mapping heading needs to be written. Each heading is numbered (`[Mapping1]`, `[Mapping2]`, and so on) and controls the backup of the respective node. All of the parameters under this header are mandatory in the configuration file. The following table lists the parameters:

Parameters	Default	Description
<code>backupHost*</code>	None	As the name implies, this tells the utility the hostname of the backup node.
<code>backupDir*</code>	None	This is the path to the directory that will host the backup files on the designated host or node where the backup will be stored. This directory must exist and the user must have all rights to it.
<code>dbNode</code>	None	This is Vertica's internal name of the host usually in the form <code>v_databasename_node00xx</code> .

Creating full and incremental backups

In this section, you will learn about the process of creating full and incremental backups. But first, let's understand the requirements.

Understanding the requirements

Before we create a snapshot, we must check the following:

- The database is up and running.
- All of the backup hosts are up and available to the cluster.
- The user account has all permissions to the target directories on the backup node.
- By default, the `vbr.py` utility searches for the config file at `/opt/vertica/config/vbr.ini`, and if it is not found, the utility exits with an error. We can supply the config file using the `--config-file` parameter with `vbr.py`.

It should be noted that if some nodes are down but the database is running in K-Safe mode, then the backup will still be created.

Running vbr.py

To run the `vbr.py` utility, use the following command:

```
> vbr.py --task backup --config-file myconfig.ini
Copying...
xxxxxxx out of xxxxxx, 100%
All child processes terminated successfully.
Committing changes on all backup sites...
backup done!
```

Incremental snapshots

If `restorePointLimit` is set to a value more than 1, then after creating a full snapshot, the `vbr.py` utility will create the future backups in an incremental manner. This incremental backup process works in the following manner:

1. The utility obtains the value of the `restorePointLimit` parameter from the configuration file.
2. If the value is more than 1, then after creating a full backup, incremental backups will be created on subsequent invocation of the

utility. It is suggested to create manual backups of snapshots periodically.

3. If after creating the next snapshot the total number of snapshots exceeds the restore point limit, `vbr.py` automatically deletes the oldest snapshot to maintain the count.

Creating schema and table snapshots

As discussed earlier, it is possible to create object-specific snapshots. These objects can be certain tables or schemas. As mentioned in the table, there are the following two configuration file parameters:

- `Objects`
- `Overwrite`

Refer to the table under the *Miscellaneous settings* section for more information.

Restoring full database snapshots

Restoring full database snapshots in Vertica can be a little tricky due to various constraints. The following are the constraints:

- The database must be down. It is good to have another database cluster to serve the application by the time the restore is complete on the primary cluster.
- The new cluster should have the same number of hosts as the base cluster of which the backup is created. This constraint is included to tackle K-Safety issues that might arise later. In fact, it is important to keep even node names and the IP addresses the same as that of the nodes in the base cluster.

We can use a full database snapshot created in Vertica 5.0 to restore into a 6.0 database. To begin a full database snapshot restore, log in using the database administrator's account created during installation. We cannot run the utility as root.

Restoring from a specific snapshot

At first, the utility restores the most recent snapshot, though it is possible to control which snapshot needs to be restored in case we have saved multiple versions. It is important to bear in mind that snapshots are restored to the same database from which they were created. In addition to that, you cannot restore a partial snapshot into an empty database. The following example exhibits restoring a database using the settings in the `myconfig.ini` config file:

```
> vbr.py --task restore --config-file  
myconfig.ini  
Copying...  
xxxxxx out of xxxxxx, 100%  
All child processes terminated successfully.  
restore done!
```

Restoring from the most recent snapshot

To restore from the most recent of several snapshots, we can use the `--archive` option of the `vby.py` command, specifying the full snapshot name with its date and time. The following example illustrates restoring a database snapshot using the settings provided in the `myconfig.ini` config file, which also contains the superuser's password:

```
> vbr.py --task restore --config-file  
myconfig.ini --archive=20131210_308564
```

The `--archive` parameter recognizes the archive directory created on October 12, 2011 (`_archive20131210`) at timestamp `308564`. Note that the configuration file contains the snapshot name of the directory, and while running the command, we just need to specify the date and timestamp. If the snapshot directory has the `_archive` suffix, the directory is archived.

Restoring schema and table snapshots

We can restore object-level snapshots only to the database from which they were created. Only the snapshots created in the same backup are compatible. We can restore object-level snapshots in the same backup location after a full-database snapshot restore. It should be noted that we cannot restore any object-level snapshot into an empty database.

Copying a database from one cluster to another

As discussed earlier, certain administration tasks require the database to shut down. In that case, it is good to have a secondary cluster. The `vbr.py` utility can do this activity of copying the database from one cluster to another using the `copycluster` command task. We can create a config file explicitly for copying the database from the base cluster to another cluster. In this config file, it is required to provide the hostnames or IP addresses of nodes of the target cluster as the backup hosts.

Please note that we can use neither incremental snapshots nor object-level snapshot files with the `copycluster` command. We can only use a full database backup file.

The following example config file can be used to copy a database on a three-node cluster to another three-node cluster (secondary cluster) comprising nodes named `bck_host01`, `bck_host02`, and `bck_host03`. Please note that although `backupHost` is used, `backupDir` is not used. This is because it is a cluster-wide doubling and not just a data backup.

```
[Misc]  
snapshotName = km_001  
tempDir = /data/tmp  
restorePointLimit = 1  
verticaConfig = False  
retryCount = 1  
retryDelay = 1  
[Database]  
dbName = km  
dbUser = dba
```



```

dbPassword = sql
dbPromptForPassword = False

[Transmission]
encrypt = True
checksum = False
port_rsync = 50000
bwlimit = 0
hardLinkLocal = False

[Mapping0]
dbNode = v_km_node0001
backupHost = bck_host01

[Mapping1]
dbNode = v_km_node0002
backupHost = bck_host02

[Mapping2]
dbNode = v_km_node0003
backupHost = bck_host03

```

Copying the database

The following example exhibits copying a cluster using a config file located in the current directory:

```

> vbr.py --config-file CopyCluster.ini --task
copycluster
Copying...
xxxxxxx out of xxxxxx, 100%
All child processes terminated successfully.
copycluster done!

```

The `vbr.py` utility should be invoked from the source cluster and not from the destination cluster, else the backup database will be copied to the primary cluster. The user must be a superuser.

Using database snapshot functions

Apart from the `vbr.py` utility, we can also use database snapshot functions that let us create snapshots and remove snapshots. It is worth noting that database snapshot functions do not provide us with great flexibility as `vbr.py` does. By using database snapshot functions, we are limited to creating a full image backup of the database on the respective nodes. For example, if we have a three-node cluster, then a backup of each node will be created on the respective nodes; this is unlike `vbr.py`, where we have an option to migrate them to other nodes.

There are the following two types of snapshots that we can create using these functions:

- **Durable snapshots:** These are like hard links to the actual data files of the database with a proper directory structure. So whatever changes are made, the storage containers will be readily available through these hard links. These snapshots remain persistent throughout.
- **Non-durable snapshots:** This type of snapshot consists of the actual files containing the database's data and not hard links. So any change made after the snapshot has been created will not be reflected in these snapshots.

There are mainly two types of database snapshot functions, as follows:

- `Database_snapshot`
- `Remove_database_snapshot`

Creating database snapshots

We will use the `DATABASE_SNAPSHOT` SQL function to create snapshots. This function requires the following two parameters:

- The name of the snapshot.
- Whether it is durable or not (true/false). In the case of true, a durable snapshot will be created. In the case of false, a non-durable snapshot will be created.

The following example illustrates the use of the `Database_snapshot` function:

```

=> SELECT DATABASE_SNAPSHOT('snapshot_1', true);
DATABASE_SNAPSHOT
-----
v_km_node0001,v_km_node0002,v_km_node0003
(1 row)

=> SELECT DATABASE_SNAPSHOT('snapshot_2', false);
DATABASE_SNAPSHOT
-----
v_km_node0001,v_km_node0002,v_km_node0003
(1 row)

```

If we wish to check what snapshots have been created, we can check the `Database_snapshots` system table. This system table will not have data of historic snapshots. The following is the sample output when we query the `Database_snapshots` system table:

```

km=> \x
Expanded display is on.
km=> select * from database_snapshots;
-[ RECORD 1 ]-----+-----
node_name           | v_km_node0001
snapshot_name       | snapshot_1
is_durable_snapshot | t
total_size_bytes    | 2528158887
storage_cost_bytes  | 1425466
acquisition_timestamp | 2013-03-08 10:45:52+05:30
-[ RECORD 2 ]-----+-----
node_name           | v_km_node0002
snapshot_name       | snapshot_1
is_durable_snapshot | t
total_size_bytes    | 2507735450
storage_cost_bytes  | 1521584
acquisition_timestamp | 2013-03-08 10:46:52+05:30
-[ RECORD 3 ]-----+-----
node_name           | v_km_node0003
snapshot_name       | snapshot_1
is_durable_snapshot | t
total_size_bytes    | 2507696427
storage_cost_bytes  | 1491310
acquisition_timestamp | 2013-03-08 10:45:31+05:30
km=> █

```

The `snapshots` subdirectory in the `catalog` directory stores both durable and non-durable snapshots on each node of the cluster. Both types of snapshots have two files, which are as follows:

- `snapshotname.txt`: This file contains an overview of the snapshot. The following is the output of a durable snapshot:

```

[dba@host01 Snapshots]$ cat snapshot_1.txt
:SnapshotInfo
isObject:false
name:snapshot_1
nodeName:v_km_node0001
time:416034941611656
backupEpoch:363
mementos:
(
  :SnapshotMemento
  oid:45035996273839628
  createdForDBSnap:true
  nodeToOidGenInfo:
  (
    :OidToOidTimeTzPair
    first:45035996273704972
    second:
      :OidTimeTzPair
      first:45035996273839628
      second:416034952035388
      .
    :OidToOidTimeTzPair
    first:45035996273713830
    second:
      :OidTimeTzPair
      first:49539595901138914
      second:416035012890715
      .
    :OidToOidTimeTzPair
    first:45035996273713892
    second:
      :OidTimeTzPair
      first:54043195528509114
      second:416034931460561
      .
  )
  sequence:1
  .
)
ahmEpoch:363
nodesInvolved:
(
  :string
  _:v_km_node0001
  .
  :string
  _:v_km_node0002
  .
  :string
  _:v_km_node0003
  .
)

```

- [snapshotname.ctlg](#): This file contains all information pertaining to the database catalog.

Removing snapshots

We can manually remove snapshots using the [REMOVE_DATABASE_SNAPSHOT](#) SQL function, as shown in the following commands:

```

=> SELECT REMOVE_DATABASE_SNAPSHOT(snapshot_1');
-
[ RECORD
1 ]-----
+-----
REMOVE_DATABASE_SNAPSHOT | Removed:
v_km_node0001,v_km_node0002,v_km_node0003

=> SELECT REMOVE_DATABASE_SNAPSHOT('snapshot_2');
-
[ RECORD
1 ]-----
+-----

REMOVE_DATABASE_SNAPSHOT | Removed:
v_km_node0001,v_km_node0002,v_km_node0003

```

Now we can check the `Database_snapshot` system tables. Because we have removed all the snapshots, we will find no rows, as shown in the following command:

```

=> SELECT * FROM DATABASE_SNAPSHOTS;
(No rows)

```

Apart from manually removing the snapshots we set, there are two parameters that allow Vertica to perform automatic removal of snapshots after a certain time interval. They are shown in the following table:

Parameters	Description	Default	Example
<code>RemoveSnapshotInterval</code>	This is the interval time in seconds between two checks that Vertica performs for automatically removing snapshots.	3600	<pre> SELECT SET_CONFIG_PARAMETER ('RemoveSnapshotInterval', 7200); </pre>
<code>SnapshotRetentionTime</code>	This is a lifetime of snapshots in seconds. Vertica will not remove a snapshot	3600	<pre> SELECT SET_CONFIG_PARAMETER ('SnapshotRetentionTime', 7200); </pre>

Parameters	Description	Default	Example
	automatically until this time limit is reached.		

Summary

A good backup strategy helps you safeguard against possible data losses in case of any catastrophic events. Vertica provides highly flexible and configurable backup and restore functions, which when optimally used can provide a satisfactory level of data backup.

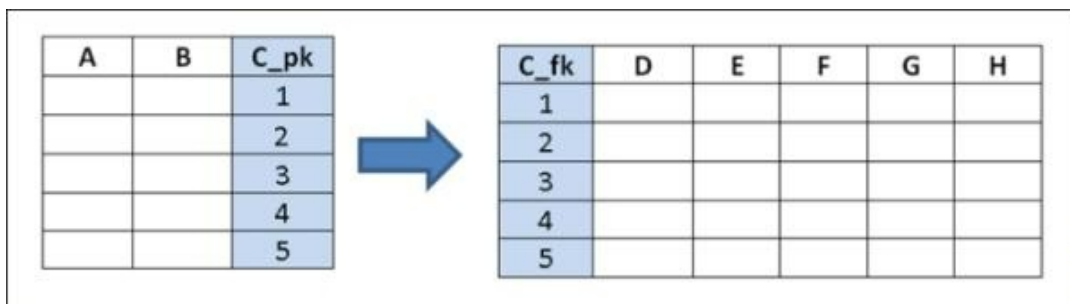
In the next chapter, we will discuss performance tuning in Vertica and some basic concepts around it.

Chapter 5. Performance Improvement

A lot has always been discussed on database performance improvement techniques. In Vertica, because there is no concept of indexes, a major thrust for performance improvement lies upon the concept of **projections**. Apart from projections, this chapter will also discuss the storage models present in Vertica.

Understanding projections

Traditional databases use row-store architecture. To process a query, a row store reads all of the columns in all of the tables mentioned in the query, regardless of the number of columns a table has. Often, analytical queries access (or require to access) only few columns of a table containing up to several hundred columns, thus making a whole-column-scan unwarranted. Additionally, a whole-column-scan also results in the retrieval of a lot of unnecessary data. Unlike other RDBMSes, Vertica reads the columns from database objects called **projections**. Consider the following example:



Now, suppose we have the following query:

```
SELECT A, D, E
FROM Table1 JOIN Table2
ON Table1.C_pk = Table2.C_fk;
```

On execution of the preceding query, a row store will typically scan through all columns of each table from physical storage, while a column store such as Vertica will only read three columns.

Projections constitute the physical schema of Vertica. In Vertica, a table only serves as a logical schema and doesn't occupy any physical space. All data is stored in the form of projections. It should be noted that there can be more than one projection for a table, that is, the same data can be present in multiple projections. Even though data is repeated, projections consume

less space than speculated. A minimum of one projection should exist for each table in the database. This makes sense as projections constitute the physical schema in Vertica. Vertica automatically creates a projection as soon as a table is created. This first projection is called a **superprojection**. A superprojection contains all columns of a table, thus making sure that all SQL queries can be answered. It should be noted that superprojections are a kind of support projection and are generally not very efficient.

Projections can be compared with **materialized views (MVs)** and indexes. One key difference between projections, MVs, and indexes is that projections are primary storage and there is no underlying table. In contrast, both MV and indexes are secondary storage and there are one or more underlying tables.

In the traditional sense, Vertica has no raw uncompressed base tables, no materialized views, and no indexes. Everything is taken care of by projections. Of course, our logical schema remains the same as with any other database, that is, tables.

Looking into high availability and recovery

The Vertica database is believed to be K-Safe if any K node(s) fail without causing the database to shut down. In a K-Safe cluster, when a failed node recovers, it joins the database cluster again. All data changes made during the time when the node was down are recovered by querying other nodes in the cluster.

In Vertica, the value of K can be 0, 1, or 2. A K-Safe value implies the maximum number of nodes that can be down at a single point in time without affecting the working of a database. So, if a database with K-safety of one ($K=1$) loses a node, the database will continue to run normally. Similarly, a database with a K-Safe value of 2 ($K=2$) will ensure that Vertica can run normally if any two nodes fail at any given point of time. For each K-Safe value, there is a minimum number of nodes that is required. The following table illustrates this:

K-level	Number of nodes required
0	1+
1	3+

K-level	Number of nodes required
2	5+
K	$(K+1)/2$

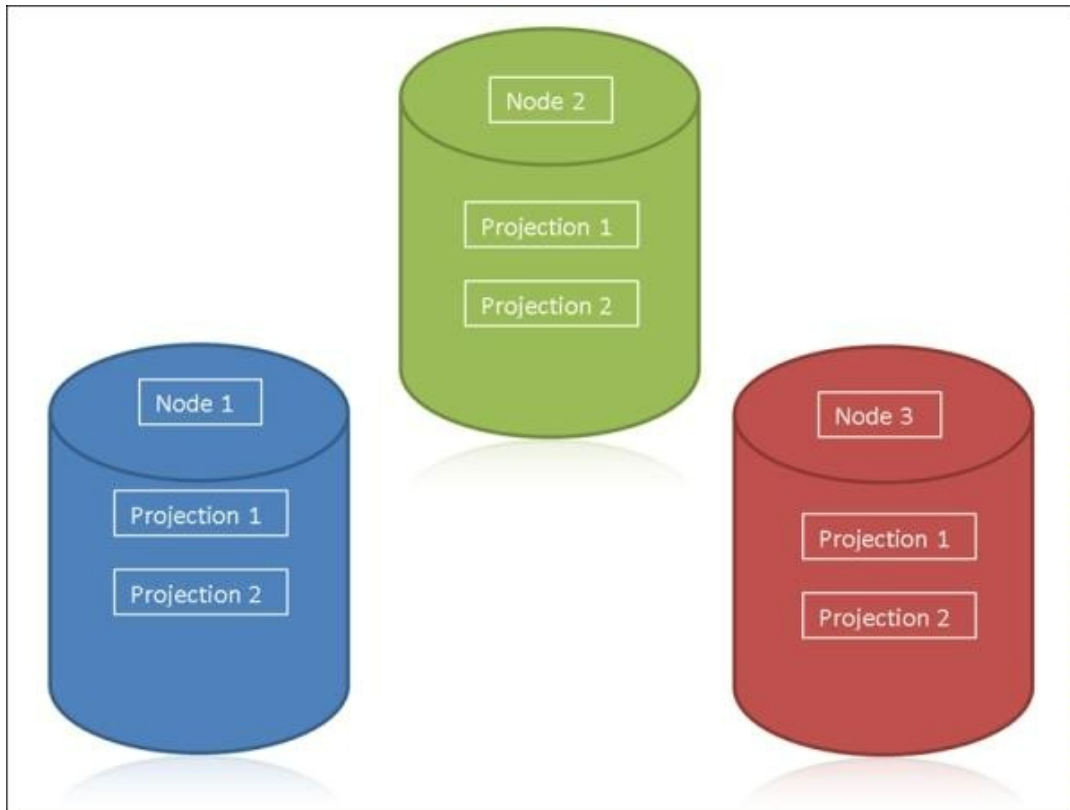
Vertica doesn't officially support values of K greater than 2.

Projections on a cluster can be stored in the following two ways:

- Unsegmented
- Segmented

Comprehending unsegmented projections

These types of projections are not broken and distributed on different nodes, but rather a copy of the entire projection is maintained on each node, that is, data is replicated across the nodes. It is suggested that small dimensional tables should remain unsegmented. The following illustration shows two projections: **Projection 1** and **Projection 2**. These are replicated across a three-node cluster but unsegmented:



Comprehending segmented projections

In segmented projections, a projection is broken into chunks or segments, and a copy of these segments is maintained on different nodes of the cluster. These replicated projections are known as **buddy projections**. These projections are recommended for fact tables and dimensional tables with a large amount of data. Consider an example where there is a cluster of three nodes and there is a projection, Proj 1, that needs to be segmented. In order to maintain data on each node, we will need to create two more copies of data: Proj 1 BP1 and Proj 1 BP2. Each of these projections, that is, Proj 1, Proj 1 BP1, and Proj 1 BP2, will be broken into three equal parts and distributed on different nodes. The following example illustrates this more clearly:



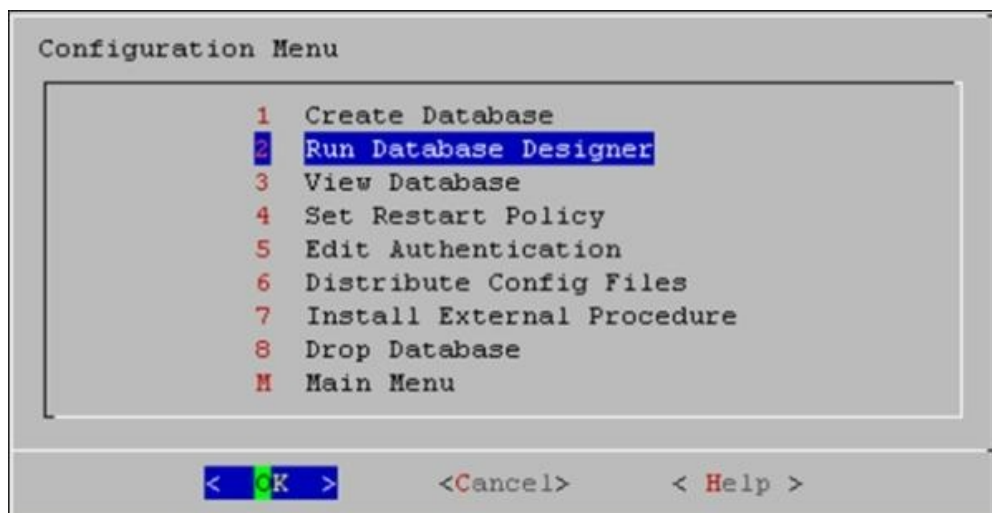
Creating projections using Database Designer

Vertica provides a UI-based tool known as **Database Designer**, which recommends projections on the basis of queries. The following information should be given to Database Designer to create the best projections:

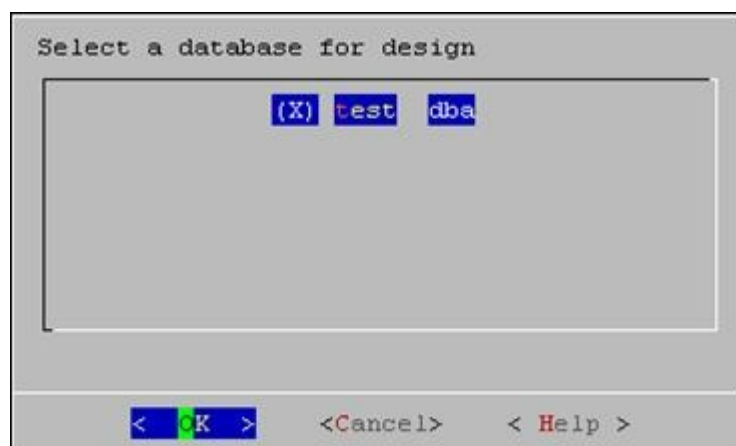
- A set of queries to be run or something similar
- A K-Safe level
- The database should already have tables ready with sample data (not more than 10 GB)

The following are the steps to work on Database Designer:

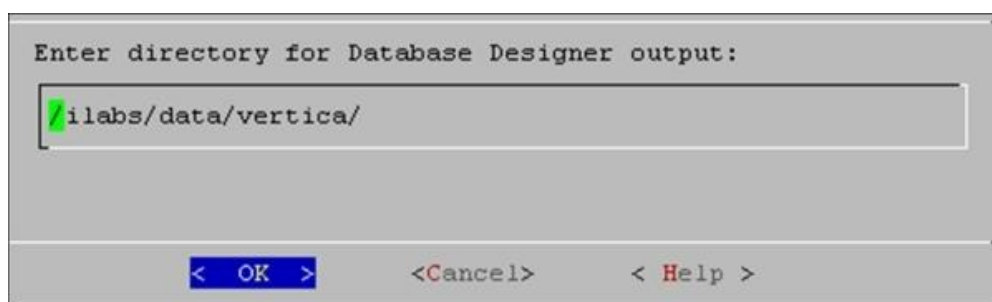
1. We can start Database Designer by navigating to **Configuration Menu | Run Database Designer** from the admin tools menu, as shown in the following screenshot:



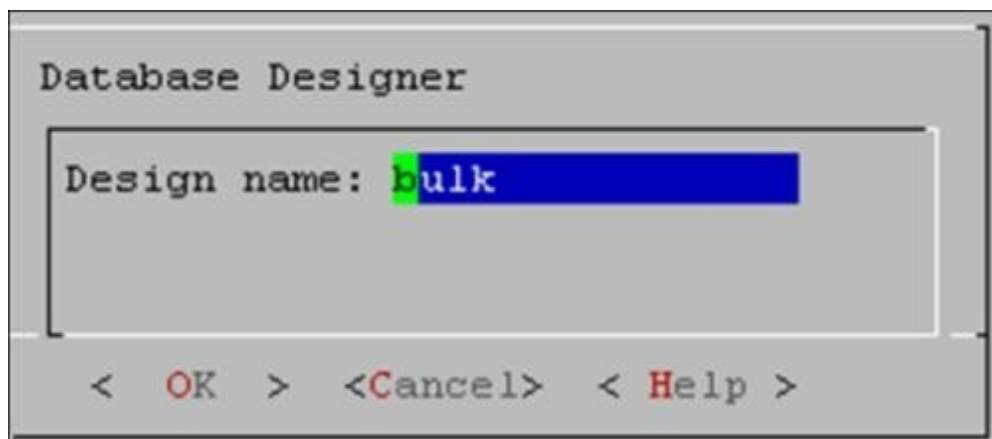
2. Select a database for the design, as illustrated in the following screenshot:



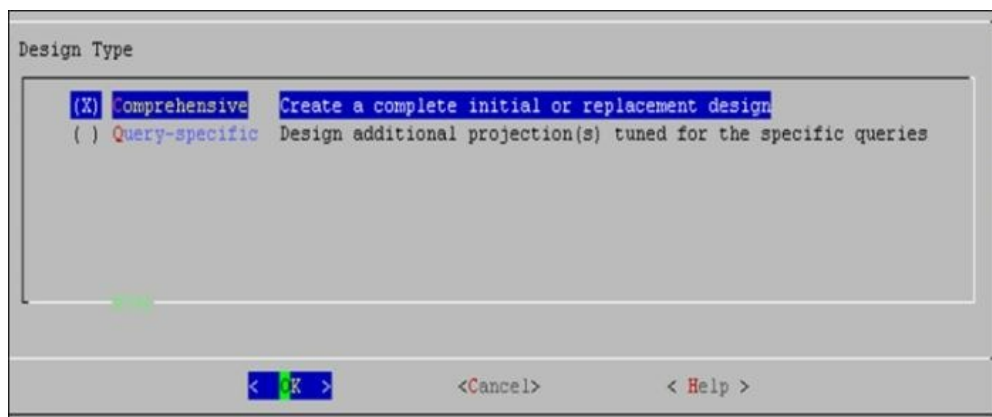
3. Enter the directory where the output of Database Designer will be stored, as shown in the following screenshot:



4. Then, enter the design name, as shown in the following screenshot:



5. The preceding steps were generic. From now, the process forks as per the design type we select:



There are two types of designs:

- Comprehensive design
- Query-specific design

The comprehensive design

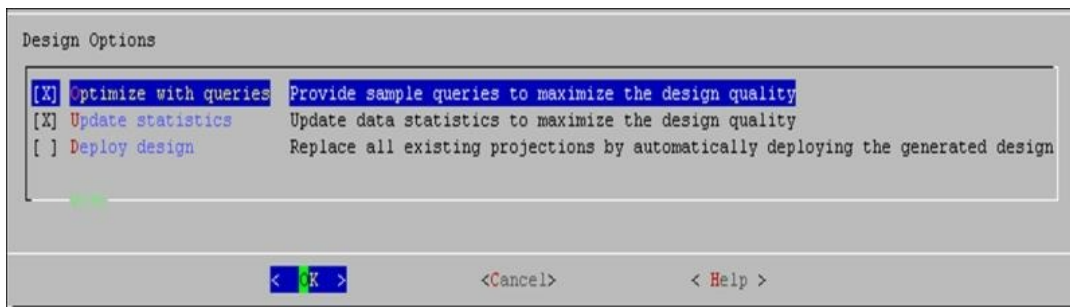
Comprehensive design lets us design projections for all the tables in the specified schemas. It is suggested to create a comprehensive design when we create a new database, but a comprehensive design for an existing database can also be created if the entire database is not performing up to the mark.

The following are the options for **Comprehensive** design that will help to control the design in a better fashion:

- **Optimize with queries:** This option lets us supply queries that will be taken into consideration by Database Designer for the design.

- **Update statistics:** When this option is selected, the statistics are collected and refreshed. Like any other database technology, precise statistics help Database Designer enhance the compression and query performance.
- **Deploy design:** When this option is selected, Database Designer deploys the new database design to our database automatically. Although Database Designer saves the SQL script for the new design (projections) in the design location, if needed, we can review the scripts and deploy them manually later.

The following screenshot shows the afore mentioned options:



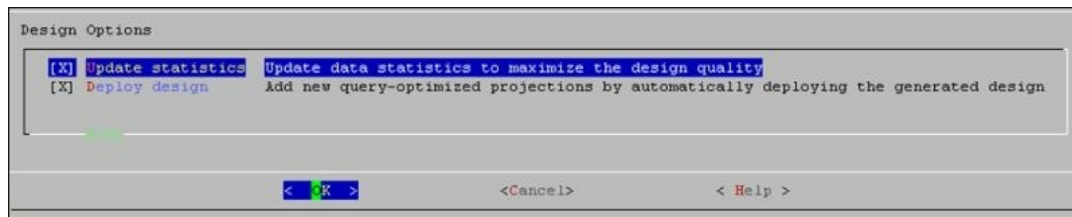
The query-specific design

Sometimes, we just need to optimize the performance of certain queries. For this purpose, a query-specific design can be employed. It should be noted that a query-specific design will limit design suggestions to the queries we supply. Projections thus created/suggested may hamper, although rarely, the performance of other queries.

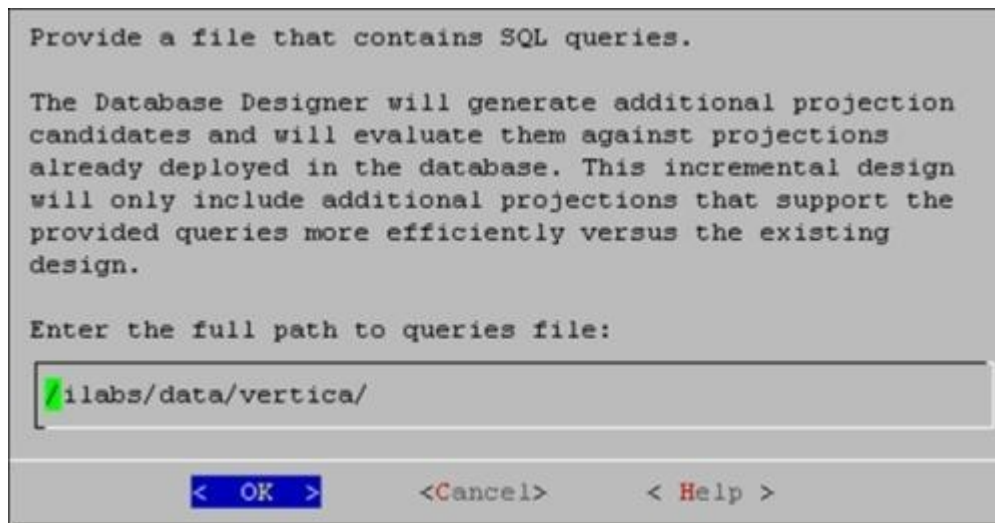
The query-specific design process lets us specify the following options:

- **Update statistics:** When this option is selected, the statistics are collected and refreshed. Like any other database technology, precise statistics help Database Designer enhance the compression and query performance.
- **Deploy design:** This option installs the new database design. In the process, new projections are added to the database. Later, these projections are populated with data. It should be noted that none of the existing projections are affected by the deployment of a new projection in this process.

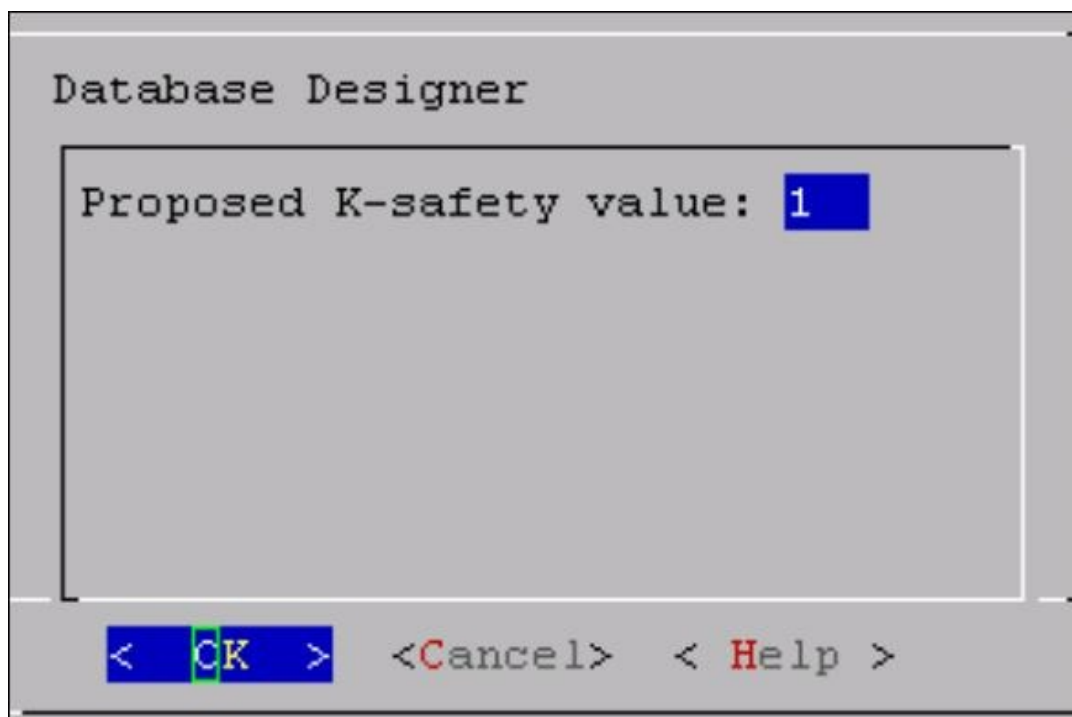
The following screenshot shows the afore mentioned options:



If we wish to select the **Comprehensive** design with the **Optimize with queries** option or the **Query-specific** design, then we need to provide a set of queries, as shown in the following screenshot:



After this, just provide the K-Safe value to start the process, as shown in the following screenshot:



Creating projections manually

If the projection created by Database Designer doesn't meet our needs, then we can write custom projections. Vertica uses the `CREATE PROJECTION` command to create projections. The following is an example:

```
CREATE Projection Projection1
(
    col1  ENCODING RLE,
col2,
    col3      // Column List and Encodings
) AS
SELECT
    t1.col1,      // Base table query
    t1.col2,
    t2.col1
FROM table1 t1
INNER JOIN table2 t2 on t1.col1 = t2.col1
ORDER BY t1.col1

SEGMENT BY HASH (t1.col1) ALL NODES OFFSET 1;
//Segmentation clause
```

Column list and encoding

The column list and encoding lists every column within the projection corresponding to the base query (described later in this section) and defines the encoding for each column. For Vertica, it is advisable to use data encoding as it results in fewer disk I/O.

The base query

The base query identifies all of the columns to incorporate in a particular projection. Base queries are just like standard SQL queries with the exception that a base query for large table projections can contain only PK/FK joins from smaller tables.

The sort order

Presorted projections are one of the key ingredients for high-performance queries. We can also optimize a query by matching the projection's sort order to the query's `GROUP BY` clause. Database Designer does not optimize for `GROUP BY` queries. Using the `GROUP BY` pipeline optimization might defeat other optimizations based on the predicate, especially if the predicate is very selective. Therefore, it is suggested not to use it or use it only when

exclusively required. So, as a tip, when predicates are not very selective or they are absent, `GROUP BY` can be used in projections.

Segmentation

As discussed earlier, the `SEGMENTATION` clause will tell a database to keep projections in a segmented or unsegmented manner.

Segmentation maximizes database performance by distributing the load. It is advisable to use `SEGMENT BY HASH` to segment large table projections.

For small tables, it is better to keep data unsegmented or non-partitioned. The `UNSEGMENT` keyword will direct Vertica to do the same. Hence, data will just be replicated and not partitioned nor segmented.

Keeping K-safety (K-Safe) in mind

We can replicate data on all nodes using the `UNSEGMENTED ALL NODES` keyword. This will ensure high availability and fault tolerance. For large tables, it is imperative to segment corresponding projections. In order to make segmented data fault tolerant and highly available or K-Safe, we need to do the following:

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

Creating buddy projections

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

1. Rename it to something similar to the name of the original projection. For example, a projection named `retail_sales_fact_P1` could have buddies named `retail_sales_fact_P1_B1` and `retail_sales_fact_P1_B2`.
2. Modify the sort order, if needed.
3. Create an offset to store the segments for the buddy on different nodes. For example, the first buddy in a projection set would have an offset of one (`OFFSET 1;`), the second buddy in a projection set would have an offset of two (`OFFSET 2;`), and so on.

A note on table partitioning

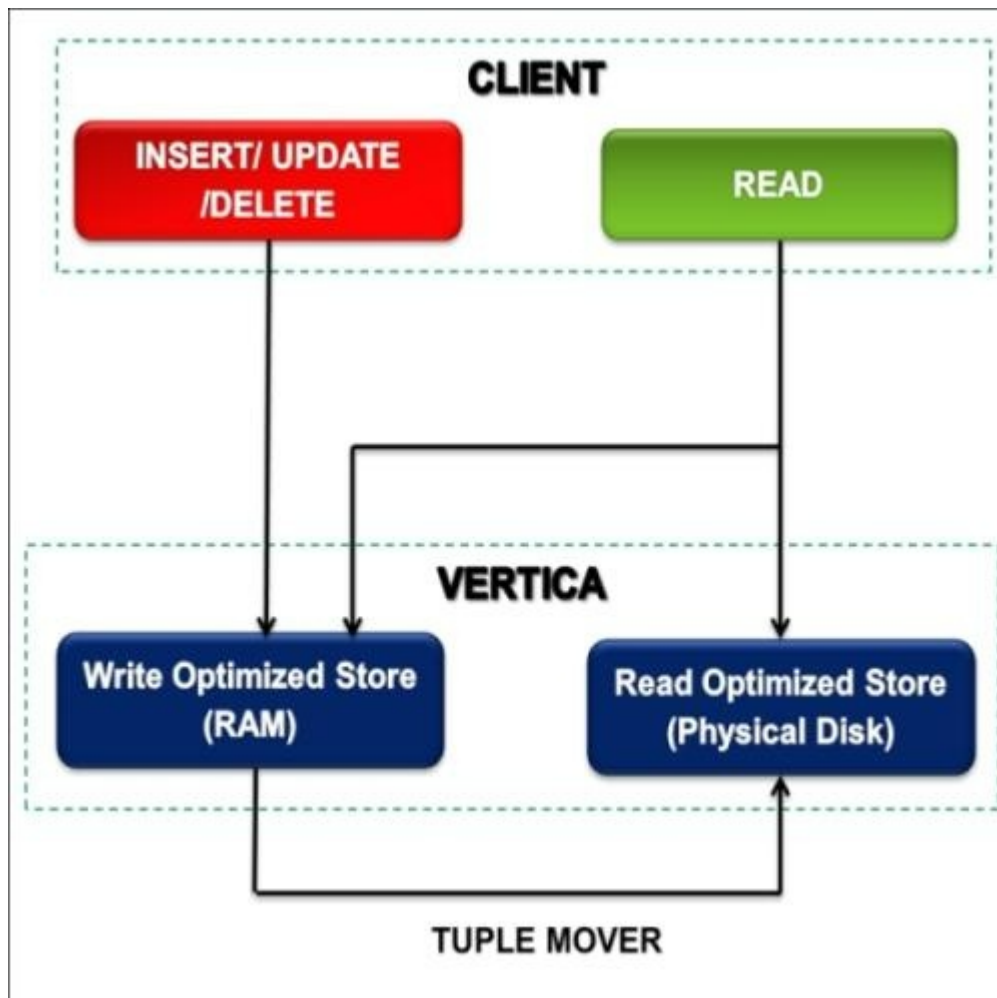
Vertica supports data partitioning at a table level, which divides one large table into smaller pieces. The partition remains exclusive to a table and

applies to all projections of a given table. In Vertica, tables just form a logical entity and not a physical one; hence, a table partition is also a logical entity. This should not be confused with the segmentation of projections as the latter works at a physical level.

A common use for partitions is to split a table by time periods. For instance, in a table that contains decades of data, we can partition by year or by month. Partitions can improve parallelism during query execution and also enable some optimizations that can improve query performance, at the same time decreasing the deletion time.

Understanding the storage model in Vertica

In order to cater to a wide array of tasks such as DML, bulk loading, and querying, Vertica implements a storage model as shown in the following illustration. This model is the same on each Vertica node.



The **Read Optimized Store (ROS)** resides on a physical disk storage structure and is organized by projection. Compression is employed at the ROS level to ensure that the disk space occupied by projections is minimal.

Unlike the ROS, the **Write Optimized Store (WOS)** resides on primary memory, again organized by projection. The WOS stores and sorts the data by epoch. It doesn't compress the data at this level, ensuring high speed writes.

To learn more on how we can control data writing to WOS and ROS, refer to the Bulk Loading section.

Tuple Mover (TM) is a Vertica background process, which is responsible for moving data from primary memory (WOS) to physical disk (ROS). Apart from that, TM is also responsible for removing deleted data and coalescing small ROS containers.

Tuple Mover operations

Tuple Mover performs the following two operations:

- [Moveout](#)
- [Mergeout](#)

Each data node is responsible for running its own TM tasks, that is, [Moveout](#) and [Mergeout](#) at its own interval of time.

Moveout

[Moveout](#) is responsible for moving data from primary memory (WOS) to physical disk (ROS). Every time a [Moveout](#) operation is performed, a new ROS container is created. In the case of [COPY...DIRECT](#) (refer [Chapter 6](#), *Bulk Loading*), new ROS containers are formed. An ROS container is nothing but a set of rows of a projection stored in a group of files. An ROS container contains data pertaining to a single table partition only; however, a table partition can be present in multiple ROS containers. The [STORAGE_CONTAINERS](#) system table stores all information regarding the execution of [Moveout](#), as shown:

```
km=> select * from storage_containers;
-[ RECORD 1 ]-----+-----
node_name          | v_km_node0001
schema_name        | public
projection_id       | 45035996273716454
projection_name     | timedata_b1
storage_type        | ROS
storage_oid         | 45035996273716511
total_row_count     | 931
deleted_row_count   | 0
used_bytes          | 5095
start_epoch         | 6
end_epoch           | 6
grouping            | PROJECTION
segment_lower_bound | 2863311530
segment_upper_bound | 4294967294
-[ RECORD 2 ]-----+-----
node_name          | v_km_node0001
schema_name        | public
projection_id       | 45035996273716440
projection_name     | timedata_b0
```

```

storage_type      | ROS
storage_oid       | 45035996273716531
total_row_count   | 945
deleted_row_count | 0
used_bytes        | 5106
start_epoch       | 6
end_epoch         | 6
grouping          | PROJECTION
segment_lower_bound | 4294967295
segment_upper_bound | 1431655764

```

Mergeout

A [Mergeout](#) operation works at the ROS level and is responsible for removing deleted data and coalescing small ROS containers. When an increased number of ROS containers hinders the performance, Tuple Mover automatically performs a [Mergeout](#) operation.

Tuning Tuple Mover

We can tune and tweak certain parameters to control Tuple Mover as per our need. The following table illustrates various configurable parameters for Tuple Mover:

Parameters	Description	Default	Example
ActivePartitionCount	By default, only a single (and the latest) partition of a partition table is loaded at any particular time. If we wish to increase the number of partitions that can be loaded at a given time, we need to change this parameter.	1	<pre> SELECT SET_CONFIG_PARAMETER('ActivePartitionC 2); </pre>
MergeOutInterval	Defined in seconds, this tells Tuple Mover to wait for a certain time interval between two Mergeout operations. It is advised to adjust it as per the frequency of ROS container creation.	600	<pre> SELECT SET_CONFIG_PARAMETER('MergeOutInterval </pre>

Parameters	Description	Default	Example
<code>MoveOutInterval</code>	Defined in seconds, this tells Tuple Mover to wait for a certain time interval to check for new data in WOS to flush data to ROS.	300	<pre>SELECT SET_CONFIG_PARAMETER ('MoveOutInterval',</pre>
<code>MoveOutMaxAgeTime</code>	Defined in seconds, this tells Tuple Mover to wait for a certain time interval between two <code>Moveout</code> operations.	1800	<pre>SELECT SET_CONFIG_PARAMETER ('MoveOutMaxAgeTime', 1200);</pre>
<code>MoveOutSizePct</code>	This defines the percentage of the primary memory or WOS that can be filled with data before Tuple Mover forcefully performs a <code>Moveout</code> operation.	0	<pre>SELECT SET_CONFIG_PARAMETER ('MoveOutSizePct',</pre>

Adding storage locations

It is imperative to add additional storage locations (read directories) as the size of the database grows. Moreover, we can keep crucial files on a high performance physical memory, such as flash drives, in order to boost performance. The following are the points that should be kept in mind before adding an extra storage location:

- Catalog and Data directories should be different.
- Catalog and Data directories should not be shared among nodes.
- A new directory is empty and Vertica has all its rights.

We can check for existing data storages for the whole cluster by running the following query:

```
SELECT * from V_MONITOR.DISK_STORAGE;
```

Before adding storage locations, we should decide what type of information we wish to store in this new location. It can be one of the following:

- Any kind of data (projections): `DATA`.
- Temporary files produced as a by-product of certain database processes: `TEMP`.
- Both data and temp files. This is the default option: `DATA, TEMP`.
- We can create locations specific for a non-admin user. It will only store data: `USER`.

Adding a new location

The `ADD_LOCATION()` function is used as a query to initialize the new data directory, the node (optional), and the type of information to be stored. The following is an example of this function:

```
SELECT ADD_LOCATION ('/newLocation/data/',
'v_test_node0003', 'TEMP');
```

Measuring location performance

We can use `MEASURE_LOCATION_PERFORMANCE()` to measure the performance of a storage location. This function has the following prerequisites:

- The storage path must be configured for the database.
- Free physical memory greater than or equal to double the RAM on the node. For example, if we have 4 GB of RAM, then the function requires 8 GB of free disk space.

Use the system table `DISK_STORAGE` to obtain information about disk storage on each database node. The following example measures the performance of a storage location on `node2`:

```
=>select
measure_location_performance('/ilabs/data/vertica/data/test/v_test_node00

WARNING 3914:  measure_location_performance can
take a long time to execute. Please check logs
for progress

      MEasure_location_performance
-----
Throughput : 38 MB/sec. Latency : 51 seeks/sec
(1 row)
```

The number generated represents the time taken by Vertica to read and write 1 MB of data from the disk, which equates to the following:


```
IO time = time to read/write 1MB + time to seek =  
1/throughput + 1/Latency
```

In the preceding command, the following is observed:

- Throughput is the average throughput of sequential reads/writes (units in MB per second)
- Latency is for random reads only in seeks (units in seeks per second)

Setting location performance

Additionally, we can set the performance of a location. Setting location performance can really boost the overall database performance as Vertica selectively stores sorted columns (in order) to faster locations of a projection and the rest of the columns to a slower location. To set the performance for a location, a superuser can use the `SET_LOCATION_PERFORMANCE()` function, as shown:

```
=> SELECT  
SET_LOCATION_PERFORMANCE(v_test_node0001, '/data/vertica/data/test/v_test,
```

In the preceding command, the following is observed:

- '38' is the throughput in MB/second
- '51' is the latency in seeks/second

Understanding storage location tweaking functions

In this section, we will understand how to tweak storage locations with the help of some functions.

Altering

We can use the `ALTER_LOCATION_USE()` function to modify existing storage locations. This following example alters the storage location on `v_test_node0003` to store data only:

```
=> SELECT ALTER_LOCATION_USE  
('/newLocation/data/', 'v_test_node0003', 'DATA');
```

Dropping

We can use the `DROP_LOCATION()` function to drop a location. The following example drops a storage location on `v_test_node0003` that was used to store temp files:

```
=> SELECT DROP_LOCATION('/newLocation/data/' ,  
    'v_test_node0003');
```

The existing data will be merged out either manually or automatically.

Retiring storage locations

To retire a storage location, use the `RETIRE_LOCATION()` function. The following example retires a storage location on `v_test_node0003`:

```
=> SELECT RETIRE_LOCATION('/newLocation/data/' ,  
    'v_test_node0003');
```

Retiring is different from dropping as in the former case Vertica ceases to store data or temp files to it. Before retiring a location, we must make sure that at least one other location on the node exists to store data and temp files.

Restoring retired storage locations

To restore an already retired location, we can use the `RESTORE_LOCATION()` function. The following example restores a retired storage location on `v_test_node0003`:

```
=> SELECT RESTORE_LOCATION('/newLocation/data/' ,  
    'v_test_node0003');
```

Summary

In Vertica, projections are the single most important topic that can help in improving performance of a Vertica deployment. As mentioned earlier, it is best to create projections using Database Designer. In the last and final chapter, we will discuss bulk loading of data in Vertica.

Chapter 6. Bulk Loading

Bulk loading is the process of inserting a huge amount of data at once. Bulk loading in Vertica is performed using the `COPY` command. This chapter will cover topics such as the use of the `COPY` command, different load methods, and the basics of data transformation.

Using the `COPY` command

The `COPY` command can only be used by a superuser. The `COPY` command provides the flexibility to load and manage data with the help of the following optional parameters:

- Format and arrangement of the incoming data
- Metadata about the data load
- Data transformation
- Error handling

The encoding of the data to be loaded should be in the `UTF-8` format. It is advisable to check the encoding of the file before loading the data. If the data present is not in the `UTF-8` format, then we can convert it using the following Linux/UNIX `iconv` command:

```
iconv -f encoding-of-old-file -t  
encoding-of-new-file old-file.txt > newfile.txt
```

This can be illustrated with the help of the following example:

```
> iconv -f WINDOWS-1251 -t UTF-8 data.txt >  
data_new.txt
```

You can also check for the various formats supported by `iconv` using `iconv -l`.

It should also be noted that data should be segregated with proper delimiter characters. Before loading the data, it should also be checked that no `CHAR(N)` or `VARCHAR(N)` data values are included in the delimiter character. The default delimiter character is the pipe character, or `|`.

The following is an example of the `COPY` command with all the possible options:

```

COPY [TARGET_TABLE]
FROM { STDIN
..... [ BZIP | GZIP | UNCOMPRESSED ]
...| 'pathToData' [ ON nodename | ON ANY NODE ]
..... [ BZIP | GZIP | UNCOMPRESSED ] [, ...]
...| LOCAL STDIN | 'pathToData'
..... [ BZIP | GZIP | UNCOMPRESSED ] [, ...]
}

```

The following is a simple example for loading the data:

```

COPY table1 FROM '/root/data/tab1.txt';

```

Here, `table1` is the target table while `'/root/data/tab1.txt'` is the source data.

To load data from the client to the Vertica database cluster, we should use `COPY...FROM LOCAL`, as shown in the following example:

```

COPY table1 FROM LOCAL '/root/data/tab1.txt'
DELIMITER '~';

```

We can provide more than one delimiter. For example, let's say we have data in the following fashion, `a|b|c|d~e|f`, with `|` and `~` being delimiters, as shown in the following example:

```

COPY table1 COLUMN OPTION (col4 DELIMITER '~')
FROM '/root/data/tab1.txt' DELIMITER '|'

```

We can also provide multiple files by giving a comma-separated list as follows:

```

COPY table1 FROM LOCAL '/root/data/tab1_1.txt',
'/root/data/tab1_2.txt'

```

We can supply archives (GZIP and BZIP) containing files as follows:

```

COPY table1 FROM LOCAL '/root/data/tab1' GZIP

```

We can supply data files from any node or a specific node by using `pathToData`. This is an optional parameter, and if not supplied, it finds files in the local node from which the command is invoked, for example:

```
COPY table1 FROM LOCAL '/root/data/tab1' GZIP ON  
ANY NODE
```

In the preceding example, `ON ANY NODE` is the `pathToData` exception.

Aborting the COPY command

If at any point in time you feel that something is wrong with the data or loading process, then you can just cancel the bulk load process. All the changes made during this process will be rolled back to its original state. Remember, it is not advisable to abort a bulk loading process, but if the situation warrants it, then go ahead.

Load methods

Depending on the size of the data, you should select one of the following load methods with the `COPY` command:

Load methods	Description and use
AUTO	This is the default option. It loads data into WOS. After WOS is full, it continues loading data into ROS. It is good for data less than 100 MB in size. (Please refer to Chapter 5, Performance Improvement , to understand more on ROS and WOS.)
DIRECT	This loads data directly into ROS containers. It is advised to use the <code>DIRECT</code> load method for data more than 100 MB in size.
TRICKLE	This loads data only into WOS. After WOS is full, it generates an error. It is suggested to use this for frequent incremental load operations.

An example of a load method is as follows:

```
COPY table1 from '/root/data/tab1.txt' DIRECT
```

For incremental loads, it is suggested to use the `NO COMMIT` command and use the `COMMIT` command at a later stage for better control at the transaction level.

Data transformation

Using the `COPY` command, we can control the columns in which the values need to be inserted for a table. Moreover, the `COPY` command supports operators, constants, Nulls, and comments. It should be noted that the `COPY` command cannot use the analytic and aggregate functions while loading the data, although it supports the following types of functions:

- Date/time
- Formatting
- Numeric
- String
- Null handling
- System information

You can also ignore columns from source files. For that, we need to use the `FILLER` option.

Summary

Since Vertica is more apt for OLAP purposes, it is imperative for you to perform bulk loading. As you must have observed, bulk loading in Vertica is quite simple and follows the same standards as followed by other relational databases.

Index

A

- --archive parameter / [Restoring from the most recent snapshot](#)
- ActivePartitionCount parameter / [Tuning Tuple Mover](#)
- ACTIVE_EVENTS system table
 - about / [Looking at events through the ACTIVE_EVENTS system table](#)
 - node_name / [Looking at events through the ACTIVE_EVENTS system table](#)
 - event_code / [Looking at events through the ACTIVE_EVENTS system table](#)
 - event_posted_timestamp / [Looking at events through the ACTIVE_EVENTS system table](#)
 - event_expiration / [Looking at events through the ACTIVE_EVENTS system table](#)
 - event_code_description / [Looking at events through the ACTIVE_EVENTS system table](#)
 - event_problem_description / [Looking at events through the ACTIVE_EVENTS system table](#)
 - reporting_node / [Looking at events through the ACTIVE_EVENTS system table](#)
- ADD_LOCATION() function
 - used, for adding new location / [Adding a new location](#)
- administration tools
 - used, to replace nodes / [Using administration tools to replace nodes with different names and IP addresses](#)
- administrative tools
 - used, for adding nodes / [Adding nodes using administration tools](#)
 - used, for removing nodes / [Removing nodes using administration tools](#)
- ALTER_LOCATION_USE() function
 - about / [Altering](#)
- AUTO load method / [Load methods](#)

B

- backupDir* parameter / [Mapping](#)
- backupHost* parameter / [Mapping](#)
- backup hosts
 - requisites / [Requirements for backup hosts](#)
- backups
 - requisites / [Understanding the requirements](#)
- base query
 - about / [The base query](#)
- Basic Input/Output System (BIOS) / [Dynamic CPU frequency scaling](#)
- buddy projections
 - about / [Comprehending segmented projections](#)
 - creating / [Creating buddy projections](#)
- bwlimit parameter / [Data transmission during the backup process](#)

C

- --config-file parameter / [Understanding the requirements](#)
- checksum parameter / [Data transmission during the backup process](#)
- cluster
 - hosts, removing from / [Removing hosts from a cluster](#)
- column encoding
 - about / [Column list and encoding](#)
- column list
 - about / [Column list and encoding](#)
- COMMIT command / [Load methods](#)
- comprehensive design
 - Optimize with queries option / [The comprehensive design](#)
 - Update statistics option / [The comprehensive design](#)
 - Deploy design option / [The comprehensive design](#)
- configuration files

- redistributing, to nodes / [Redistributing configuration files to nodes](#)
- copycluster command / [Copying a database from one cluster to another](#)
- COPY command
 - using / [Using the COPY command](#)
 - aborting / [Aborting the COPY command](#)
- CPU throttling / [Dynamic CPU frequency scaling](#)
- Current Fault Tolerance at Critical Level / [Looking at events](#)

D

- database
 - copying, from one cluster to other / [Copying a database from one cluster to another](#)
 - copying / [Copying the database](#)
- database access
 - dbName parameter / [Database access settings](#)
 - dbUser* parameter / [Database access settings](#)
 - dbPromptForPassword* parameter / [Database access settings](#)
 - dbPassword* parameter / [Database access settings](#)
- database access settings / [Database access settings](#)
- Database Designer
 - used, for creating projections / [Creating projections using Database Designer](#)
 - design type, selecting / [Creating projections using Database Designer](#), [The comprehensive design](#), [The query-specific design](#)
- database snapshot functions
 - using / [Using database snapshot functions](#)
- database snapshots
 - creating / [Creating database snapshots](#)
- Database_snapshot function / [Creating database snapshots](#)
- Data Collector
 - enabling / [Enabling and disabling Data Collector](#)
 - disabling / [Enabling and disabling Data Collector](#)

- components, monitoring / [Monitoring data collection components](#)
- Data Definition Language (DDL) / [Monitoring through the system tables](#)
- Data Manipulation Language (DML) / [Monitoring through the system tables](#)
- data retention policy
 - viewing / [Viewing the current data retention policy](#)
 - configuring / [Configuring data retention policies](#)
- data transformation
 - about / [Data transformation](#)
- data transmission
 - encrypt parameter / [Data transmission during the backup process](#)
 - checksum parameter / [Data transmission during the backup process](#)
 - port_rsync parameter / [Data transmission during the backup process](#)
 - bwlimit parameter / [Data transmission during the backup process](#)
 - hardLinkedLocal parameter / [Data transmission during the backup process](#)
- dbName parameter / [Database access settings](#)
- dbNode parameter / [Mapping](#)
- dbPassword* parameter / [Database access settings](#)
- dbPromptForPassword* parameter / [Database access settings](#)
- dbUser* parameter / [Database access settings](#)
- design types
 - comprehensive design / [The comprehensive design](#)
 - query-specific design / [The query-specific design](#)
- DIRECT load method / [Load methods](#)
- DISABLE_LOCAL_SEGMENTATION function / [Enabling and disabling local segmentation](#)
- DROP_LOCATION() function
 - about / [Dropping](#)
- Durable snapshots / [Using database snapshot functions](#)
- Dynamic CPU frequency scaling / [Dynamic CPU frequency scaling](#)

E

- elastic cluster
 - enabling / [Enabling and disabling an elastic cluster](#)
 - disabling / [Enabling and disabling an elastic cluster](#)
- elastic cluster rebalancing
 - monitoring / [Monitoring elastic cluster rebalancing](#)
- ENABLE_LOCAL_SEGMENTS function / [Enabling and disabling local segmentation](#)
- encrypt parameter / [Data transmission during the backup process](#)
- End-user License Agreement (EULA) / [Steps to install Vertica](#)
- events
 - about / [Looking at events](#)
 - list / [Looking at events](#)
 - Low Disk Space / [Looking at events](#)
 - Read Only File System / [Looking at events](#)
 - Loss Of K Safety / [Looking at events](#)
 - Current Fault Tolerance at Critical Level / [Looking at events](#)
 - Too Many ROS Containers / [Looking at events](#)
 - Node State Change / [Looking at events](#)
 - Recovery Failure / [Looking at events](#)
 - Recovery Error / [Looking at events](#)
 - Recovery Lock Error / [Looking at events](#)
 - Recovery Projection Retrieval Error / [Looking at events](#)
 - Refresh Error / [Looking at events](#)
 - Refresh Lock Error / [Looking at events](#)
 - Tuple Mover Error / [Looking at events](#)
 - Timer Service Task Error / [Looking at events](#)
 - Stale Checkpoint / [Looking at events](#)
 - through logfiles / [Looking at events through logfiles](#)
 - through ACTIVE_EVENTS system table / [Looking at events through the ACTIVE_EVENTS system table](#)
- event_code / [Looking at events through the ACTIVE_EVENTS system table](#)
- event_code_description / [Looking at events through the ACTIVE_EVENTS system table](#)
- event_expiration / [Looking at events through the ACTIVE_EVENTS system table](#)
- event_posted_timestamp / [Looking at events through the ACTIVE_EVENTS system table](#)
- event_problem_description / [Looking at events through the ACTIVE_EVENTS system table](#)

F

- failed node
 - replacing, different name used / [Replacing a failed node using a different name and IP address](#)
 - replacing, IP address used / [Replacing a failed node using a different name and IP address](#)
- full database snapshots
 - restoring / [Restoring full database snapshots](#)
 - specific snapshot, restoring from / [Restoring from a specific snapshot](#)
 - most recent snapshot, restoring from / [Restoring from the most recent snapshot](#)
 - schema, restoring / [Restoring schema and table snapshots](#)
 - table snapshots, restoring / [Restoring schema and table snapshots](#)

G

- GET_DATA_COLLECTOR_POLICY() function / [Viewing the current data retention policy](#)

H

- hardLinkedLocal parameter / [Data transmission during the backup process](#)
- hosts
 - removing, from cluster / [Removing hosts from a cluster](#)

I

- iconv command / [Using the COPY command](#)
- increment backup
 - working / [Incremental snapshots](#)
- installation
 - Vertica / [Steps to install Vertica](#)
- IP addresses

- changing, of Vertica cluster / [Changing the IP addresses of a Vertica cluster](#)

K

- K-safety
 - about / [Keeping K-safety \(K-Safe\) in mind](#)
 - buddy projections, creating / [Creating buddy projections](#)
 - table partitioning / [A note on table partitioning](#)
- K-safety level
 - lowering / [Lowering the K-safety level](#)

L

- load methods
 - AUTO load method / [Load methods](#)
 - DIRECT load method / [Load methods](#)
 - TRICKLE load method / [Load methods](#)
- local segmentation
 - enabling / [Enabling and disabling local segmentation](#)
 - disabling / [Enabling and disabling local segmentation](#)
 - best practices / [Understanding the best practices in cluster management](#)
- location performance
 - measuring / [Measuring location performance](#)
 - setting / [Setting location performance](#)
- logfiles
 - events, looking at / [Looking at events through logfiles](#)
- Loss Of K Safety / [Looking at events](#)
- Low Disk Space / [Looking at events](#)

M

- management console

- used, for adding nodes / [Using the Management Console to add nodes](#)
- used, for removing nodes / [Removing nodes using the Management Console](#)
- mapping / [Mapping](#)
 - backupHost* parameter / [Mapping](#)
 - backupDir* parameter / [Mapping](#)
 - dbNode parameter / [Mapping](#)
- MARK_DESIGN_KSAFE function / [Lowering the K-safety level](#)
- MAXIMUM_SKEW_PERCENT parameter / [Comprehending the elastic cluster scaling factor](#)
- MEASURE_LOCATION_PERFORMANCE() function
 - used, for location performance measuring / [Measuring location performance](#)
- MergeOutInterval parameter / [Tuning Tuple Mover](#)
- Mergeout operation
 - about / [Mergeout](#)
- method / [Method](#)
- miscellaneous
 - snapshotName* parameter / [Miscellaneous settings](#)
 - tempDir parameter / [Miscellaneous settings](#)
 - verticaBinDir parameter / [Miscellaneous settings](#)
 - verticaConfig parameter / [Miscellaneous settings](#)
 - restorePointLimit* parameter / [Miscellaneous settings](#)
 - objects* parameter / [Miscellaneous settings](#)
 - overwrite parameter / [Miscellaneous settings](#)
 - retryCount parameter / [Miscellaneous settings](#)
 - retryDelay parameter / [Miscellaneous settings](#)
- miscellaneous settings / [Miscellaneous settings](#)
- monitoring
 - through system tables / [Monitoring through the system tables](#)
- most recent snapshot
 - restoring from / [Restoring from the most recent snapshot](#)
- MoveOutInterval parameter / [Tuning Tuple Mover](#)
- MoveOutMaxAgeTime parameter / [Tuning Tuple Mover](#)
- Moveout operation

- about / [Moveout](#), [Mergeout](#)
- MoveOutSizePct parameter / [Tuning Tuple Mover](#)
- MVs
 - and indexes versus projections / [Understanding projections](#)

N

- NO COMMIT command / [Load methods](#)
- nodes
 - adding, in Vertica / [Adding nodes in Vertica](#)
 - method / [Method](#)
 - adding, management console used / [Using the Management Console to add nodes](#)
 - adding, administrative tools used / [Adding nodes using administration tools](#)
 - removing, in Vertica / [Removing nodes in Vertica](#)
 - replacing, same name used / [Replacing a node using the same name and IP address](#)
 - replacing, IP address used / [Replacing a node using the same name and IP address](#)
 - configuration files, redistributing to / [Redistributing configuration files to nodes](#)
 - replacing, administration tools used / [Using administration tools to replace nodes with different names and IP addresses](#)
- nodes, removing
 - K-safety level, lowering / [Lowering the K-safety level](#)
 - administrative tools used / [Removing nodes using administration tools](#)
 - management console used / [Removing nodes using the Management Console](#)
 - hosts, removing from cluster / [Removing hosts from a cluster](#)
- Node State Change / [Looking at events](#)
- node_name / [Looking at events through the ACTIVE_EVENTS system table](#)
- Non-durable snapshots / [Using database snapshot functions](#)

O

- objects* parameter / [Miscellaneous settings](#)
- overwrite parameter / [Miscellaneous settings](#)

P

- port_rsync parameter / [Data transmission during the backup process](#)
- preinstallation steps, Vertica
 - swap space / [Swap space](#)
 - Dynamic CPU frequency scaling / [Dynamic CPU frequency scaling](#)
 - disk space, requisites / [Understanding disk space requirements](#)
- projections
 - superprojection / [Understanding projections](#)
 - versus MVs and indexes / [Understanding projections](#)
 - unsegmented projections / [Comprehending unsegmented projections](#)
 - segmented projections / [Comprehending segmented projections](#)
 - creating, Database Designer used / [Creating projections using Database Designer](#)
 - manually creating / [Creating projections manually](#)
 - column list / [Column list and encoding](#)
 - column encoding / [Column list and encoding](#)
 - base query / [The base query](#)
 - sort order / [The sort order](#)
 - segmentation / [Segmentation](#)
 - K-safety / [Keeping K-safety \(K-Safe\) in mind](#), [A note on table partitioning](#)

Q

- query-specific design
 - about / [The query-specific design](#)
 - Update statistics option / [The query-specific design](#)
 - Deploy design option / [The query-specific design](#)

R

- Read Only File System / [Looking at events](#)
- Read Optimized Store (ROS) / [Looking at events](#)
- Recovery Error / [Looking at events](#)
- Recovery Failure / [Looking at events](#)
- Recovery Lock Error / [Looking at events](#)
- Recovery Projection Retrieval Error / [Looking at events](#)
- Refresh Error / [Looking at events](#)
- Refresh Lock Error / [Looking at events](#)

- Remove node button / [Removing nodes using the Management Console](#)
- RemoveSnapshotInterval parameter / [Removing snapshots](#)
- reporting_node / [Looking at events through the ACTIVE_EVENTS system table](#)
- restorePointLimit* parameter / [Miscellaneous settings](#)
- restorePointLimit parameter / [Incremental snapshots](#)
- RESTORE_LOCATION() function
 - about / [Restoring retired storage locations](#)
- RETIRE_LOCATION() function
 - about / [Retiring storage locations](#)
- retryCount parameter / [Miscellaneous settings](#)
- retryDelay parameter / [Miscellaneous settings](#)
- ROS
 - about / [Understanding the storage model in Vertica](#)

S

- scaling factor
 - about / [Comprehending the elastic cluster scaling factor](#)
- scaling factor settings
 - viewing / [Viewing and setting the scaling factor settings](#)
 - setting / [Viewing and setting the scaling factor settings](#)
- schema
 - creating / [Creating schema and table snapshots](#)
 - restoring / [Restoring schema and table snapshots](#)
- segmentation, projections
 - about / [Segmentation](#)
- segmented projections
 - about / [Comprehending segmented projections](#)
- SET_CONFIG_PARAMETER() function / [Enabling and disabling Data Collector](#)
- SET_DATA_COLLECTOR_POLICY() function / [Configuring data retention policies](#)

- SET_LOCATION_PERFORMANCE() function
 - using / [Setting location performance](#)
- SET_SCALING_FACTOR function / [Viewing and setting the scaling factor settings](#)
- snapshotName* parameter / [Miscellaneous settings](#)
- SnapshotRetentionTime parameter / [Removing snapshots](#)
- snapshots
 - Durable snapshots / [Using database snapshot functions](#)
 - Non-durable snapshots / [Using database snapshot functions](#)
 - removing / [Removing snapshots](#)
- sort order
 - about / [The sort order](#)
- specific snapshot
 - restoring from / [Restoring from a specific snapshot](#)
- Stale Checkpoint / [Looking at events](#)
- storage locations
 - adding / [Adding storage locations](#)
 - performance, measuring / [Measuring location performance](#)
 - performance, setting / [Setting location performance](#)
 - tweaking, functions used / [Understanding storage location tweaking functions](#), [Restoring retired storage locations](#)
- storage location tweaking functions
 - ALTER_LOCATION_USE() function / [Altering](#)
 - DROP_LOCATION() function / [Dropping](#)
 - RETIRE_LOCATION() function / [Retiring storage locations](#)
 - RESTORE_LOCATION() function / [Restoring retired storage locations](#)
- storage model
 - about / [Understanding the storage model in Vertica](#)
 - TM operations / [Tuple Mover operations](#), [Mergeout](#), [Tuning Tuple Mover](#)
 - storage locations, adding / [Adding storage locations](#), [Measuring location performance](#), [Understanding storage location tweaking functions](#)
- sudo command / [Steps to install Vertica](#)
- swap space / [Swap space](#)

- system table
 - about / [Monitoring through the system tables](#)
 - schemas / [Monitoring through the system tables](#)
 - example / [Understanding a system table example](#)

T

- table partitioning
 - about / [A note on table partitioning](#)
- table snapshots
 - creating / [Creating schema and table snapshots](#)
 - restoring / [Restoring schema and table snapshots](#)
- tempDir parameter / [Miscellaneous settings](#)
- Timer Service Task Error / [Looking at events](#)
- TM
 - about / [Understanding the storage model in Vertica](#)
 - parameters, tuning / [Tuning Tuple Mover](#), [Adding storage locations](#)
- TM operations
 - Moveout / [Moveout](#), [Mergeout](#)
 - Mergeout / [Mergeout](#)
- TM parameters
 - ActivePartitionCount / [Tuning Tuple Mover](#)
 - MergeOutInterval / [Tuning Tuple Mover](#)
 - MoveOutInterval / [Tuning Tuple Mover](#)
 - MoveOutMaxAgeTime / [Tuning Tuple Mover](#)
 - MoveOutSizePct / [Tuning Tuple Mover](#)
- Too Many ROS Containers / [Looking at events](#)
- TRICKLE load method / [Load methods](#)
- Tuple Mover Error / [Looking at events](#)

U

- unsegmented projections
 - about / [Comprehending unsegmented projections](#)

V

- vbr.py configuration file
 - generating / [Generating the vbr.py configuration file](#)
 - miscellaneous settings / [Miscellaneous settings](#)
 - database access settings / [Database access settings](#)
 - data transmission / [Data transmission during the backup process](#)
 - mapping / [Mapping](#)
- vbr.py utility
 - running / [Running vbr.py](#)
- vby.py command / [Restoring from the most recent snapshot](#)
- Vertica
 - preinstallation, steps / [Understanding the preinstallation steps](#)
 - installing / [Steps to install Vertica](#)
 - nodes, adding / [Adding nodes in Vertica](#)
 - nodes, removing / [Removing nodes in Vertica](#)
- verticaBinDir parameter / [Miscellaneous settings](#)
- Vertica cluster
 - IP addresses, changing / [Changing the IP addresses of a Vertica cluster](#)
- verticaConfig parameter / [Miscellaneous settings](#)
- Vertica Management Console
 - about / [Monitoring Vertica through the Management Console](#)
- V_CATALOG schema / [Monitoring through the system tables](#)
- V_MONITOR schema / [Monitoring through the system tables](#)

W

- WOS
 - about / [Understanding the storage model in Vertica](#)
- Write Optimized Store (WOS) / [Looking at events](#)