WHITEPAPER

VECTORWISE

Developer Guide

A technical whitepaper

TABLE OF CONTENTS:

INTRODUCTION

Welcome to the VectorWise 1.5 Developer Guide.

VectorWise is a relational database that transparently utilizes performance features in CPUs to perform extremely fast reporting and data analysis. This document contains practical information, guidelines and best practices to get the most out of VectorWise. You should read this document when you plan to evaluate or deploy VectorWise.

The Developer Guide covers the following topics:

- System sizing: how to choose the right hardware configuration for VectorWise.
- Database configuration: considerations around database settings.
- Schema design: best practices for database table design and index strategy.
- Data loading: how to most efficiently load and manage your data in VectorWise.
- High availability: how to minimize downtime for VectorWise.

This guide does not replace the VectorWise User Guide; it is a supplement to that guide.

SYSTEM SIZING

You have 3 options to install and run VectorWise:

1) Use a dedicated server (and dedicated storage) and the base Operating System to run VectorWise.
2) Create a virtual machine and run VectorWise inside the virtual environment.
3) Use resources in the Cloud to run VectorWise. For more information, see: http://community.ingres.com/wiki/Ingres_VectorWise_Amazon_EC2_Images.

This section focuses on Option 1: VectorWise on a dedicated server with dedicated storage, as this produces the best results. Most of the discussion is also relevant for options 2 and 3. A virtual environment is not as ideal as a dedicated hardware environment, especially as it relates to storage throughput. When using VectorWise in the cloud, you may not be able to influence the physical storage throughput. If most or all of your data will fit in memory in a virtual environment or in the Cloud then you may not need to be concerned about this.

The maximum possible data processing throughput VectorWise can achieve depends on a number of factors including the data types involved in the query, the data, the query itself including the type of computation. VectorWise has been observed processing data analysis at incredible throughputs reaching over 1.5 GB/s per CPU core.

Determining the core count

The core count is determined by a combination of two factors:

1) What degree of parallelism do you plan to use to satisfy query response times?
2) How many operations do you want to be executed concurrently?

In theory, if a system is not currently experiencing bottlenecks, then setting parallelism to factor x will roughly reduce the query execution time linearly by a factor x. In practice you will likely see a less than perfect linear scalability, most certainly once you go beyond a degree of parallelism of 8. You may still see improvements in query response times above that level, but adding 50% more resources would not shrink the execution time by 50% but rather less than 50%.

Degree of parallelism is a changeable setting which allows you to do some experiments. A first-cut estimate of the Degree Of Parallelism (DOP) you should set can be calculated by the following formula:

```
DOP = column data retrieved in GB (uncompressed) / desired response time / 1.5
```

Note 1: This formula assumes the maximum data processing rate for VectorWise is 1.5 GB/s per core which may be too high for your query workload. Adjust the 1.5 factor in the formula if necessary.

Note 2: Only calculate the size of the data in the columns involved in the query. VectorWise uses column-based storage and does not retrieve column data that is not retrieved in the query.

For example, if the largest query in my system runs against a single denormalized table with a billion rows and the query summarizes sales by product code with:

- product code that is on average 20 bytes,
- sales number that is on average 10 bytes,
- required response time is 5 seconds,

then the DOP should be set to:

$$\text{DOP} = (20+10) / 1000^3 \text{ (GB)} * 1{,}000{,}000{,}000 / 5 / 1.5 = 4$$

When you set DOP to 4 then every query will use 4 CPU cores for the duration of the query, and the number of concurrent queries that can be executed is:

```
number of concurrent queries = number of cores / DOP
```

DOP is set with the `max_parallelism_level` configuration parameter (see the VectorWise 1.5 User Guide for more details).

More queries can be submitted at any point in time, but queries will wait for CPU core resources to become available. At present automatic queuing cannot be controlled or prioritized and operations are started in the order in which they are received.

If you disable parallel query by setting `max_parallelism_level` to 1 then any operation will use only one CPU core. In that case the maximum number of queries that can be executed concurrently is equal to the number of cores in the server. More queries can be submitted at any point in time, but queries will wait for CPU core resources to become available. At present automatic queuing cannot be controlled or prioritized and operations are started in the order in which they are received.

Balance the required query response times (via the degree of parallelism you set) with the number of concurrent queries you want to run in order to determine the ideal core count for your system. Note that many applications, most notably ones that require user input such as ad-hoc query applications, don’t execute queries continuously. Users will query some data, analyze results and run more queries. As a result you can often support many more concurrent *users* than the system supports concurrent *queries* executing. Depending on the application you may be able to support 3x to 10x more concurrent users than your system can support concurrent queries.

Keep in mind that a user may submit multiple concurrent queries to the system, so every distinct, concurrent operation, independent of its origin must be counted.

How much memory do I need?

At present time it is extremely important to size memory appropriately for a VectorWise deployment. Insufficient memory allocated to VectorWise may result in out-of-memory conditions for queries which can lead to unhappy users or unmet service level agreements. VectorWise development is working to address the majority of the out-of-memory conditions in a subsequent release, due out in 2011.

Your server must have enough memory to support the Operating System (OS), VectorWise and any other applications running on the server. Whilst the OS can deal with insufficient memory by swapping memory to disk, this must be avoided as it will cause a slowdown of the system and result in poor and unpredictable performance.

VectorWise uses two memory pools:

- Execution memory: the execution memory pool must be large enough to support concurrent hash joins and aggregations against your biggest tables.
- Column buffer manager (CBM) memory: a portion of main memory that is allocated to store data (compressed).

To identify how much memory is needed, identify how much memory you need for every pool (as discussed below), then add 2 GB for the OS and memory for any other applications you plan to run on the same server. To simplify system sizing and to achieve the most predictable performance from VectorWise, you should dedicate your server to run VectorWise.

Sizing execution memory

There is no simple formula that determines the amount of execution memory needed for a query. VectorWise 1.5 does not spill hash joins or aggregations to disk which means that these operations have to be executed and thus fit in execution memory. The execution memory must be large enough to support query operations against your biggest tables and must take into account concurrent query execution.

For a large hash join the smallest side of the join must fit in memory. For example, if a query retrieves on average 30 bytes per row for the smallest side of the join and the query retrieves 100 million rows, then to complete a hash join the query will use:

```
30 * 100,000,000 = 3,000,000,000 bytes = 3 GB
```

In order for VectorWise to correctly identify which side of the join is the smallest (after applying filter criteria) it is important that table statistics are reliable. See the VectorWise 1.5 User Guide for more information about collecting and maintaining statistics.

For a large aggregation the entire (intermediate) result set must fit in memory. For example, if a query retrieves 50 bytes per row out of a 1 billion rows table (with no filter criteria), and the aggregation reduces the cardinality by a factor of 20, then the query will use:

```
50 * 1,000,000,000/20 = 2,500,000,000 bytes = 2.5 GB
```

To achieve the fastest query response times VectorWise does not compress data in execution memory.

Multiple queries running in parallel will each consume memory. Take into account query concurrency and the workload when determining the total amount of memory required for execution memory.

A query will consume as much memory as it can. If a query needs more memory than is available, it will return an out-of-memory error. Unless configured otherwise, VectorWise allocates 50% of your physical memory to execution memory by default.

Sizing the Column Buffer Manager (CBM) memory

The first question to ask when you want to determine how much CBM memory you need is whether you require the data set to be able to fit in memory. IO performance is very important to achieve maximum query performance and using main memory to store the data (in addition to disk for high availability and recovery reasons) can avoid a potential data storage layer bottleneck.

By default VectorWise keeps data compressed in CBM memory. Maximum performance can be achieved when data is stored uncompressed in memory (see the Database configuration section below), but this will increase the memory size requirement.

If your data set is significantly greater than your planned CBM memory size, then consider the size of the frequently accessed data which should ideally fit into CBM memory.

Unless configured otherwise, VectorWise will configure 25% of your available memory to CBM memory by default.

Sizing disk storage

The primary storage for your VectorWise database is disk storage. Besides the storage space requirement, it is important that your storage solution satisfies your performance requirements. Note that any component in the storage infrastructure may become a bottleneck, including individual drives, disk controllers, Host Bus Adapters (HBAs – if applicable), switches (if applicable), etc.

IO performance requirement

Depending on the query and the data VectorWise can process data at more than 1.5 GB/s per CPU core. In order to achieve this data processing rate the CPU cores have to be fed at a rate fast enough to keep them busy. If data is compressed, which it is by default, then the cost of decompression is about 25% of the CPU core's processing capacity.

Use the following formula to calculate the maximum required bandwidth:

```
 maximum bandwidth per core = core processing speed / compression ration * 0.75
```

Multiply this number by the number of cores in your system to calculate the total required disk throughput to drive the maximum bandwidth across all CPU cores.

Note, that such a high IO consumption requirement is typically found with relatively simple queries such as single-table scans with filters and aggregations. For more complex queries (e.g. performing many joins against small memory-resident tables), the IO consumption requirement may be significantly less.

Note also that the data compression ratio you achieve depends on the data types in your tables as well as the actual data. It is common to achieve 2-3x compression ratios, although both higher and lower compression ratios have been observed.

You will only achieve the maximum data processing throughput if all cores are fully occupied and all concurrently running queries are relatively simple. If you choose to configure your system to not achieve absolute maximum performance, you should know that if fully busy executing relatively simple queries the system will be IO-bound.

Spinning disks

The primary development focus for spinning disks over the last few years has been on increasing the capacity of a single disk. Data transfer rates have improved only slightly. In fact, simply due to the laws of physics larger spinning disks show a greater variability for data transfer rates between data stored on the outer slices versus the inner slices of the disk. Therefore, to achieve good consistent performance you should choose smaller rather than bigger disks e.g. choose 73 GB or 146 GB disks over 500 GB (or larger) disks.

Faster spinning disks at 15k RPM have higher throughput rates than slower 10k RPM or 7.2k RPM disks. In an ideal case a single spinning 15k RPM disk can sustain up to 120 MB/s data transfer.

Solid State Drives (SSDs)

A single Solid State Drive (SSD) can sustain up to 250 MB/s, independent of size. Although prices are coming down SSDs are still more expensive per storage unit than spinning disks and generally come in smaller capacities. For maximum performance configurations – unless most of the frequently accessed data can reside in memory – SSDs are often the best available option to drive VectorWise's data processing capacity using only internal storage, as they provide excellent bandwidth for their physical size and power consumption.

RAID configuration

Unless your entire data set fits in memory, in order to fully drive the data processing capacity of VectorWise, you must have multiple drives working in parallel. The way to do this is by using striping across multiple drives through RAID (Redundant Array of Inexpensive/Independent Disks) volumes. Use a large stripe size of 1 MB in order to optimize IOs sent by the OS down to disk.

Hardware controllers have a limited throughput capacity which may limit the IO throughput below the level you need for VectorWise. Therefore, opt for *software*-based RAID with multiple non-RAID controllers to avoid a bottleneck at a single *hardware* RAID controller.

Also consider choosing a RAID configuration that protects against a drive failure (unless you use a different approach to high availability); RAID5 or RAID6 setups are typical RAID configurations that provide a good trade-off between storage overhead, performance and availability.

Storage requirement

The minimum number of disks is determined by the amount of storage space required and the IO throughput.

You will also likely need storage space for a data staging area on a fast RAID array to load data into VectorWise.

Also be sure to take into consideration the expected database growth in conjunction with the expected challenge of extending the file system if and when you need more storage space for your database.

File system

There is no particular file system that you are required to use. You could use ext2, ext3 or xfs. Keep the following in mind:

- Some file systems have a maximum file size e.g. ext3 has a 2TB limit, and VectorWise 1.5 stores its entire database in a single file.
- Do not use ext4 – as of December 1st 2010 there is a known bug that can cause data corruption.
- Choose an xfs filesystem if you need maximum flexibility for your file size.

Guideline

To configure a completely balanced system that can deliver maximum performance when all CPU cores are busy you should plan for:

- 4 (or more) 15k RPM disk per CPU core
- 2 (or more) SSDs per CPU core

DATABASE CONFIGURATION

VectorWise configuration is on a per-database basis. This section discusses considerations for a few database settings that are commonly set or modified.

The VectorWise software installation process generates an initial vectorwise.conf configuration file that includes several configuration parameters with default settings based on your system's size. Configuration parameters that are not explicitly included in the configuration file assume a default value. Please refer to the VectorWise 1.5 User Guide for details and default values to see if it makes sense for you to change the default values.

By default, vectorwise.conf settings apply for every database you create (you may change this – see the VectorWise 1.5 User Guide for details). As such and because VectorWise dedicates a process per database, you must take into account the total sum of resources required for each database. For example, VectorWise by default allocates 50% of physical memory to its execution memory. This is on a per-database basis. If you actively use three databases, the assumption would then be 150% of memory, which is not reasonable. In this case make sure to set the execution memory size for each database small enough, so that if used all together at the same time, they consume a reasonable amount of memory.

Access to data

When data is accessed and is not in CBM memory, the data is read compressed from the database into memory (RAM) before it is decompressed and processed using the CPU cache as the processing memory.

Large block IO

VectorWise always performs large block IOs to read data from the disk. Large IOs are most efficient if the data for a single IO is stored together. The `block_size` parameter in the vectorwise.conf configuration file identifies the size of the block (page). The default value is 512 KB.

Typical data warehouse queries scan through very large volumes of data. If subsequent data blocks have to be accessed then the most efficient way to retrieve the data is if the blocks are stored together on disk. This is particularly important for spinning disks, where the cost of moving the disk head is expensive and sequential IO delivers at least 2 times more throughput than random IO. VectorWise uses the parameter `group_size` to control the number of blocks (pages) that are stored together. The default value is 8.

For VectorWise 1.5 the parameters `block_size` and `group_size` cannot be changed after the database has been created. If you want to change these parameters after database creation it is necessary to unload/destroy/recreate/reload the database.

Data allocation

By default VectorWise 1.5 uses a pure column-based data store for most situations[1] so data is allocated per column. In a pure column-based data store the minimum size that a table will occupy on disk, irrespective of the amount of data in the table, is:

```
number of columns * block_size * group_size
```

Take this into consideration if your database has a large number of table columns, due to the number of tables and/or the number of columns per table.

You can reduce the minimum allocation by reducing the parameters `group_size` and/or `block_size`, at the cost of slower performance for large table scans.

VectorWise 1.5 also introduces experimental functionality for a storage allocation mechanism that stores entire rows of data in a single data block. Within the block data is still stored column-by-column in order to optimize data compression. Consider using row-based storage for extremely wide tables with relatively few rows in order to limit storage allocation or for tables with relatively few columns for which queries always retrieve the majority of the columns. The row-based storage approach is used if you use `WITH STRUCTURE = VECTORWISE_ROW` at the end of a create table statement.

VectorWise always performs IO on a block by block basis. Unnecessary IO is performed if the block is not full (e.g. there is not enough data to fill a block after compression) or if data is retrieved that is not required to satisfy the query (e.g. when using the row-based storage mechanism but not all columns are required in the query). Choose your storage allocation to optimize IO for maximum performance.

Column Buffer Manager (CBM) memory

The Column Buffer Manager (CBM) memory is a portion of the server's memory (RAM) that is used to store data that was pre-fetched from disk. A large CBM memory size can improve query performance as only data not yet in CBM memory will have to be read from disk in order to satisfy the query.

[1] The exception to the default is indexed tables with multi-column indexes. All indexed columns are always stored together in a block.

Use the vectorwise.conf initialization parameter `bufferpool_size` to indicate the size of CBM memory. By default 25% of the server's memory will be allocated for CBM memory.

Data in CBM memory mirrors data on disk i.e. if the data resides compressed on disk then it will be compressed in CBM memory to make optimum use of the memory buffer. Data will only be decompressed when it is taken out of CBM memory to be processed. If you want to avoid the small cost of decompression then you can choose to store data uncompressed by using the command `set trace point qe82` before you create a table. Note that an uncompressed table may take up more disk space and hence may take longer to read from disk, so you should only use this for small or medium size tables.

Database file size

VectorWise 1.5 automatically extends the size of the database file if there is not currently enough space available in the file.

The database file is used for two purposes:

1) Store data: by default data is compressed on a block-by-block basis, but storage is always allocated in blocks. As a result the amount of space occupied by the database simply to store the data may be larger than the amount of source data if the storage allocation is non-optimal (possibly because of partially full blocks). For a typical reporting or data warehouse database such a situation should be rare.
2) Temporary storage: large sort operations that would otherwise run out of memory are spilled to disk and use space in the database file. While this use can be significant the space is released back for data storage after the operation has completed. Temporary space is also used during execution of a `COMBINE` statement (see below) for up to as much of a full copy of the table on which the statement operates.

Once the database file has grown to a certain size it will never shrink. However, available space within the file is always reused, both for permanent and for temporary purposes, before the file is extended again.

If you have to shrink the database file then you must unload, destroy, recreate and reload the database.

SCHEMA DESIGN

There is no need to design a specialized schema to work with VectorWise. VectorWise does not require materialized views or projections to deliver exceptional query performance so there is no need to design these. You may choose to use a star schema or a snowflake schema, but you may also choose these other extreme cases that are often not feasible with other relational databases:

- Denormalize all columns into a single wide table.
- Don't make any changes to the schema you currently use and just load the data and run queries against it.

VectorWise's vectorised in-cache processing benefits computations as well as table joins. Other data access optimizations VectorWise transparently provides include:

- Column-based storage: queries only access relevant table columns.
- Min/max indexes: based on the query's filter criteria, either explicit as part of the query definition, or implicit based on table joins or sub-select statements, VectorWise identifies candidate data blocks that must be read from disk. If there is a correlation between the order in which data is added to the database and the data values (e.g. time-based data is an obvious example) then the min/max indexes may filter large amounts of data blocks very efficiently. In extreme cases min/max indexes provide the same performance benefit that partition elimination provides for other databases.
- Compression: by default, data is stored in a compressed format so that it takes less time to read the data from disk.

The combination of column-based access and min/max indexes can make queries against a single large denormalized table extremely efficient for some workloads. Because of efficient data compression, the storage overhead caused by denormalization may be minimal depending on the data.

Indexing

VectorWise does not rely on indexes to achieve good performance. In fact, most reporting and data analysis workloads don't benefit from explicitly indexed tables so for these scenarios you should not create indexes.

VectorWise does allow you to index a table. An indexed table stores data physically ordered on the index definition. You may know this as an index-organized table or clustered index. With VectorWise 1.5 you can only create one index per table. Internal min/max indexing is always created for every column and doesn't affect whether or not an explicit index is created on the table. Only if a table is predominantly accessed using a filter or join against a particular column or set of columns then an index on that column or set of columns may result in a performance gain.

Indexes also affect query execution plans, causing the selection of a sort merge join over a hash join when both tables in the join are indexed on the join key. The sort merge join uses less memory than the hash join and is more efficient because data is already sorted.

Creating an index on a table will introduce considerations on update operations and can increase the memory utilization to load and update data. Refer to the subsequent section on Data loading in this document for more details.

Constraints

A database for data analysis is typically loaded through a controlled process that may be used to validate the data. As a result, from a data integrity point of view, there should be no need to define primary, unique or foreign key constraints at the database level.

On the other hand constraints do provide extra information to the optimizer which may result in better query execution plans. If data constraints do not cause issues with your data loading strategy then it is better to define constraints than not.

VectorWise does allow you to create constraints, but at present time VectorWise does not allow dropping them. To remove a constraint the table has to be unloaded, dropped, recreated and then reloaded.

DATA LOADING

Databases for reporting and data analysis are often loaded incrementally using large data sets at the time. VectorWise 1.5 works well in such an approach, so that the database can optimize space allocation and maximize data compression. This section discusses best practices for loading data.

Initial data load

In order to initially load your data efficiently into VectorWise you should always use a bulk approach. This section discusses various methods for the initial load.

copy statement

If your data resides in flat files then use of the copy command is a flexible and efficient way to bulk load the data into VectorWise.

The syntax for the copy command may be different than the syntax for other bulk loaders you are used to. Below is an example of using the copy statement. To create the table:

```
CREATE TABLE region (

        r_regionkey INTEGER not null,

        r_name CHAR(25) not null,

        r_comment VARCHAR(152) not null)

\g
```

To load the following set of data in a data file region.tbl:

```
0|AFRICA|Continent of the elephants
1|AMERICA|Both North and South America
2|ASIA|Where tigers live
3|EUROPE|Many languages are spoken here
4|MIDDLE EAST|Sunny and very warm
5|AUSTRALIA|For goodness sake, please don’t leave us out
```

Use the following copy statement:

```
copy table region (

        r_regionkey = 'c0|',

        r_name = 'c0|',

        r_comment = 'c0nl'

)

from 'region.tbl' \g
```

In the above example, 'c0|' represents a free format character string (versus fixed length) followed by a pipe delimiter, and 'c0\n' a free format character string followed by the newline at the end of the line.

The copy statement also supports the commonly used comma or semicolon separated files that use double quote enclosures and the backslash as the escape character. Please refer to the VectorWise User Guide for more examples, and the Ingres Command Reference Guide for complete syntax and options for the copy statement

At present copy only supports date, time and timestamp fields formatted using the ANSI standard:

- yyyy-mm-dd for date fields unless you use II_DATE_FORMAT to override the default,
- hh24:mi:ss for time fields,
- yyyy-mm-dd hh24:mi:ss.ffffff for timestamp fields.

Handling data load issues

The copy command attempts to load the data file using the parameters you provide when you issue the statement. If the load fails for whatever reason, for example due to a bad record, then the default behavior is to stop processing and rollback the entire statement.

To diagnose data load issues and identify the data load problem you should use the options copy provides in the with-clause. Use ON_ERROR = CONTINUE and LOG = 'filename' to continue the load when hitting an issue, logging bad records to the file 'filename'. Include ERROR_COUNT if you want to stop after a certain number of errors.

For example:

```
copy table region (
        r_regionkey = 'c0|',
        r_name = 'c0|',
        r_comment = 'c0nl'
)
from 'region.tbl'
with on_error = continue
, log = 'region_bad_records.log' \g
```

If you see unexpected data load issues please report these to Ingres directly or on the VectorWise forum at http://community.ingres.com/forum/vectorwise/.

iivwfastload - experimental

Another way to load file-based data into VectorWise is through the use of the iivwfastload utility. iivwfastload bypasses SQL and loads directly into VectorWise. This approach can be faster than the use of the copy command, in particular if there are many empty column values in the data file.

At present the iivwfastload utility is experimental and has a number of restrictions:

- The utility must run on the database server. Due to security reasons there is no remote iivwfastload utility.
- iivwfastload does not support the use of enclosures or escape characters.
- Date, time and timestamp fields must be formatted using the ANSI standard in the data file.
- File components must match the number and order of the target table's columns.

To load the region table in a database called dbt3 as shown in the example under copy can be done using the following statement:

```
 iivwfastload -database dbt3 -table region \
-datafile region.tbl -fdelim '|' -rdelim '\n'
```

INSERT/SELECT

In addition to create table as select and INSERT/SELECT between VectorWise tables VectorWise 1.5 also supports these statements between Ingres and VectorWise tables (bidirectionally).

To move data into VectorWise tables INSERT/SELECT will efficiently append data to the VectorWise tables.

JDBC batch inserts

VectorWise 1.5 introduces support for JDBC batch statements. Through the JDBC batch interface inserts are internally transformed into efficient data appends. From a data storage perspective applying a large transaction that (amongst others) consists of many inserts into the same table is as efficient through the JDBC batch interface as it is through an intermediate staging step on the file system followed by a bulk load using the copy statement or iivwfastload.

Third-party tools

As an alternative to writing programs and scripts to unload data followed by using copy or iivwfastload to load the data into VectorWise tables, you can use third-party tools that support VectorWise “bulk” loading, for example: Ingres High Volume Replicator (HVR), Pentaho and Talend all support the “bulk” load operation.

Incremental data load

The most efficient way to incrementally load subsequent data into VectorWise 1.5 is through the use of incremental bulk data operations e.g. copy, or through the JDBC batch interface. You can also apply single-row DML statements using regular insert/update/delete commands, but these incremental “batch” operations are much less efficient than incremental ”bulk” operations. See the VectorWise User Guide, Chapter 11: Methods for Updating Data, concerning batch versus bulk data loads.

To insert additional data efficiently you can choose one of the following approaches:

- Use the copy command (or experimental iivwfastload utility) to load from data files and append the data to the table. If data is already present in the table and the table is indexed then using this method data changes will be loaded into memory and not directly written to disk. See the section DML and memory utilization below for considerations on incrementally loading indexed tables.
- Use the JDBC batch interface to load data directly through a JDBC-based application. If data is already present in the table and the table is indexed then using this method data changes will be loaded into memory and not directly written to disk. See the section DML and memory utilization below for considerations on incrementally loading indexed tables.
- Use insert into <table> as select <columns> from <other table>. If data is already present in the table and the table is indexed then using this method data changes will be loaded into memory and not directly written to disk. See the section DML and memory utilization below for considerations on incrementally loading indexed tables.
- Use the COMBINE statement to add the data to the table. If your table is indexed and already contains data, then the use of the COMBINE statement is the only way to append the data directly to the table.

Both the use of the COMBINE statement and the use of insert as select require you to create a staging table. When you use insert as select then the staging table can be an Ingres table or a VectorWise table. When you use the COMBINE statement then all tables involved must be VectorWise tables.

COMBINE statement

The COMBINE statement is processed against a table and performs the following operations:

- Writes the data from all completed, cached, insert/update/delete statements against a table that reside in CBM memory to disk and optimizes the table layout on disk.
- Performs the transactions requested in the arguments to the COMBINE statement as bulk operations.

The use of COMBINE is also the most efficient way to perform updates and deletes to data.

Because the COMBINE statement will rewrite the entire table it can be an expensive operation, both from a time and a storage utilization point of view. Also COMBINE can use a significant amount of temporary space within the database file for data operations e.g. sorting. Altogether this extra space utilization can be 2 or more times higher than the size of all the tables participating in the COMBINE.

For more information on the use of COMBINE, including practical examples, please refer to the VectorWise User Guide.

DML and memory utilization

Any DML that is not a data append through the bulk loader, insert as select, or the result of executing the COMBINE statement will be stored per table in an optimized in-memory structure called the Positional Delta Tree (PDT). This includes all DML except for COMBINE executed against an indexed table that already has data in it.

VectorWise supports multi-version read consistency. Committed data in PDTs is efficiently merged with data read from disk to answer queries. Any data that is committed to PDTs is also written to the transaction log and persisted on disk for recovery purposes.

Every VectorWise database reserves a portion of the server's memory to store PDTs. The amount of memory reserved is controlled by the parameter `max_global_update_memory` as an upper bound of query execution memory that can be used for PDTs. The default is 0.5%.

When the amount of memory used by PDTs exceeds the available memory pool VectorWise automatically triggers a COMBINE against all tables with outstanding transactions. Depending on the size of the tables this can be a very costly operation and it may occur at an inopportune time.

Avoid running out of PDT memory by planning a data loading strategy using a combination of the following strategies:

- Append data whenever possible.
- Use the COMBINE statement whenever possible to delete or update data.
- Explicitly call the COMBINE statement against a table with outstanding DML at a time when the system has surplus resources available to perform the expensive operation.

Moving window of data

A moving window of data is very common for fact tables in a data warehouse or data mart. For example: you want to keep the most recent 36 months of data on-line. There are 3 approaches to achieve a moving window data loading strategy with VectorWise:

1) Append data directly to the fact table in VectorWise as it arrives. E.g. add data daily or multiple times per day. This approach assumes that the fact table is not indexed.

   On a monthly basis, create a staging table by selecting the rows from the fact table for the month you want to purge (create table … as select …) and then use the COMBINE statement to remove the data from the original table.

   ```
   CALL VECTORWISE (COMBINE 'basetable - staging')
   ```

   Then drop the staging table.

   The COMBINE operation will rewrite the entire fact table containing 36 months worth of data using a lot of storage resources. Consider performing this operation when few users access the system to minimize the impact.

2) Append data to staging table STAGING1 rather than the base fact table as it arrives. E.g. add data daily or multiple times per day. This approach assumes that staging table STAGING1 is not indexed. Create a view to union (all) the results of staging table STAGING1 and the base fact table.

   On a monthly basis, create a second staging table STAGING2 by selecting the rows from the fact table for the month you want to purge (create table … as select …) and then use the COMBINE statement to add the data in staging table STAGING1 and remove the data from the original table which is in staging table STAGING2.

   ```
   CALL VECTORWISE (COMBINE 'basetable + staging1 - staging2')
   ```

The COMBINE operation will rewrite the entire fact table containing 36 months worth of data using a lot of storage resources. Consider performing this operation when few users access the system to minimize the impact.

Drop staging table STAGING2.

Then remove the data from staging table STAGING1:

```
CALL VECTORWISE (COMBINE ‘staging1 - staging1’)
```

3) Use multiple tables with the same structure to represent all data for the fact table (e.g. one table per month – 36 tables total for 36 months of data).

   Create a view to union (all) the results of all tables and present the data for all tables as a single fact table view to an application. Add data as it arrives to the most recent month of data. If the table is indexed then you have to use COMBINE to add the data, but since it is only a month’s data the overhead will be limited. On a monthly basis drop or truncate the oldest table.

The first and second approaches are easier, but take up a lot of time and storage resources to rewrite the table when using the COMBINE statement. The first approach cannot be used if the fact table is indexed. The third approach utilizes storage space more efficiently but requires more maintenance (which can be automated). In addition the second approach may in some cases result in different, less optimal query execution plans.

HIGH AVAILABILITY

Any database may become unavailable due to a variety of planned and unplanned outages. Planned outages include system and software upgrades which as they are planned, should have little or no impact on the users. It is prudent to pursue a high availability strategy.

This section discusses strategies to minimize downtime due to unplanned outages. Examples of unplanned outages include:

- Hardware problems including server or storage failures.
- Data corruption issues due to hardware or software problems.
- System outages due to power or network failures.
- Data issues due to data or operational errors.

The primary goal of high availability is to ensure you can recover your system after an outage. The secondary goal is to minimize the time of any outages. The strategies discussed below attempt to minimize if not prevent the impact of unplanned outages.

Also, make sure you have a plan for outage prevention. Test both your prevention and recovery plans, so that you to know that they will work and that you are ready, if and when an actual unplanned outage occurs.

Hardware protection

There are multiple technologies to safeguard your hardware from unavailability should a component fail. Examples include:

- Use a backup power supply to protect the server from a power failure.
- Bond multiple network ports or use a software-based solution to failover network connectivity to surviving ports.
- Use a RAID configuration that provides protection against disk failure, such as RAID5 or RAID6.
- Use a separate standby system, ideally in a different, remote location. See Standby Configuration in this document.

Backup and restore

A database backup enables you to go back and restore a database to a point in time. Doing so protects you from data issues and/or corruptions that you may have had since the last backup. In addition, you can restore the database on different hardware in case the current hardware has undergone a complete and unrecoverable system crash. A database restore will cause an outage that will take time to perform, how long depending on the data size and speed of your system.

The design of your backup strategy starts with an analysis of the risk of an outage and the amount of time you can afford the system to be unavailable. In addition, consider the cost of the backup solution itself.

VectorWise 1.5 supports full backups but with no concurrent DML or DDL activity. VectorWise 1.5 does not support incremental backups.

The following sections describe various backup strategies. In some cases it makes sense to use a combination of these approaches for optimum high availability.

Full operating system database backup

A full Operating System backup takes a backup of the database file. The advantage of a full backup is that it is self-contained and simple to understand. Disadvantages to a full backup are the amount of storage space required to store the backup and the lack of options to restore only a subset of the database.

To backup the database:
- Stop the application(s) accessing the database.
- Use COMBINE to flush any insert/update/delete in CBM.
- Shutdown the database.
- Copy the database file and configuration files using tar or cpio or something else to a safe place.
- Startup the database.
- Resume the application(s) accessing the database.

To restore the database
- Stop the application(s) accessing the database.
- Shutdown the database.
- Copy the database file and configuration files using tar or cpio or something else from the safe place.
- Startup the database.
- Resume the application(s) accessing the database.

If the server has completely and unrecoverably crashed then you may have to reinstall the VectorWise software before you can restore the database.

Ingres copydb/unloaddb database backup

An ingres copydb or unloaddb will create scripts with database object definitions and commands to unload the database or individual tables. The advantage of this approach is that it is self-contained and simple to understand. In addition you can backup a subset of the database. Disadvantages to this backup mechanism are the amount of storage space required to store the resulting data files and the time to perform the unload and reload of the database.

To backup the database:
- Stop the application(s) accessing the database.
- Run copydb or unloaddb and the associated scripts on the database.
- Resume the application(s) accessing the database.

To restore the database:
- Stop the application(s) accessing the database.
- Either drop the database objects, or destroy/create the database.
- Run the scripts to reload the database.
- Resume the application(s) accessing the database.

For more information, see the Ingres documentation on copydb and unloaddb.

Ingres database backup utilities (ckpdb and rollforwarddb)

Alternatively the Ingres ckpdb utility can be used to take the backup, and rollforwarddb to restore and recover the database. Please refer to the VectorWise User Guide on how to use these utilities.

Partial database backup

If you want the option of a partial database restore then you have to take a partial database backup. For example, if you have to perform some data maintenance on a table and you want to make sure that you can revert to the data set prior to the change, then you should take a backup of the table's data set. Use create table as select from to store a copy of the data in a different table, or use the copy command to export the data from the table into a file. To restore the table, use the COMBINE command to delete all the data from the table:

```
CALL VECTORWISE (COMBINE 'basetable – basetable')
```

Then, use insert as select from to reinsert the data, or the copy command to read the data from the file back into the table.

Warning: if you drop the table rather ran use the COMBINE statement, any dependent object will be affected, e.g. views, grants, constraints.

Database reload

Your backup strategy may be to reload the database from the data source(s). Such a scenario requires that the source data sets are still available for your entire database.

Hybrid approach: full backup with incremental reload

A full database backup is the easiest way to backup your VectorWise 1.5 database. However your database may be terabytes in size and as a result, depending on the hardware you use, the backup may take quite some time. A full database backup will also take up a significant amount of storage space.

Data warehouses commonly use a strategy of a regular (e.g. weekly, monthly) full database backup and store incremental data loads (e.g. data files) until the next full backup. A database restore and recovery consists of restoring the most recent full database backup followed by a replay of the data loads since the most recent full backup.

Standby configuration

The best protection against a variety of both planned and unplanned outages is the use of a standby configuration. The standby configuration should be located in a different data center than your primary system, ideally far away from the primary system. This provides protection against disasters such as flooding, earthquakes and large power or network outages.

The simplest way to implement the standby configuration is to duplicate the primary system and use the same load routines to populate the standby system. Such an implementation minimizes the risk of data corruption caused by replication technologies, as well as the risk of an accident. E.g. the DBA thought that he/she was connected to the test environment when emptying a table but was in fact connected to the production environment. If replication or mirroring were being used this would result in the data deletion being replicated/mirrored to the backup machine which would negate the disaster recovery strategy. At this point it would be necessary to recover from a backup which results in an outage waiting for restore and recovery to happen.

MANAGING AND MONITORING VECTORWISE

At a high level there are two ways to manage VectorWise: remotely using the Ingres DBA tools, and locally on the server.

At present the Ingres DBA tools are only available on the Windows platform, look somewhat outdated, and provide only minimum management capabilities. Ingres is working on a new revision of the tools for which a first version is due out in 2011.

Management and monitoring on the server provides richer and more specific information, but is only available in command-line mode. Going forward VectorWise will continue to improve monitoring capabilities.

DBA tools

VectorWise 1.5 installs by default using a UTF8 character set. In order for the DBA tools to connect to VectorWise 1.5 the installation must also use a UTF8 character set. If the character set for the DBA tools installation is not set to UTF8 then you can do so using the following steps:

- Open the Ingres DBA tools command prompt.
- Run the command ingprenv, and verify the value for II_CHARSET<II>, where <II> is your installation identifier.
- Run the command ingsetenv, and set the value for II_CHARSET<II> to UTF8.
- Start (or restart) the Ingres DBA tools installation using ingstop and ingstart, or using the Ingres Service Manager.
- Run the tools to connect to the VectorWise database.

The Ingres Visual DBA provides visibility into schema and table definitions as well as users, groups and roles. Visual DBA provides limited capabilities to make configuration changes.

iivwinfo utility

The iivwinfo utility is a command-line utility that must be run on the database server. Following is information the utility provides. Please refer to the VectorWise User Guide for detailed information about the iivwinfo utility.

Statistics

The default mode for iivwinfo is to retrieve statistics (or use the –s option). The statistics provide insight in memory and storage utilization as well as the number of sessions. For example:

```
vectorwise@vanma01-m6400:~$ ./iivwinfo -s fmvw

./iivwinfo
Using settings:
database   : 'fmvw'
table      : ''
Connecting to VW server at port '44216'
Executing query
+------------------------------------+----------------------------------------+
|stat                                |value                                   |
|varchar(36)                         |varchar(44)                             |
+------------------------------------+----------------------------------------+
|memory.query_allocated              |47318944                                |
|memory.query_maximum                |8589934592                              |
|memory.query_virtual_allocated      |47482112                                |
|memory.query_virtual_maximum        |70368744177664                          |
|memory.update_allocated             |0                                       |
|memory.update_maximum               |2147483648                              |
|memory.bufferpool_allocated         |206569472                               |
|memory.bufferpool_maximum           |4294967296                              |
|bm.block_size                       |524288                                  |
|bm.group_size                       |8                                       |
|bm.columnspace_total_blocks         |262144                                  |
|bm.columnspace_free_blocks          |253526                                  |
|bm.bufferpool_total_blocks          |8192                                    |
|bm.bufferpool_free_blocks           |8192                                    |
|bm.bufferpool_used_blocks           |0                                       |
|bm.bufferpool_cached_blocks         |394                                     |
|bm.columnspace_location
/u01/vw1.5/ingres/data/vectorwise/fmvw/CBM/default/0|
|system.active_sessions              |1                                       |
|system.log_file_size                |1908426                                 |
|system.threshold_log_condense       |33554432                                |
+------------------------------------+----------------------------------------+
(20 rows)

vectorwise@vanma01-m6400:~$
```

Database configuration

Use iivwinfo –c <db name> to retrieve database configuration information. Below is an example of the beginning of iivwinfo –c.

```
vectorwise@vanma01-m6400:~$ ./iivwinfo -c fmvw

./iivwinfo
Using settings:
database   : 'fmvw'
table      : ''
Connecting to VW server at port '44216'
Executing query
+-----------------------------------------------------+----------------+
|config                                               |value           |
|varchar(53)                                          |varchar(60)     |
+-----------------------------------------------------+----------------+
|cbm.append_fill_threshold                            |0.500000        |
|cbm.append_max_tuples_to_recompress                  |1048576        |
|cbm.append_reuse_last_block                          |true           |
|cbm.auto_pax_storage                                 |false          |
|cbm.block_size                                       |524288         |
...
```

Table block usage

The –tbu option can be used to obtain insight in the size of tables on disk. Multiply the block_count output with the block size to know the storage utilization. For example:

```
vectorwise@vanma01-m6400:~$ ./iivwinfo -tbu fmvw

./iivwinfo
Using settings:
database   : 'fmvw'
table      : ''
Connecting to VW server at port '44216'
Executing query
+-------------------------+--------------------+
|table_name               |block_count         |
|str                      |slng                |
+-------------------------+--------------------+
|account                  |                   6|
|calendar                 |                   3|
|category                 |                   4|
|currency                 |                   4|
|customer                 |                  29|
|days                     |                   2|
|department               |                   2|
|employee                 |                  17|
|employee_closure         |                   3|
|expense_fact             |                   7|
|position                 |                   6|
|product                  |                  15|
|product_class            |                   5|
|promotion                |                   7|
|region                   |                   7|
|reserve_employee         |                  16|
|salary                   |                   8|
|salesdetails             |                6749|
|shipmethod               |                   2|
|store                    |                  24|
|time_by_day              |                  11|
|warehouse                |                  15|
|warehouse_class          |                   2|
|sales                    |                1674|
+-------------------------+--------------------+
(24 rows)

vectorwise@vanma01-m6400:~$
```

About Ingres Corporation

Ingres develops and markets the leading open source enterprise-grade relational database and a breakthrough analytic database, VectorWise, designed to support the growing need for analytics and business intelligence. VectorWise customers claim performance improvements of up to 70X that of comparable databases, without the traditional overhead. The product is suitable for SMBs and enterprises and as an embedded database for ISVs and SaaS providers. VectorWise is quickly becoming a leader in interactive reporting and analysis for data-driven companies. More information is available at www.ingres.com

Ingres Corporation	Ingres Europe Limited	Ingres Germany GmbH	Ingres France	Ingres Australia
500 Arguello Street, Suite 200	215 Bath Road	Ohmstrasse 12	7C Place Du Dôme	Level 8, Suite 1
Redwood City, California 94063	Slough, Berks SL 1 4AA	63225 Langen	Immeuble Elysées La Défense	616 St. Kilda Road
USA	United Kingdom	Germany	92056 Paris La Défense Cedex	Melbourne, Victoria, 3004
Phone: +1.650.587.5500	Phone: +44 (0) 1753 559550	Phone: +49 (0) 6103.9881.0	France	Australia
			Phone: +33 (0) 1.72.75.74.54	Phone: +61 3 8530.1700

For more information, contact ingres@ingres.com