Red Hat Reference Architecture Series

Deploying a Vertica®/Jaspersoft® based Business Intelligence Solution on Red Hat® Enterprise Linux® 5

JasperSoft® Business Intelligence
Vertica® Analytic Database
Red Hat® Enterprise Linux® 5
X86-64 Servers

Version 1.0

November 2008

Deploying a Vertica™ / Jaspersoft®
based Business Intelligence Solution
on Red Hat® Enterprise Linux® 5

1801 Varsity Drive
Raleigh NC 27606-2072 USA
Phone: +1 919 754 3700
Phone: 888 733 4281
Fax: +1 919 754 3701
PO Box 13588
Research Triangle Park NC 27709 USA

Table of Contents

1. Introduction

The Vertica Analytic Database is the only database built from scratch to handle today's heavy business intelligence workloads. In customer benchmarks, Vertica has been shown to manage terabytes of data running on extraordinarily less expensive hardware and answers queries 50 to 200 times faster than competing row-oriented databases and specialized analytic hardware

This document summarizes the key aspects of Vertica's technology that allows it to provide such dramatic performance benefits, and compares the design of Vertica to other popular relational systems.

The key to Vertica's performance is three fold:

1. It organizes data on disk as columns of values from the same attribute, as opposed to storing it as rows of tabular records. This organization means that when a query needs to access only a few columns of a particular table, only those columns need to be read from disk. Conversely, in a row-oriented database, all values in a table are typically read from disk, which wastes I/O bandwidth.
2. Vertica employs **aggressive compression** of data on disk, as well as a query execution engine that is able to keep data compressed while it is operated on. Compression in Vertica is particularly effective, as values within a column tend to be quite similar to each other and compress very well—often by up to 90%. In a traditional row-oriented database, values within a row of a table are not likely to be very similar, and hence are unlikely to compress well. Direct operation on data combined with Vertica's compression algorithms – which are especially designed to impose negligible CPU overhead -- means that compression not only reduces the amount of data read from disk but actually decreases the amount of work the CPU has to do to process a query.
3. Because data is compressed, Vertica has sufficient storage to store each table on disk in **multiple sort orders**. Tables are decomposed into overlapping groups of columns, called projections, and each projection is sorted on a different attribute (or set of attributes). Each projection is optimized for answering queries with predicates on its sort attributes.

Of course, compression, column-orientation, and table decomposition are transparent to the end-user. In fact, Vertica provides a standard SQL interface to users, as well as compatibility with existing ETL, reporting, and business intelligence (BI) tools. This makes it easy to migrate existing databases to Vertica and use other BI technologies with Vertica databases. Vertica is designed to run on inexpensive clusters or "grids" of off-the-shelf Linux servers—no expensive SANs high-end servers are required to run large data warehouses on Vertica. Vertica both reduces hardware costs (often by up to 90% relative to other data warehouse databases) and improves the ability to answer more queries for more people against more data.

Besides remarkable performance on a variety of database workloads, Vertica includes several other features designed to offer performance, scalability, reliability, and ease of use. These include:

1. A **shared nothing, grid-based parallel database architecture** that allows Vertica to scale effectively on clusters of commodity CPUs.
2. A **hybrid data store**, where newly inserted records are added to a write-optimized portion of the database to allow continuous, high-performance insert operations concurrently with querying to enable real-time analytics.
3. **Automated physical database design tools** that recommends how data should be organized both locally on each node in a cluster, as well as partitioned across a cluster. In addition to choosing projections and sort orders, these tools ensure **k-safety**, meaning that each table is replicated on k nodes so that k-1 node failures can be tolerated by the system without interrupting functionality. These tools reduce administrative costs by simplifying physical database design decisions. They also allow Vertica to automatically adapt to on-the-fly the addition or removal of database nodes.
4. **High-performance, ACID-compliant replication-based concurrency control, recovery, and high-availability** algorithms that exploit k-safety rather than using traditional log-based methods.
5. **Flexible deployment options –**
    a. Downloaded and installed on Linux servers of your choice
    b. Pre-configured and shipped on HP or Sun hardware
    c. Licensed and used on an on-demand basis, hosted in the Amazon Elastic Compute Cloud (EC2).
6. A collection of backup and disaster recovery tools.

In the remainder of this white paper, we describe the basic architecture of the Vertica Analytic Database, and describe these key features of Vertica in more detail. With these features in mind, we then describe how it is that Vertica can provide such remarkable performance compared to other relational systems.

1.1 Vertica Analytic Database Architecture and System Overview

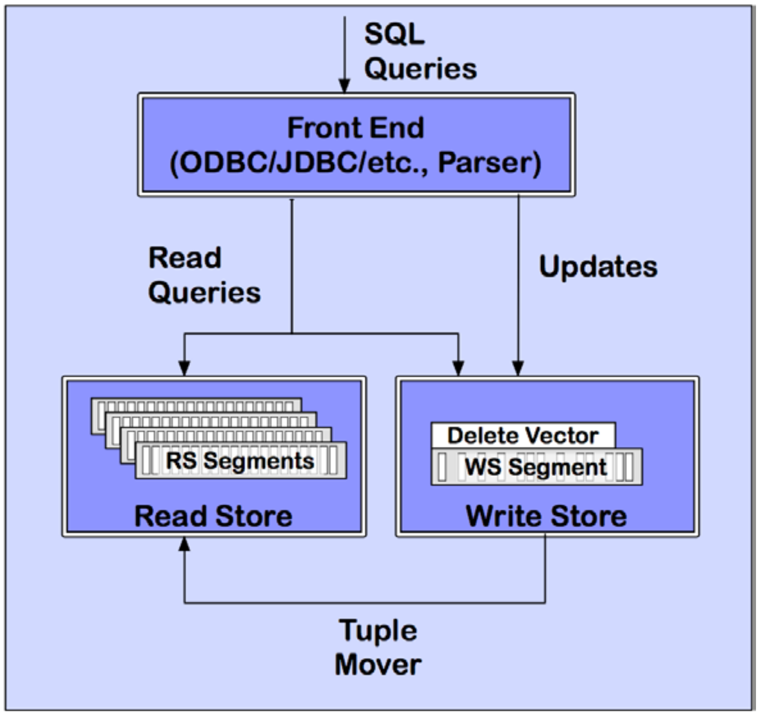
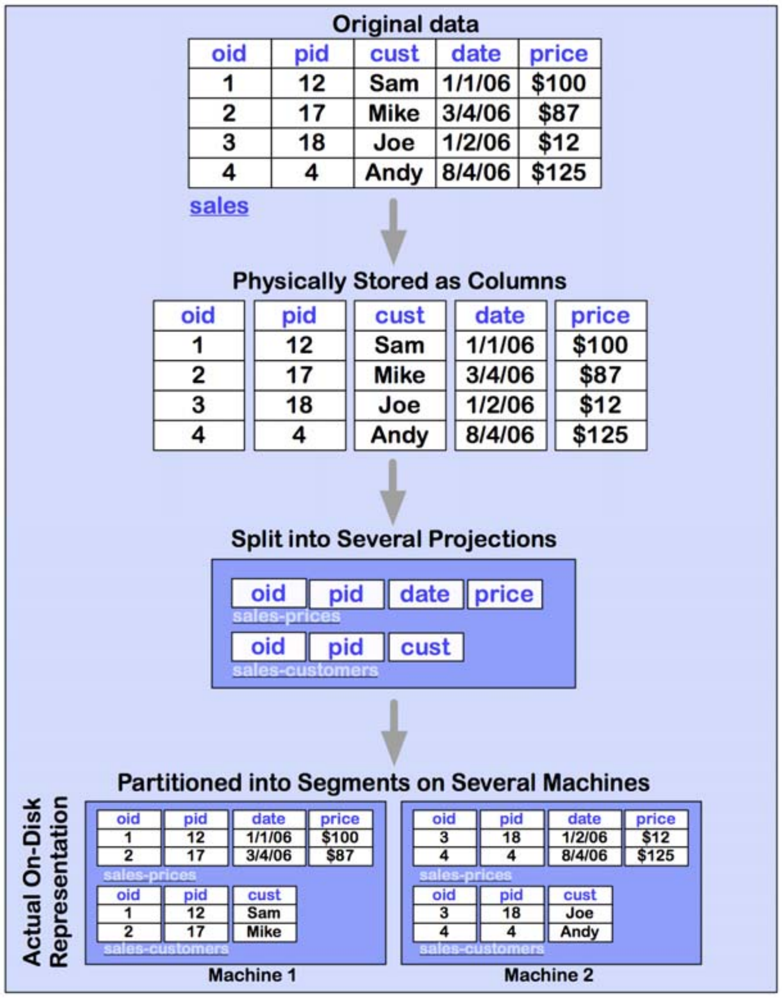


***Figure 1:* The Vertica Analytic Database Architecture**

Figure 1 illustrates the basic system architecture of Vertica on a single node. As in a traditional relational database, queries are issued in SQL to a front end that parses and optimizes queries. Unlike a traditional relational database, however, Vertica is internally organized into a **hybrid store** consisting of two distinct storage structures. The *Write-Optimized Store* (WOS) is a data structure that generally fits into main memory and is designed to efficiently support insert and update operations, but is relatively slow to query because it is unsorted and uncompressed. Conversely, the *Read-Optimized Store* (ROS) contains the bulk of the data in the database, and is both sorted and compressed, making it efficient to read and query. Periodically, a **Tuple Mover** process moves data out of the WOS and into the ROS in large blocks. Because it operates on the entire WOS, the tuple mover can be very efficient, sorting many records at a time and writing them to disk as a batch.

Internally, both the WOS and ROS are organized into columns, with each column representing one attribute of a table. For example, for a table of *sales* data with the schema (order-id, product-id, sales-date, customer-id, sales-price), Vertica would store this as at least five distinct columns (though they would logically appear as one table to the user). By storing these columns separately, Vertica is able to access only those columns that are needed to answer a particular query. Figure 2 illustrates how logical data in an example *sales* table is physically stored as columns. It also shows data being distributed across several projections and servers, which further improves performance, as we describe below.


Original data
sales
Physically Stored as Columns
Split into Several Projections
sales-prices
sales-customers
Partitioned into Segments on Several Machines
Actual On-Disk Representation
Machine 1
Machine 2

oid	pid	cust	date	price
1	12	Sam	1/1/06	$100
2	17	Mike	3/4/06	$87
3	18	Joe	1/2/06	$12
4	4	Andy	8/4/06	$125

Figure 2: **Logical tables are physically stored as columns, and partitioned into segments on several machines and in several different projections.**

In Vertica, each column may be stored in a number of **Projections** that represent partially redundant copies of the data in the database. For example, as shown in Figure 2, our sales table might be stored as two projections, one called *sales-prices with* the schema (order-id, product-id, sales-date, sale-price), and one called *sales-customers* with the schema (order-id, product-id, customer-id). Each of these projections also has a sort order that specifies how the data in the projection is arranged on disk. For example, the *sales-customers* projection might be stored sorted on customer id; this makes it particularly efficient for finding all of the products that a customer bought. By storing several overlapping projections of a table in different sort orders, Vertica can be efficient at answering many different types of queries. Vertica's **Database Designer** automatically selects a good set of overlapping projections for a particular table based on the set of queries issued to that table over time.

It may seem that redundantly storing data in multiple projections is very wasteful of disk space. However, Vertica includes a suite of aggressive **column-oriented compression schemes** that allow it to reduce the amount of space a particular projection takes up in the ROS by as much as 90%. This also improves query performance by reducing the amount of data that must be read off disk . These compression schemes, described in more detail below, have low CPU overhead and allow direct operation by the database on the compressed data. This means that Vertica can often process queries over compressed data substantially faster than over uncompressed data.

Beyond projections, there are several additional optimizations employed by Vertica that involve storing each individual column in Vertica across several physical objects. First, on a single machine, the ROS is horizontally partitioned into several *Storage Containers* by time. This allows the Tuple Mover to insert new data into the ROS without having to merge together and rewrite all old containers. To keep the total number of containers small, from time to time, the tuple mover merges these ROS containers together in the background. This process is illustrated in Figure 3.

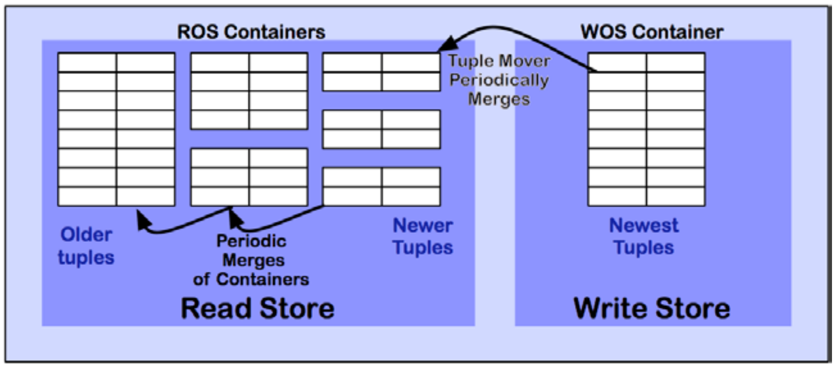


***Figure 3:* A hybrid storage architecture features a Tuple mover process that asynchronously moves batches of new data from a memory-based WOS to disk-based ROS containers**

Second, each projection is horizontally partitioned into *segments,* each of which is stored on a different machine in the cluster based on the value of one attribute in the table. This makes it possible to parallelize queries, allowing many applications to scale linearly with the number of nodes in the database. One possible partitioning across machines is shown for our example *sales* database at the bottom of Figure 2.

Now that we have covered the basic architecture of the Vertica database system, we look at the key features of the system that allow it provide **outstanding performance** as well as **simplify administration and reduce total cost of ownership**.

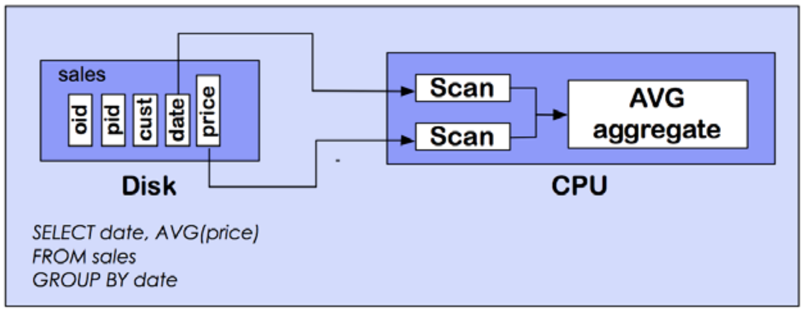
1.2 Performance Features

Vertica's performance on read-intensive workloads is due to a number of factors. To help understand its performance properties, let's look at the execution of a simple query over the *sales* table described above. Suppose we wish to compute the average sales price grouped by date in our *sales* database. The steps involved in processing this query are shown in Figure 4. There are two primary costs in processing a typical query: disk access and CPU cycles (in a distributed database, network I/O may also be significant). On most modern computers, it is possible to perform disk I/O in parallel with CPU operations. Thus, if answering a query takes D seconds of disk time and C seconds of CPU time, the total query time will be the maximum of C and D.

1.2.1 Column-orientation

Since many database queries are "disk bound," meaning D > C, the most obvious performance advantage of Vertica is that it only needs to read the two columns involved in the query from disk. In a row-store database, the disk scans all five columns. This extra three columns of scanning can represent as much as a 250% slowdown in performance. Since most databases have many more than five columns (hundreds, even), the performance of a row-store is often even worse.

Other database vendors include support for materialized views and data cubes, which are data structures designed to optimize the performance of a few common queries. They work by reducing the need to scan over unused columns by partially pre-computing query answers into data structures stored on disk. These approaches are optimized for answering a few queries well and are unable to provide comparable performance to Vertica across many different query workloads. Besides Vertica's column-orientation, one of the major ways in which Vertica achieves this performance advantage is via aggressive columnar compression, which we describe next.



```
SELECT date, AVG(price)
FROM sales
GROUP BY date
```

Figure 4: **A query plan for a simple query in Vertica**

1.2.2 Aggressive Compression

In our simple model of database performance, compression helps to dramatically reduce both C and D. As an example, suppose the *sales* table accessed by the plan in Figure 4 is sorted by date, and then within each date by price. If a database contains 1 year of data, and each order has one of about 1,000 different prices on a given day, and there are 10 million total records in the database, there will on average be about 27 different orders from each of the 1,000 prices during each day. This means that, rather than representing the price column as a list of the form (price1, price2, price3, …), we can represent it as a *run-length encoded (RLE)* list of the form (<count, price1>, <count, price2>, ….), which will be about 27 times smaller than the initial representation. We can also compress date in a similar way, storing just 365 different <count, date> pairs to store the entire column of 10 million dates! Compressing data in this way greatly reduces the disk I/O (D). However, we also reduce the amount of CPU work, C, because database operators (like *scan* and *aggregate* in Figure 2) in Vertica are specially optimized to work directly with compressed data. For example, the *AVG* aggregate does not need to convert the RLE compressed list of prices back into uncompressed form, but can instead compute averages directly from the compressed records, reducing C by about a factor of 27.

Vertica includes many other compression methods, including delta encoding of successive values and an efficient Lempel Ziv implementation, that are well-suited to use with sorted columns with more distinct values as well as with unsorted columns. Most of these methods allow direct operation on the data within the query processor. Other database systems – even those that have a column-oriented architecture – cannot operate on compressed data in this way.

1.2.3 Read-Optimized Storage

Recall that the bulk of the data in Vertica is stored in the ROS, which is optimized for efficient read operations. Keeping the ROS sorted and compressed is one way in which its structure is optimized. However, Vertica includes a number of other optimizations. For example, data in the ROS is *dense packed*, meaning that no empty spaces are left at the ends of disk pages; traditional databases leave a substantial portion of each page empty to allow insert operations to be performed without reorganizing all data on the disk. Vertica does not need to do this because inserts to the ROS are only done in bulk via the tuple mover. As another optimization, Vertica pre-fetches large blocks during most reads to avoid unnecessary seeks when large portions of several columns are being scanned.

1.2.4 Ability to exploit multiple sort orders

As described above, Vertica's approach to high availability and recovery involves the use of replicas which store the same data in different sort orders. Not only does Vertica's recovery scheme avoid expensive logging operations, these additional sort orders can be used to further improve query performance. Furthermore, Vertica's aggressive compression schemes mean that different projections can be created on a single node that store the overlapping sets of columns in different sort orders. Vertica can then select the best sort order for processing a given query.

1.2.5 Parallel shared-nothing design on off-the-shelf hardware

Vertica's performance scales linearly to a large number of CPUs. Because Vertica does not depend on custom hardware, it will benefit over time from the dramatic performance improvements that commodity hardware has traditionally enjoyed. It is much harder for custom hardware solutions to reap these same benefits. Furthermore, because it is designed from the ground up to run on off-the-shelf, Linux-based (including kernel-based I/O and file systems) servers, it will maintain compatibility and scalability over time.

1.2.6 Bottom Line

In summary, many vendors offer non-column store systems. For read-mostly workloads, these designs typically offer about 1/100th the performance of Vertica when using the same amount of disk space, or 1/10th the performance of Vertica in 10 times the disk space (for example, when they have many materialized views). A few column-oriented systems do exist, but these first-generation designs lack aggressive compression features, a hybrid WOS/ROS design, or Vertica's sophisticated recovery and replication strategies. Typically, Vertica offers 10 times the performance of these column-based systems in about ¾ the disk space.

The table below illustrates the performance wins in a typical data warehouse application. The study comes from a Vertica customer in the financial services industry, and it compares Vertica to a popular row-oriented data warehouse DBMS. Here, queries against 1.5 terabytes of stock market trade and quote history are 270X faster when Vertica queries the data. Vertica allows them to answer 33x more queries per day, and data is available in real-time rather than being a day late. All of this can be achieved on hardware costing over $1M (USD) less. Similar benefits can be reviewed for other industries and applications by visiting www.vertica.com/benchmarks.

	Row-Oriented Data Warehouse DBMS	Vertica® Analytic Database	Vertica Advantage
Avg Query Response Time	37 minutes	9 seconds	270x faster answers
Reports per Day	30	1000	33x more reports
New Market Data Availability	Next day	1 minute	Real-time views
Hardware Cost	$1.4M (2x 6 servers + SAN)	$50,000 (6 HP ProLiant servers)	1/28th of the hardware, built-in redundancy

1.3 Administrative and Management Features

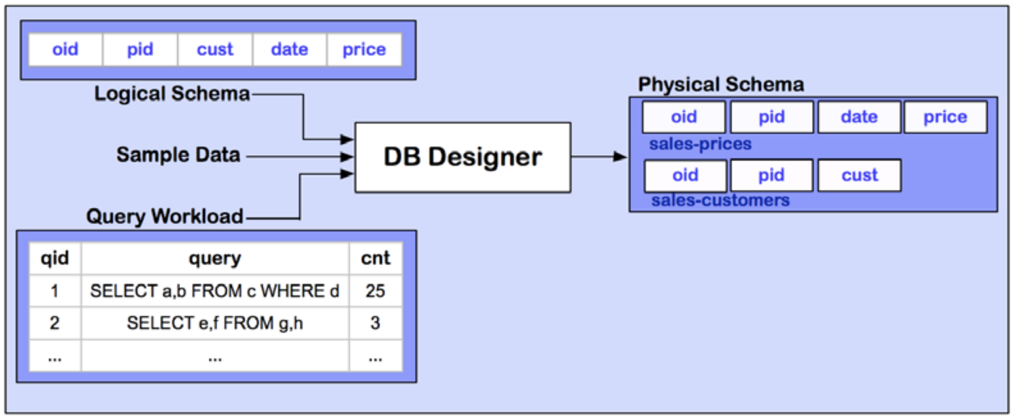
In addition to outstanding performance on a range of applications, Vertica includes a number of features designed to simplify database administration and reduce total cost of database ownership.

1.3.1 Vertica Database Designer

The primary role of the Vertica Database Designer (DB Designer) is to recommend a set of projections (including encoding, sort order and segmentation values) that will provide good performance for the queries that the user most commonly issues over his or her database and cover any ad-hoc queries that users may choose to run. Of course, the database administrator is able to override any of the decisions made by the DB Designer, but in many cases, the designer can do a remarkably good job of optimizing performance, thus reducing the time administrators must spend on physical database tuning.

The basic algorithm the DB Designer uses takes in a deployed logical schema, a set of sample data from those tables, a "training set" of typical queries, and a maximum space budget for each machine in the cluster. It then recommends a set of projections that provide good performance on the training set, subject to satisfying the space constraints and ensuring that any SQL query can be answered (which means all of the columns of the database must be stored in at least one projection) even in the event that *k* server nodes fail (i.e., recommends a k-safe design).

Figure 5 illustrates the basic process: the DB Designer takes in a logical schema and a query and data workload, and produces a physical schema consisting of a set of projections and a partitioning of those projections across multiple machines.



qid	query	cnt
1	SELECT a,b FROM c WHERE d	25
2	SELECT e,f FROM g,h	3
...	...	...

Figure 5: **The DB Designer produces a physical schema consisting of a set of projections and a partitioning of those projections from a logical schema, a query workload, and a collection of sample data.**

1.3.2 Recovery and High Availability through k-Safety

The traditional method to ensure that a database system can recover from a crash is to use logging and (in the case of a distributed databases), a protocol called *two-phase commit*. The main idea is to write in a sequential log a *log record* for each update operation before the operation is actually applied to the tables on the disk. These log records are a redundant copy of the data in the database, and when a crash occurs, they can be replayed to ensure that transactions are *atomic* – that is, all of the updates of a transaction appear to have occurred, or none of them do. The two-phase commit protocol is then used to ensure that all of the nodes in a distributed database agree that a transaction has successfully committed; it requires several additional log records to be written. Log-based recovery is widely used in other commercial systems, as it provides strong recoverability guarantees at the expense of modest (but not insignificant) performance and disk space overhead.

Vertica has a unique approach to recovery that avoids these costs while still providing distributed recoverability. The basic idea is to exploit the distributed nature of a Vertica database. The Vertica DB Designer ensures that every column in every table of the database is stored on at least k+1 machines in the Vertica cluster. We call such a database *k-safe*, because if k machines crash or otherwise fail, a complete copy of the database is still available. As long as fewer than k-1 other machines fail simultaneously, a crashed machine can recover its state by copying data about transactions that committed or aborted while it was crashed from other machines in the system. This approach does not require logging because nodes replicating the data ensure that a recovering machine always has another (correct) copy of the data to compare against, replacing the role of a log in a traditional database. As long as k-safety holds, there is always one machine that knows the correct outcome (commit or abort) of every transaction.

k-safety also means that Vertica is *highly available*: it can tolerate the simultaneous crash of up to any k machines in a grid without interrupting query processing. The value of k can be configured to provide the desired tradeoff between hardware costs and availability guarantees. Note, however, that adding additional machines to increase the k-value of a Vertica database not only improves availability but also improves performance. This is because all that is required for k-safety is that each column is replicated K times, but each copy of a column may be stored in a different projection, with a different sort order. As noted above, different sort orders are good for answering different queries.
It is instructive to contrast Vertica's high-availability schemes with traditional database systems where high availability is achieved through the use of active standbys – essentially completely unused hardware that has an exact copy of the database and is ready to take over in the event of a primary database failure. Unlike Vertica's k-safe design employing different sort orders, active standbys simply add to the cost of the database system without improving performance.

Because Vertica is k-safe, it supports *hot-swapping* of nodes. A node can be removed and the database will continue to process queries (at a lower rate). A node can be added, and the database designed will automatically allocate a collection of objects to that node so that it can begin processing queries, increasing database performance automatically.

1.3.3 Continuous Load: Snapshot Isolation and the WOS

In a traditional database system, adding new data is done in one of two ways. Either, it is inserted a record-at-a-time, which is convenient for users, since they can update the database whenever they need to, but substantially slows the database. This slowness comes from two factors: first, inserts usually require locking operations that block other transactions running in the system and second, because *every* insert requires expensive updates to indices, materialized views, and cubes. As an alternative to record-at-a-time inserts, conventional databases are often loaded via a bulk-load process, where the database system is taken offline for several hours so that new records can be added without acquiring locks and so that indices can be updated in bulk.

Vertica, in contrast, offers a **continuous load** feature that does not require the system to ever be taken offline but still provides excellent insert performance. This is achieved through two techniques. First, inserts do not affect the performance of most queries in the system. This is because read-only operations in Vertica usually run in a **snapshot isolation** mode, where they read a consistent snapshot of the database from just before any concurrent insert or update operation ran. This means that read-only queries do not block for updates or inserts. It also means that read-only operations are not required to acquire locks before reading from the database, further improving performance.

The second way in which Vertica achieves good continuous load performance is through the WOS. Since the WOS is unsorted and un-indexed, individual insert operations are fast. When the tuple mover runs, it updates the (sorted and indexed) ROS very efficiently because it can apply all of the WOS operations in bulk (much as the bulk-load process does in traditional databases, but without taking the system offline.)

1.3.4 Backup/Restore for Disaster Recovery

Finally, Vertica includes a set of administrative tools for performing database archival and restore for use in disaster recovery. These tools support parallel backup of multiple nodes to a single image stored on a remote disk (e.g., a SAN). They also support an incremental backup mode, where new data added to the database is automatically written to an archival image.

1.4 Who Should Use Vertica?

Vertica is a relational SQL database system that is best suited to read-intensive analytic database applications such as data warehouses and data marts. It is optimized for databases with OLAP-style workloads that include a small fraction of update operations (a good rule of thumb is that fewer than 1% of the total SQL statements should be DELETEs or UPDATEs). For database-backed applications that meet these requirements, Vertica offers a substantial performance increase over row-oriented OLTP databases, other column databases and even proprietary data warehouse appliance hardware, while using significantly less disk space and without requiring a large investment in hardware or annual database administration overhead.

1.5 Additional Resources

The Vertica Analytic Database supports SQL and integrates with 3rd-party ETL, analytic and BI reporting tools and applications via JDBC, ODBC and specific language bindings. Therefore, using all your existing SQL knowledge and technology, a Vertica database can be very quickly created and loaded with data.

If you would like to learn more about the Vertica Database or if you would like to evaluate it yourself, then visit the following links:

Gartner on Vertica and EDWs	www.vertica.com/gartner	Watch a recording of Don Feinberg of Gartner explain why supplementing an EDW with column-DB based data marts improves the ROI on EDWs
Vertica Resource Library	www.vertica.com/resourcelibrary	White papers, demos, webcasts, system requirements
Vertica Benchmarks	www.vertica.com/benchmarks	See customer-submitted cost and performance comparisons between Vertica and other databases
Vertica for the Cloud	www.vertica.com/cloud	Get a Vertica database instance provisioned instantly on the Amazon Cloud and use it on a month-to-month basis
Vertica Customers	www.vertica.com/customers	See who's using Vertica
Request a Vertica Evaluation	www.vertica.com/download	Request a free evaluation copy of the Vertica Analytic Database to download and install

1.6 About Vertica Systems

Vertica Systems is the market innovator for high-performance analytic database management systems that run on industry-standard hardware. Co-founded by database pioneer Dr. Michael Stonebraker, Vertica has developed grid-based, column-oriented analytic database technology that lets companies of any size store and query very large databases orders of magnitude faster and more affordably than other solutions. The Vertica Analytic Database's unmatched speed, scalability, flexibility and ease of use helps customers like JP Morgan Chase, Verizon, Mozilla, Comcast, Level 3 Communications and Vonage capitalize on business opportunities in real time. For more information, visit the company's Web site at http://www.vertica.com.

2. Vertica and Red Hat Linux: System Requirements and Configuration

This section contains recommendations and guidelines for installation and configuration of a Linux machine for use with Vertica.

2.1 Representative Commercial Systems

Vertica is optimized for distributed computing environments such as grids and clusters of shared-nothing computers connected via TCP/IP. Vertica takes advantage of multiple processors, cores, and disks.

A production deployment should feature a cluster of at least 3 nodes (to ensure high availability), and each computer within the cluster should be identically equipped and meet the following minimum system requirements:

CPU:	64-bit dual-core or quad-core 1.6GHz Recommended CPUs: (AMD: any Opteron™ 1000, 2000, 8000 series; Intel: any Core 2 or Pentium D, or any Xeon 3000,3200, 5300 or 7000 series)
RAM:	2GB per CPU core
DISK:	1TB of local storage (4x250GB 10K RPM, SATA or SAS)
OS:	Red Hat Enterprise Linux 5

HP ProLiant DL 385

- 2 x AMD Opteron™ Processor 2210
- 8 GB RAM - 533MHz 4x1GB
- 7 x 146GB SAS 10K RPM hard disks
- Red Hat Enterprise Linux 5 (64-bit)

Dell PowerEdge 2950

- 1 x Quad-core Intel Xeon 5310 1.6GHz
- 8GB RAM
- 4 x 250GB SATA II
- Red Hat Enterprise Linux 5 (64-bit)

2.2 Trickle Loads

For high rate trickle loads (> 500MB/min) Vertica recommends a minimum of 4 cores and 16GB of RAM (4GB per CPU core) to allow for larger merge chunks.

2.3 Swap Space Recommendations

Swapping is undesirable and causes performance to degrade. Vertica recommends that you allocate minimum swap space as follows:

RAM	Swap Space
2 GB	4 GB
4 GB	8 GB
8+ GB	RAM + 2GB

2.4 Package Requirements

Vertica requires the following packages installed on the server

- libreadline
- ntp
- pam
- python (version 2.5 or greater)
- ssh
- sudo
- bc

Vertica also recommends the following optional packages

- gcc
- perl
- sysstat
- zip

2.5 Conflicting Packages

Vertica recommends against installing or running other processor or memory intensive services the same system including

- Application Servers,
- Web Servers and
- Other Database Engines

2.6 Deployment Directory

Vertica installs in **/opt/vertica**. This directory must **not** be shared across the machines and should be mounted on a local disk. Vertica will automatically install the RPM on all nodes in the cluster and replicate the contents of the configuration directory as well as the license key file. Minimum disk requirements for Vertica directories are as follows:

Path	Description	Minimum Free Space
/opt	Vertica installs in /opt/vertica	50MB for install
/catalog	Path to directory for database catalog	2x RAM always free
/data	Path to directory for database files	10x RAM always free

2.7 Creating a Striped Data Disk

For storing database files, Vertica recommends using multiple dedicated physical drives for each machine. These drives may be striped (RAID0) using either a hardware RAID controller if it can stripe in 1MB chunks or in software using **mdadm**. Follow these instructions to configure a stripe set using **md**.

1. Determine the device names of the disks to be striped. These are listed during install and sometimes with the machine is started (check dmsg). They may also be listed using dmesg or

```
mdadm –query /dev/sd*
```

A four disk set may have device names `/dev/sda1, /dev/sdb1, /dev/sdc1,`

```
/dev/sdd1
```

2. Run the **mdadm** command to create a stripe set. This command creates a new device (/dev/md0) that is a RAID level 0 stripe set with chunk size of 1024k. In this example the stripe set is composed of the four disk devices listed. The 1MB chunk size is chosen because Vertica reads and writes in 1MB chunks.

   Repeat once for each `/dev/sd[b,c,d]`
   ```
   $ printf 'n\np\n1\n\n\nt\nfd\nw\n' | fdisk /dev/sda
   ```

   Note the "1"
   ```
   $ /sbin/mdadm --create /dev/md0 --level=0 --chunk=1024 --raid-devices=4
   /dev/sd[bcde]1
   ```

3. With the new device created, format it ext3

   ```
   $ mkfs.ext3 /dev/md0
   ```

4. Create a **/data** directory and add mount information to **/etc/fstab**

   ```
   /dev/md0     /data        ext3    defaults      0 0
   ```

2.8 Locale and Time zone

Vertica recommends that you set the LANG and TZ environment variables are correctly configured. The preferred LANG is en_US.UTF-8. Ensure that the environment variables are set on all machines in the cluster or you may experience inconsistent results in data returned from different machines in the cluster.

2.9 Network Time Protocol

Vertica recommends keeping the time on all machines in the cluster in sync using NTP. While Vertica itself is resilient to clock skew, each process uses its local clock for time based functions. Time variances may cause inconsistent query results when using date and time functions.

2.10 Database Administration User

The Vertica catalog and database directories must be owned by the **dbadmin** user. The user may be created at installation by the Vertica installer or can be deployed prior to installation.

When deploying a user manually, ensure the user has password-less ssh access between all nodes in the cluster and owns both the catalog and database directories. The installer will check for this at install time.

2.11 Networking

Vertica recommends connecting the database hosts on a private GigE network, using a secondary interface for client communications. The private network may be configured in a high availability deployment using multiple physical interfaces and switch switches bound to each host as a single IP using NIC bonding.

Check open sockets on each node in the cluster, in particular look for servers listening on ports 4803 (the spread port) and 5433 (the Vertica Database port). You can use the following command:

```
# netstat -nap | grep -e 4803 -e 5433
```

Before installing Vertica the command should have no output indicating that both ports are available. After installing Vertica and before creating a database, you should see spread bound to port 4803:

```
tcp      0    0 0.0.0.0:4803  0.0.0.0:* LISTEN      2206/spread
udp      0    0 0.0.0.0:4803  0.0.0.0:*             2206/spread
```

After creating a Vertica database, you should see both spread bound to port 4803 and Vertica bound to port 5433:

```
tcp      0    0 0.0.0.0:5433  0.0.0.0:* LISTEN      2840/vertica
tcp      0    0 :::5433       :::*      LISTEN      2840/vertica
tcp      0    0 0.0.0.0:4803  0.0.0.0:* LISTEN      2206/spread
udp      0    0 0.0.0.0:4803  0.0.0.0:*            2206/spread
```

2.12 Hostnames

Names for all hosts must be forward and reverse resolvable and should have consistent case names. When possible make sure host identify themselves and can resolve each other using canonical names rather than fully qualified names.

Vertica recommends setting up hostnames using **/etc/hosts** and configuring **/etc/resolv.conf** to resolve hostnames using **/etc/hosts**.

2.13 IP Addressing

Vertica host IP addresses must be permanent. This can be configured as a static IP address or using DHCP with statically bound IP address. The IP address for a Vertica host may not change and any replacement nodes must assume the IP address of the original host.

2.14 Firewalls

Vertica recommends disabling SELinux and Linux firewalls on all database host and recommends against using iptables. Review your Linux installation guide for information on how to disable any other system firewalls that may be present.

If you choose to configure IP Tables, allow ports 4803 UDP and 5433 TCP as well as higher ports for session connections.

Check whether iptables is running by issuing the following command:

```
# /etc/init.d/iptables status
Firewall is stopped.
```

If iptables is not stopped you may stop it by issuing the command

```
# /etc/init.d/iptables stop
Firewall is stopped.
```

You may also remove it from the appropriate run levels in /etc/rc.d.

2.15 SE Linux

Most Linux systems offer a feature called Secure Linux or SELinux. This feature offers fine grained control over system security. You may configure SELinux to allow Vertica access to the network though Vertica recommends disabling it. On RedHat Enterprise Linux you can configure SELinux using the following command:

```
# /usr/bin/system-config-securitylevel-tui
```

Follow the instructions to disable the Security Level and SELinux.

2.16 Installation

Root access is required to install the Vertica RPM and to run the `install_vertica` script. These can both be performed by a system administrator or scripted to for automated deployment. Both operations may be completed using sudo provided it has privileges to install the Vertica binaries and configure the spread service. To install the RPM issue the following command

```
rpm -i {rpm file}
```

Following rpm installation run the `install_vertica` command providing valid arguments as detailed in the Vertica Installation Guide.

2.17 Replacing Nodes

When replacing a failed machine the replacement machine must be configured identically to the machine being replaced. Follow these steps to configure the new host:

1. Install the same OS and use the same IP address, hostname and catalog and data paths.
2. Ensure password-less ssh connectivity between all machines including the new host.
3. Install the same version of the Vertica RPM.
4. Create the dbadmin user.
5. Create the catalog and data paths and set the same permissions as the original host.
   If the original catalog directory was:
   `/catalog/devdb/DATABASE_NAME/sitexx_catalog/`
   then create this directory:
   `/catalog/devdb/DATABASE_NAME/`
6. From the administration tools select "Start a Node"

Vertica will automatically initialize the configuration and create the catalog and data files based on the configuration in the rest of the cluster.

Note: If you have made any changes to the vertica.conf file these will not be reflected on the new host. After the new host has recovered you may stop the Vertica process on that host, replace the Vertica.conf file and restart the Vertica node.

3. Jaspersoft / Vertica Integration

[This document describes a standard installation of JasperServer on Red Hat Enterprise Linux which will install a copy of the Apache Tomcat application server. It is also possible to manually deploy the JasperServer application to an existing application server. JBoss, WebSphere, GlassFish, and other application servers are supported. Manual deployment of JasperServer is outside the scope of this document.]

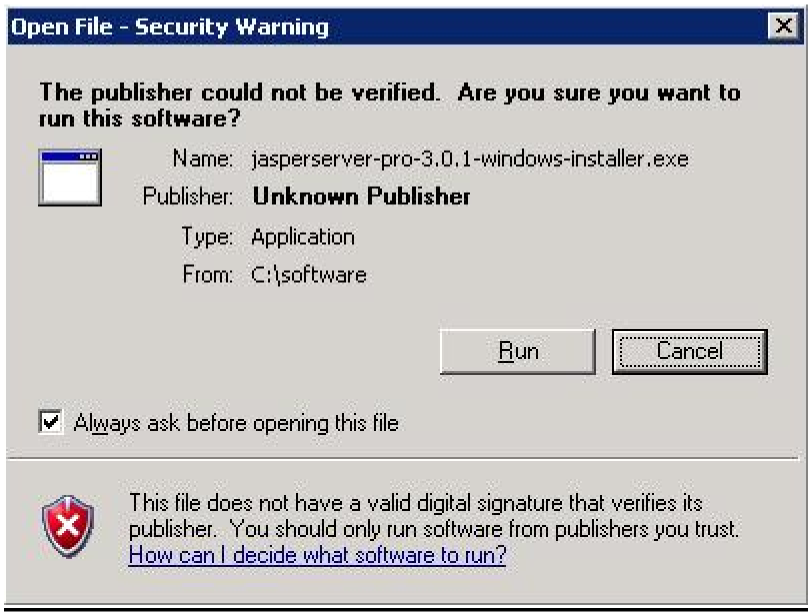
Pre-Requirements:

- Jaspersoft Software
- Vertica DBMS
- Vertica JDBC Driver

Please ensure that you have downloaded and installed the latest copy of the Vertica JDBC driver. The driver and the latest version number can be found on the Vertica Systems website at http://www.vertica.com/v-zone/download_vertica link using your username and password. Also, download the Jaspersoft software from the link http://www.jaspersoft.com/downloads.html and follow the steps given below to install the Jasper soft Software.

3.1 Steps to Install Jaspersoft Software

1) Double click the exe downloaded from http://www.jaspersoft.com/downloads.html and click on Run.



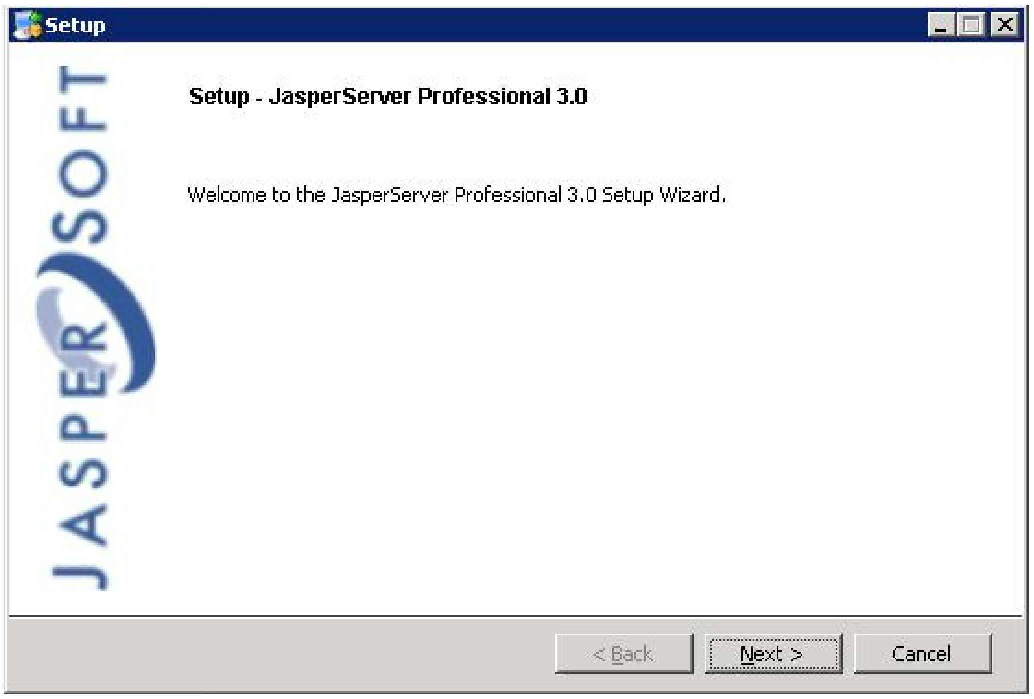
Open File - Security Warning

The publisher could not be verified. Are you sure you want to run this software?

Name: jasperserver-pro-3.0.1-windows-installer.exe
Publisher: Unknown Publisher
Type: Application
From: C:\software

This file does not have a valid digital signature that verifies its publisher. You should only run software from publishers you trust. How can I decide what software to run?

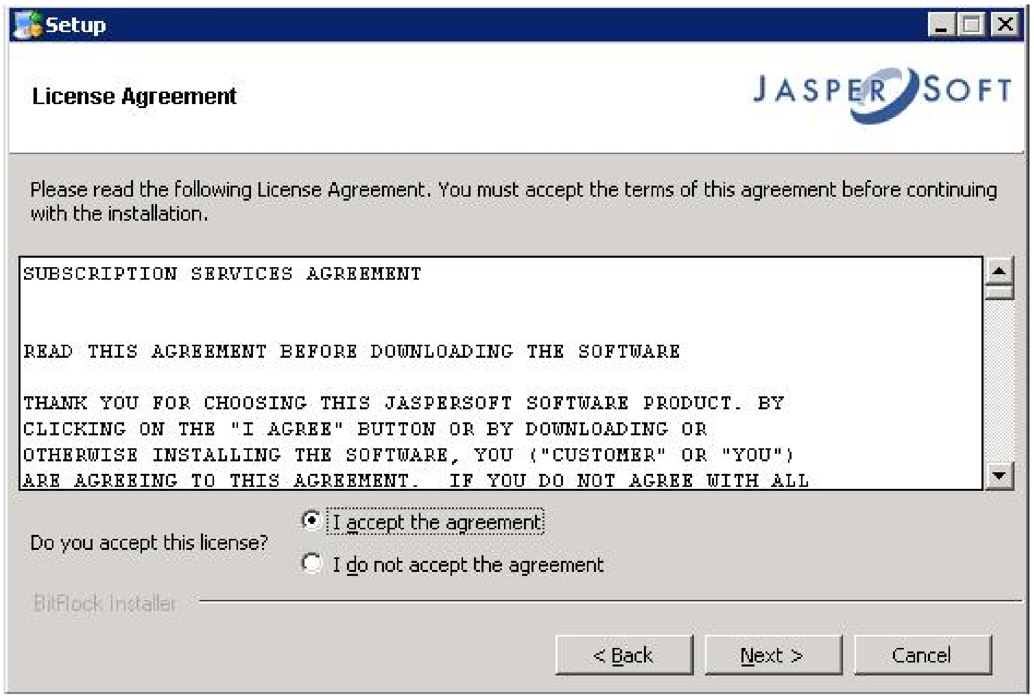
2) A Setup window of Jasper Server Professional 3.0 appears. Click on Next in the window below.



Setup - JasperServer Professional 3.0

Welcome to the JasperServer Professional 3.0 Setup Wizard.

3) Accept the License Agreement by checking the radio button as shown in the window below.



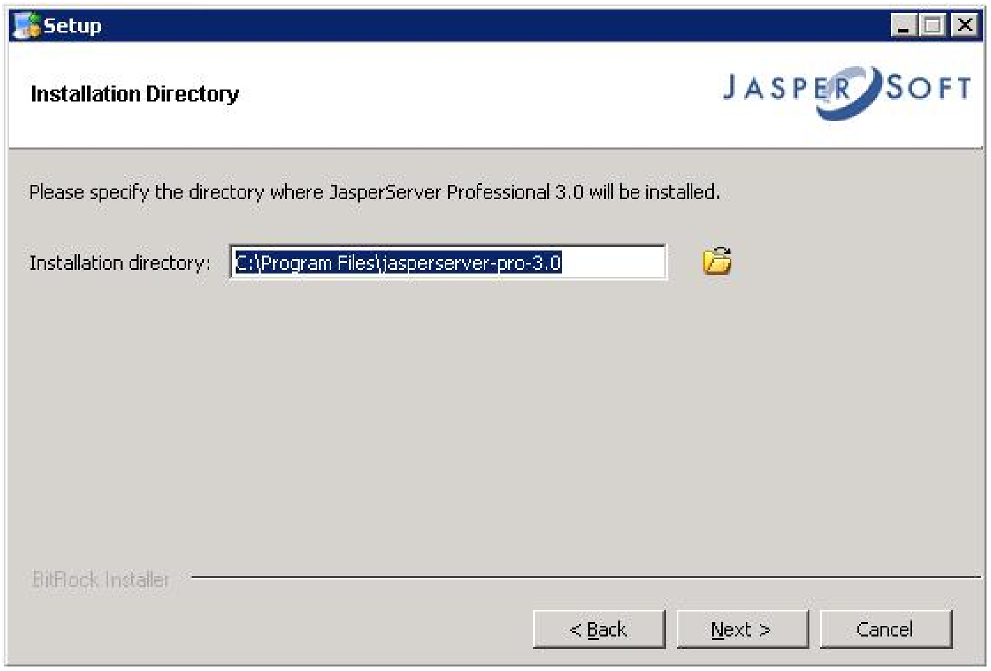
License Agreement

Please read the following License Agreement. You must accept the terms of this agreement before continuing with the installation.

```
SUBSCRIPTION SERVICES AGREEMENT


READ THIS AGREEMENT BEFORE DOWNLOADING THE SOFTWARE

THANK YOU FOR CHOOSING THIS JASPERSOFT SOFTWARE PRODUCT. BY
CLICKING ON THE "I AGREE" BUTTON OR BY DOWNLOADING OR
OTHERWISE INSTALLING THE SOFTWARE, YOU ("CUSTOMER" OR "YOU")
ARE AGREEING TO THIS AGREEMENT.  IF YOU DO NOT AGREE WITH ALL
```

Do you accept this license? [✓ I accept the agreement] I do not accept the agreement

4) Select a path of the directory (other than the default) if you want to change the path of the installation directory and then click Next.

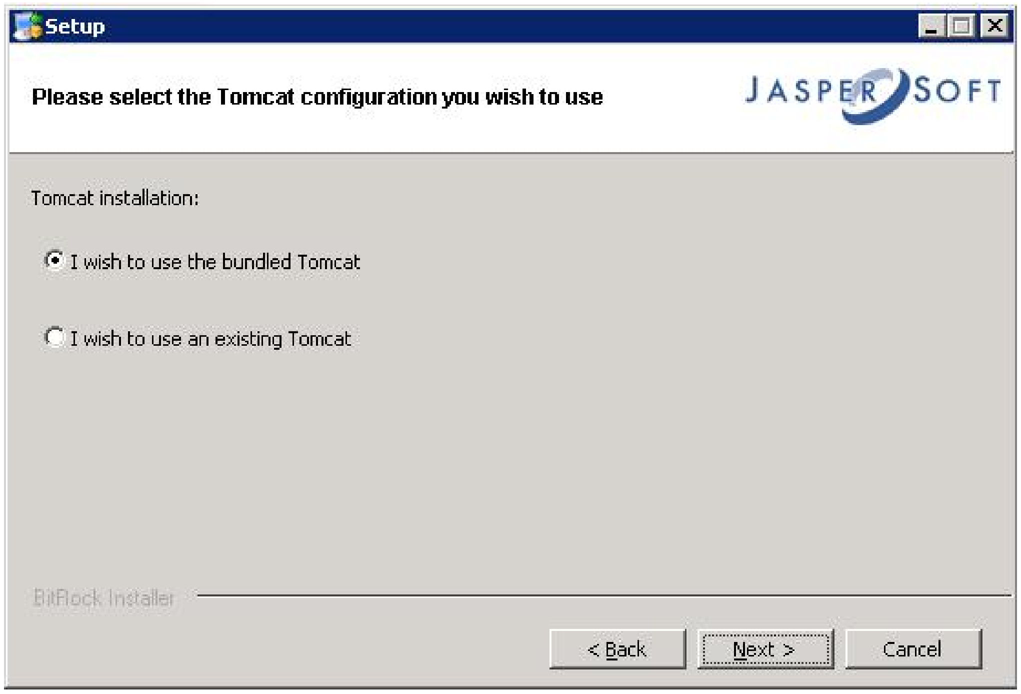


Installation Directory

Please specify the directory where JasperServer Professional 3.0 will be installed.

Installation directory: C:\Program Files\jasperserver-pro-3.0

5) Select the first option if you want to use the bundled Tomcat with the Software otherwise if you wish to use the existing installed Tomcat then select second option in the window below.



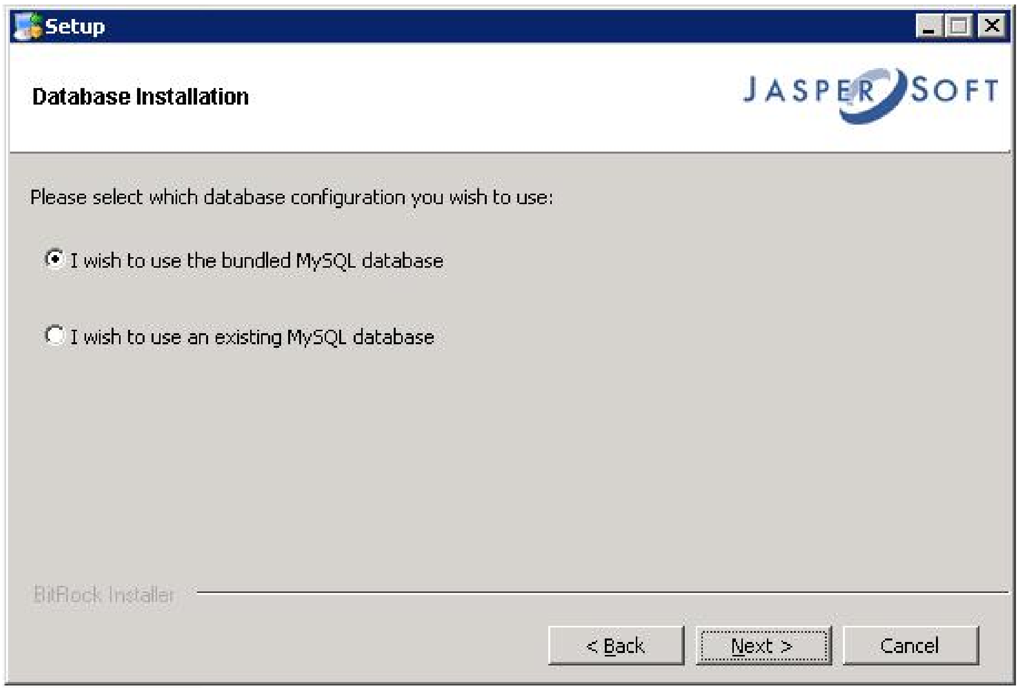
Please select the Tomcat configuration you wish to use

Tomcat installation:

● I wish to use the bundled Tomcat

○ I wish to use an existing Tomcat

6) Select the first option if you want to use the bundled MySQL Database with the Software otherwise if you wish to use the existing installed MySQL Database then select second option in the window below.


Setup

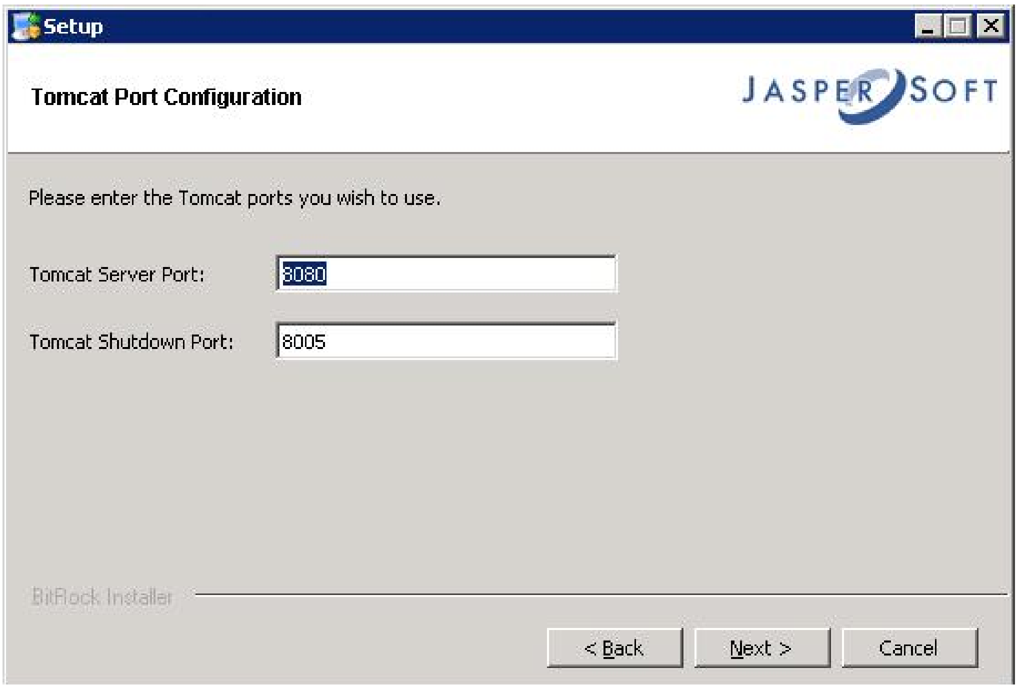
Database Installation

Please select which database configuration you wish to use:

(●) I wish to use the bundled MySQL database

( ) I wish to use an existing MySQL database

7) The window below shows the Tomcat Port Configuration. Click Next.


Setup

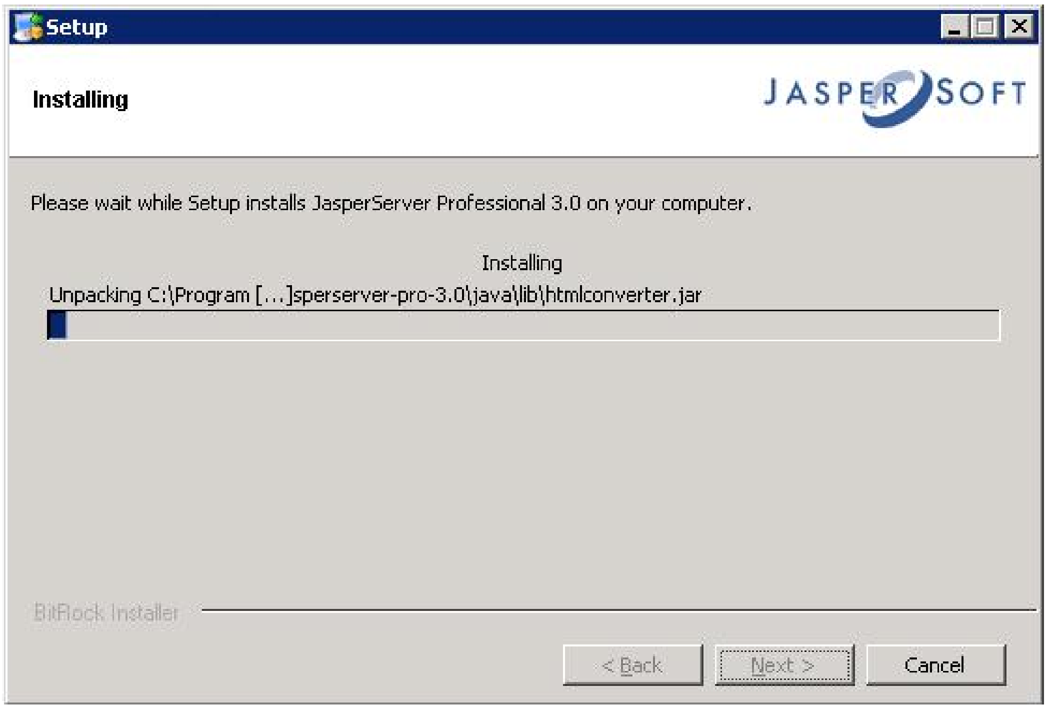
Tomcat Port Configuration

Please enter the Tomcat ports you wish to use.

Tomcat Server Port: 8080

Tomcat Shutdown Port: 8005

8) Keep Clicking "Next" in the windows ahead. Installation will start as shown in the window below.

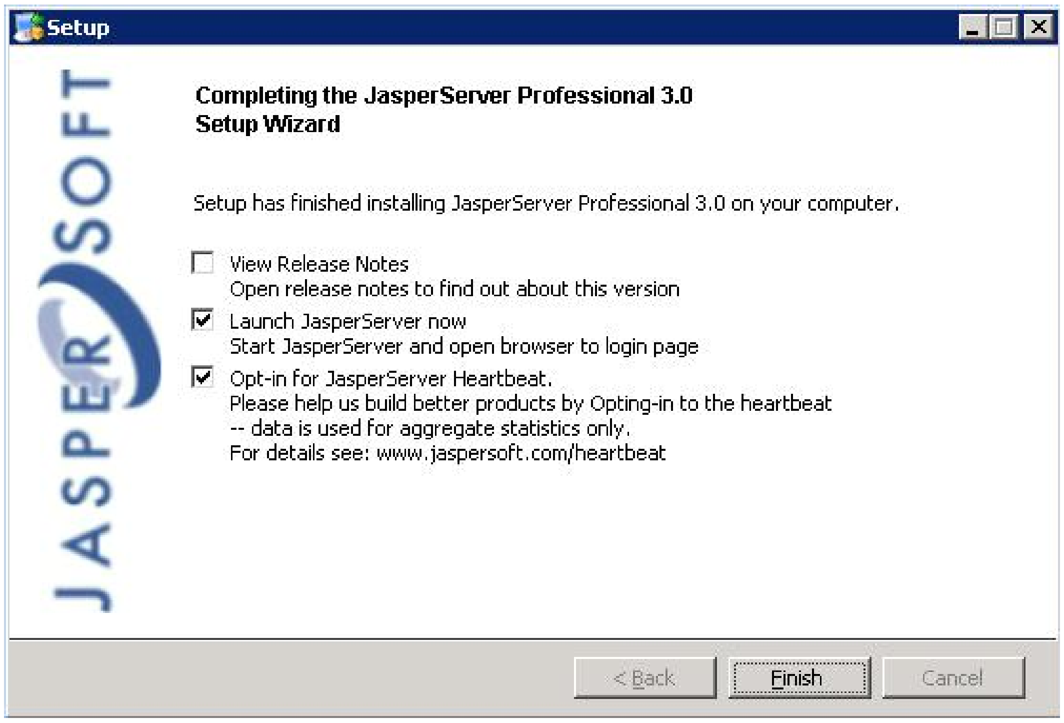


Installing

Please wait while Setup installs JasperServer Professional 3.0 on your computer.

Installing

Unpacking C:\Program [...]sperserver-pro-3.0\java\lib\htmlconverter.jar



Completing the JasperServer Professional 3.0 Setup Wizard

Setup has finished installing JasperServer Professional 3.0 on your computer.

☐ View Release Notes
Open release notes to find out about this version

☑ Launch JasperServer now
Start JasperServer and open browser to login page

☑ Opt-in for JasperServer Heartbeat.
Please help us build better products by Opting-in to the heartbeat
-- data is used for aggregate statistics only.
For details see: www.jaspersoft.com/heartbeat

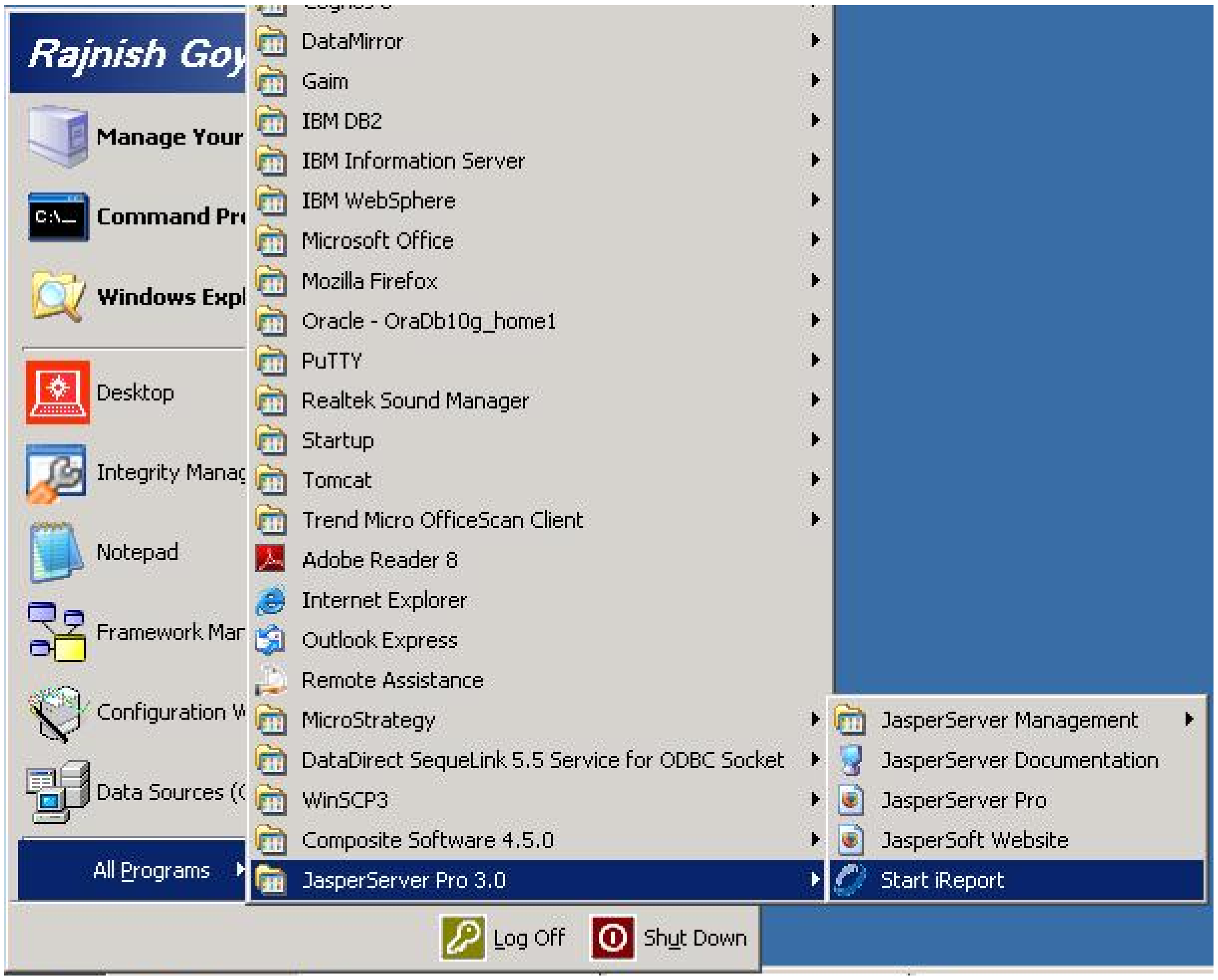
9) Click Finish. This completes the installation of Jasper Soft Software.

3.2 Steps to Connect Jaspersoft with Vertica

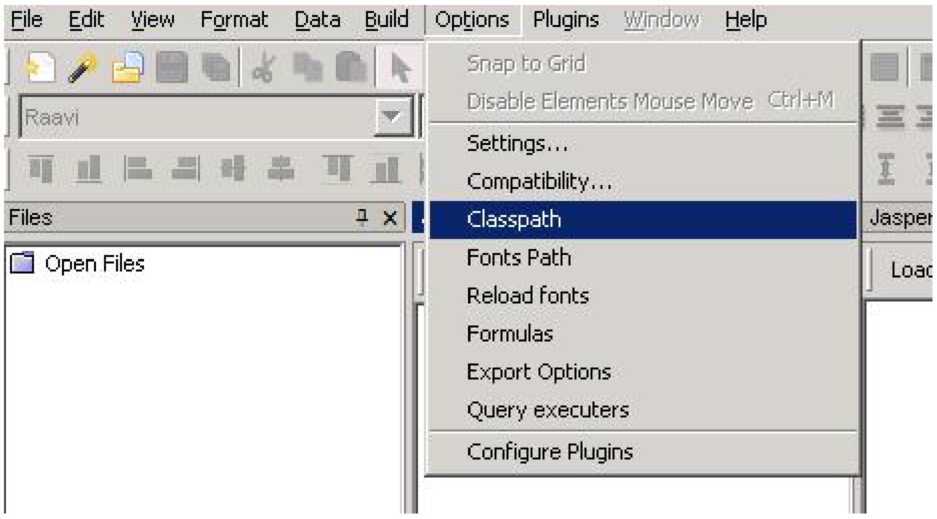
Below are steps to connect vertica to Jaspersoft and check the connection by listing the schema and tables.

1) First of all to configure Jasper with Vertica, copy the Vertica JDBC jar's file (vertica_x.x_jdk-_5.jar where x.x is the version of the JDBC Driver) from "C:\Program Files\Vertica Systems\Vertica Client Drivers 2.2\lib" to "InstallationDirectory_Jaspersoft"\ireport\lib (example "C:\Program Files\jasperserver-pro-3.0\ireport\lib").

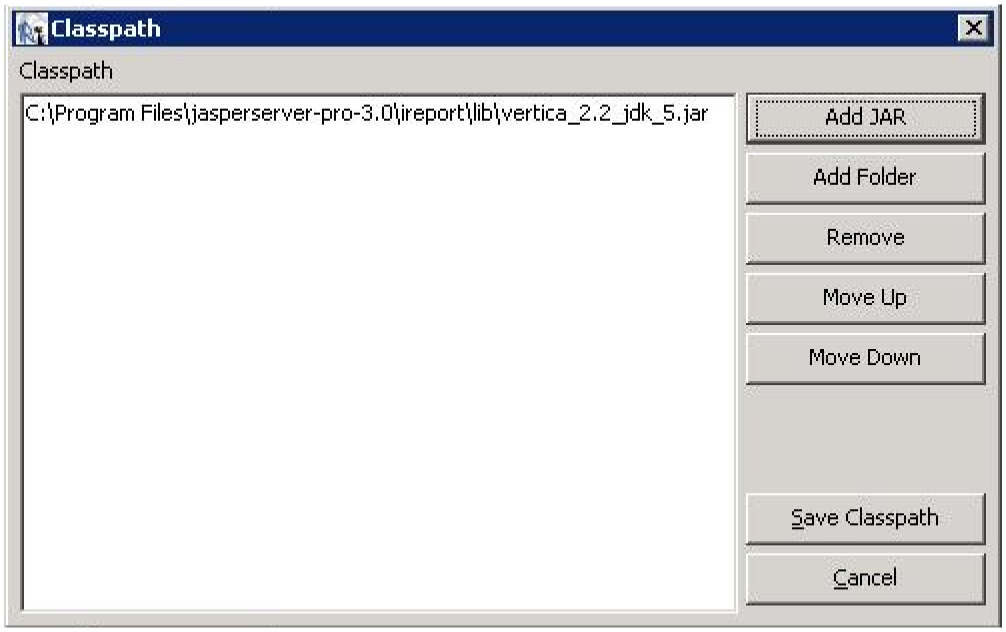
2) From Windows Start up, Select All Programs -> JasperServer Pro 3.0 - > Start iReport. This will open iReport 3.0.



3) Go to Options - > Classpath. This will open the window to set the classpath.

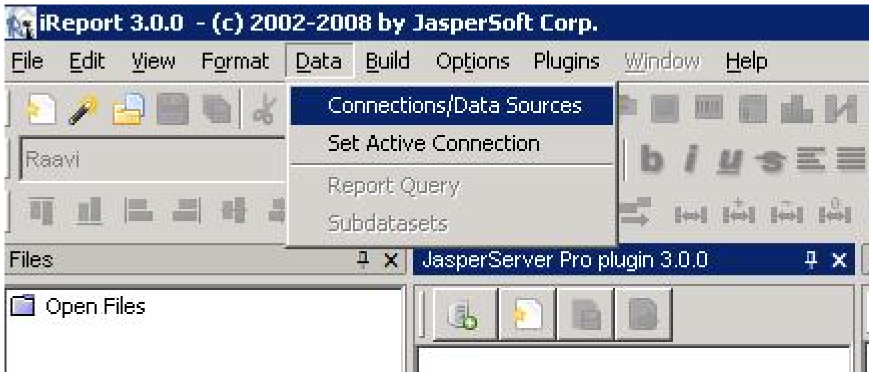


4) The window below opens. Click Add JAR and select "InstallationDirectory_Jaspersoft"\ireport\lib\vertica_x.x_jdk_5.jar where x.x is the version of the JDBC Driver. This will set the classpath of the driver.

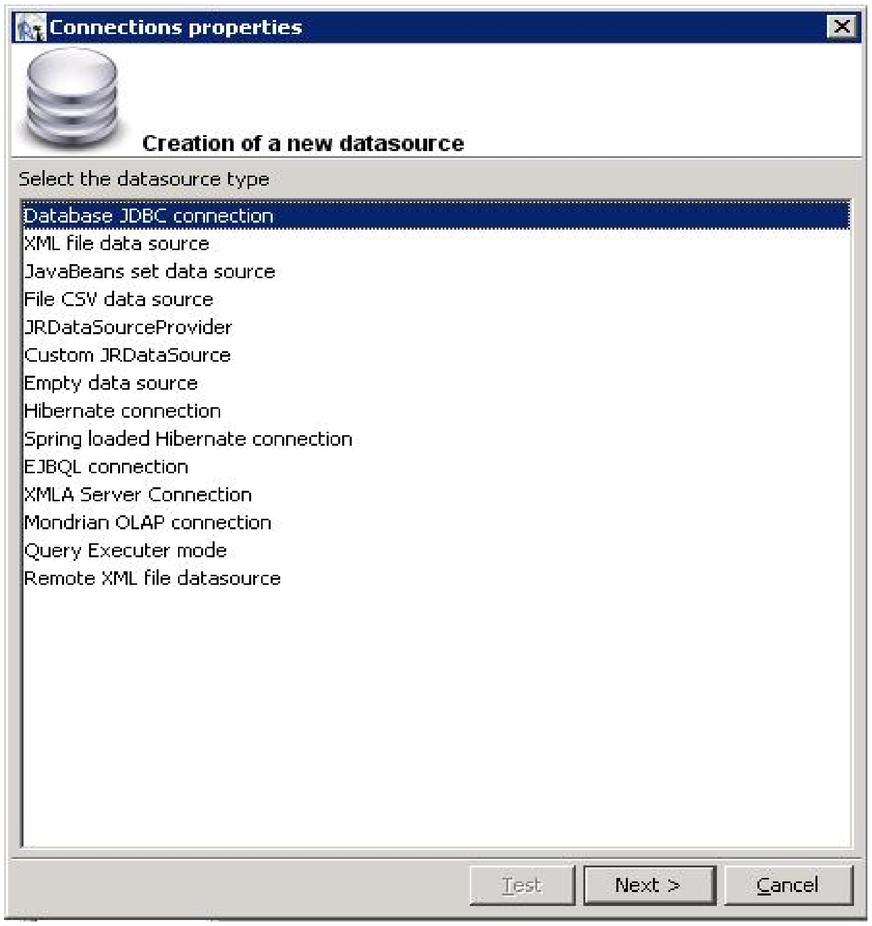


C:\Program Files\jasperserver-pro-3.0\ireport\lib\vertica_2.2_jdk_5.jar

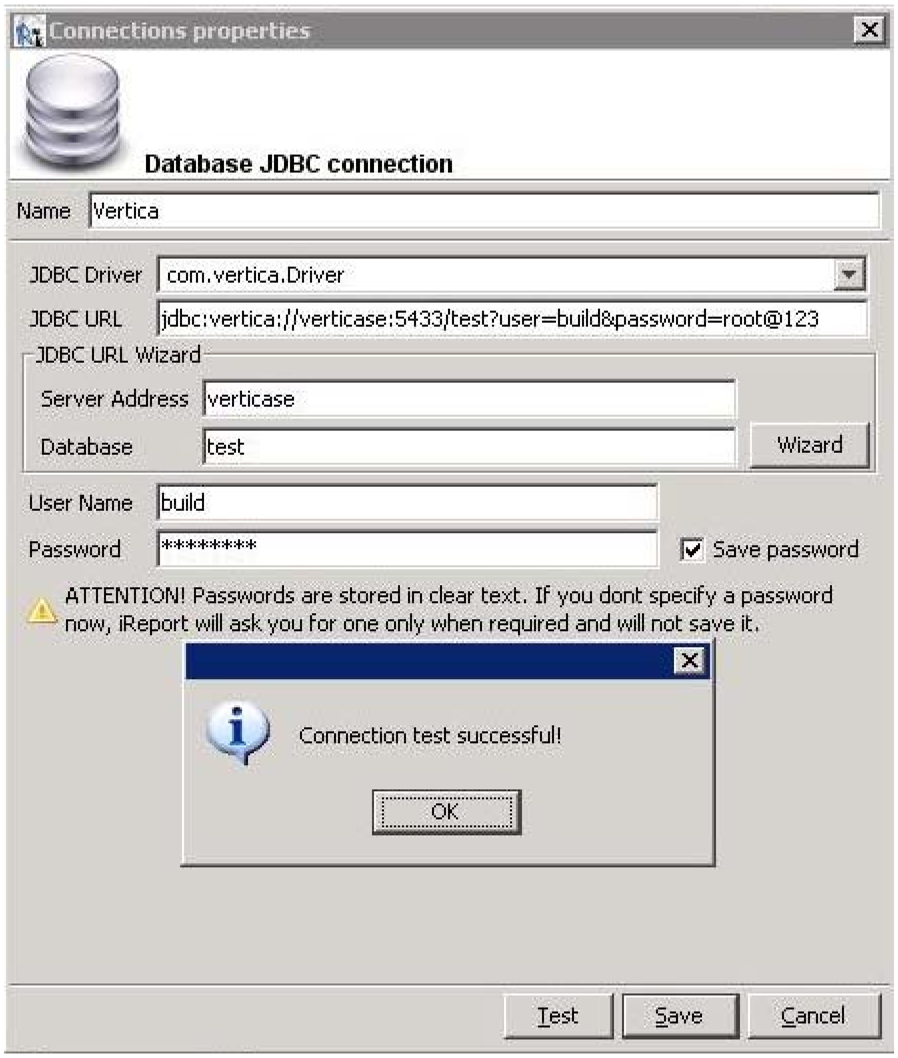
5) Go to Data - > Connections/Data Sources to create a new data source. This will open a new window. Click on "New" in new window opened.


iReport 3.0.0 - (c) 2002-2008 by JasperSoft Corp.
File Edit View Format Data Build Options Plugins Window Help
Connections/Data Sources
Set Active Connection
Report Query
Subdatasets
Raavi
Files
Open Files
JasperServer Pro plugin 3.0.0

6) Select Database JDBC connection as the new data source and click “Next”.



7) Enter the Database JDBC Driver Connection Properties as in the below figure and Click on “Test” to test the connection.



Name: Vertica

JDBC Driver: com.vertica.Driver

JDBC URL: jdbc:vertica://verticase:5433/test?user=build&password=root@123

JDBC URL Wizard

Server Address: verticase

Database: test

User Name: build

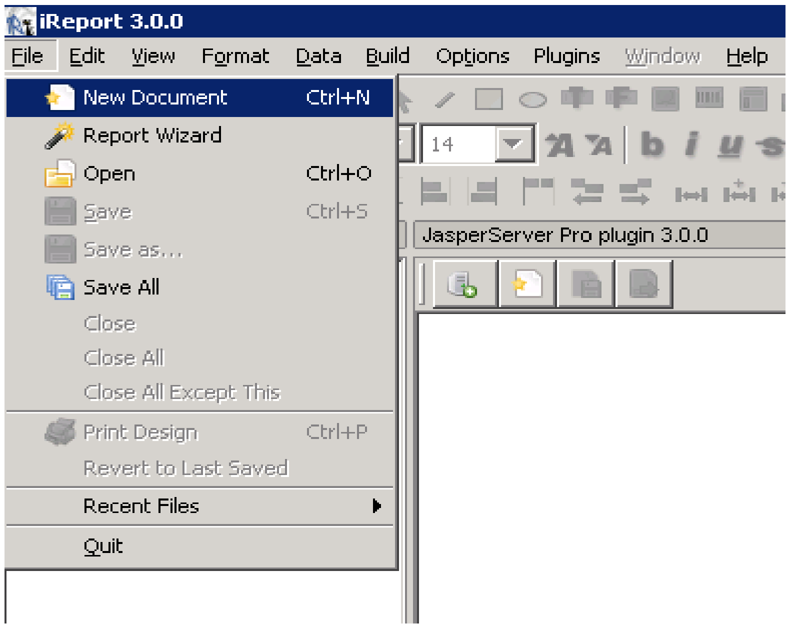
Password: ********

[✓] Save password

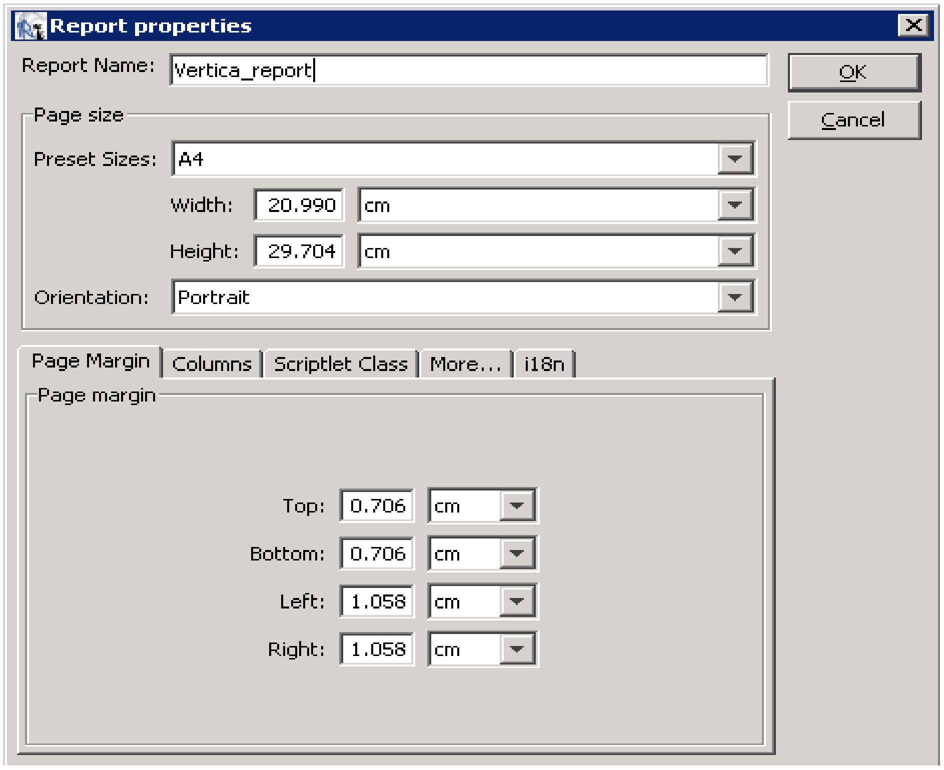
ATTENTION! Passwords are stored in clear text. If you dont specify a password now, iReport will ask you for one only when required and will not save it.

Connection test successful!

8) Go to File - > New Document.



9) Enter the report name “Vertica_report” in the Report Name field.


Report properties

Report Name: Vertica_report

Page size

Preset Sizes: A4

Width: 20.990 cm

Height: 29.704 cm

Orientation: Portrait

Page Margin | Columns | Scriptlet Class | More... | i18n

Page margin

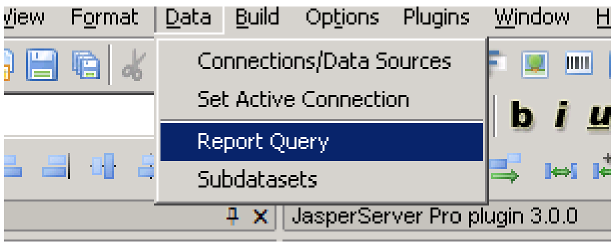
Top: 0.706 cm

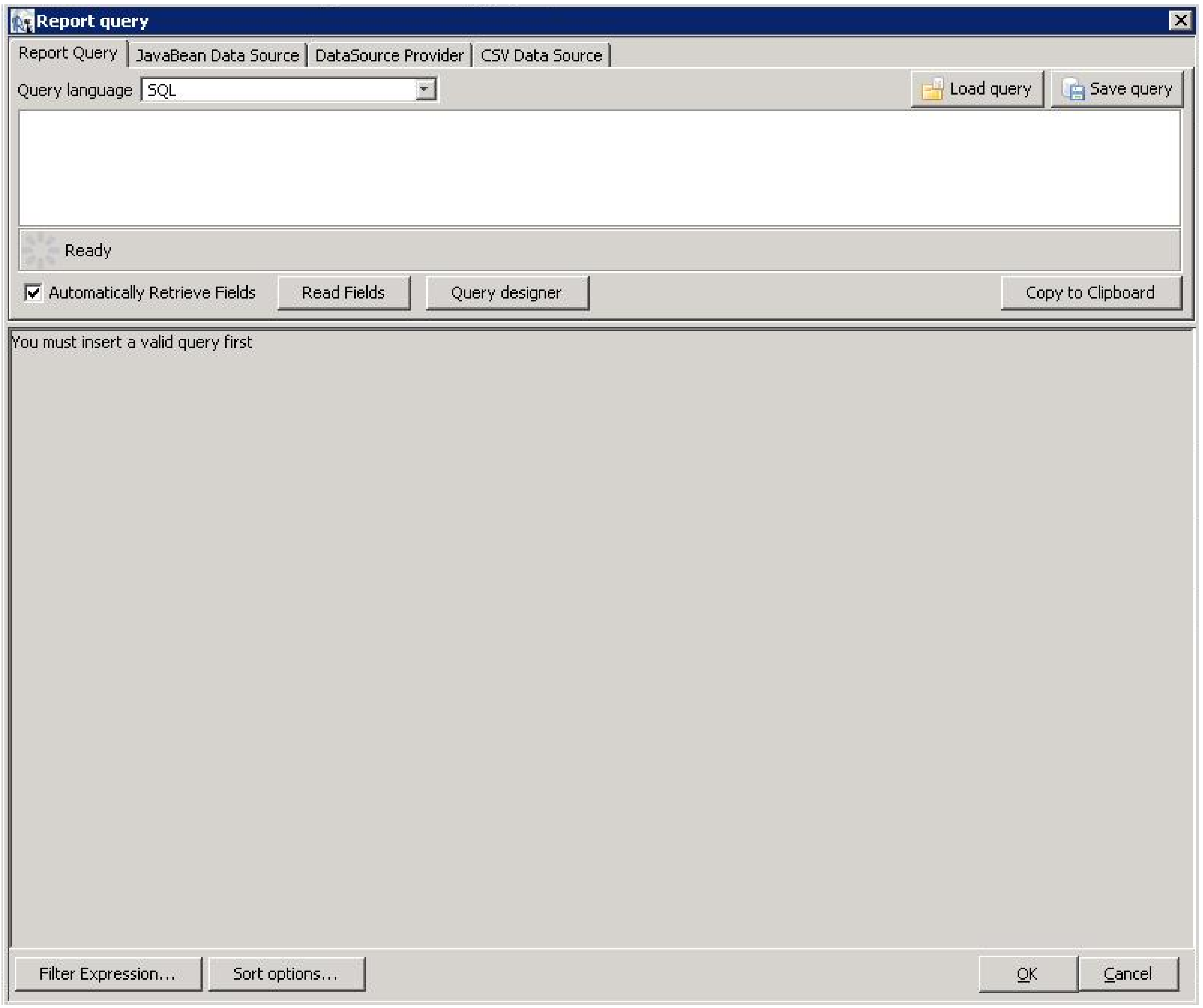
Bottom: 0.706 cm

Left: 1.058 cm

Right: 1.058 cm

10) Go to Data - > Report Query.


View Format Data Build Options Plugins Window H
Connections/Data Sources
Set Active Connection
Report Query
Subdatasets
JasperServer Pro plugin 3.0.0

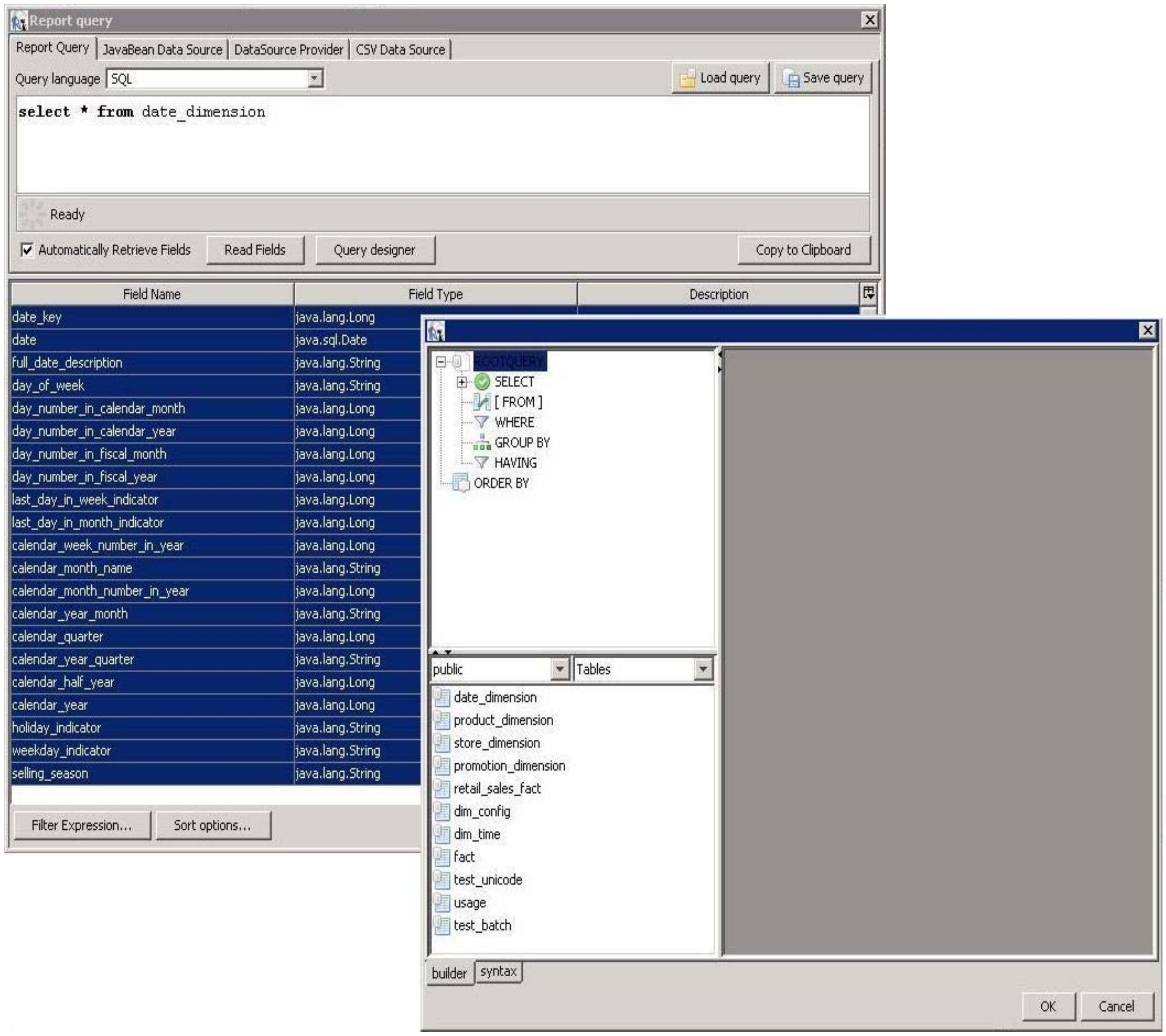


Query language: SQL

Ready

You must insert a valid query first

11) Enter a valid query in the query window example “select * form date_dimension”.
 Click Query designer which will open the second window given below. It shows the tables and schema information and is used to design the query.


Report query

Report Query | JavaBean Data Source | DataSource Provider | CSV Data Source

Query language SQL

Load query | Save query

```
select * from date_dimension
```

Ready

☑ Automatically Retrieve Fields | Read Fields | Query designer | Copy to Clipboard

Field Name	Field Type	Description
date_key	java.lang.Long	
date	java.sql.Date	
full_date_description	java.lang.String	
day_of_week	java.lang.String	
day_number_in_calendar_month	java.lang.Long	
day_number_in_calendar_year	java.lang.Long	
day_number_in_fiscal_month	java.lang.Long	
day_number_in_fiscal_year	java.lang.Long	
last_day_in_week_indicator	java.lang.Long	
last_day_in_month_indicator	java.lang.Long	
calendar_week_number_in_year	java.lang.Long	
calendar_month_name	java.lang.String	
calendar_month_number_in_year	java.lang.Long	
calendar_year_month	java.lang.String	
calendar_quarter	java.lang.Long	
calendar_year_quarter	java.lang.String	
calendar_half_year	java.lang.Long	
calendar_year	java.lang.Long	
holiday_indicator	java.lang.String	
weekday_indicator	java.lang.String	
selling_season	java.lang.String	

Filter Expression... | Sort options...

ROOTQUERY
- SELECT
- [FROM]
- WHERE
- GROUP BY
- HAVING
- ORDER BY

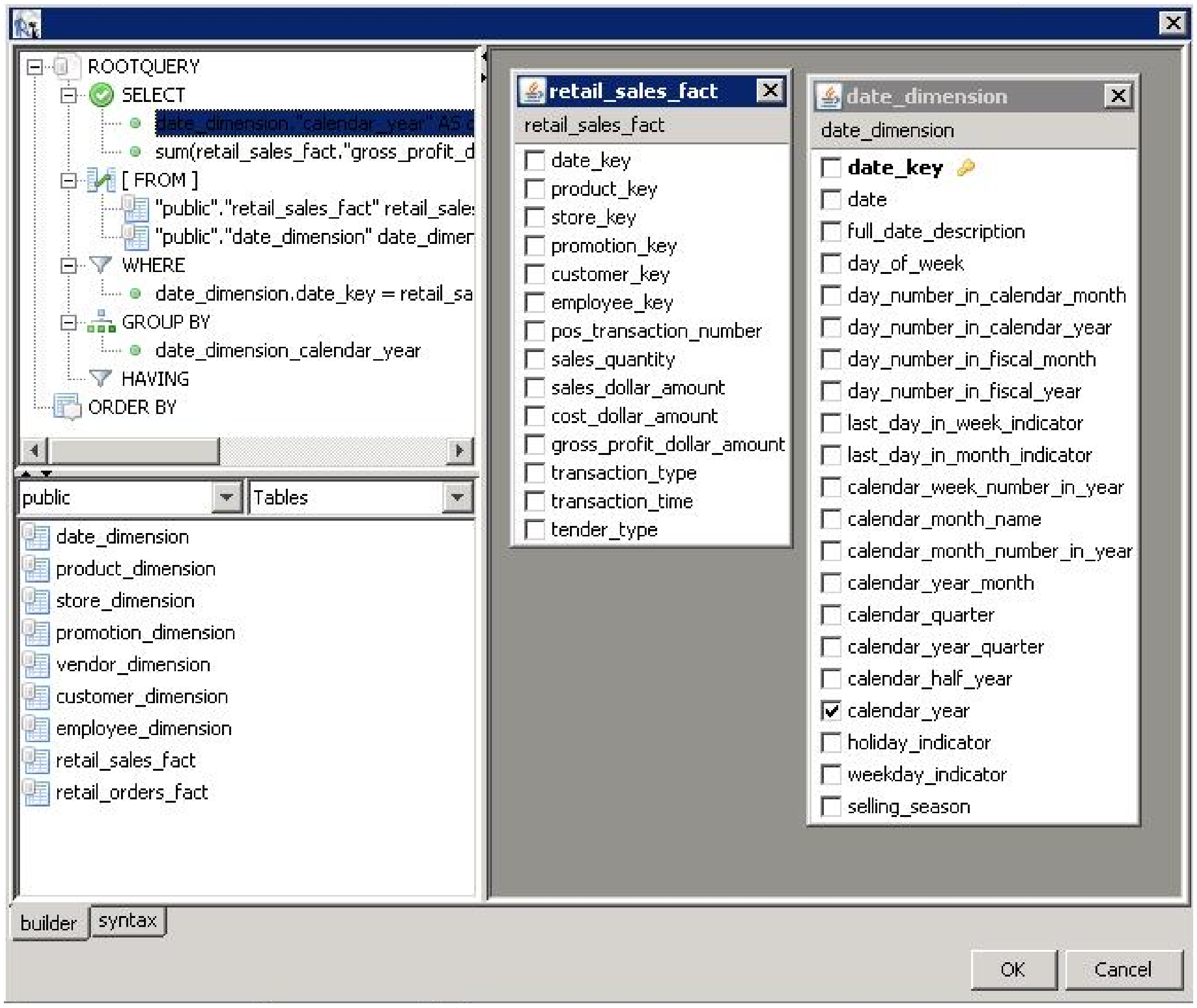
public | Tables

- date_dimension
- product_dimension
- store_dimension
- promotion_dimension
- retail_sales_fact
- dim_config
- dim_time
- fact
- test_unicode
- usage
- test_batch

builder | syntax

OK | Cancel

12) Window below shows how to create query in the Query Designer. It is just drag and drop of tables and checking boxes for selecting the particular columns. We can also create the query with conditions, group by clause etc as given below.



ROOTQUERY
SELECT
date_dimension."calendar_year" AS c
sum(retail_sales_fact."gross_profit_d
[FROM]
"public"."retail_sales_fact" retail_sale:
"public"."date_dimension" date_dimer
WHERE
date_dimension.date_key = retail_sa
GROUP BY
date_dimension_calendar_year
HAVING
ORDER BY

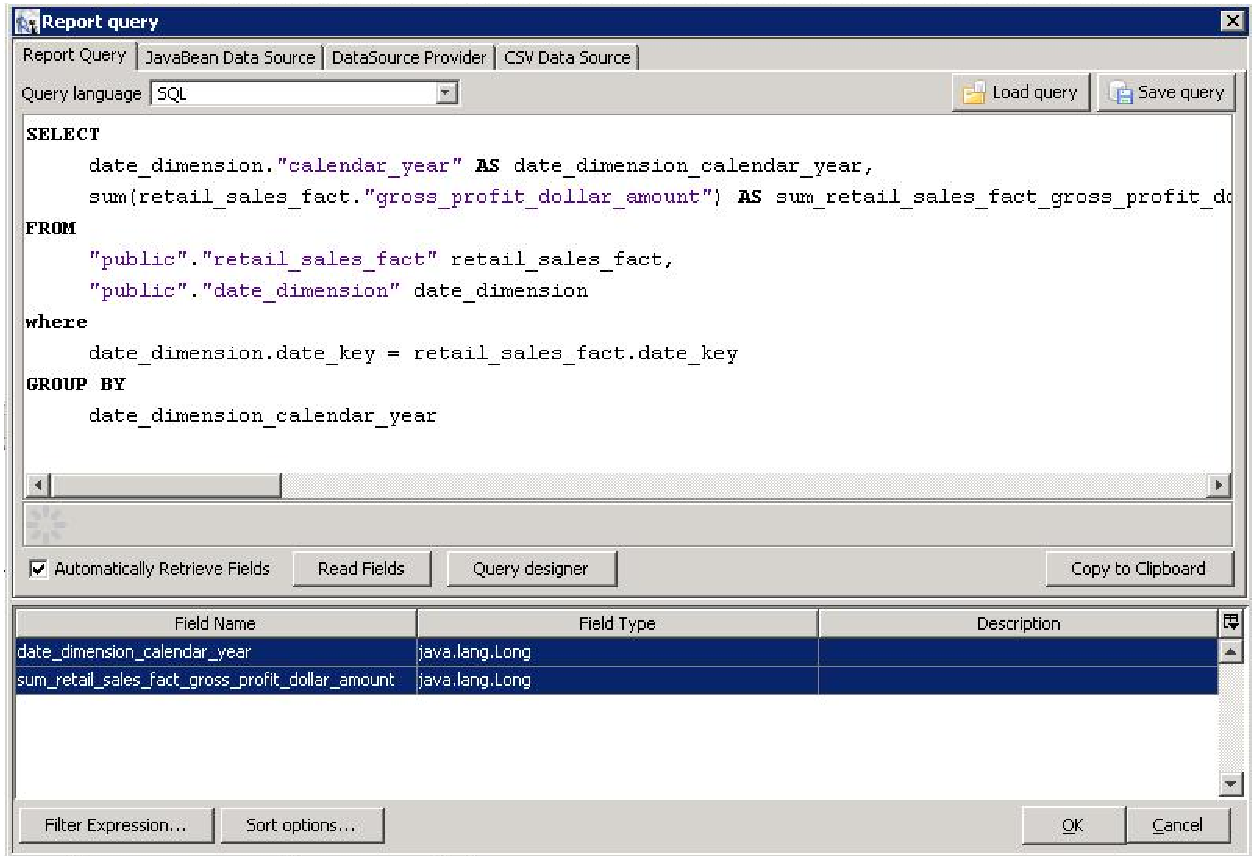
public | Tables

date_dimension
product_dimension
store_dimension
promotion_dimension
vendor_dimension
customer_dimension
employee_dimension
retail_sales_fact
retail_orders_fact

retail_sales_fact
retail_sales_fact
- [ ] date_key
- [ ] product_key
- [ ] store_key
- [ ] promotion_key
- [ ] customer_key
- [ ] employee_key
- [ ] pos_transaction_number
- [ ] sales_quantity
- [ ] sales_dollar_amount
- [ ] cost_dollar_amount
- [ ] gross_profit_dollar_amount
- [ ] transaction_type
- [ ] transaction_time
- [ ] tender_type

date_dimension
date_dimension
- [ ] **date_key**
- [ ] date
- [ ] full_date_description
- [ ] day_of_week
- [ ] day_number_in_calendar_month
- [ ] day_number_in_calendar_year
- [ ] day_number_in_fiscal_month
- [ ] day_number_in_fiscal_year
- [ ] last_day_in_week_indicator
- [ ] last_day_in_month_indicator
- [ ] calendar_week_number_in_year
- [ ] calendar_month_name
- [ ] calendar_month_number_in_year
- [ ] calendar_year_month
- [ ] calendar_quarter
- [ ] calendar_year_quarter
- [ ] calendar_half_year
- [x] calendar_year
- [ ] holiday_indicator
- [ ] weekday_indicator
- [ ] selling_season

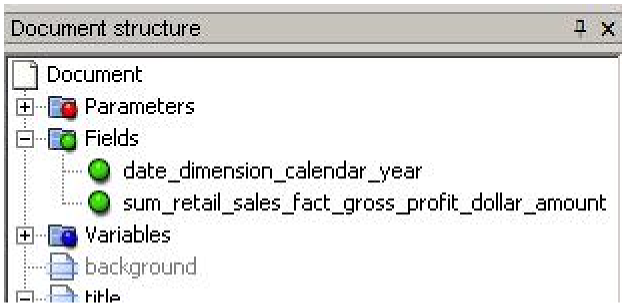
13) The below window shows the query designed by the query designer.



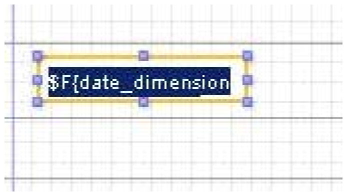
```
SELECT
     date_dimension."calendar_year" AS date_dimension_calendar_year,
     sum(retail_sales_fact."gross_profit_dollar_amount") AS sum_retail_sales_fact_gross_profit_dc
FROM
     "public"."retail_sales_fact" retail_sales_fact,
     "public"."date_dimension" date_dimension
where
     date_dimension.date_key = retail_sales_fact.date_key
GROUP BY
     date_dimension_calendar_year
```

Field Name	Field Type	Description
date_dimension_calendar_year	java.lang.Long	
sum_retail_sales_fact_gross_profit_dollar_amount	java.lang.Long	

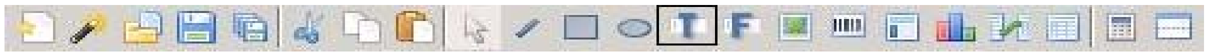
14) Now the fields selected in the query is shown in the Document structure Pane as given below.

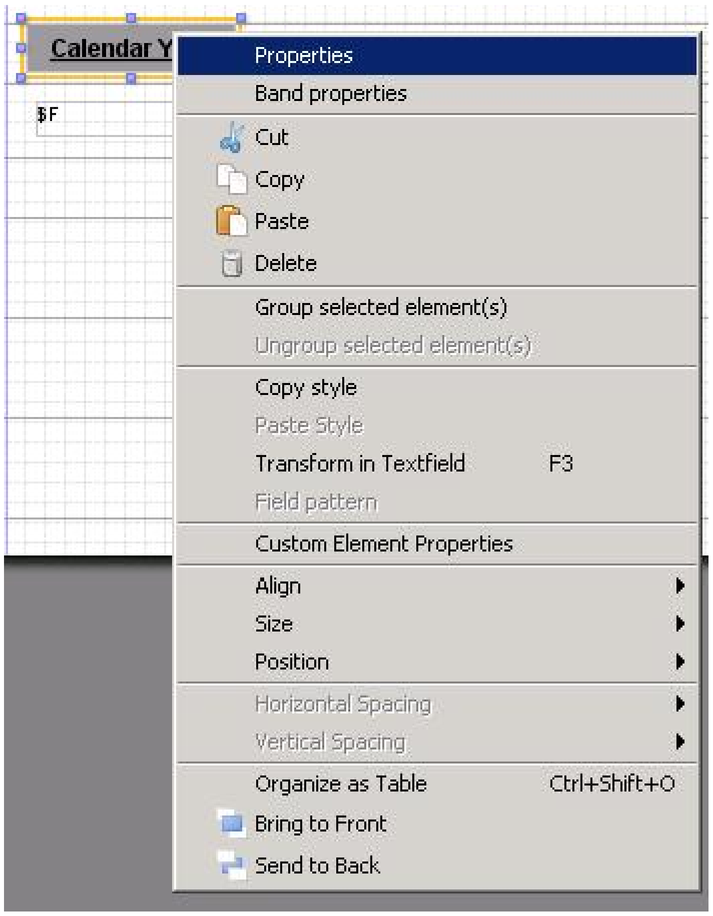

Document structure
Document
Parameters
Fields
date_dimension_calendar_year
sum_retail_sales_fact_gross_profit_dollar_amount
Variables
background
title

15) Drag and drop these fields from Document Structure Pane to Report Window.

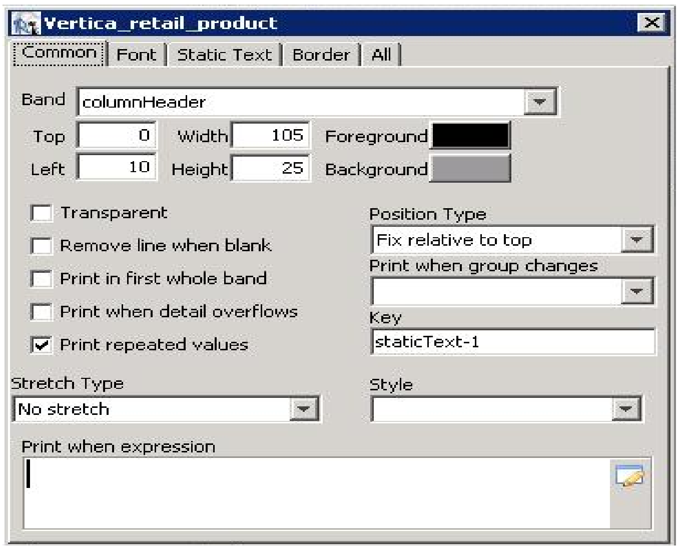


16) For entering Static Text in the Colum headings click "T" icon. Then make a box of a particular size in the Report window and right click that box to set the properties.





17) The window below is the Properties window to change the properties of the static text box.



Vertica_retail_product

Common | Font | Static Text | Border | All

Band: columnHeader

Top: 0 Width: 105 Foreground

Left: 10 Height: 25 Background

- [ ] Transparent
- [ ] Remove line when blank
- [ ] Print in first whole band
- [ ] Print when detail overflows
- [x] Print repeated values

Position Type: Fix relative to top

Print when group changes:

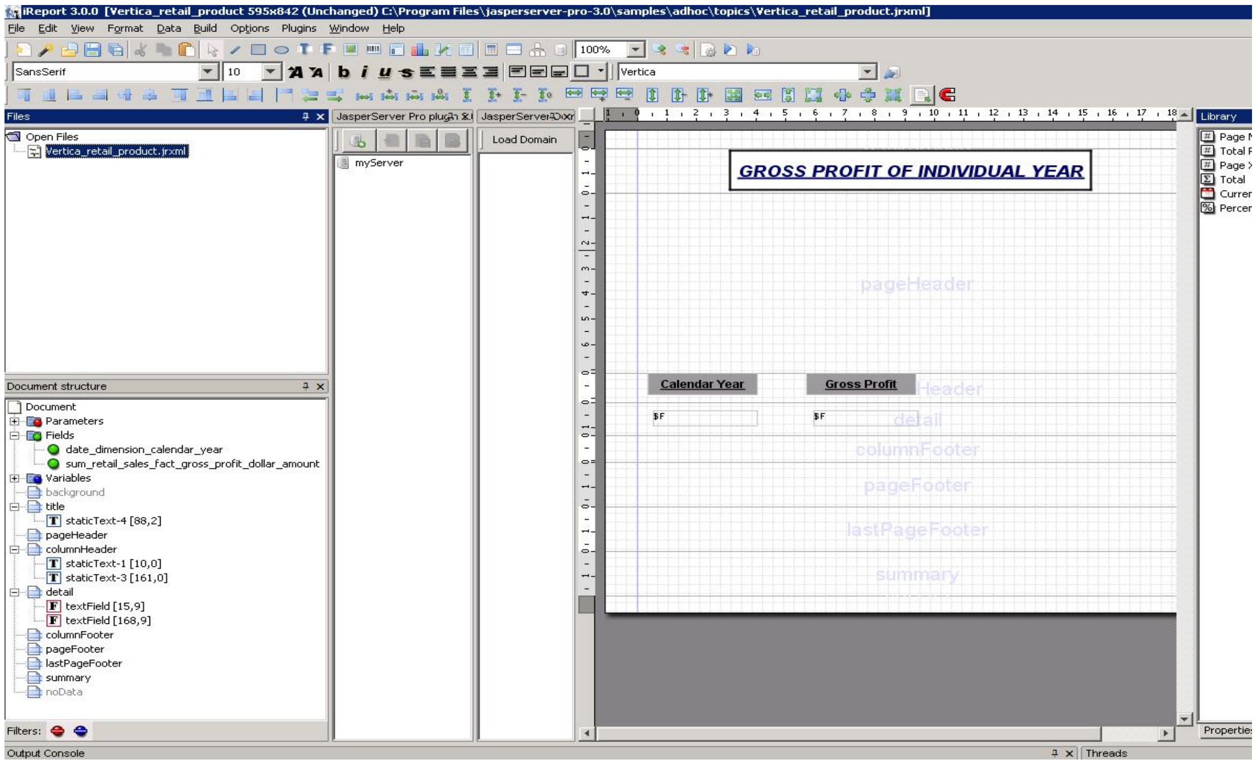
Key: staticText-1

Stretch Type: No stretch

Style:

Print when expression:

18) Using the above 2 steps put the columns heading and title of the Report as shown in the figure below.



iReport 3.0.0 [Vertica_retail_product 595x842 (Unchanged) C:\Program Files\jasperserver-pro-3.0\samples\adhoc\topics\Vertica_retail_product.jrxml]

File Edit View Format Data Build Options Plugins Window Help

Files: Open Files — Vertica_retail_product.jrxml

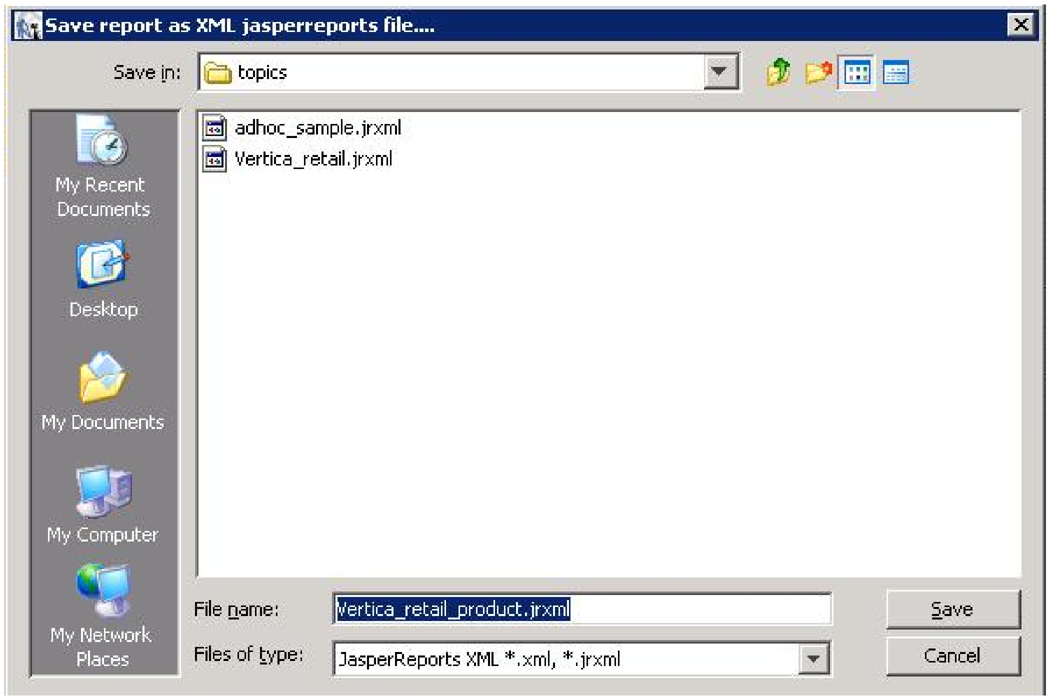
Document structure:

```
Document
  Parameters
  Fields
    date_dimension_calendar_year
    sum_retail_sales_fact_gross_profit_dollar_amount
  Variables
  background
  title
    staticText-4 [88,2]
  pageHeader
  columnHeader
    staticText-1 [10,0]
    staticText-3 [161,0]
  detail
    textField [15,9]
    textField [168,9]
  columnFooter
  pageFooter
  lastPageFooter
  summary
  noData
```

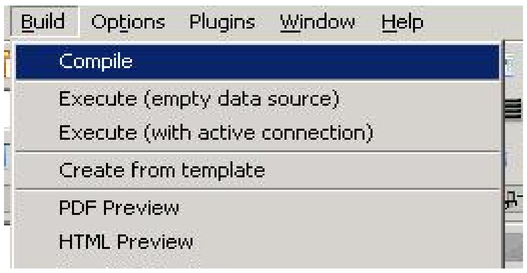
GROSS PROFIT OF INDIVIDUAL YEAR

Calendar Year	Gross Profit
$F	$F

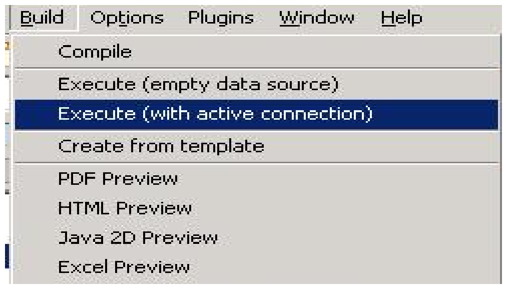
19) Save the report by selecting a particular path as shown below.



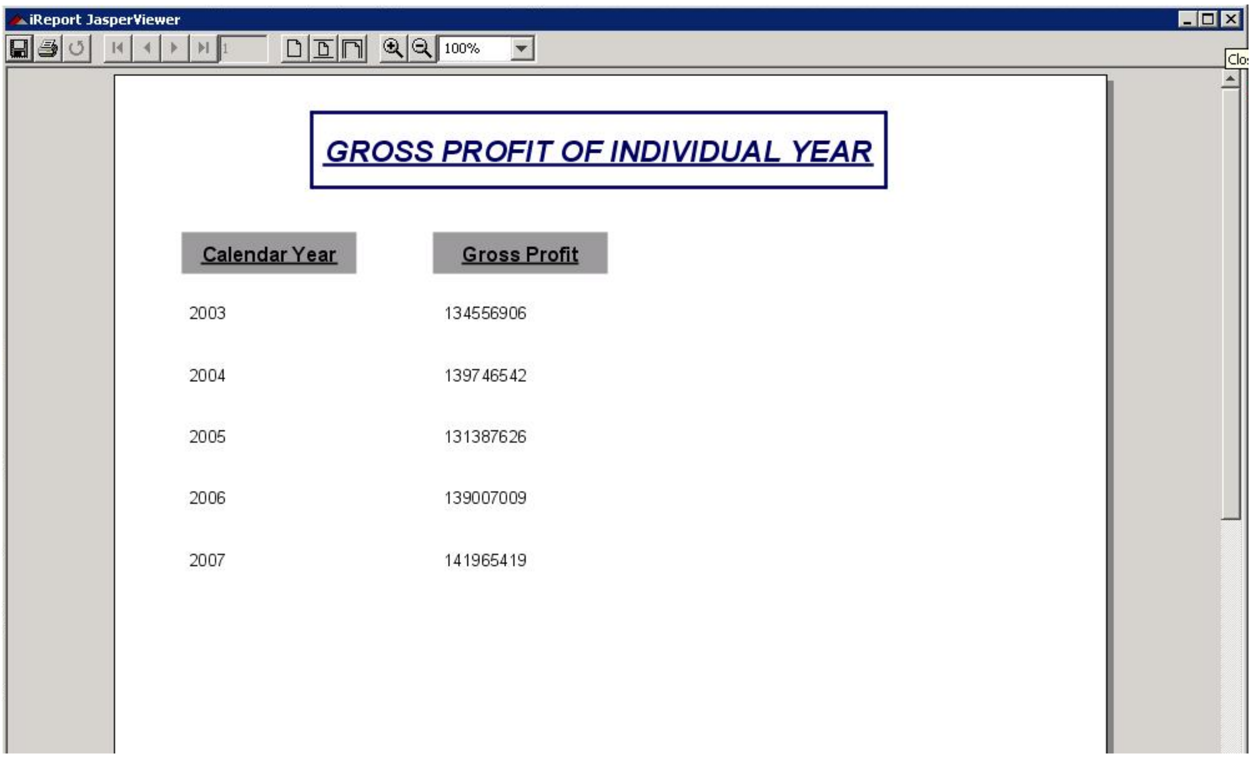
20)To Compile the Report Go to Build - > Compile.



21)Then to run the report with the active connection Go to Build -> Execute(with active connection)



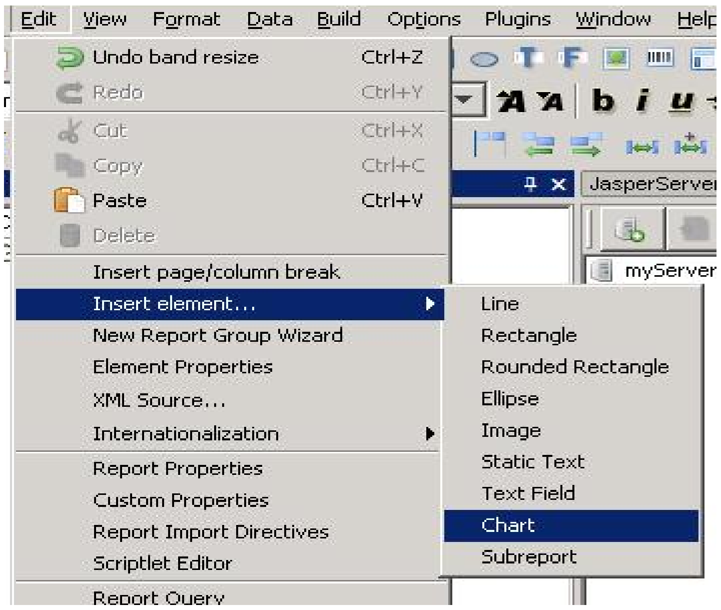
22) After executing the reports we get iReport Jasper Viewer given below.


iReport JasperViewer

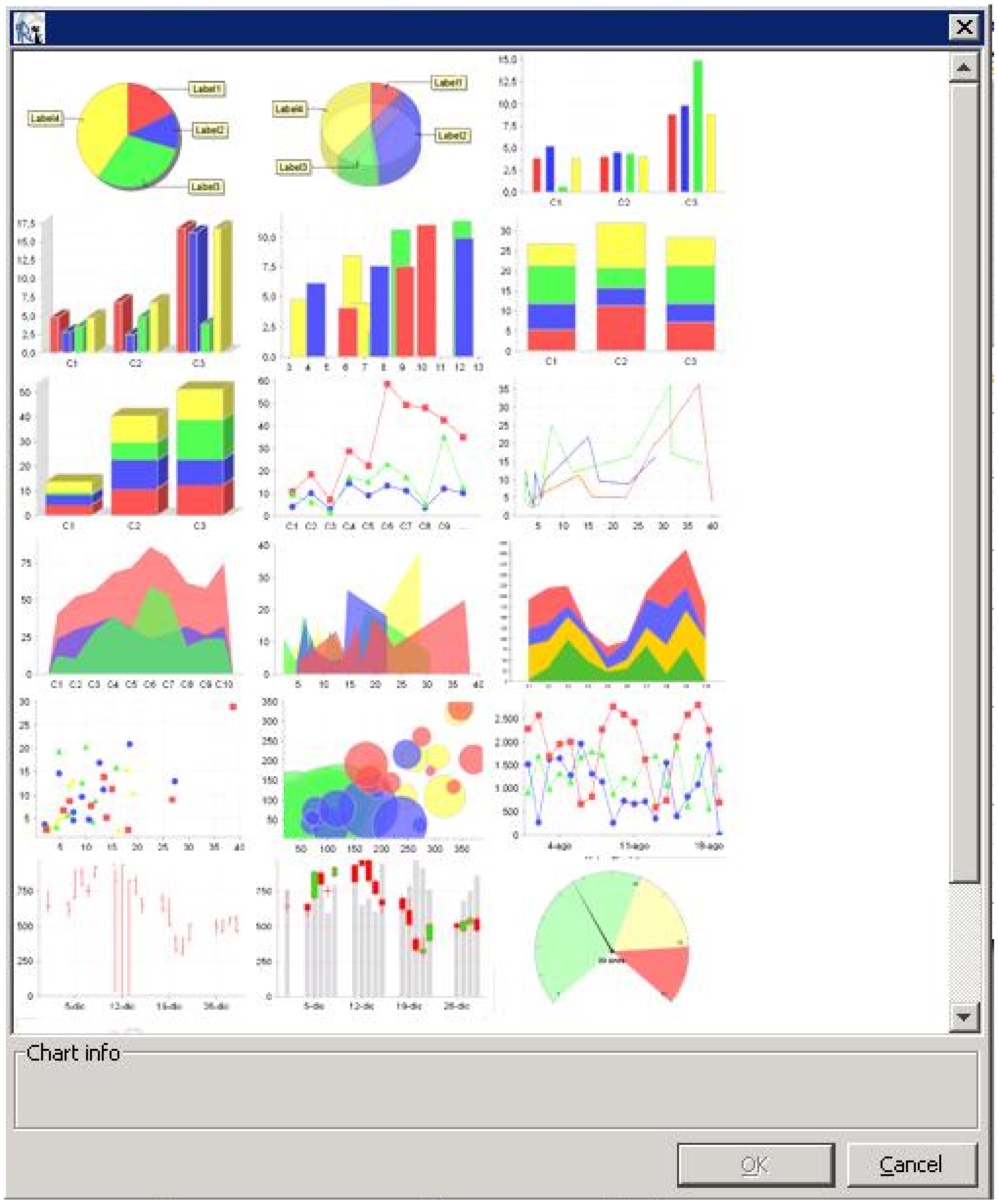
GROSS PROFIT OF INDIVIDUAL YEAR

Calendar Year	Gross Profit
2003	134556906
2004	139746542
2005	131387626
2006	139007009
2007	141965419

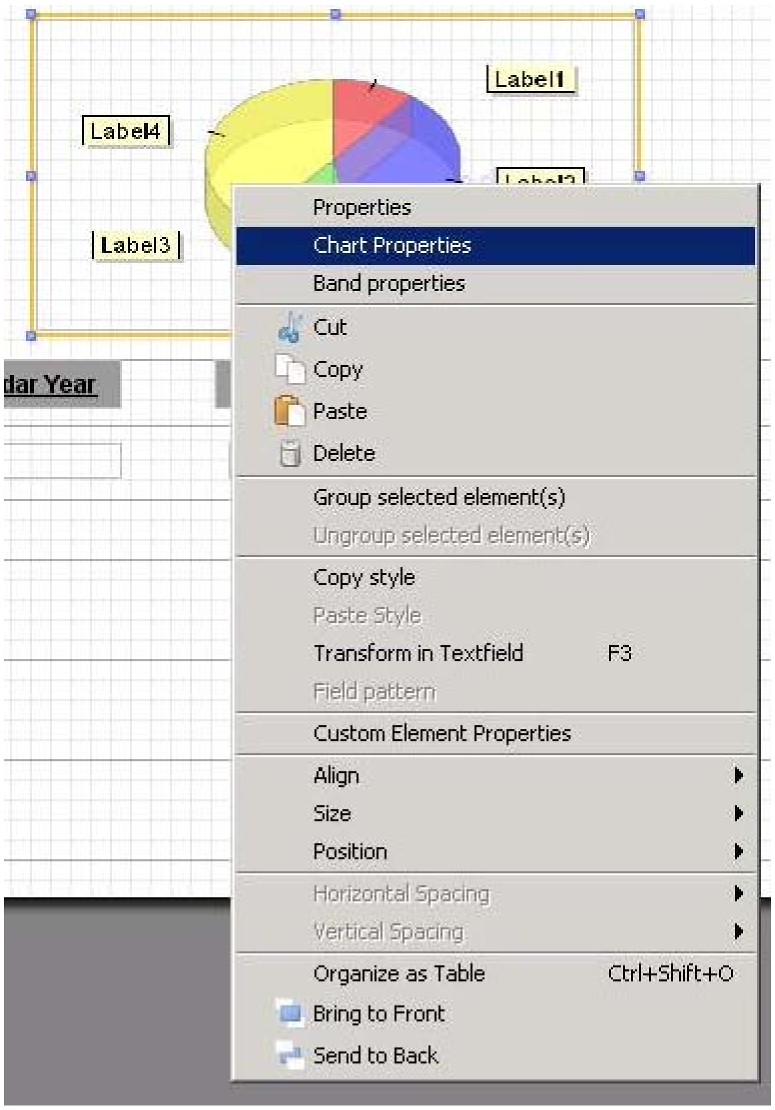
23) To include charts in the report go to Edit -> Insert Element -> Chart as shown below.



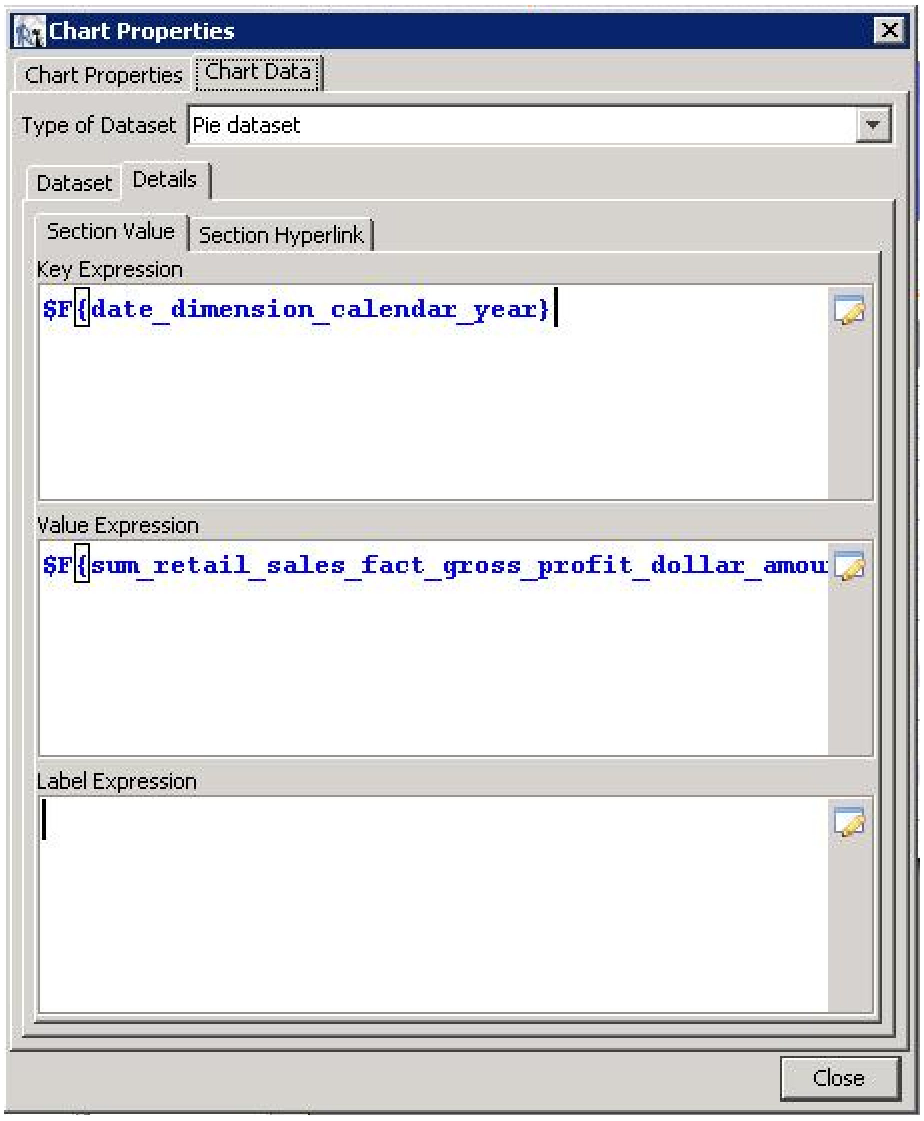
24)Go to Report window and click anywhere. It will open the window given below. Choose the template of the chart and Click “OK”.



25) Right click on the chart in the report window and click Chart Properties to set the properties of the chart.



26) A new window of properties open shown below. In Chart Data Tab, set the key expression and value expression for the chart.



Type of Dataset: Pie dataset

Key Expression

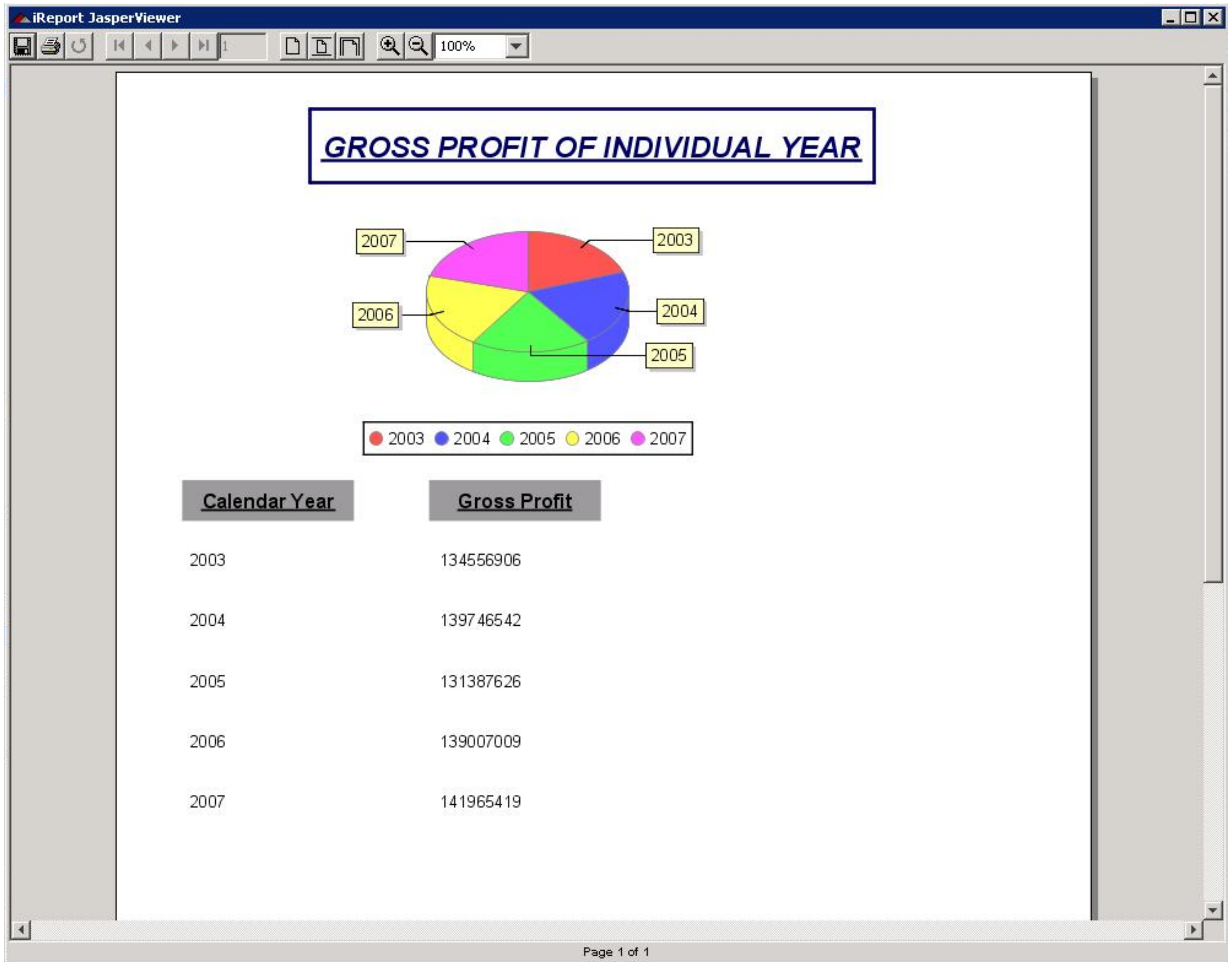
```
$F{date_dimension_calendar_year}
```

Value Expression

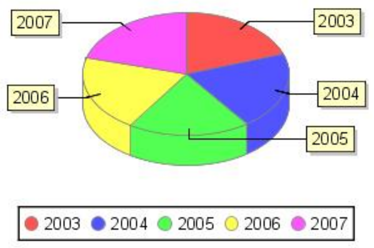
```
$F{sum_retail_sales_fact_gross_profit_dollar_amou
```

Label Expression

27) After setting the properties of the chart again compile and execute the report to get the final report as shown below. We can save these reports in various file formats by clicking save button.



GROSS PROFIT OF INDIVIDUAL YEAR



Calendar Year	Gross Profit
2003	134556906
2004	139746542
2005	131387626
2006	139007009
2007	141965419

4. Vertica Analytic Database on SAN

This section following document provides important information on how to configure Storage Area Network (SAN) for use with the Vertica Analytic Database. It also provides insight on how it may directly impact application performance and how to best setup and configure this scenario for optimum utilization.

4.1 Shared Nothing Architecture

The Vertica Analytic Database is optimized for share nothing configurations. The database is deployed on a cluster of industry standard servers connected via Gigabit Ethernet. Each server in the cluster manages its own persistent data and does not rely on shared disks in order to communicate with other servers in the cluster. When loading data or executing queries, Vertica distributes each operation so as to maximize parallelism on the various cluster servers. Each server in an N-server system manages 1/Nth of the data. In a Highly Available (HA) configuration, each server manages 2/Nths of the data, including a primary copy and a different secondary copy. The HA configuration (known as K-safety) accommodates up to K server failures in the cluster without any downtime or the need to restore from backup.

4.2 Direct Attached Storage vs. Storage Area Network

When considering a Vertica deployment a critical decision is whether to use local direct attached storage - either in a paired blade or pizza box configuration - or if there is a requirement to store the data on a SAN. For local storage configurations, follow the "Vertica Machine Requirements and Config" document. The remainder of this document describes considerations and guidelines for deploying Vertica on a SAN.

4.3 Storage Area Network Configuration Guidelines

The primary factor when using a SAN as the storage location for a Vertica deployment is physical contention for spindles. Vertica strongly recommends using a dedicated SAN or a dedicated segment of a restricted SAN. Aside from contention for disk, Vertica performance will be limited by available cache, I/O and network traffic in any shared configuration.

Each server in the Vertica cluster should mount its own LUN backed by dedicated physical drives. Sharing drives across LUNs can lead to hotspots since all servers issue multiple concurrent I/O requests. Depending on the expected user load, each LUN should be striped (RAID 0) with a minimum of eight disks. Stripes should be composed of as large a chunk size as possible – ideally 1mb per disk as described in the Machine Requirements document. When executing a query each server issues multiple simultaneous I/O requests for 1mb blocks of data, processing the results in a pipelined fashion. Running 30 concurrent queries will incur as many as 300 simultaneous requests. The response time for each query is principally gated by the latency incurred in the disk subsystem.

While latency is the primary concern when configuring a SAN, throughput is the gating factor in total execution time. Vertica is typically CPU bound while running both loads and queries so long as disk throughput is adequate. When processing data on a SAN, Vertica should be connected via dual 4gbps HBAs running in parallel to prevent a network bottleneck.

4.4 Storage Area Network Diagram

The following diagram illustrates the recommended deployment model for connecting Vertica to a SAN.

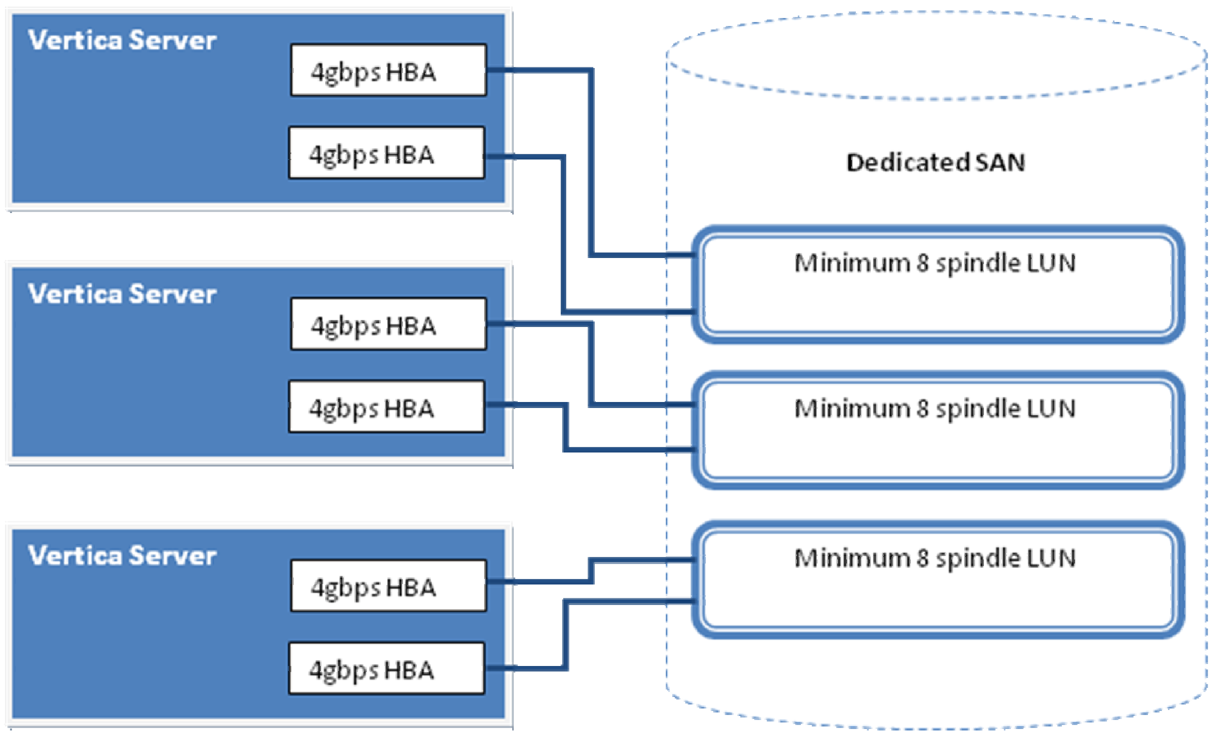


Figure 6

For specific questions about configuring a SAN for your Vertica installation or for more information about deploying Vertica, contact Vertica Field Engineering at http://www.vertica.com/support.

5. SAN Storage Configuration

This section provides general technical guidance to database administrators, system administrators, storage management personnel, and Linux IT professionals regarding deployment and maintenance of the Vertica® Analytic Database configured with SAN-based storage systems.

This section includes:

- Vertica minimum requirements for the storage subsystem;
- A characterization of the Vertica I/O pattern, enabling IT engineers to apply storage system vendors' recommendations in the context of Vertica requirements; and
- Recommended tools and methods of validating the storage system configuration.

5.1 Vertica Database Cluster with SAN Storage

The example below outlines the recommended connectivity within an environment that includes a distributed 4 node Vertica database cluster and a generic SAN storage system. The minimum number of nodes in the Vertica cluster is 3. The connectivity pattern should be repeated in the case of a larger Vertica cluster. The maximum number of cluster nodes is determined by the capacity of the FC switching equipment and SAN units, and not by intrinsic restrictions in the Vertica product design.

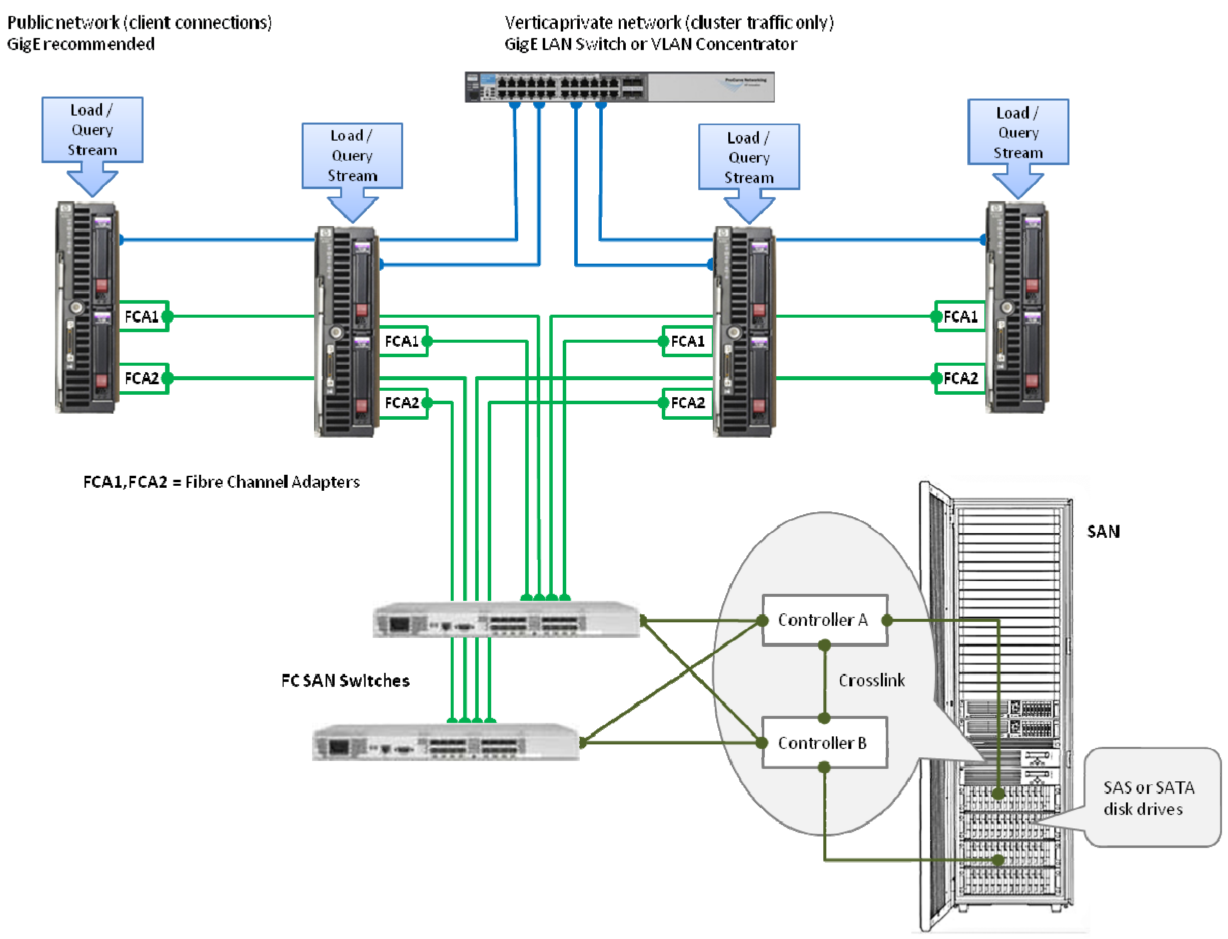


Figure 7

Each system representing a node in the Vertica cluster has two FC HBAs that provide two FC connections to the SAN array via two FC SAN switches. The modern SAN storage arrays are typically quipped with at least 2 controllers, which are cross connected to each other and are also attached to each of the FC switches.

The logical volumes (LUNs) are allocated on the SAN unit and presented to its own Linux hosts. Per the above connectivity chart, each host would have a total of 8 FC connection paths to its LUNs across 2 HBAs, 2 switches, and 2 controllers.

This configuration is consistent with commonly accepted practices of attaching SAN storage. It provides the opportunity for High Performance storage access by spreading out the I/O across all available resources while offering redundancy to satisfy High Availability requirements.

5.2 Vertica I/O Profile Characterization

Vertica is purposely designed to shift the burden of data processing from the storage tier to the computing tier, i.e. CPU and RAM. Vertica dramatically minimizes the amount of file I/O by operating only on columns referenced by the query. Vertica further reduces the amount of I/O by storing the data in a highly compressed form, only possible with a columnar structure.

During query execution Vertica is primarily CPU bound, rarely requiring more than 150MB/s read I/O bandwidth per query per each cluster node.

During direct data load, the Vertica database goes through several internal phases, some of which may be constrained by I/O bandwidth. The Vertica database server may drive 150 MB/s or more of write I/O throughput.

Vertica's I/O pattern is dominated by sequential 1 MB block reads and writes.

Vertica requires only a moderate sustainable level of IOPS (input/output operations per second).

Vertica clustering is based on the shared-nothing model. Vertica does not require nor need a shared/clustered file system.

5.3 SAN Performance Requirements and Configuration Recommendations

1. Each node of the Vertica database cluster needs 2 top level directories to accommodate its segment of the database – a directory for the database catalog files and a separate directory for the database data files. While not technically a requirement, we strongly recommend placing the catalog and data files on separate LUNs.
2. Per above, the total number of LUNs allocated on the storage array would be 2 x [the number of nodes], i.e. 1 catalog and 1 data LUN per node. For catalog and database data space allocation guidance refer to the Vertica documentation.
3. The LUNs must be presented to each Linux host so that the paths to the catalog and data on each host are identical.
4. Vertica requires a **minimum** of **150 MB/s read and 150 MB/s write throughput** on each node, in full **duplex** (i.e. reading and writing at 150 MB/s simultaneously), concurrently on all nodes of the cluster.
5. Vertica recommends a minimum of 2 FC HBAs on each node of the cluster, with each port rated at **2Gb** or better. A single dual port FC HBA may be acceptable.
6. Benchmark data suggests that Vertica I/O performance benefits from larger RAID stripe sizes. Setting stripe size (or even stripe unit size, if available) to 1 MB may yield additional performance gains on some SAN arrays.

5.4 SAN Configuration Considerations

Traditionally, the SAN array performance tuning focuses on two mutually exclusive strategies – contention management vs. workload distribution.

More recently, however, the manufacturers have been claiming that the enterprise level storage arrays, particularly high end models, have sufficient intelligence in the array management software to offer a balance between the two strategies, often by being able to determine the type of workload presented to the unit.

We have seen sufficient evidence to agree that given Vertica's predictable and consistent I/O pattern, it may be best to avoid attempts to manage perceived contention at the array by overwriting manufacturer's recommendations, as noted in the section Impact of Disk Group Allocations.

While the focus of SAN configuration optimization may not necessarily be the storage array itself, there is substantial opportunity for misconfiguration en route between the host and the storage array, e.g. in the software and drivers on the host side and in the FC switched fabric (switches/controllers/physical FC connections).

Vertica recommends validating that each of the cluster nodes receives the minimum required I/O bandwidth by using the SAN Configuration Verification Tool. This test must be run **concurrently** on all cluster nodes.

5.4.1 Host Side Driver and Multipathing Software

Selection and proper configuration of a Linux side driver and multipathing software is one of the critical factors in delivering the required I/O bandwidth for the host.

The Linux FC driver and the optional multipathing software may be available from a variety of sources, particularly from:

- the storage array manufacturer (e.g. HP, EMC, Hitachi, SUN, etc.);
- the host system manufacturer (e.g. HP, Dell, SUN, IBM, etc.);
- the HBA manufacturer (e.g. QLogic, Brocade, Emulex, etc.);
- a Linux kernel distribution.

The wide selection of configuration choices may lead to confusion at times. In our experience the best overall results are achieved when storage administrators follow good practices provided by storage array manufacturers.

Some device drivers may offer built-in multipath capabilities, which have to be carefully managed. For instance, if the driver implements the failover functionality, it may present all FC paths as a single device, thus denying the opportunity to load balance I/O traffic across all available paths. Each host may be independently choosing the same primary connection path to its LUNs, resulting in all I/O concentrating on a single FC connection rather than being spread across available FC paths. Consequently, the multipath/failover option in the driver becomes counterproductive in a bandwidth focused environment and should be disabled. When the multipath/failover option is disabled, each path is presented to the Linux host as a separate block device, thus allowing all FC paths to the LUN to be managed by the optional multipathing software.

The multipathing software (such as HP Device Mapper (DM) Multipath Software, EMC PowerPath, etc.) enables automatic load balancing and explicit path management (via device "blacklist", for instance). These capabilities are typically required to achieve the best results with SAN storage.

5.4.2 Impact of Disk Group Allocations

When configuring a Vertica cluster with storage arrays, follow good practices of disk group allocations from the array manufacturer. Manufacturers often suggest creating a single large disk group and allow the array's software to manage striping.

With Vertica's shared-nothing clustering model the goal may intuitively be towards segregating resources attached to individual nodes. We are aware of situations when attempts to manage perceived contention at the array's spindle level by allocating separate disk groups led to the overall inferior Vertica database performance.

5.4.3 Impact of RAID Type

Vertica field studies show that enterprise level SAN arrays, when configured as RAID0, RAID10, or RAID5 yield essentially similar Vertica performance levels.
While use of RAID5 as database storage is often considered a controversial topic, the benchmark data shows that RAID5 is a viable, perhaps even a preferred storage configuration option for Vertica because of RAID5's excellent storage utilization characteristics.

Vertica cluster performance is practically not impacted with a failure of a physical drive. Both queries and loads may continue while the storage that suffered a failure is operating in a degraded configuration.

A sufficient amount of at Battery Backed Write Cache (BBWC/NVRAM), 1GB or more, should be considered a minimum requirement for RAID5 configuration.

5.5 SAN Configuration Verification Tool

After the SAN storage is configured and the required catalog and data LUNs are presented to each Linux host of the Vertica cluster, a standalone utility that emulates Vertica I/O patterns may be used to validate that each of the cluster nodes receives the minimum required I/O bandwidth.

This utility performs read, write, and re-write tests and requires a minimum amount of free disk space equal to 4 times the amount of RAM.

The test must be run concurrently on all nodes of the Vertica cluster.

Please contact your Vertica sales consultant for the download, installation and configuration instructions.

6. Vertica Cluster Network Configuration

When preparing a cluster for a Vertica installation it is important to review the following checklist to ensure compatibility and avoid issues creating a database. You will need to have root access to your cluster to change most of these configurations. If you do not have root access, please review this document with your IT administrator.

6.1 IP Configuration

Check the IP configuration on each node in your cluster using the `/sbin/ifconfig` command. You will see output similar to the following:

```
# /sbin/ifconfig
eth0      Link encap:Ethernet  HWaddr 00:0C:29:52:37:63
          inet addr:192.168.163.128  Bcast:192.168.163.255  Mask:255.255.255.0
          inet6 addr: fe80::20c:29ff:fe52:3763/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:14418 errors:0 dropped:0 overruns:0 frame:0
          TX packets:4069 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:14106172 (13.4 MiB)  TX bytes:359902 (351.4 KiB)
          Interrupt:177 Base address:0x1400

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:3926 errors:0 dropped:0 overruns:0 frame:0
          TX packets:3926 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:380270 (371.3 KiB)  TX bytes:380270 (371.3 KiB)
```

Pay attention to eth0 or similarly named interfaces. Check that the **inet addr** on line two has a similar prefix on all nodes in your cluster. The **Bcast** and **Mask** must be identical on all nodes in the cluster. Vertica requires that all nodes in a cluster be on the same subnet. You may consult with your IT administrator if you are not sure whether all nodes are on the same subnet.

6.2 Routing

Print the routing table with the 'route' command and check the default route. For example, if you are running on a 192.168.163.x network your route table may appear as follows:

```
# route
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
```

```
192.168.163.0   *               255.255.255.0   U     0      0        0 eth0
default         192.168.163.2   0.0.0.0         UG    0      0        0 eth0
```

If you have multiple interfaces you may see other entries besides eth0 in the last column. Make sure that the default route for all machines in your cluster is the same and that the destination and genmask for the '*' gateway is the same.

6.3 Hostnames

Hostnames must be set to non-local host (not the 127.0.0.1 address) and consistent across the cluster. Your hostnames may be configured via DNS or using /etc/hosts. If you are using DNS, please consult your IT administrator. If you are using /etc/hosts, it should look like the following:

```
127.0.0.1             localhost.localdomain localhost
192.168.163.128         node01.domain.com node01
192.168.163.129         node02.domain.com      node02
192.168.163.130         node03.domain.com      node03
```

This configuration names three nodes with their corresponding ip addresses.

6.4 SSH

The database administrator user must be able to ssh from each node in the cluster to every other node using the hostname and without being prompted for a password. To test this in our example cluster log in as the database administrator on node01 and ssh to both node02 and node03, ssh-ing back to node01 from each. Repeat this process on node02, ssh-ing to node3. If you find that ssh is not setup correctly please consult the Vertica documentation or use the db-ssh-config tool setup your database administrator user.

6.5 Open Sockets

Look for open sockets on each node in the cluster, in particular look for servers listening on ports 4803 (the spread port) and 5433 (the Vertica Database port). You can use the following command:

```
# netstat -nap | grep -e 4803 -e 5433
```

Before installing Vertica the command should have no output indicating that both ports are available. After installing Vertica and before creating a database, you should see spread bound to port 4803:

```
tcp     0    0 0.0.0.0:4803  0.0.0.0:* LISTEN      2206/spread
udp     0    0 0.0.0.0:4803  0.0.0.0:*             2206/spread
```

After creating a Vertica database, you should see both spread bound to port 4803 and Vertica bound to port 5433:

```
tcp     0    0 0.0.0.0:5433  0.0.0.0:* LISTEN      2840/vertica
tcp     0    0 :::5433       :::*      LISTEN      2840/vertica
tcp     0    0 0.0.0.0:4803  0.0.0.0:* LISTEN      2206/spread
udp     0    0 0.0.0.0:4803  0.0.0.0:*            2206/spread
```

6.6 IP Tables

You may configure IP Tables to allow ports 4803 and 5433 as well as higher level ports though Vertica recommends disabling it completely. Check whether iptables is running by issuing the following command:

```
# /etc/init.d/iptables status
Firewall is stopped.
```

If iptables is not stopped you may stop it by issuing the command

```
# /etc/init.d/iptables stop
Firewall is stopped.
```

And remove it from the appropriate run levels in `/etc/rc.d.` Check with your IT administrator before making changes to your system startup scripts.

6.7 Other Firewalls

Other firewalls may be present. You should review your Linux installation guide for information on how to disable any additional system firewalls that are present.

6.8 SELinux and Security Levels

Most Linux systems offer a feature called Secure Linux or SELinux. This feature offers fine grained control over system security. You may configure SELinux to allow Vertica access to the network though Vertica recommends disabling it. On RedHat Enterprise Linux you can configure SELinux using the following command:

```
# /usr/bin/system-config-securitylevel-tui
```

Follow the instructions to disable the Security Level and SELinux.

6.9 Network Configuration

Vertica recommends that you run your database cluster on a private subnet with a dedicated switch to prevent any network contention and issues with custom network configurations. If this is not possible, review the Vertica installation and quickstart procedure with your IT administrator and check for dropped or delayed packets, spread errors or warnings in the vertica.log or problems creating a database.

Vertica supports systems that have multiple network cards in a number of configurations. The most common configurations are:

- Private subnets which do not use host name resolution.
- Configurations where each IP address is given a unique host name.
- Configurations where all IP addresses on a given system share a host name.

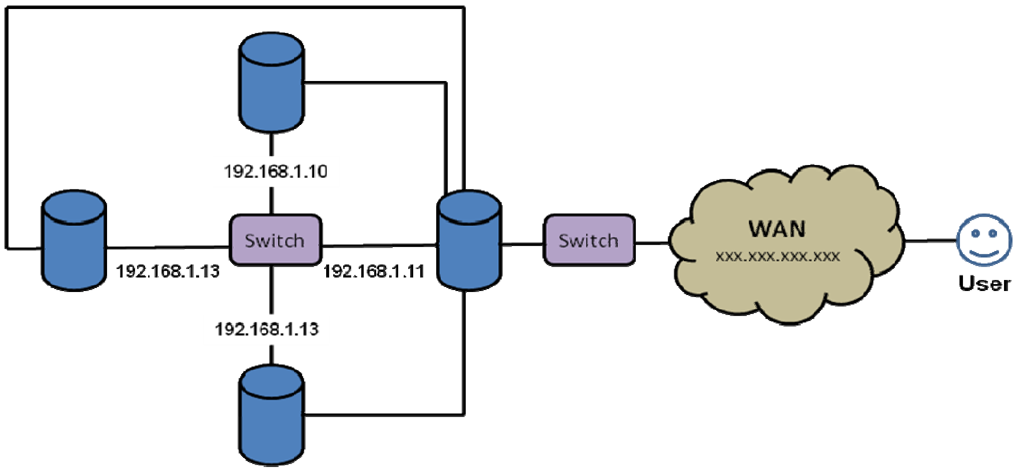


Figure 8: **Sample 4 Node Cluster**

The illustration provided in Figure 4 is useful when discussing these types of network configurations.

6.9.1 Private subnets which do not use host name resolution

In the illustration provided, the private subnet is the network identified by the 192.168.1.255 network. In this configuration, each of the systems will be provided with a hostname corresponding to the ip address of the public network (xxx.xxx.xxx.xxx). There are no host names defined for the private network.

In this configuration, the -s option of the `install_vertica` script would consist of a series of 4 ip addresses.

For example:

```
  /opt/vertica/sbin/install_vertica -s
192.168.1.10,192.168.1.11,192.168.1.12,192.168.1.13 -r .....
```

6.9.2 Configurations where each IP address is given a unique host name

Again using the illustration provided, the private subnet is identified by the 192.168.1.255 subnet. In this configuration the private and public ip addresses are given unique hostnames. A sample /etc/hosts file might look like this:

```
bash#> cat /etc/hosts

# local loopback do not delete this line
127.0.0.1    localhost localhost.localdomain
# public network ip addresses.
10.10.10.10  host01.company.com host01
10.10.10.11  host02.company.com host02
10.10.10.12  host03.company.com host03
10.10.10.13  host04.company.com host04
# private network – all vertica comms go here
192.168.1.10      cluster01
192.168.1.11      cluster02
192.168.1.12      cluster03
192.168.1.13      cluster04
```

It should be noted that all possible network paths are described here. It is possible to reach the other members of the cluster with both the public and private hostnames. To ensure that vertica installs using the private network, the install line should include the hostnames of the private network. For example;

```
 /opt/vertica/sbin/install_vertica -s cluster01,cluster02,cluster03,cluster04 -r
...
```

6.9.3 Configurations where all IP addresses on a given system share a host name

Again using the illustration provided, the private subnet is identified by the 192.168.1.255 subnet. In this configuration the private and public ip addresses are given shared host names. A sample /etc/hosts file might look like this:

```
bash#> cat /etc/hosts

# local loopback do not delete this line
127.0.0.1    localhost localhost.localdomain

# public network ip addresses.
10.10.10.10  host01.company.com host01
```

```
# private network - all vertica comms go here
192.168.1.10      host01
192.168.1.11      host02
192.168.1.12      host03
192.168.1.13      host04
```

Notice that in this case only the local public ip address is provided. If you want to include the entire spectrum of ip addresses, be sure to put the private network first in /etc/hosts resolution as the vertica install utility will pick the first set of ip addresses that share a common subnet.

Each host in the cluster that is configured this way must have its /etc/host.conf file modified to turn on multi-home capabilities*. To do this, edit the /etc/host.conf file and add the line **multi on** to the file. You must also make sure that the order of resolution is **hosts,bind** in this file. For example;

```
bash#> cat /etc/host.conf
order hosts,bind
multi on
```

In addition, the /etc/nsswitch.conf will need to contain file resolution first.

To ensure that Vertica installs using the private network, the install line should include the hostnames of the private network. For example;

```
/opt/vertica/sbin/install_vertica -s host01,host02,host03,host04 -r ...
```

6.10 Network Troubleshooting

6.10.1 Issue: Install reports "Expected n heard from n-y"

This is caused by 2 possible errors. The most common error is that there is no route from one of the hosts in the cluster to another – probably because a firewall is still running. Make sure that the firewall has been turned off.

The second error is less common and results when a system is heavily loaded or limited in resources. Because the throughput test server may lag when starting up on heavily loaded systems, the client will occasionally try to connect before that server has started up. This is especially prevalent on VM images on laptops. To resolve, either wait for the system to quiet down or provide more resources.

6.10.2 Issue: In multi-home scenarios when a hostname is shared by many IP's, Vertica install chooses the wrong IP address to install to

The install script is able to identify the public subnet as a subnet common among all members of the cluster first. To resolve, modify the /etc/hosts file and place the private network addresses first.

7. High Availability Recovery

The Vertica Analytic Database has a unique approach to recovery that takes advantage of its distributed architecture. Every data value in every table of a K-Safe database is stored on multiple nodes. This form of distribution ensures that a complete and correct copy of the database is available, even if up to K nodes fail. A recovering node automatically recovers its lost objects by querying the other nodes in the cluster.

7.1 K-Safety

A Vertica database is said to be K-safe if one node can fail at any given time without causing the database to shut down. If a database having K=1 loses a node, the database continues to run normally. When the failed node returns and successfully recovers, it can participate in database operations again. However if more than one node fails, the system is no longer K-safe and will shut itself down to prevent inconsistency. The entire database must then be started again as described in "Restart the Database" section.

7.1.1 Requirements for K-Safety

Vertica supports a k-safety level of 1, meaning that only one node can go down regardless of the number of nodes in your cluster.

When creating projections with the Database Designer, projection definitions which meet k-safe design requirements will be recommended and marked with a k-safety level of 1. Notice the output from executing the optimized design script generated by the Database Designer.

A physical schema consists of a set of projections used to store data on disk. The projections in the physical schema are based on the objects in the Logical Schema.

A projection is a special case of a materialized view that provides physical storage for data. A projection can contain some or all of the columns of one or more tables. A projection that contains all of the columns of a table is called a superprojection. A projection that joins one or more tables is called a pre-join projection. Most projections are used for ad-hoc query processing and K-safety but it is possible to have query-specific projections.

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

A materialized view is similar to a standard SQL view with one major exception: the data is actually stored on disk rather than computed each time the view is used in a query. A materialized view, then, must be refreshed whenever the data in the underlying tables is changed. A projection is a special case of a materialized view.

```
Stock_Schema=> \i Stock_Schema_design_opt_1.sql
```

```
CREATE PROJECTION

CREATE PROJECTION

        ⋮

  mark_design_ksafe

----------------------

 Marked design 1-safe

(1 row)
```

If you have created your projections manually, executing `select mark_design_ksafe(1);` will either mark the system as 1-safe or will return an error which will indicate which projections do not have buddy projections and are not k-safe. For more information on buddy projections and how to add additional buddies using the db designer, refer to the Modifying Existing Database Design section in the Vertica Product Documentation, DB Administrator Guide.

7.1.2 Determining K-Safety

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

```
SELECT current_fault_tolerance FROM vt_system;
current_fault_tolerance
----------------
              1
(1 row)
```

7.1.3 Monitoring K-Safety

Monitoring tables can be accessed programmatically to enable external actions such as alerts. The K-Safety level can be monitored by polling the vt_system table column and checking the value.

For example, the following script will monitor the k-safety level of a Vertica cluster and email the appropriate person if it drops to 1:

```
#!/usr/bin/perl

$email = @ARGV[0];

sub sendEmail {
       my ($to, $from, $subject, $message) = @_;
       my $sendmail = '/usr/lib/sendmail';
       open(MAIL, "|$sendmail -oi -t") or die "Cannot open $sendmail: $!";
       print MAIL "To: $to\n";
       print MAIL "From: $from\n";
       print MAIL "Subject: $subject\n\n";
       print MAIL "$message\n";
       close(MAIL);
}


$ksafety = `vsql -c "select current_fault_tolerance from vt_system;" | awk '/[0-9]/
{print $1}'`;
while ($ksafety > 0) {
      sleep(1);
      $ksafety = `vsql -c "select current_fault_tolerance from vt_system;" | awk '/[0-9]/
{print $1}'`;
}
sendEmail($email,$email,'K-Safety Level 0','K-Safety Level 0');
```

7.1.4 Loss of K-Safety

Should you lose a node and your cluster's K-Safety level drops to 0, all DML operations will remain available. However performance will be impacted.

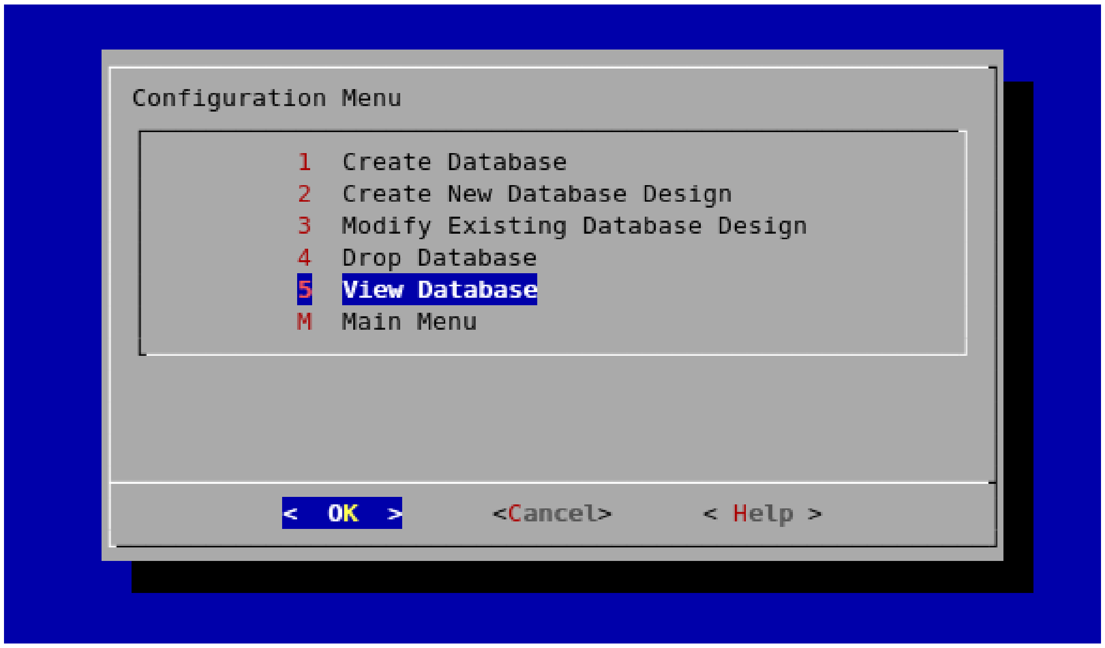
7.2 Catalog and Data Files

For the recovery process to complete successfully it is essential to ensure that catalog and data files are in the proper directories. In Vertica, the catalog is a set of files that contain information (metadata) about the objects in a database, such as the nodes, tables, constraints, and projections. The catalog files are replicated on all nodes in a cluster while the data files are unique to each node. They can be found in the following directories:

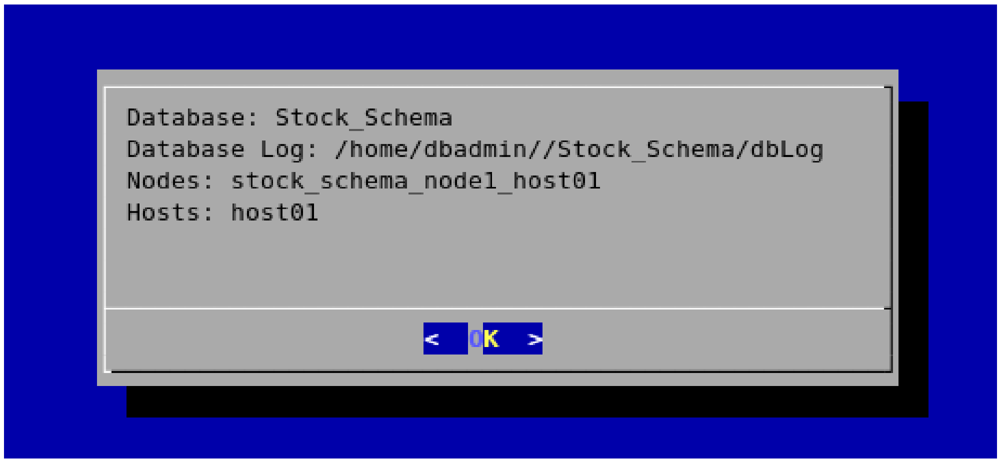
```
      /DATABASE_HOME_DIR/DATABASE_NAME/site<xx>_catalog/

      /DATABASE_HOME_DIR/DATABASE_NAME/site<xx>_data/
```

Where DATABASE_HOME_DIR is the path that can be found from the admintools "Configuration" -> "View Database" menu.

To view the path from adminTools, select "Configuration" from the main menu and then select "View Database".



Select the database that you would like to view and click “OK” to see the database profile.



```
Database: Stock_Schema
Database Log: /home/dbadmin//Stock_Schema/dbLog
Nodes: stock_schema_node1_host01
Hosts: host01
```

In this example, the catalog files are found in the following directory:

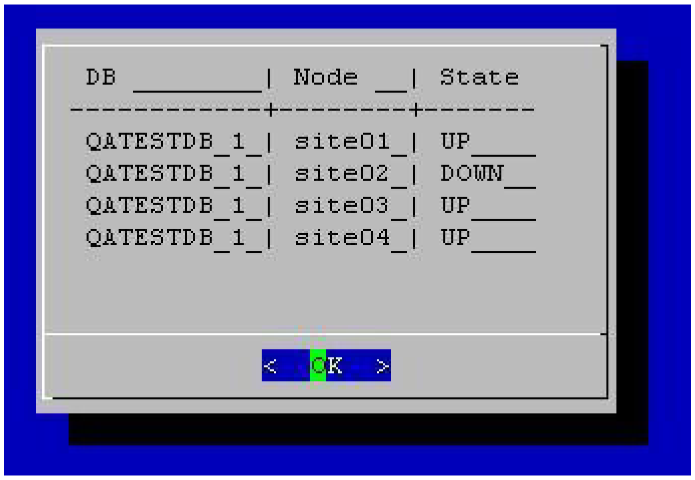
```
/home/dbadmin/Stock_Schema/stock_schema_node1_host01_catalog/
```

7.3 Recovery Scenarios

In the following sections we detail four recovery scenarios describing the steps to recovering or replacing a failed node.

7.3.1 Restarting a Node

When one node in a running database cluster fails or if any files from the catalog or data directories are lost from any one of the nodes the admin tools will reflect the status of down node in admin tools within a few minutes. This can be checked with the "View Database Custer State" in the admintools main menu.



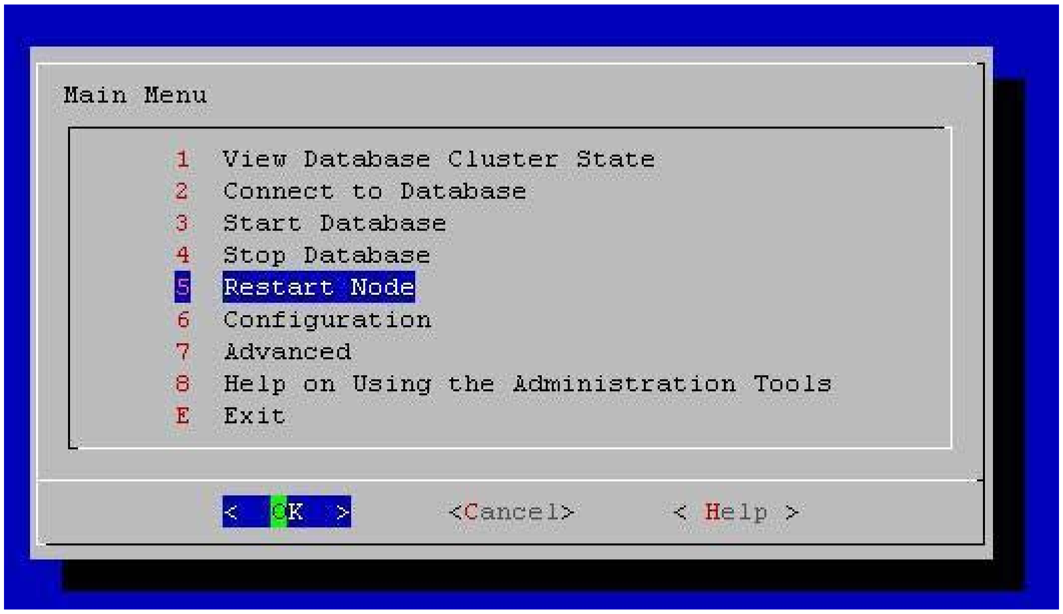
```
DB ________| Node __| State
------------+--------+-------
QATESTDB_1_| site01_| UP____
QATESTDB_1_| site02_| DOWN__
QATESTDB_1_| site03_| UP____
QATESTDB_1_| site04_| UP____
```

With one node of a K=1 cluster down the value of K is 0.

Note: The k-safety value is a property of the schema. Should a node fail, the new K value will not be reflected in the vt_system table.

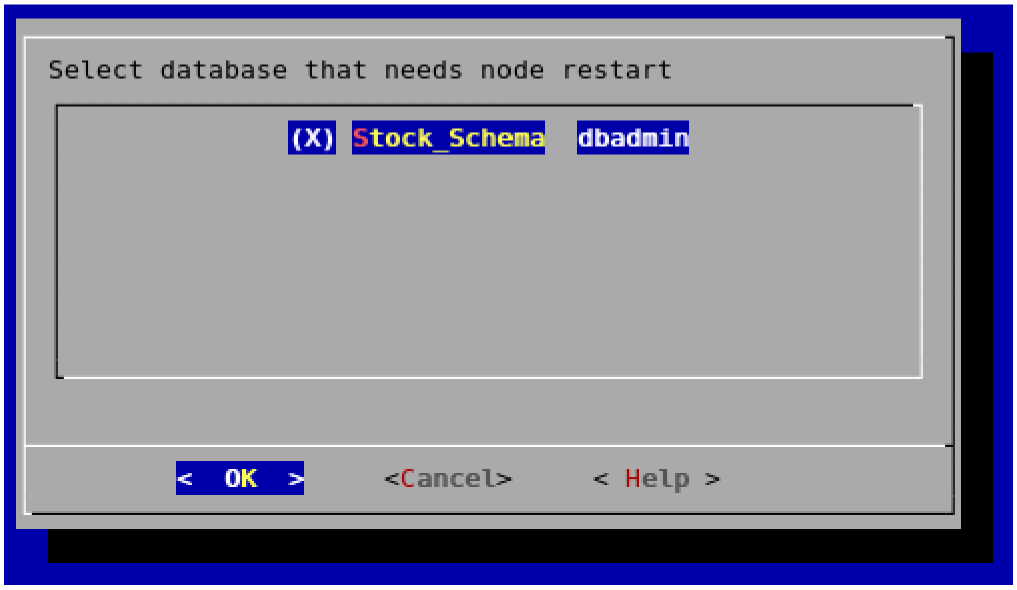
To begin the recovery process select "Restart Node" from admin tools as detailed in the steps below. This will recover the failed node.

1. Run admin tools and select “Restart Node”



```
Main Menu

        1  View Database Cluster State
        2  Connect to Database
        3  Start Database
        4  Stop Database
        5  Restart Node
        6  Configuration
        7  Advanced
        8  Help on Using the Administration Tools
        E  Exit

        <  OK  >      <Cancel>      < Help >
```

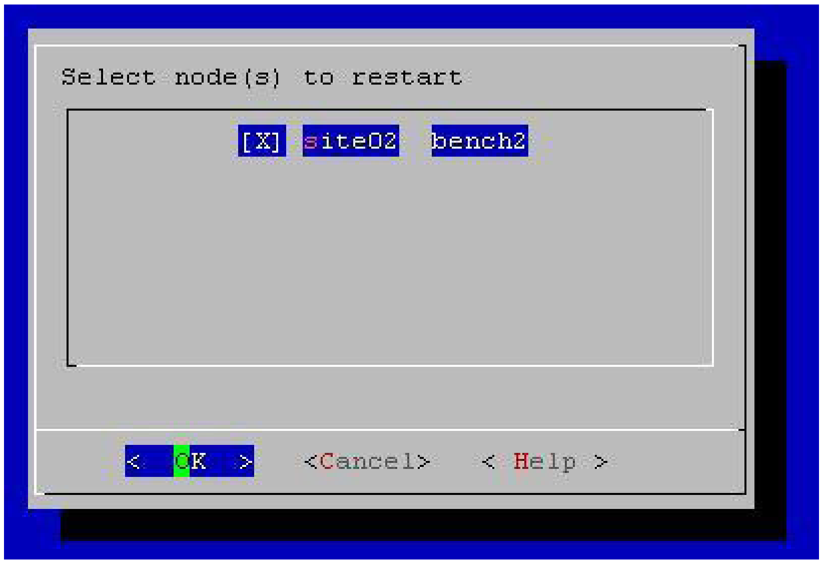
2. Select the database node that you wish to recover.



```
Select database that needs node restart

            (X) Stock_Schema  dbadmin

        <  OK  >      <Cancel>      < Help >
```

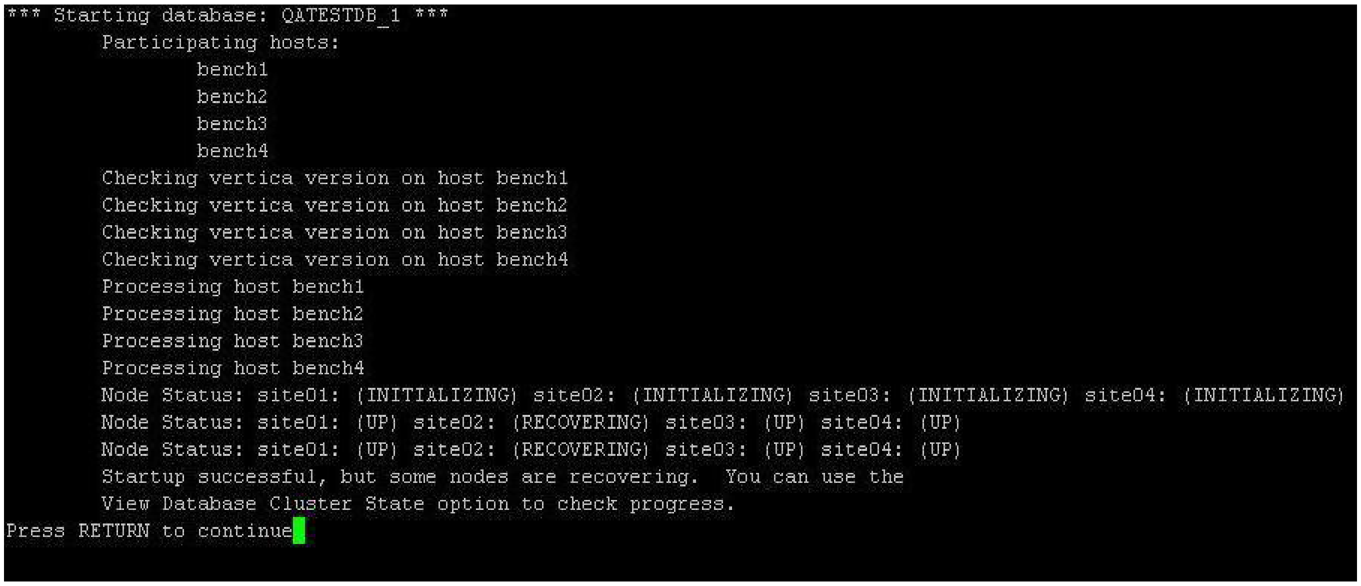
3. Select Node to be restarted.

Note: You may see additional nodes in the list because they are used internally by the admin tools.



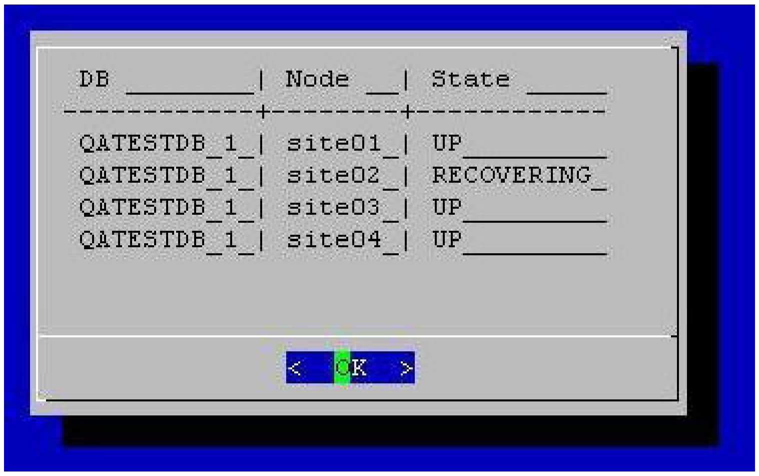
```
Select node(s) to restart

[X] site02  bench2

<  OK  >   <Cancel>   < Help >
```

4. The starting database message includes the recovery message.



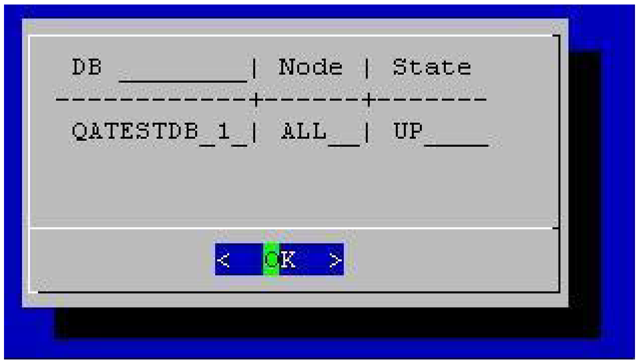
```
*** Starting database: QATESTDB_1 ***
        Participating hosts:
                bench1
                bench2
                bench3
                bench4
        Checking vertica version on host bench1
        Checking vertica version on host bench2
        Checking vertica version on host bench3
        Checking vertica version on host bench4
        Processing host bench1
        Processing host bench2
        Processing host bench3
        Processing host bench4
        Node Status: site01: (INITIALIZING) site02: (INITIALIZING) site03: (INITIALIZING) site04: (INITIALIZING)
        Node Status: site01: (UP) site02: (RECOVERING) site03: (UP) site04: (UP)
        Node Status: site01: (UP) site02: (RECOVERING) site03: (UP) site04: (UP)
        Startup successful, but some nodes are recovering.  You can use the
        View Database Cluster State option to check progress.
Press RETURN to continue
```

5. Recovery state can be viewed again by selecting "View Database Cluster State" from the main menu.



```
 DB __________| Node __| State _____
-------------+---------+------------
 QATESTDB_1_| site01_| UP_________
 QATESTDB_1_| site02_| RECOVERING_
 QATESTDB_1_| site03_| UP_________
 QATESTDB_1_| site04_| UP_________
```

6. After the database is fully recovered, the status can be checked again through "View Database Cluster State"



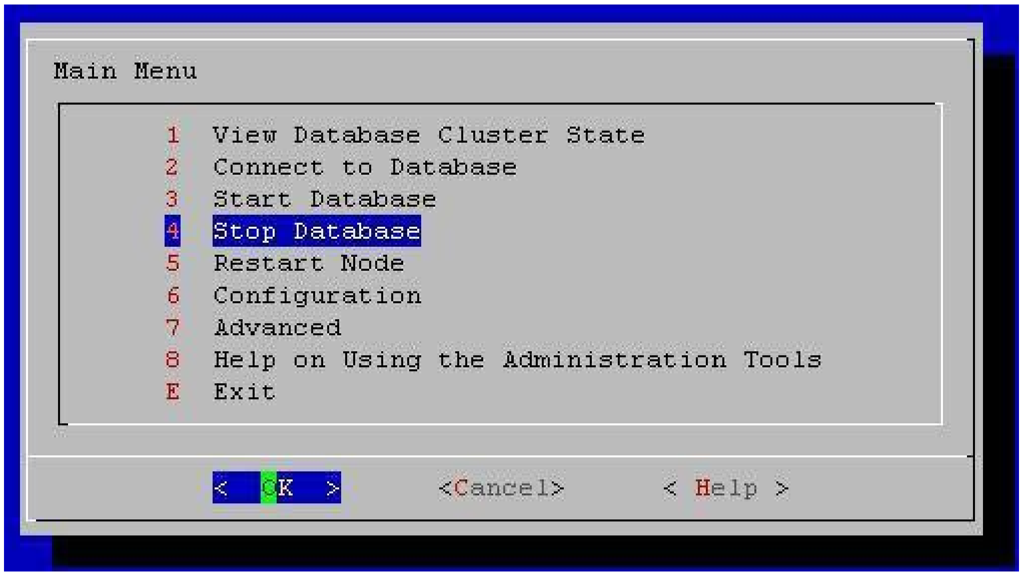
```
 DB __________| Node | State
-------------+------+-------
 QATESTDB_1_| ALL__| UP____
```

7.3.2 Restarting a Database

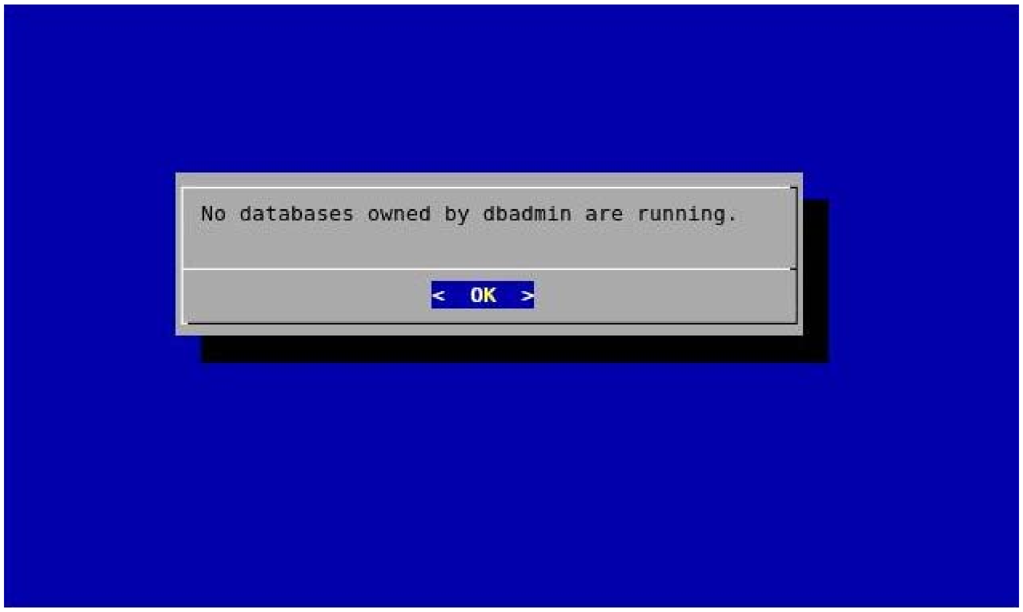
Should you lose the Vertica process on more than one node (for example, due to power loss), or if the servers are shut down without properly shutting down the Vertica database first, the database cluster will identify that it was not cleanly shutdown during startup. The database will automatically detect when the cluster was last in a consistent state and then shutdown. At this point it can be restarted by an administrator.

From the main menu in admintools:

1. Verify that the database has been stopped.

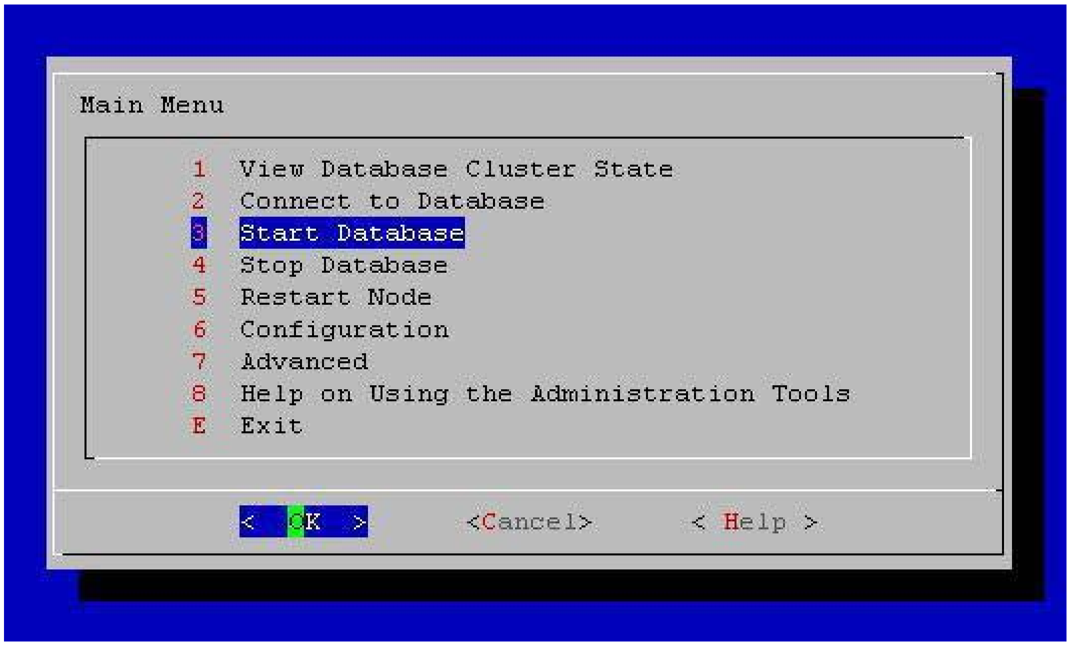


```
Main Menu

        1  View Database Cluster State
        2  Connect to Database
        3  Start Database
        4  Stop Database
        5  Restart Node
        6  Configuration
        7  Advanced
        8  Help on Using the Administration Tools
        E  Exit

        <  OK  >        <Cancel>        < Help >
```



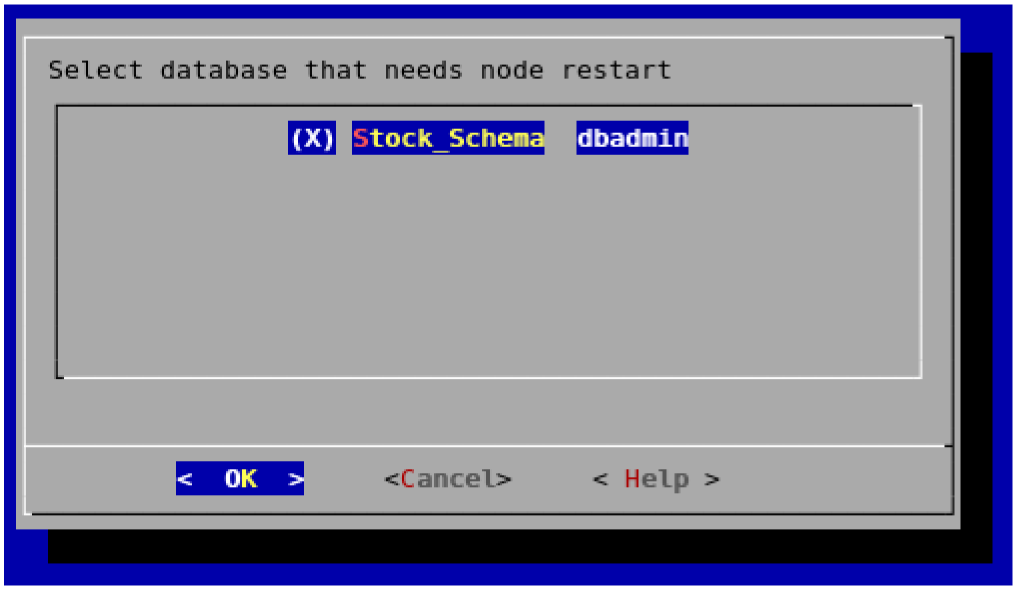
```
No databases owned by dbadmin are running.

                <  OK  >
```

2. Then start the database by selecting "Start Database" from the Main Menu.



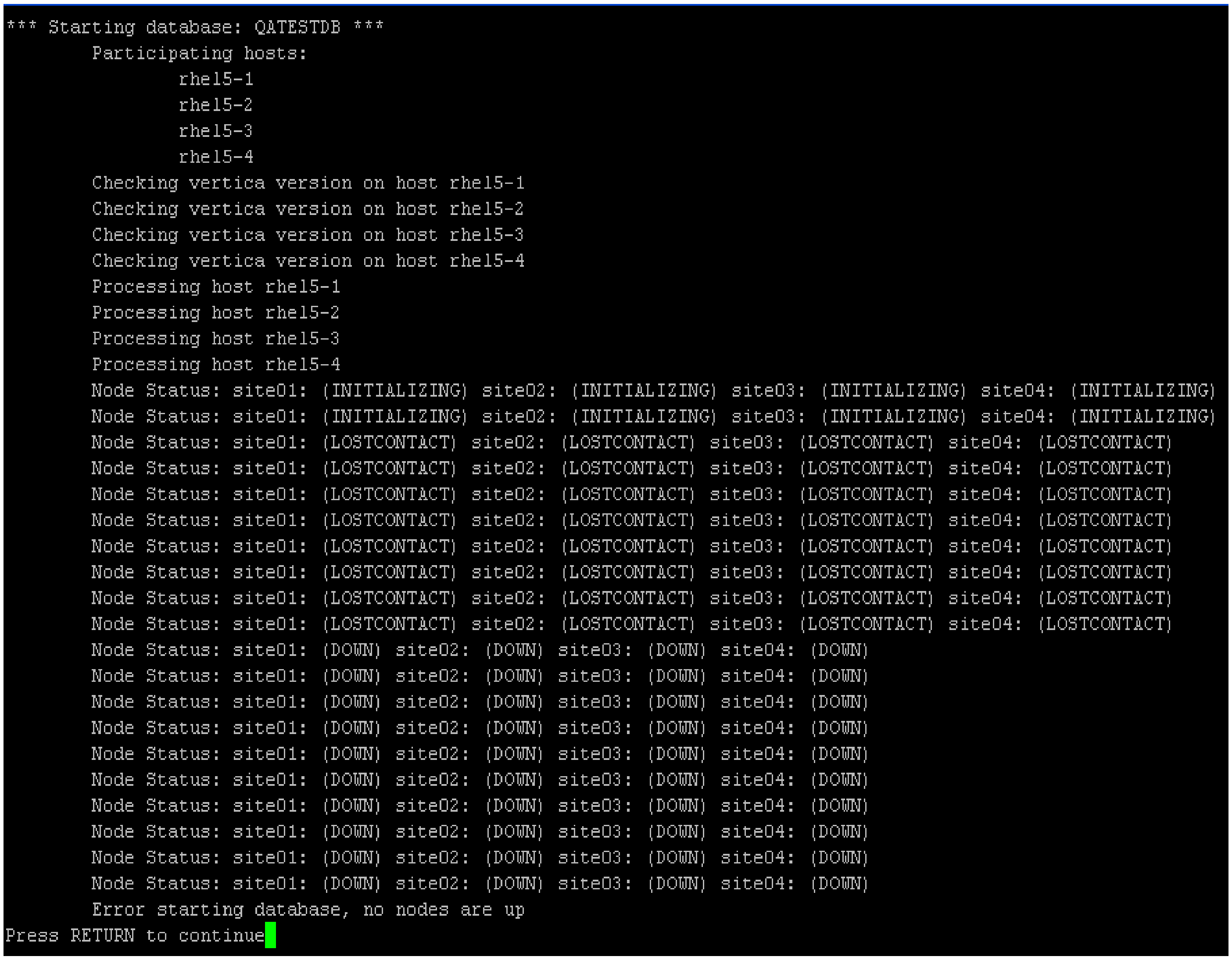
```
Main Menu

      1  View Database Cluster State
      2  Connect to Database
      3  Start Database
      4  Stop Database
      5  Restart Node
      6  Configuration
      7  Advanced
      8  Help on Using the Administration Tools
      E  Exit

      <  OK  >      <Cancel>      < Help >
```

3. Select the database to be restarted.



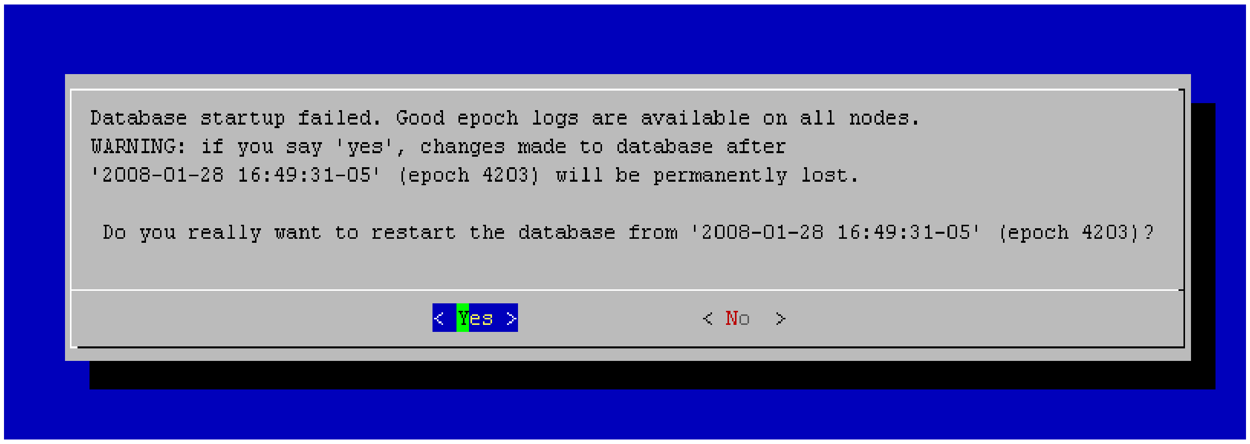
```
Select database that needs node restart

          (X) Stock_Schema  dbadmin

     <  OK  >     <Cancel>     < Help >
```

If you are starting the database after an unclean shutdown, you will see messages which will indicate that the startup has failed. Press RETURN to continue with the recovery process.



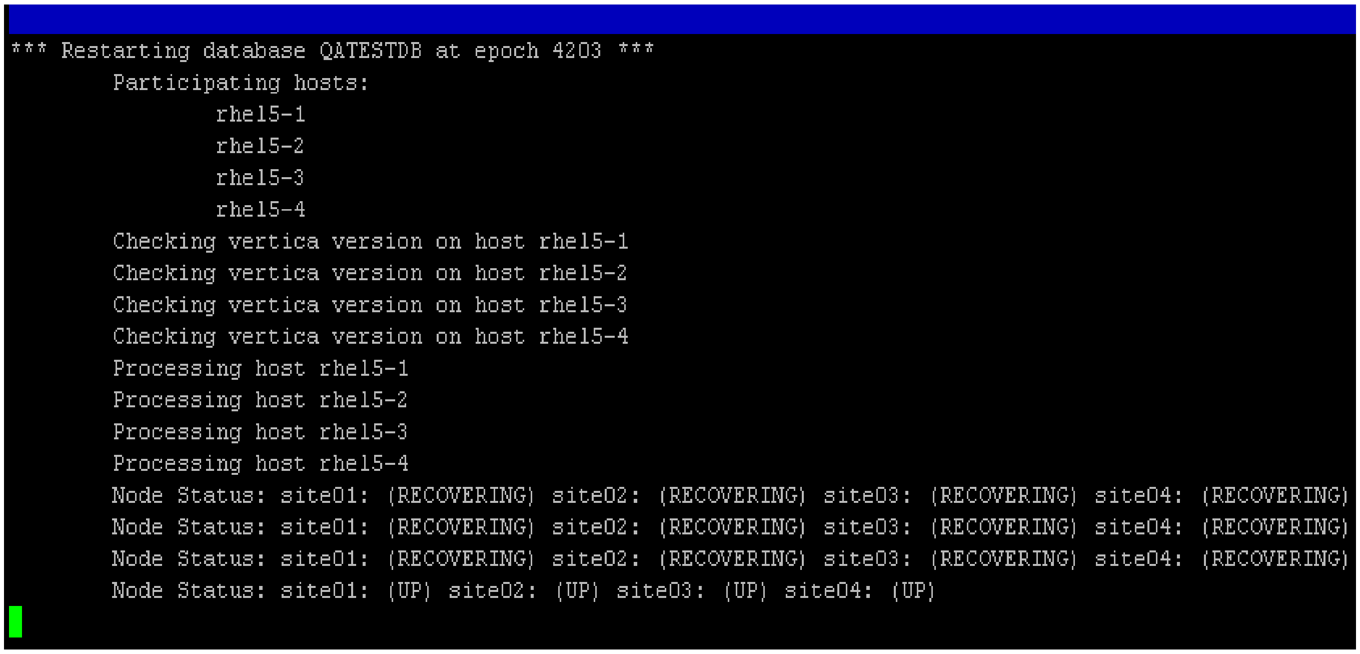
```
*** Starting database: QATESTDB ***
        Participating hosts:
                rhel5-1
                rhel5-2
                rhel5-3
                rhel5-4
        Checking vertica version on host rhel5-1
        Checking vertica version on host rhel5-2
        Checking vertica version on host rhel5-3
        Checking vertica version on host rhel5-4
        Processing host rhel5-1
        Processing host rhel5-2
        Processing host rhel5-3
        Processing host rhel5-4
        Node Status: site01: (INITIALIZING) site02: (INITIALIZING) site03: (INITIALIZING) site04: (INITIALIZING)
        Node Status: site01: (INITIALIZING) site02: (INITIALIZING) site03: (INITIALIZING) site04: (INITIALIZING)
        Node Status: site01: (LOSTCONTACT) site02: (LOSTCONTACT) site03: (LOSTCONTACT) site04: (LOSTCONTACT)
        Node Status: site01: (LOSTCONTACT) site02: (LOSTCONTACT) site03: (LOSTCONTACT) site04: (LOSTCONTACT)
        Node Status: site01: (LOSTCONTACT) site02: (LOSTCONTACT) site03: (LOSTCONTACT) site04: (LOSTCONTACT)
        Node Status: site01: (LOSTCONTACT) site02: (LOSTCONTACT) site03: (LOSTCONTACT) site04: (LOSTCONTACT)
        Node Status: site01: (LOSTCONTACT) site02: (LOSTCONTACT) site03: (LOSTCONTACT) site04: (LOSTCONTACT)
        Node Status: site01: (LOSTCONTACT) site02: (LOSTCONTACT) site03: (LOSTCONTACT) site04: (LOSTCONTACT)
        Node Status: site01: (LOSTCONTACT) site02: (LOSTCONTACT) site03: (LOSTCONTACT) site04: (LOSTCONTACT)
        Node Status: site01: (LOSTCONTACT) site02: (LOSTCONTACT) site03: (LOSTCONTACT) site04: (LOSTCONTACT)
        Node Status: site01: (DOWN) site02: (DOWN) site03: (DOWN) site04: (DOWN)
        Node Status: site01: (DOWN) site02: (DOWN) site03: (DOWN) site04: (DOWN)
        Node Status: site01: (DOWN) site02: (DOWN) site03: (DOWN) site04: (DOWN)
        Node Status: site01: (DOWN) site02: (DOWN) site03: (DOWN) site04: (DOWN)
        Node Status: site01: (DOWN) site02: (DOWN) site03: (DOWN) site04: (DOWN)
        Node Status: site01: (DOWN) site02: (DOWN) site03: (DOWN) site04: (DOWN)
        Node Status: site01: (DOWN) site02: (DOWN) site03: (DOWN) site04: (DOWN)
        Node Status: site01: (DOWN) site02: (DOWN) site03: (DOWN) site04: (DOWN)
        Node Status: site01: (DOWN) site02: (DOWN) site03: (DOWN) site04: (DOWN)
        Node Status: site01: (DOWN) site02: (DOWN) site03: (DOWN) site04: (DOWN)
        Error starting database, no nodes are up
Press RETURN to continue
```

An epoch represents committed changes to the data stored in a database between two specific points in time. When starting the database, Vertica will search for last good epoch. Upon determining the last good epoch, you will be prompted to verify that you would like to start the database from the good epoch date. Select "Yes" to continue with the recovery.



```
Database startup failed. Good epoch logs are available on all nodes.
WARNING: if you say 'yes', changes made to database after
'2008-01-28 16:49:31-05' (epoch 4203) will be permanently lost.

 Do you really want to restart the database from '2008-01-28 16:49:31-05' (epoch 4203)?
```

Vertica will then continue to initialize and recover all data prior to the last good epoch.



```
*** Restarting database QATESTDB at epoch 4203 ***
        Participating hosts:
                rhel5-1
                rhel5-2
                rhel5-3
                rhel5-4
        Checking vertica version on host rhel5-1
        Checking vertica version on host rhel5-2
        Checking vertica version on host rhel5-3
        Checking vertica version on host rhel5-4
        Processing host rhel5-1
        Processing host rhel5-2
        Processing host rhel5-3
        Processing host rhel5-4
        Node Status: site01: (RECOVERING) site02: (RECOVERING) site03: (RECOVERING) site04: (RECOVERING)
        Node Status: site01: (RECOVERING) site02: (RECOVERING) site03: (RECOVERING) site04: (RECOVERING)
        Node Status: site01: (RECOVERING) site02: (RECOVERING) site03: (RECOVERING) site04: (RECOVERING)
        Node Status: site01: (UP) site02: (UP) site03: (UP) site04: (UP)
```

If recovery takes more than a minute, you will be prompted to answer <Yes> or <No> to "Do you want to continue waiting?". When all the nodes' status have changed to RECOVERYING or UP, answering <No> will allow you to exit this screen and monitor progress via the adminTool main menu. Otherwise, answering <yes> will continue to display the database recovery window.

Note: Be sure to reload any data that was added after the last good epoch date to which you have recovered.

7.3.3 Partial Disk Loss

If the disk where the data or catalog directory resides fails, replace the disk and recreate the data or catalog directory. Then copy the following files from any other node to the new node:

```
vertica.conf
debug_log.conf
```

These files can be found in the following directory on the other nodes:

```
/DATABASE_HOME_DIR/DATABASE_NAME/site<xx>_catalog/
```

Note: Examples for finding your DATABASE_HOME_DIR can be found in the "Catalog and Data Files" section of this document.

After replacing these files, restart the database following the steps in the previous "Restart the Database" section.

7.3.4 Replacing a Failed Machine

When replacing a failed machine, the replacement machine must be identically configured to the machine being replaced. Follow these steps to configure the new host:

Note: Detailed instructions for steps 1 through 4 can be found in the Vertica Product Documentation, Installation Guide.

1. Install the same OS and use the same IP address, hostname.
2. Install the same version of the Vertica RPM but do not run `/opt/vertica/sbin/install_vertica.`
3. Create the dbadmin user.
4. Create the catalog and data paths using the same permissions as the original machine. For example, if the original catalog directory was:
   `/DATABASE_HOME_DIR/DATABASE_NAME/site<xx>_catalog/`

   then create this directory:
   `/DATABASE_HOME_DIR/DATABASE_NAME/`

   Note: Examples for finding your DATABASE_HOME_DIR can be found in the "Catalog and Data Files" section of this document.

      a. To ensure password-less ssh, execute the following:

b. Copy the fixkeys.py script, available for download on the Online Training Section of the Vertica website, to the `/opt/vertica/bin/` directory. The script must be owned and executable by root.

c. Execute the fixkeys.py script as root:

```
fixkeys.py <comma separated list of all hosts in cluster, including
this one> <db admin user>
```

5. Restart the node. (Please refer to the previous section "Restart the Node" for more details.)

6. Copy the contents of `/opt/vertica/config` over from another site

7. Install and start spread as root user. To install spread, execute the `/opt/vertica/sbin/spread_install.sh` script as follows:

```
spread_install.sh <install,stop,remove,start> <prefix_dir>
<RHEL4|RHEL5|FC4|FC5|FC6|SUSE|DEBIAN>
```

For example, to install on a redhat 5 site:

```
spread_install.sh install /opt/vertica/ RHEL5
```

To start spread, execute the following as root:

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

7.3.5 Adding Disk Space

The following procedure details the steps required to add disk space to a node in the Vertica cluster.

1. Add a disk to the system
   Shutdown Vertica and power off system if it is required by the hardware.
   Insert the new disk and power on the system if required
2. Partition/format/mount new disk as required by the hardware environment
3. Create a database path on the new volume
4. Restart Vertica on site if required
5. Add the new database path on the system with the new disk from an open session:
   `select add_location('<database path on new volume'>);`

Note: Add_location is a local command and needs to be run on each site that space is added to.

For example, given the following configuration;

Site: site01
Database Name: myDB
Existing data path: `/myPath/myDB/site01_data/`

Path to new volume being added: `/myNewPath/`

We would follow the steps below to add a new disk.

1. From admintools stop database on site01
2. Power off site01
3. Insert new disk
4. Power on site01
5. Partition as needed
   `mount /myNewPath/ on new volume`
6. Create a data directory path on the new volume
   `mkdir -p /myNewPath/myDB/site01_data2/`
7. From admintools start database on site01
8. Open a database connection on site01 and add new data location
   `select add_location('/myNewPath/myDB/site01_data2/');`

7.3.6 Adding Disk Space Across the Cluster

The following steps can be performed to add disk space to all sites in an optimal cluster environment:

If the cluster can be taken offline:

1. Shutdown the cluster from admintools
2. Follow "Base Case" steps 1-3 for adding a disk on each site
3. Start the Vertica database from admintools
4. Follow "Base Case" step 5 for adding a location on each site

If the cluster cannot be taken offline:

1. Follow "Bases Case" procedure for each site, 1 site at a time.

7.3.7 Recovering the from a Backup

Please refer to the Backup and Recovery guide available on the Online Training section of the Vertica Website for additional information on recovering a cluster from backup.

7.4 Monitoring Recovery

During recovery, Vertica adds logging information to the vertica.log on each site. Monitoring your Recovery progress is possible by viewing these messages. Recovery status messages can be identified by the string '[Recovery]'. For example,

```
$ tail -f catalog-path/database-name/node-name_catalog/vertica.log
```

```
01/23/08 10:35:31 thr:Recover:0x2a98700970 [Recover] <INFO> Changing site node01
startup state from INITIALIZING to RECOVERING
01/23/08 10:35:31 thr:CatchUp:0x1724b80 [Recover] <INFO> Recovering to specified
epoch 0x120b6
01/23/08 10:35:31 thr:CatchUp:0x1724b80 [Recover] <INFO> Running 1 split queries
01/23/08 10:35:31 thr:CatchUp:0x1724b80 [Recover] <INFO> Running query: ALTER
PROJECTION proj_tradesquotes_0 SPLIT node01 FROM 73911;
```

When recovery has completed, selecting "View Database Cluster State" from the Main Menu in the adminTools utility will report your node's status as "UP".

8. Backup and Restore

The following document details the steps required to perform a backup on and restore to the Vertica Database. In this guide we will be using the Stock_Schema sample database, included with the Vertica distribution. For the purposes of demonstration, the cluster to be backed up will include hosts; hostA01, hostA02, hostA03, and hostA04 and the target cluster, where the database will be restored, includes hosts; hostB01, hostB02, hostB03, and hostB04.

8.1 Introduction

8.1.1 Cold Backup

The most straightforward way to backup in Vertica is to perform a cold backup. A cold backup involves shutting down the Vertica Database completely and backing up the catalog and data directories. For this approach, the standard UNIX utility tar is used.

The advantages of a cold backup is that it is relatively easy to perform and restore with very little site-specific customization required for implementation.

The disadvantage of cold backups is that the database must be shut down. If you can afford to shut down a database for backups, cold database backups usually offer the best and easiest backup strategy.

8.1.2 Hot Backup

During Vertica's hot backup procedure the database is up and available for selects to users. However, the tuple mover is shutdown and the system is not available for operations that may modify the database and its data. For hot backups, the rdiff-backup tool, which is provided by http://rdiff-backup.nongnu.org/, is used to demonstrate the procedures to accomplish an incremental backup of the database's catalog and data directories. However, the tool is not a part of Vertica software.

The advantages of a hot backup are;

- The database can continue to handle select queries while the database is being backed up.
- The data can be recovered to a more recent time period.
- Requires less time especially for large databases.

On the other hand, hot backups have several disadvantages including;

- Requiring a utility that is not included with the standard LINUX distribution or Vertica software.
- Slightly more complex to implement.
- The need to periodically test in order to ensure viability.

8.2 Backup Procedures

8.2.1 Prepare the Database

a.) Stop the Database (Cold Backup)

Before a cold backup is performed, the Vertica database process will need to be stopped to ensure that the backup represent a single-moment snapshot of the db. In this example, we will stop the Stock_Schema database on the hostA01 through hostA04 cluster using the command line argument:

```
hostA01:/ $ /opt/vertica/bin/adminTools -t stop_db -d Stock_Schema -p [password]
       Issuing shutdown command to database
Database Stock_Schema stopped successfully
```

b.) Stop the Tuple Mover (Hot Backup)

Alternatively, if no operation that changes the database is performed, the database can be placed into a steady state during the backup procedure. The approach is preferred in reporting or analytic applications wherein data are only loaded or changed by the DBA at scheduled times.

Before the backup, the DBA must issue a 'select stop_tuple_mover();' to stop the ATM (automatic tuple mover) and wait for the currently running tuple mover mergeout and moveout to finish.

The tuple mover can be stopped manually from modifying files in the background:

```
select stop_tuple_mover();
```

In addition, the following is an example script that will wait for the tuple mover mergeout and moveout to finish.

stopTM.sh

```
#!/bin/bash

usage()
{
    echo  "usage: $0 -n<db_name> -u<username> -p<port>"
}

DB_NAME=""; PORT=""; USERNAME=""
while getopts n:u:p: opt
do
    case "$opt" in
      n)  DB_NAME=$OPTARG;;
      p)  PORT=$OPTARG;;
      u)  USERNAME=$OPTARG;;
      \?) # unknown flag
          usage; exit 1;;
    esac
done
[[ "$DB_NAME" ]] ||
    { echo "Must specify a database name with -n<db_name>"; exit 1; }
[[ "$PORT" ]] ||
    { echo "Must specify the database port with -p<port>"; exit 1; }
[[ "$USERNAME" ]] ||
    { echo "Must specify a username with -u<username>"; exit 1; }

MERGEOUT =`/opt/vertica/bin/vsql -d $DB_NAME -p $PORT -U $USERNAME \
               -c "select dump_configuration;" | grep MergeOutInterval | awk '{print $3}'`

MOVEOUT =`/opt/vertica/bin/vsql -d $DB_NAME -p $PORT -U $USERNAME \
               -c "select dump_configuration;" | grep MoveOutInterval | awk '{print $3}'`

if ((MERGEOUT>MOVEOUT)); then
        ((MAX_WAIT = MERGEOUT+3))
else
        ((MAX_WAIT =MOVEOUT+3))
fi

WAIT_INTERVAL=10;
TOTAL_WAIT=0;

# Wait for currently running tuple mover mergeout and moveout to
# finish

while true; do
       RUNNING=`/opt/vertica/bin/vsql -d $DB_NAME -p $PORT -U $USERNAME \
                                      -c "select * from vt_tuple_mover;" | grep Running | wc -l`
       if (( ! RUNNING )); then
           break
       fi
       sleep $WAIT_INTERVAL
       ((TOTAL_WAIT += WAIT_INTERVAL))
done
# Wait for tuple mover process to reset.

if ((TOTAL_WAIT < MAX_WAIT)); then
   ((TIME_REMAINING = MAX_WAIT - TOTAL_WAIT))
   sleep $TIME_REMAINING
fi
```

Next, advance the epoch to make sure all modified rows are flagged with an epoch:
`select advance_epoch();`
To ensure all the data in the WOS have been moved out to the ROS on the disk, run 'alter projection moveout' for each projection that has data in the WOS. The following script can be used to automate this process:

```
#! /bin/bash

usage()
{
    echo  "usage: $0 -n<db_name> -u<username> -p<port>"
}

DB_NAME=""; PORT=""; USERNAME=""
while getopts n:u:p: opt
do
    case "$opt" in
      n)  DB_NAME=$OPTARG;;
      p)  PORT=$OPTARG;;
      u)  USERNAME=$OPTARG;;
      \?) # unknown flag
          usage; exit 1;;
    esac
done
[[ "$DB_NAME" ]] ||
    { echo "Must specify a database name with -n<db_name>"; exit 1; }
[[ "$PORT" ]] ||
    { echo "Must specify the database port with -p<port>"; exit 1; }
[[ "$USERNAME" ]] ||
    { echo "Must specify a username with -u<username>"; exit 1; }

PROJECTION_NAMES=`/opt/vertica/bin/vsql -d $DB_NAME -p $PORT -U $USERNAME \
       -c "select wos_rows, projection_name from vt_column_storage;" | \
       awk '{ if (($1 > 0) && ($1 !~ /wos/)) print $3}' | uniq`

For PROJECTION in ${PROJECTION_NAMES[*]}; do
       /opt/vertica/bin/vsql -d $DB_NAME -p $PORT -U $USERNAME \
              -c "alter projection $PROJECTION moveout;"
Done
```

Once this is completed, the database itself is ready to be backed up. It is important that in the duration of the backup, there are no copy, insert, delete, or update commands executed. When the backup is complete, issue the following command to start the tuple mover again:

```
select start_tuple_mover();
```

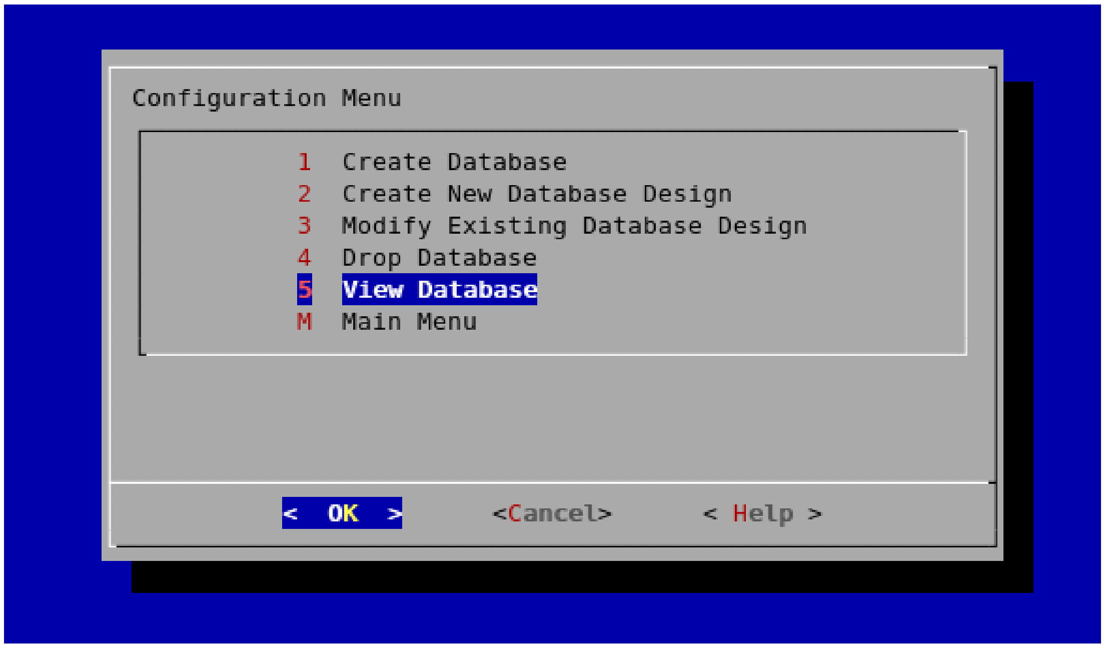
8.2.2 Determine the Database Directory

In Vertica, the catalog is a set of files that contain information (metadata) about the objects in a database, such as the nodes, tables, constraints, and projections. The catalog files are replicated on all nodes in a cluster while the data files are unique to each node. They can be found in the following directories:

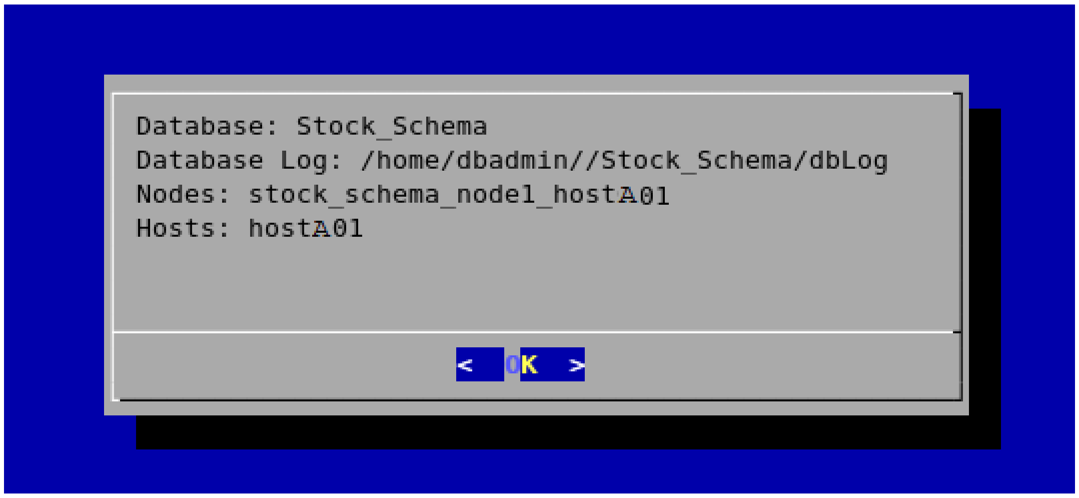
```
/DATABASE_HOME_DIR/DATABASE_NAME/site<xx>_catalog/

/DATABASE_HOME_DIR/DATABASE_NAME/site<xx>_data/
```

Where `DATABASE_HOME_DIR` is the path that can be found from the adminTools "Configuration" -> "View Database" menu.

To view the path from adminTools, select "Configuration" from the main menu and then select "View Database".



Select the database that you would like to view and click "OK" to see the database profile.



```
Database: Stock_Schema
Database Log: /home/dbadmin//Stock_Schema/dbLog
Nodes: stock_schema_node1_hostA01
Hosts: hostA01
```

In this example, the catalog files are found in the following directory:

```
/home/dbadmin/Stock_Schema/stock_schema_node1_hostA01_catalog/
```

So the directory that will need to be archived is:

```
/home/dbadmin/Stock_Schema
```

8.2.3 Archive the Database Directory

a.) Full Backup

Using the GNU utility "tar", archive the directory tree which contains your catalog and data files for the database to be backed up.

The following script can be used to do this in an automated fashion. Note that you will have to change the parameters for your environment.

fullBackup.sh

```
SOURCE_HOSTS=(hostA01 hostA02 hostA03 hostA04)
BACKUP_HOST=hostA01          # The name of the host to store backups
BACKUP_DIR=/home/dbadmin/backup # Directory to store the backed up database.
    for host in ${SOURCE_HOSTS[*]}; do
        ssh $host tar -C /home/dbadmin/ -czf - Stock_Schema | \
            ssh $BACKUP_HOST dd of=$BACKUP_DIR/Stock_Schema.${host}.tz
    done
```

The Stock_Schema database has now been successfully backed up.

b.) Incremental Backup

The following incremental backup procedures are demonstrated by using the rdiff-backup tool. For downloading information and additional documentation on the rdiff-backup utility, please refer to http://rdiff-backup.nongnu.org/

Using the `rdiff-backup` tool, the following script will run an incremental backup of the /catalog and /data directories for the specified database. This script can also be used to set up a daily cron job.

Refer to the previous section, "Determining your Database Location" for information on finding your database directories.

incrementalBackup.sh

```
#!/bin/bash

usage()
{
    echo  "usage: $0 -B<host1,host2,host3,...> -b<backup_dir> -n<db_name> -d<db_dir> -u<username>"
}

BACKUP_CLUSTER_HOSTS=(); BACKUP_DIR=""; DB_NAME=""; DB_DIR=""; USERNAME=""
while getopts B:b:n:d:u: opt
do
    case "$opt" in
      B)  BACKUP_CLUSTER_HOSTS=(`echo $OPTARG | tr , " "`) ;;
      b)  BACKUP_DIR=$OPTARG;;
      n)  DB_NAME=$OPTARG;;
      d)  DB_DIR=$OPTARG;;
      u)  USERNAME=$OPTARG;;
      \?) # unknown flag
          usage; exit 1;;
    esac
done
[[ "${#BACKUP_CLUSTER_HOSTS[*]}" -gt 0 ]] ||
    { echo "Must specify the backup cluster hosts with -B<host1,host2,...>"; exit 1; }
[[ "$BACKUP_DIR" ]] ||
    { echo "Must specify a backup directory name with -b<backup_dir>"; exit 1; }
[[ "$DB_NAME" ]] ||
    { echo "Must specify a database name with -n<db_name>"; exit 1; }
[[ "$DB_DIR" ]] ||
    { echo "Must specify the database directory with -d<db_dir>"; exit 1; }
[[ "$USERNAME" ]] ||
    { echo "Must specify a username with -u<username>"; exit 1; }

rdiff-backup  /opt/vertica/config/users $BACKUP_DIR/users

for HOST in ${BACKUP_CLUSTER_HOSTS[*]}; do
    rdiff-backup --print-statistics --include '**/*catalog' \
                                    --include '**/*data' \
                                    --exclude $DB_DIR/$DB_NAME \
                 $USERNAME@${HOST}::$DB_DIR/$DB_NAME $BACKUP_DIR/$HOST
Done
```

8.3 Restore Procedures

8.3.1 Restore the Database

Please note that when restoring you must restore to an empty directory. Restoring over your old directory files may corrupt your database.

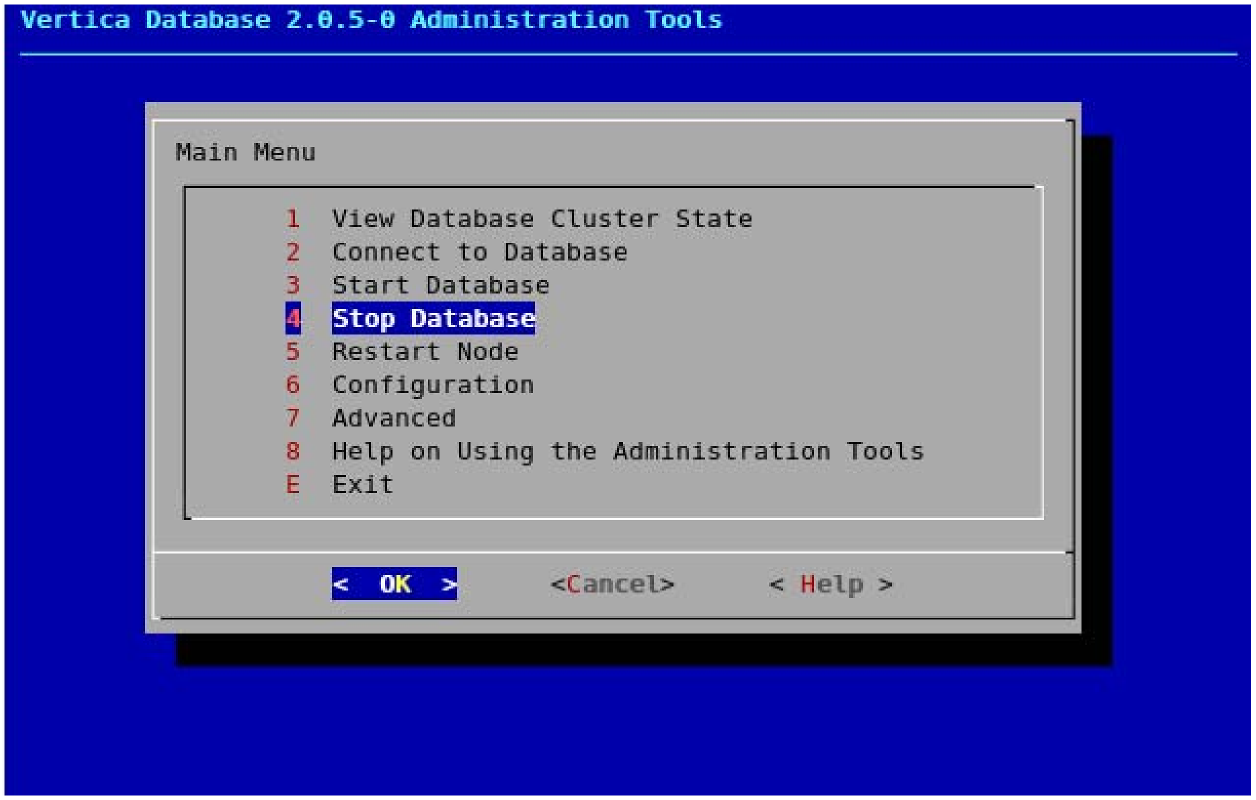
a.) Restoring to a New Cluster

1. Install Vertica
If Vertica is not already installed on the target cluster, install the Vertica database with the same configuration as the original cluster following the steps as detailed in the Vertica Product Documentation Installation Guide. In this example, Vertica will be installed on the target cluster which consists of hosts hostB01 through hostB04.
Note: The cluster must have the same number of nodes as the original cluster.

2. Create the Database
Once Vertica has been installed on the target cluster, create the database using the adminTools user interface with the same database name and the same data and catalog directories. In our example, the directories would be /home/dbadmin/Stock_Schema.
For detailed instructions on creating a database, please refer to the "Create the Database" section of the Vertica Production Documentation DBA Guide.
During the database creation process, be sure to make note of the port number. This information will be required when restoring the backup. For example, the following is a sample output from creating the Stock_Schema database:

```
*** Creating database: Stock_Schema ***
   Running integrity checks
        Will create database on port 5433
        Checking that nodes are defined and installed
        . . .
```

3. Stop the Database
Once the database is created it must be stopped before proceeding. From the adminTools utility Main Menu, select the "Stop Database" option and then exit adminTools.



4A. Restore the Database

On the backup host, change to the backup directory and copy the tar files to the nodes in the destination cluster into the restore directory (for this example, /home/dbadmin) and untar them.
The following script may be used to automate this process after it has been edited for your environment.

fullRestore.sh

```
BACKUP_DIR=/home/dbadmin/backup # Where the tar files are stored.
SOURCE_HOSTS=(hostA01 hostA02 hostA03 hostA04)
DEST_HOSTS=(hostB01 hostB02 hostB03 hostB04)
NHOSTS=${#SOURCE_HOSTS[*]}
    cd $BACKUP_DIR
    i=0
    while [ $i -lt $NHOSTS ]; do
        scp Stock_Schema.${SOURCE_HOSTS[$i]}.tz  ${DEST_HOSTS[$i]}:/home/dbadmin
        ssh  ${DEST_HOSTS[$i]} tar -C /home/dbadmin -xf   \
                               /home/dbadmin/Stock_Schema.${SOURCE_HOSTS[$i]}.tz
        ((i++))
    done
```

4B. Incremental Restore

Viewing Available Backups

The `--list-increments` option will list the times of the available backups.
This option may be useful if you need to restore an older backup of the database, but aren't sure which one.   The following is an example of the command and its output:

```
$ BACKUP_DIR=/backup_vertica
$ HOST=hostA01
$ rdiff-backup --list-increments $BACKUP_DIR/$HOST
Found 6 increments:
    increments.2008-02-28T18:31:39-05:00.dir   Thu Feb 28 18:31:39 2008
    increments.2008-02-29T14:58:41-05:00.dir   Fri Feb 29 14:58:41 2008
    increments.2008-03-01T09:38:21-05:00.dir   Sat Mar  1 09:38:21 2008
    increments.2008-03-02T10:18:25-05:00.dir   Sun Mar  2 10:18:25 2008
    increments.2008-03-03T10:23:46-05:00.dir   Mon Mar  3 10:23:46 2008
    increments.2008-03-04T11:26:16-05:00.dir   Tue Mar  4 11:26:16 2008
 Current mirror: Mon Mar  4 12:13:15 2008
```

Incremental Restore Procedure

Login to the host which has the backup files (hostA01). Restore the database onto the hostB cluster using the `rdiff-backup --restore-as-of` option. The --restore-as-of option is used to restore the database to a specific backup. It is recommended to specify a backup session identifier when restoring the database. For example, a backup session valueof "0B" specifies the time of the current backup, while "3B" specifies the time of the third newest backup.

Here is an example script which restores the database to the specified restore time.
incrRestoreNew.sh

```
#!/bin/bash

usage()
{
    echo "usage: $0 -B<host1,host2,host3,...> -R<host1,host2,host3,...> -b<backup_dir> -n<db_name> -
d<db_dir> -u<username> -t<restore_time>"
}

BACKUP_CLUSTER_HOSTS=(); RESTORE_CLUSTER_HOSTS=()
BACKUP_DIR=""; DB_NAME=""; DB_DIR=""; USERNAME=""; RESTORE_TIME=""

while getopts B:R:b:n:d:u:t: opt
do
    case "$opt" in
      B)  BACKUP_CLUSTER_HOSTS=(`echo $OPTARG | tr , " "`) ;;
      R)  RESTORE_CLUSTER_HOSTS=(`echo $OPTARG | tr , " "`) ;;
      b)  BACKUP_DIR=$OPTARG;;
      n)  DB_NAME=$OPTARG;;
      d)  DB_DIR=$OPTARG;;
      u)  USERNAME=$OPTARG;;
      t)  RESTORE_TIME=$OPTARG;;
      \?) # unknown flag
          usage; exit 1;;
    esac
done
[[ "${#BACKUP_CLUSTER_HOSTS[*]}" -gt 0 ]] ||
    { echo "Must specify the backup cluster hosts with -B<host1,host2,...>"; exit 1; }
[[ "${#RESTORE_CLUSTER_HOSTS[*]}" -gt 0 ]] ||
    { echo "Must specify the restore cluster hosts with -R<host1,host2,...>"; exit 1; }
[[ "$BACKUP_DIR" ]] ||
    { echo "Must specify a backup directory name with -b<backup_dir>"; exit 1; }
[[ "$DB_NAME" ]] ||
    { echo "Must specify a database name with -n<db_name>"; exit 1; }
[[ "$DB_DIR" ]] ||
    { echo "Must specify the database directory with -d<db_dir>"; exit 1; }
[[ "$USERNAME" ]] ||
    { echo "Must specify a username with -u<username>"; exit 1; }
[[ "$RESTORE_TIME" ]] ||
    { echo "Must specify a restore time with -t<restore_time>"; exit 1; }

i=0
for BACKUP_HOST in ${BACKUP_CLUSTER_HOSTS[*]}; do

    RESTORE_HOST=${RESTORE_CLUSTER_HOSTS[$i]}
    rdiff-backup --force --restore-as-of $RESTORE_TIME \
                 $BACKUP_DIR/$BACKUP_HOST  $USERNAME@${RESTORE_HOST}::$DB_DIR/$DB_NAME
    ((i++))

done
```

b.) Restoring on the Same Cluster (Incremental)

The `--restore-as-of` option is used to restore the database to a specific backup. It is recommended to specify a backup session identifier when restoring the database. For example, a backup session value of "0B" specifies the time of the current backup while "3B" specifies the time of the third newest backup.

The following script will restore the database with the 3rd newest backup. Following the output given from the `--list-increments` command found in the “Incremental Restore” -> “Viewing Available Backups” section, this corresponds to the backup taken on Sunday March 2nd.

incrRestoreSame.sh

```
#!/bin/bash

usage()
{
    echo "usage: $0 -B<host1,host2,host3,...> -b<backup_dir> -n<db_name> -d<db_dir> -u<username> -
t<restore_time>"
}

BACKUP_CLUSTER_HOSTS=(); BACKUP_DIR=""; DB_NAME=""; DB_DIR=""; USERNAME=""; RESTORE_TIME=""
while getopts B:b:n:d:u:t: opt
do
    case "$opt" in
      B)  BACKUP_CLUSTER_HOSTS=(`echo $OPTARG | tr , " "`) ;;
      b)  BACKUP_DIR=$OPTARG;;
      n)  DB_NAME=$OPTARG;;
      d)  DB_DIR=$OPTARG;;
      u)  USERNAME=$OPTARG;;
      t)  RESTORE_TIME=$OPTARG;;
      \?) # unknown flag
          usage; exit 1;;
    esac
done
[[ "${#BACKUP_CLUSTER_HOSTS[*]}" -gt 0 ]] ||
    { echo "Must specify the backup cluster hosts with -B<host1,host2,...>"; exit 1; }
[[ "$BACKUP_DIR" ]] ||
    { echo "Must specify a backup directory name with -b<backup_dir>"; exit 1; }
[[ "$DB_NAME" ]] ||
    { echo "Must specify a database name with -n<db_name>"; exit 1; }
[[ "$DB_DIR" ]] ||
    { echo "Must specify the database directory with -d<db_dir>"; exit 1; }
[[ "$USERNAME" ]] ||
    { echo "Must specify a username with -u<username>"; exit 1; }
[[ "$RESTORE_TIME" ]] ||
    { echo "Must specify a restore time with -t<restore_time>"; exit 1; }

for HOST in ${BACKUP_CLUSTER_HOSTS[*]}; do
    rdiff-backup --force --restore-as-of $RESTORE_TIME $BACKUP_DIR/$HOST \
               $USERNAME@${HOST}::$DB_DIR/$DB_NAME

    sudo rdiff-backup --force --restore-as-of $RESTORE_TIME $BACKUP_DIR/users \
               root@${HOST}::/opt/vertica/config/users
done
```

8.3.2 Replace the Database Directories

In the case that the target and source directories are different, replace the catalog and data directories for the target cluster on each node with the backed up ones. Ensure that the directory names accurately reflect the target host name. For this example, the directory names on the original host were:
`/home/dbadmin/Stock_Schema/stock_shema_node1_hostA01_catalog`

```
/home/dbadmin/Stock_Schema/stock_shema_node1_hostA01_data
```

And will need to be changed to reflect the target host name:

```
/home/dbadmin/Stock_Schema/stock_shema_node1_hostB01_catalog
/home/dbadmin/Stock_Schema/stock_shema_node1_hostB01_data
```

The following script can be used to update the directory names after it has been modified to reflect your environment.

replaceDBDirs.sh

```
#!/bin/bash

usage()
{
    echo "usage: $0 -B<host1,host2,host3,...> -R<host1,host2,host3,...> -n<db_name> -d<db_dir>"
}

BACKUP_CLUSTER_HOSTS=(); RESTORE_CLUSTER_HOSTS=()
DB_NAME=""; DB_DIR=""

while getopts B:R:n:d: opt
do
    case "$opt" in
      B)  BACKUP_CLUSTER_HOSTS=(`echo $OPTARG | tr , " "`) ;;
      R)  RESTORE_CLUSTER_HOSTS=(`echo $OPTARG | tr , " "`) ;;
      n)  DB_NAME=$OPTARG;;
      d)  DB_DIR=$OPTARG;;
      \?) # unknown flag
          usage; exit 1;;
    esac
done
[[ "${#BACKUP_CLUSTER_HOSTS[*]}" -gt 0 ]] ||
    { echo "Must specify the backup cluster hosts with -B<host1,host2,...>"; exit 1; }
[[ "${#RESTORE_CLUSTER_HOSTS[*]}" -gt 0 ]] ||
    { echo "Must specify the restore cluster hosts with -R<host1,host2,...>"; exit 1; }
[[ "$DB_NAME" ]] ||
    { echo "Must specify a database name with -n<db_name>"; exit 1; }
[[ "$DB_DIR" ]] ||
    { echo "Must specify the database directory with -d<db_dir>"; exit 1; }

DB_DIR=$DB_DIR/$DB_NAME
NHOSTS=${#BACKUP_CLUSTER_HOSTS[*]}
# Stock_Schema -> stock_schema
DB_NAME_META=`echo $DB_NAME | tr [:upper:] [:lower:] | tr - _`
# Replace the catalog and data dir for RESTORE_CLUSTER on each node with the backed up ones.
i=0; n=1
while (( i < $NHOSTS )) ;do
    BACKUP_HOST=${BACKUP_CLUSTER_HOSTS[${i}]}
    RESTORE_HOST=${RESTORE_CLUSTER_HOSTS[${i}]}
    BACKUP_HOST_META=`echo $BACKUP_HOST | tr [:upper:] [:lower:] | tr - _ | tr . _`
    RESTORE_HOST_META=`echo $RESTORE_HOST | tr [:upper:] [:lower:] | tr - _ | tr . _`
    ssh ${RESTORE_HOST} "cd $DB_DIR && rm -rf  ${DB_NAME_META}_node${n}_${RESTORE_HOST_META}_catalog \
                                               ${DB_NAME_META}_node${n}_${RESTORE_HOST_META}_data"
    ssh ${RESTORE_HOST} "cd $DB_DIR && mv ${DB_NAME_META}_node${n}_${BACKUP_HOST_META}_catalog   \
                                         ${DB_NAME_META}_node${n}_${RESTORE_HOST_META}_catalog"
    ssh ${RESTORE_HOST} "cd $DB_DIR && mv ${DB_NAME_META}_node${n}_${BACKUP_HOST_META}_data  \
                                      ${DB_NAME_META}_node${n}_${RESTORE_HOST_META}_data"
    ((i++)); ((n++))
done
```

8.3.3 Change the Meta-data Definitions

Next, the path and address settings in the *_6.xml and *_7.xml meta-data files, located in the catalog directories, will need to be modified to reflect the new environment. These files will need to be updated on each node in the target cluster.

For this example, before updating, the original *_6.xml file will look similar to the following:

```
hostB01: /home/dbadmin/Stock_Schema/stock_schema_node1_hostB01_catalog/Catalog/$ cat *_6.xml
:Site
old:45035996273704964
name:stock_schema_node1_hostA01
type:6
isPersistent:true
IsTemp:fals
isMutable:false
isGlobal:true
database:0
schema:0
address:hostA01
catalogPath:/home/dbadmin//Stock_Schema/stock_schema_node1_hostA01_catalog/Catalog
hasCatalog:false
dbdPath:/home/dbadmin//Stock_Schema/stock_schema_node1_hostA01_data/SAL
siteUniqueID:10
.
```

Notice the references throughout to the original host, “hostA01”. These will all need to be changed to reflect the target hostname, “hostB01”.

The following script may be used to make these updates but must first be edited for use in your environment.

modifyMetaData.sh

```
#!/bin/bash
#
# Edit the four  *_6.xml and four *_7.xml on each of the hosts
#

usage()
{
    echo "usage: $0 -B<host1,host2,host3,...> -R<host1,host2,host3,...> -n<db_name> -d<db_dir>"
}

BACKUP_CLUSTER_HOSTS=(); RESTORE_CLUSTER_HOSTS=()
DB_NAME=""; DB_DIR=""

while getopts B:R:n:d: opt
do
    case "$opt" in
      B)  BACKUP_CLUSTER_HOSTS=(`echo $OPTARG | tr , " "`) ;;
      R)  RESTORE_CLUSTER_HOSTS=(`echo $OPTARG | tr , " "`) ;;
      n)  DB_NAME=$OPTARG;;
      d)  DB_DIR=$OPTARG;;
      \?) # unknown flag
          usage; exit 1;;
    esac
done
[[ "${#BACKUP_CLUSTER_HOSTS[*]}" -gt 0 ]] ||
    { echo "Must specify the backup cluster hosts with -B<host1,host2,...>"; exit 1; }
[[ "${#RESTORE_CLUSTER_HOSTS[*]}" -gt 0 ]] ||
    { echo "Must specify the restore cluster hosts with -R<host1,host2,...>"; exit 1; }
[[ "$DB_NAME" ]] ||
    { echo "Must specify a database name with -n<db_name>"; exit 1; }
[[ "$DB_DIR" ]] ||
    { echo "Must specify the database directory with -d<db_dir>"; exit 1; }

DB_DIR=$DB_DIR/$DB_NAME
NHOSTS=${#BACKUP_CLUSTER_HOSTS[*]}

# Build the sed -e edits
i=0;
while (( i < $NHOSTS )) ;do
    RESTORE_HOST=${RESTORE_CLUSTER_HOSTS[${i}]}
    BACKUP_HOST=${BACKUP_CLUSTER_HOSTS[${i}]}
    EDIT="$EDIT -e s%${BACKUP_HOST}%${RESTORE_HOST}%"
    ((i++))
done
i=0;
while (( i < $NHOSTS )) ;do
    RESTORE_HOST=${RESTORE_CLUSTER_HOSTS[${i}]}
    # host-1.domain.com -> host_1_domain_com
    RESTORE_HOST_META=`echo $RESTORE_HOST | tr [:upper:] [:lower:] | tr - _ | tr . _`
    BACKUP_HOST=${BACKUP_CLUSTER_HOSTS[${i}]}
    BACKUP_HOST_META=`echo \$BACKUP_HOST   | tr [:upper:] [:lower:] | tr - _ | tr . _`
    EDIT="$EDIT -e s%${BACKUP_HOST_META}%${RESTORE_HOST_META}%"
    ((i++))
donei=0;
while (( i < $NHOSTS )) ;do
    RESTORE_HOST=${RESTORE_CLUSTER_HOSTS[${i}]}
    # fc6-1.verticacorp.com -> fc6_1_verticacorp_com
    RESTORE_HOST_META=`echo $RESTORE_HOST | tr [:upper:] [:lower:] | tr - _ | tr . _`
    BACKUP_HOST=${BACKUP_CLUSTER_HOSTS[${i}]}
    BACKUP_HOST_META=`echo \$BACKUP_HOST   | tr [:upper:] [:lower:] | tr - _ | tr . _`
    EDIT="$EDIT -e s%${BACKUP_HOST_META}%${RESTORE_HOST_META}%"
    ((i++))
done

# Now do the edits on each host
#
# Stock_Schema -> stock_schema
DB_NAME_META=`echo $DB_NAME | tr [:upper:] [:lower:]`
i=0; n=1
while (( i < $NHOSTS )); do

    RESTORE_HOST=${RESTORE_CLUSTER_HOSTS[${i}]}
    RESTORE_HOST_META=`echo $RESTORE_HOST | tr [:upper:] [:lower:] | tr - _ | tr . _`

    ssh $RESTORE_HOST "
        cd ${DB_DIR}/${DB_NAME_META}_node${n}_${RESTORE_HOST_META}_catalog/Catalog
        for input_file in *_6.xml *_7.xml; do
            sed -i "$EDIT" \${input_file}
        done
    "
    ((i++)); ((n++))
done
```

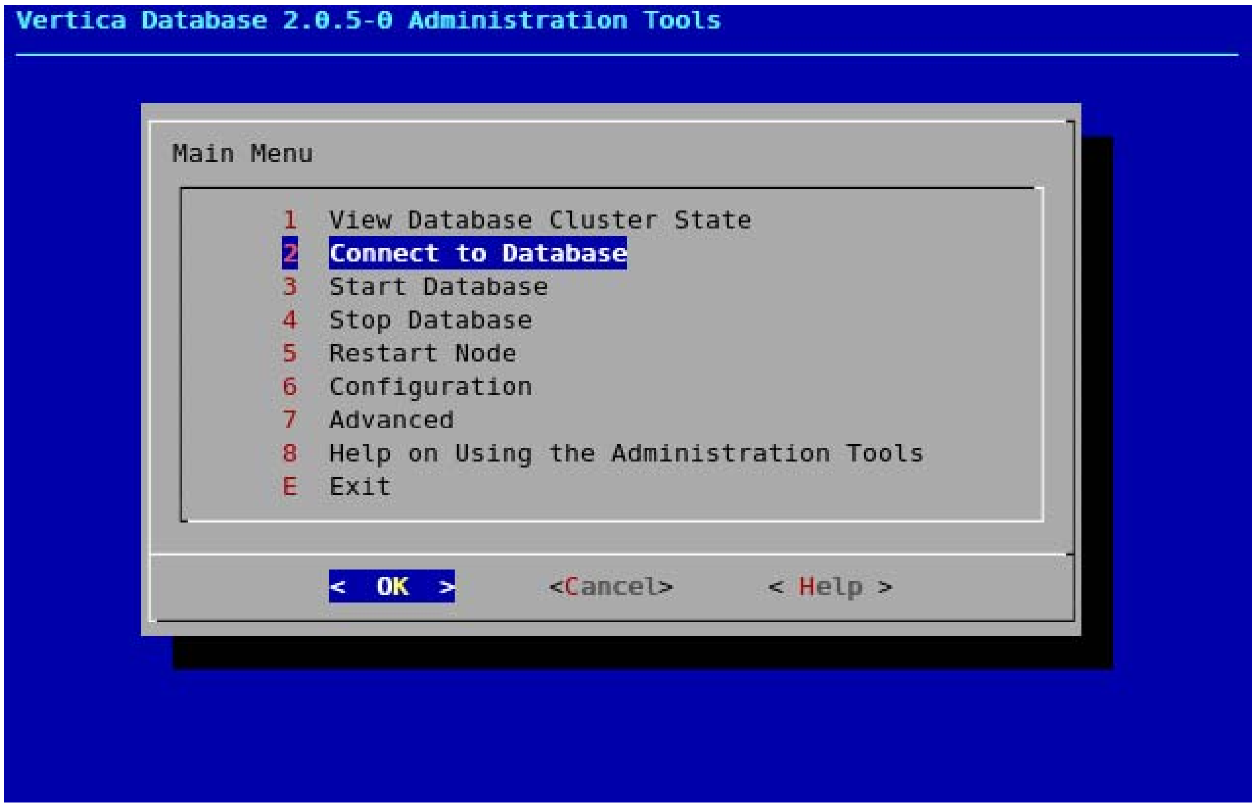
8.3.4 Start the Database

The database on the target cluster is ready to be started. This can be done either from the Admintools utility or from the command line;

```
hostB01:/ $ /opt/vertica/bin/adminTools -t start_db -d Stock_Schema –p [password]
Participating hosts:
        hostB01
        hostB02
        . . .
        Checking vertica version on host hostB01
        Processing host hostB1
        Node Status: stock_schema_node1_hostB01: (DOWN)
        . . .
        Node Status: stock_schema_node1_hostB01: (UP)
        . . .
Database Stock_Schema started successfully
```

8.3.5 Verify the Database

After the database starts, connect to it by selecting "connect to Database" through the AdminTools main menu, and execute some commands to verify the database restore was successful.



```
Stock_Schema => select count(*) from stocktransaction_fact;
    count
 ---------
  5000000
 (1 row)
```

8.4 Removing Old Incremental Backups

Although `rdiff-backup` only stores file differences, you will probably want to remove backups older than a certain date. The `--remove-older-than` option can be used to delete old backups. The following is an example script that tells rdiff-backup to remove any backups older than the specified backup time.

removeOldIncrBackup.sh

```
#!/bin/bash
#
# A script to remove old incremental backups.
#
usage()
{
    echo "usage: $0 -B<host1,host2,host3,...> -b<backup_dir> -t<backup_time>"
}

BACKUP_CLUSTER_HOSTS=()
BACKUP_DIR=""; BACKUP_TIME=""

while getopts B:b:t: opt
do
    case "$opt" in
      B)  BACKUP_CLUSTER_HOSTS=(`echo $OPTARG | tr , " "`) ;;
      b)  BACKUP_DIR=$OPTARG;;
      t)  BACKUP_TIME=$OPTARG;;
      \?) # unknown flag
          usage; exit 1;;
    esac
done
[[ "${#BACKUP_CLUSTER_HOSTS[*]}" -gt 0 ]] ||
    { echo "Must specify the backup cluster hosts with -B<host1,host2,...>"; exit 1; }
[[ "$BACKUP_DIR" ]] ||
    { echo "Must specify the backup directory with -b<backup_dir>"; exit 1; }
[[ "$BACKUP_TIME" ]] ||
    { echo "Must specify a backup time with -t<backup_time>"; exit 1; }

for HOST in ${BACKUP_CLUSTER_HOSTS[*]}; do
    rdiff-backup --force --remove-older-than $BACKUP_TIME $BACKUP_DIR/$HOST
done
```