

# VectorWise 1.5

---

## User Guide



VW-15-UG-06

This Documentation is for the end user's informational purposes only and may be subject to change or withdrawal by Ingres Corporation ("Ingres") at any time. This Documentation is the proprietary information of Ingres and is protected by the copyright laws of the United States and international treaties. It is not distributed under a GPL license. You may make printed or electronic copies of this Documentation provided that such copies are for your own internal use and all Ingres copyright notices and legends are affixed to each reproduced copy.

You may publish or distribute this document, in whole or in part, so long as the document remains unchanged and is disseminated with the applicable Ingres software. Any such publication or distribution must be in the same manner and medium as that used by Ingres, e.g., electronic download via website with the software or on a CD-ROM. Any other use, such as any dissemination of printed copies or use of this documentation, in whole or in part, in another publication, requires the prior written consent from an authorized representative of Ingres.

To the extent permitted by applicable law, INGRES PROVIDES THIS DOCUMENTATION "AS IS" WITHOUT WARRANTY OF ANY KIND, INCLUDING WITHOUT LIMITATION, ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NONINFRINGEMENT. IN NO EVENT WILL INGRES BE LIABLE TO THE END USER OR ANY THIRD PARTY FOR ANY LOSS OR DAMAGE, DIRECT OR INDIRECT, FROM THE USER OF THIS DOCUMENTATION, INCLUDING WITHOUT LIMITATION, LOST PROFITS, BUSINESS INTERRUPTION, GOODWILL, OR LOST DATA, EVEN IF INGRES IS EXPRESSLY ADVISED OF SUCH LOSS OR DAMAGE.

The manufacturer of this Documentation is Ingres Corporation.

For government users, the Documentation is delivered with "Restricted Rights" as set forth in 48 C.F.R. Section 12.212, 48 C.F.R. Sections 52.227-19(c)(1) and (2) or DFARS Section 252.227-7013 or applicable successor provisions.

Copyright © 2011 Ingres Corporation. All Rights Reserved.

Ingres, OpenROAD, and EDBC are registered trademarks of Ingres Corporation. All other trademarks, trade names, service marks, and logos referenced herein belong to their respective companies.

# Contents

---

<b>PART 1: Concepts</b>	<b>9</b>
<b>Chapter 1: Introducing VectorWise</b>	<b>11</b>
What Is VectorWise?.....	11
VectorWise Technology .....	11
Vectorised Processing—Calculating Query Answers Fast.....	12
Storage Innovations—Beating the Disk Bottleneck .....	13
How VectorWise Differs from Traditional Ingres .....	14
VectorWise Storage Type.....	14
Ingres Database Version .....	14
Purpose of This Guide .....	15
Path Notation in This Guide .....	15
Your Support Options .....	15
<b>Chapter 2: VectorWise Concepts</b>	<b>17</b>
VectorWise Table Structures.....	17
Data Storage Format.....	18
Raw Table Storage Format.....	18
Data Compression .....	19
Querying Data .....	21
Statistics for Query Optimization .....	21
Data Manipulation .....	21
Granularity of DML Operations .....	21
Snapshot Isolation.....	22
Persistence and Recovery .....	22
Management of In-memory Updates .....	22
Transaction Isolation Model .....	23
Snapshot Isolation.....	23
Optimistic Concurrency Control .....	23
Transaction Examples .....	24
Transactions and DDL Operations .....	25
<b>Chapter 3: VectorWise Features</b>	<b>27</b>
VectorWise Storage Structures .....	27
Default Database Characteristics.....	28
Transaction Management .....	28
Update Restrictions .....	28

---

SQL Statements.....	29
SQL Data Types .....	30
SQL Functions.....	30
Command and Utility Summary .....	32
New Features in Version 1.5 .....	32
Unsupported Features.....	34
Unsupported SQL Statements .....	34
Unsupported Data Types .....	34
Unsupported SQL Functions .....	35
Restrictions and Known Incompatibilities.....	35
SQL Cursors.....	35
Some Nested Queries Do Not Work.....	36
Cross-system Queries .....	36
Host Variables in REPEATED Queries .....	36
SQL Data Type Incompatibilities .....	36
SQL Function Incompatibilities .....	37
Differences in Constraint Checking.....	37
Data Size Limitation for Some Queries .....	37

## **Chapter 4: Developing Applications** **39**

Application Programming Languages .....	39
Application Programming Interfaces .....	39
SQL Overview .....	40
Connectivity Drivers .....	41

## **Chapter 5: Understanding Usage Scenarios** **43**

Typical Use Scenarios .....	43
Reporting on DBT-3 Database Benchmark .....	43
How to Run the Benchmark .....	44
Generate Data.....	45
Start VectorWise .....	45
Create a Database .....	46
Create a VectorWise Table .....	46
Bulk Load the Data into the Table.....	47
Create Statistics .....	48
Run a Reporting Query .....	49

---

## **PART 2: Installation** **51**

### **Chapter 6: Recommended Platforms** **53**

RAID File System .....	53
Hardware versus Software RAID .....	53
Database Location and Linux Symbolic Link .....	54
Supported CPUs .....	55
Disk Subsystems .....	55
Guidelines for a Balanced Platform .....	56

### **Chapter 7: Installing VectorWise** **59**

Download VectorWise .....	59
Installation Considerations .....	59
VectorWise Data Location (II_VWDATA) .....	60
Character Set .....	60
Installing VectorWise for Linux .....	61
Packages That Can Be Installed .....	61
Install libaio .....	62
Install VectorWise Using the Installation Wizard on Linux .....	63
ingres_express_install Command—Install VectorWise Quickly .....	66
Install VectorWise Using RPM Commands .....	68
Install VectorWise Interactively Using ingbuild .....	70
How You Access the Instance on Linux .....	71
Response File—Define Configuration for VectorWise Instance .....	72
Upgrade to VectorWise 1.5 .....	73
Post-installation Tasks .....	73
How You Safely Uninstall VectorWise .....	74
View Installation Environment Settings .....	75

### **Chapter 8: Accessing VectorWise** **77**

Startup and Shutdown .....	77
Start VectorWise as a Service .....	78
Network Connections .....	78

## **PART 3: Administration** **79**

### **Chapter 9: Configuring and Managing VectorWise** **81**

Configuration File (vectorwise.conf) .....	81
Configuration Options .....	82
[memory] Settings .....	82

---

[system] Settings .....	83
[cbm] Settings .....	85
[engine] Settings.....	86
When to Change the Default Configuration Values .....	87
Memory Settings.....	88
Memory Configuration Guidelines.....	89
I/O Settings .....	89
View Information about a Database with VectorWise Tables .....	90
Using Large Pages.....	94
Requirements for Huge Pages in Linux .....	94
Designate Memory for Huge Pages on Linux.....	95
Using Multiple Databases .....	96
Error Reporting—Database Log File .....	96
VectorWise SQL Settings.....	97
Performance Tips .....	98

## **Chapter 10: Creating a Database 101**

How to Create a Database.....	101
Create a Database.....	102
Database Default Configuration .....	102
Connect to a Database.....	103
Create a Table .....	104
Create an Index .....	105
Methods for Loading Data .....	105
Load Data with COPY Statement.....	106
Load Data with iivwfastload Utility .....	106
Create Statistics with Optimizedb .....	107

## **Chapter 11: Managing the Database 109**

Methods for Updating Data .....	109
Batch and Bulk Updates.....	109
Combining Tables .....	112
How to Avoid Propagation.....	113
COMBINE Command—Merge and Update Data .....	113
Best Practices for Updates .....	116
Methods for Backing Up and Restoring Data .....	117
Backup and Recovery with Checkpoints .....	117
Copy Files to a Backup Location.....	120
Copy the Database with Copydb .....	121

---

## **PART 4: Reference** **123**

### **Appendix A: SQL Reference** **125**

ALTER TABLE .....	125
CALL VECTORWISE.....	126
COMMIT .....	126
COPY .....	127
CSV and SSV Delimiters .....	127
CREATE INDEX .....	128
CREATE TABLE.....	129
CREATE VIEW .....	130
DECLARE GLOBAL TEMPORARY TABLE .....	130
DELETE .....	131
DROP.....	131
INSERT .....	131
ROLLBACK.....	132
SELECT (Interactive) .....	132
SELECT (Embedded).....	135
UPDATE .....	136

### **Appendix B: Command Reference** **137**

iivwfastload Command—Load Data into VectorWise Tables .....	137
iivwinfo Command—Display Information about a Database with VectorWise Tables .....	138

### **Appendix C: VectorWise Limits** **139**

VectorWise Limits.....	139
------------------------	-----

## **Index** **141**





# PART 1: Concepts

---



# Chapter 1: Introducing VectorWise

---

This section contains the following topics:

[What Is VectorWise?](#) (see page 11)

[VectorWise Technology](#) (see page 11)

[How VectorWise Differs from Traditional Ingres](#) (see page 14)

[Ingres Database Version](#) (see page 14)

[Purpose of This Guide](#) (see page 15)

[Path Notation in This Guide](#) (see page 15)

[Your Support Options](#) (see page 15)

## What Is VectorWise?

VectorWise is the next generation database management system from the Ingres family of database products.

VectorWise is targeted at analytical database applications—applications that need to process large volumes of data and perform complex operations on it to derive useful information. Typical examples include data warehousing, data mining, and reporting. The main benefit of VectorWise over traditional Ingres solutions is significantly higher performance on data analysis tasks.

VectorWise is optimized to work with both memory- and disk-resident datasets, allowing it to efficiently process large amounts of data (hundreds of gigabytes).

**Note:** Although it is very fast for data analysis, VectorWise is not meant to be used for transaction processing, that is, applications that perform a large number of inserts, updates, and deletes. For OLTP, you can open a session to a traditional Ingres database from your VectorWise client.

## VectorWise Technology

VectorWise introduces a new way of storing data in Ingres and a completely new mechanism for evaluating queries.

Innovations such as vectorised processing, compression, and columnar data layout allow analytical queries to be run very fast on a single server, even a laptop.

## Vectorised Processing—Calculating Query Answers Fast

The most distinctive feature of VectorWise is the "vectorised" method it uses for evaluating queries. Rather than operating on single values from single table records at a time, VectorWise makes the CPU operate on "vectors," which are arrays of values from many different records. Such vectorised execution brings out the best in modern CPU technology. It brings to the world of databases the high performance that modern computers exhibit for scientific calculation, gaming, and multimedia applications.

The technical basis for efficiency of vectorised processing is that modern chip technology (be it Intel, AMD, or IBM manufactured) now uses deeply pipelined CPU designs. Keeping all pipelines full—and thus efficiency near peak—is impossible for traditional database engines primarily due to code complexity. Similarly, crucial CPU features such as Intel's Streaming SIMD Extensions (SSE) are not used well by traditional database systems.

Vectorised processing changes that. It provides efficiency that traditionally is only obtained by computer programs handwritten for one particular task. Additionally, because of the high clock frequency of current CPUs, database systems now need to treat main memory access as a significant cost factor. VectorWise tackles this by ensuring that the vectors it operates on fit inside the CPU caches, avoiding unnecessary (and in multi-core systems, often contended) main memory access. VectorWise takes advantage of multi-core systems by handling multiple queries concurrently or by running single queries in parallel.

The improved overall computational efficiency of VectorWise over traditional Ingres and other commercial relational database technology is at least an order of magnitude for long running analytical queries.

## Storage Innovations—Beating the Disk Bottleneck

Any database system with such a high computational speed runs the risk of becoming I/O bound. For this reason, the second major component of VectorWise consists of storage innovations designed for high I/O throughput. These innovations include:

- Columnar data layout
- Advanced compression
- Storage indexes

VectorWise introduces a new storage mechanism that uses columnar data layout, allowing analytical queries to avoid disk access for columns not involved in a query. While you can generally think of VectorWise storage as a column-store, VectorWise can mix columnar and row-based storage so that certain columns that are always accessed together get stored in the same disk block. Layout decisions are largely handled automatically by the system, but can also be controlled by the user.

To further avoid I/O becoming a performance bottleneck, VectorWise introduces a number of advanced compression schemes. These schemes are designed for very fast decompression. Therefore, accessing compressed data in VectorWise means that less data needs to come from disk, yet queries do not slow down due to decompression effort.

Finally, VectorWise uses storage indexes. The storage indexes are very small and store the minimum and maximum value per data block. The storage index, which is automatically created and maintained, enables the execution engine to rapidly identify candidate data blocks.

## How VectorWise Differs from Traditional Ingres

Even though VectorWise uses a radically new way of storing data and evaluating queries, it is fully integrated into traditional Ingres. The higher software layers—the layer database administrators, database users, or database applications interact with, such as management tools and the JDBC API—are identical to traditional Ingres.

The difference from traditional Ingres is that the data is stored in VectorWise tables. This data is processed using the specialized VectorWise engine, allowing much higher performance for analytical database tasks.

Although it is possible to have both VectorWise and Ingres Database installations on the same server, this is not desirable. We recommend that VectorWise be installed on a dedicated server.

While a database can contain a mixture of both VectorWise and traditional Ingres tables, version 1.5 does not support mixing both types of tables in the same query. You can, however, easily move data between Ingres and VectorWise tables directly and also create tables in traditional Ingres or VectorWise using CREATE TABLE AS SELECT or INSERT...SELECT statements.

VectorWise also differs from traditional Ingres in its intended usage scenario and in its VectorWise-specific configuration options and files.

### VectorWise Storage Type

VectorWise introduces a new VECTORWISE storage structure to Ingres.

The VECTORWISE storage type differs from traditional Ingres storage types in the following ways:

- Queries accessing VECTORWISE tables use an alternative execution module that is optimized for analytical processing.
- A query that accesses tables using VECTORWISE storage cannot access tables using other Ingres storage types.
- Some operations are not available when using the VECTORWISE storage type. For details about unsupported features, see the chapter "VectorWise Features."

### Ingres Database Version

VectorWise 1.5 is based on Ingres Database 10.0.

## Purpose of This Guide

Working with VectorWise is in many ways the same as working with traditional Ingres. The purpose of this guide is to:

- Give an overview of VectorWise.
- List the features of traditional Ingres that are currently supported and unsupported by VectorWise.
- Describe the aspects of VectorWise that are different from traditional Ingres.

This guide is designed to be the starting point for working with VectorWise. Consult the standard Ingres documentation (<http://docs.ingres.com>) for details about Ingres Database.

## Path Notation in This Guide

The directory structure of a VectorWise installation is the same regardless of operating system. Rather than showing path examples for all environments, this guide uses Linux notation only.

For example: When describing the location of the configuration file for the installation, the guide shows:

```
II_VWDATA/ingres/data/vectorwise/vectorwise.conf
```

where II\_VWDATA is the value of the VectorWise environment variable II\_VWDATA displayed by the `ingrenv` command.

## Your Support Options

Enterprise customers with active maintenance and support contracts have full access to Ingres Support, including telephone support and online use of our call tracking system and knowledge base, Service Desk. For Customer Support contact details, see <http://ingres.com/support-services/support#contacts>.

If you have an active support contract and want to register for access to Service Desk (<https://servicedesk.ingres.com>), use the enrollment form at <http://www.ingres.com/user/register.php>. (Your six-digit Account Number/Site ID is required.)

If you do not have a support agreement for Ingres Corporation products and are interested in purchasing support, contact us at [sales@ingres.com](mailto:sales@ingres.com).

For more information about support options, visit <http://ingres.com/support-services/support#>.

Free support is available from the Ingres Open Source Community. Community members may obtain assistance for Ingres Corporation products by registering with the Ingres Community Site and using the available tools. To register, go to <http://community.ingres.com/forum/> and click Register.

The Community Forums also provide Ingres Open Source Community members the opportunity to ask questions and interact with other community members and Ingres Corporation technical staff. For more information, visit <http://community.ingres.com/forum/index.php>.



# Chapter 2: VectorWise Concepts

---

This section contains the following topics:

[VectorWise Table Structures](#) (see page 17)

[Querying Data](#) (see page 21)

[Data Manipulation](#) (see page 21)

[Transaction Isolation Model](#) (see page 23)

## VectorWise Table Structures

In broad terms, VectorWise uses columnar storage. While data is stored and retrieved in familiar relational rows, the internal storage is different. Instead of storing all column values for a single row next to each other, all rows for a single column are stored together. This storage format has benefits for data warehouse applications, such as reduced I/O and improved efficiency.

A relational table may have several dozen or even hundreds of columns. In traditional row-oriented relational storage, the entire row must be read even if a particular query requests only a few columns. With column-oriented storage, a query needing only a few columns will not read the remaining columns from disk, and will not waste memory storing the unnecessary values. A side benefit of this columnar storage is that compression (see page 19) can be more effective.

While VectorWise is fundamentally a column store, it is not strictly so. For certain types of tables, VectorWise stores data from more than one column together in a data block on disk. Even so, within a data block the values for each column are kept together so that data for a column can be processed efficiently in vectors. VectorWise 1.5 introduces a new storage type that lets you indicate you want to store all columns for a table in a single block.

## Data Storage Format

For each database, VectorWise keeps data in a single directory, which you can find under *II\_VWDATA/ingres/data/vectorwise/dbname*. All data for a given database is stored in a single file in this directory (CBM/default/0). Additionally, log information is stored under CBM/LOG.

The data file consists of a number of blocks, which can be seen as the equivalent of pages. Each block contains possibly compressed data from one or more attributes. The size of the block can be configured with the [cbm] *block\_size* parameter (see page 85) before creating a database.

Additionally, for better sequential access performance, multiple blocks from the same column can be stored contiguously in a block group. The size of this group can be configured with the [cbm] *group\_size* parameter (see page 85).

Database size is unlimited. Once space is allocated by the database, the file storing the data can only grow; it is never freed to the operating system. If a table is dropped, its space within the file is reused first.

## Raw Table Storage Format

The default storage format for a VectorWise table (VECTORWISE) is also known as a RAW table. This format is similar to an Ingres heap table, in that the data is stored on disk in the order in which it is inserted. There are, however, some differences.

First, the data is stored in columns and the data is compressed.

The second difference is that for each column, the system automatically maintains simple statistics about the data distribution by storing the minimum and maximum value for ranges of tuples. If a query includes a filter on the column data value, the execution engine uses these min-max data values to decide whether to examine the data in a given block. Since min-max information is maintained in memory outside of the data block, this simple form of automatic indexing can dramatically reduce disk I/O and memory usage. You can configure the granularity of the min-max index with the [cbm] *minmax\_maxsize* parameter (see page 86).

## Data Compression

Columnar storage inherently makes compression (and decompression) more efficient than does row-oriented storage.

For row-oriented data, choosing a compression method that works well for the variety of data types in a row can be challenging, because compression for text and numeric data work best with different algorithms.

Column storage allows the algorithm to be chosen according to the data type and the data domain and range, even where the domain and range are not declared explicitly. For example, an alphabetic column GENDER defined as CHAR(1) will have only two actual values (M and F), and rather than storing an eight-bit byte, the value can be compressed to a single bit, and then the bit string can be further compressed.

VectorWise uses different types of algorithms from those found in other products. Because VectorWise processes data so efficiently, compression—and in particular decompression—are designed to use very little CPU and to reduce disk I/O. While on-disk compression ratios may be slightly lower than other products, overall performance is improved.

Compression in VectorWise is automatic, requiring no user intervention. VectorWise chooses a compression method for each column, per data block, according to its data type and distribution.

## Data Type Storage Format and Compression Type

Data types are stored internally in a specific format. The type of compression used depends on the data type and distribution.

VectorWise automatically uses any of the following compression methods:

- RLE (Run Length Encoding)—This method is efficient if many duplicate adjacent tuple values are present (such as in ordered columns with few unique values).
- PFOR (Patched Frame Of Reference)—This method encodes values as a small difference from a page-wide base value. The term "Patched" indicates that FOR is enhanced with a highly efficient way to handle values that fall outside the common frame of reference. PFOR is effective on any data distribution with some value distribution locality.
- PFOR-DELTA (delta encoding on top of PFOR)—In this method, the integers are made smaller by considering the differences between subsequent values. PFOR-DELTA is highly effective on ordered data.
- PDICTIONARY dictionary encoding (offsets into a dictionary of unique values)—This method is very efficient in case the value distribution is dominated by a limited amount of very frequent values. This compression method is currently the only one that applies for character types. Character data of size one—CHAR(1)—is internally represented as a number, and is an exception to this rule.

Most INTEGER, DECIMAL, and DATE/TIME types internally are compressed using any of the four compression methods.

Data types stored without compression in VectorWise tables are DECIMAL with precision greater than 18, FLOAT, and FLOAT4.

Character types (CHAR, VARCHAR, NCHAR, NVARCHAR) of lengths larger than one are stored internally as variable width strings. For data sets with repeating values, this data is automatically compressed using a per-block dictionary.

NULL values are stored internally as a single byte column, which is also automatically compressed using any of the four compression methods. The null indicator, if needed, is represented internally as a separate column. Loading and processing of nullable columns can be slower than non-nullable columns.

## Querying Data

### Statistics for Query Optimization

Analytical queries are the focus of VectorWise. It is essential that such queries have good information about the characteristics of the underlying data because the amount of data accessed can be great, and the sizes of intermediate processing stages must be properly estimated to produce optimal query plans.

By comparison, queries used in standard transactional processing typically access only a handful of records, making the estimates less important.

To allow the system to generate optimal query plans, we highly recommend running the `optimizedb` command if a significant percentage of the data (10% or greater) in a table has changed since the last time you ran `optimizedb`.

## Data Manipulation

Despite its focus on analytical rather than transactional processing, VectorWise provides a rich set of data updating capabilities.

### Granularity of DML Operations

VectorWise distinguishes between two granularities of DML operations:

- **Batch** – Operations that involve a relatively small number (for example, thousands or tens of thousands) of records. Such updates are buffered in the system memory and merged with the data from disk on the fly. If an operation is executed in batch mode, and the number of records is too high, the query will fail.

INSERT, DELETE, and UPDATE are always executed as batch operations. COPY FROM and INSERT...SELECT are executed as batch operations if the destination table has indexes declared and if it is not empty.

- **Bulk** – Operations that involve a large number of records (for example, one million or more), in which the data changes do not fit in the system memory.

CREATE TABLE AS SELECT is always executed as a bulk operation. COPY FROM and INSERT...SELECT are executed as bulk operations if the destination table has no indexes declared or if it is empty.

For more information, see Batch and Bulk Updates (see page 109).

## Snapshot Isolation

VectorWise provides update capabilities based on the concept of snapshot isolation. This means that the state of the system the user sees when starting a query stays the same even under concurrent updates. For details, see Transaction Isolation Model (see page 23).

## Persistence and Recovery

A COMMIT statement makes all changes from a transaction become persistent (including batch operations). In addition, VectorWise provides full automatic recovery. If a system crashes during a transaction, any changes made by that transaction will not be visible after the system restart, but all successfully committed transactions will be restored.

## Management of In-memory Updates

Batch updates are stored in a special highly-optimized differential data structure, called the Positional Delta Tree (PDT), in main memory. Data is also written to disk immediately to ensure recoverability.

Batch updates that reside in memory can potentially consume a significant percentage of useful system memory. Also, when the memory limit for these types of operations is reached, no more operations of this type are possible.

As a result, if the user is performing batch operations, VectorWise automatically propagates the changes buffered in memory to the disk-resident table. Propagation is triggered by factors such as table sizes and the number of DML operations buffered.

The volume of batch updates in such a propagation cycle should fit in main memory.

In the current version of VectorWise, update propagation is a relatively expensive operation, because it rewrites the entire contents of a table, even those blocks that have not changed.

The best practice is to avoid propagation to large tables (see page 113).

The amount of memory reserved for Positional Delta Trees and the amount of differences after which propagation is triggered can be configured with system configuration options. For more information, see PDT Parameters (see page 84). In cases with very high update rates, the memory required for PDTs can become too high, in which case you will have to work with raw tables or use bulk operations (see page 109), such as the COMBINE command.

## Transaction Isolation Model

VectorWise uses a *snapshot isolation* transaction isolation model with *optimistic concurrency control*.

### Snapshot Isolation

*Snapshot isolation* is a transaction isolation model in which a given transaction always sees the same, consistent state of the system representing the moment when a transaction has started. As a result, phenomena seen in certain other isolation levels (dirty, non-repeatable, and phantom reads) are avoided. Snapshot isolation is closely related to multiversion concurrency control.

Snapshot isolation is slightly weaker than the serializable isolation level available in some systems. It can lead to anomalies, where two transactions make decisions based on data that is modified by another transaction.

The main benefit of snapshot isolation is that, while providing a very high transaction isolation level, it avoids locking for read operations. Such locks can be very detrimental to database performance. By combining it with the optimistic concurrency control model, it allows VectorWise to achieve high performance of update operations without affecting the performance of read-only queries.

For more information, see:

- [http://en.wikipedia.org/wiki/Snapshot\\_isolation](http://en.wikipedia.org/wiki/Snapshot_isolation)
- [http://en.wikipedia.org/wiki/Multiversion\\_concurrency\\_control](http://en.wikipedia.org/wiki/Multiversion_concurrency_control)
- [http://en.wikipedia.org/wiki/Isolation\\_%28database\\_systems%29](http://en.wikipedia.org/wiki/Isolation_%28database_systems%29)

### Optimistic Concurrency Control

Unlike some systems, VectorWise uses the *optimistic concurrency control* model. In this model, transactions are allowed to perform changes to the data without checking for conflicts possibly caused by other transactions that have committed in the meantime. Conflicts are checked only when a transaction commits. This refers not only to conflicts caused by changing the same data, but also to constraint violations that occur only as a result of two transactions committing (for example, two concurrent transactions inserting a record with the same UNIQUE field).

Compared to other approaches, optimistic concurrency control avoids locking records for concurrently working transactions. This results in very good performance in cases where transactions typically do not conflict with each other.

The most visible aspect of optimistic concurrency control that sometimes causes problems for users accustomed to other solutions is that the conflicts are detected only at commit time. The application logic must take this into account.

For more information, see  
[http://en.wikipedia.org/wiki/Optimistic\\_concurrency\\_control](http://en.wikipedia.org/wiki/Optimistic_concurrency_control).

## Transaction Examples

### Non-dirty Read Example

This example demonstrates how committing transactions are not visible to concurrently running transactions. In this case, Transaction A will insert a value, but Transaction B will not see it until a new transaction is started.

Transaction A	Transaction B
BEGIN TRANSACTION; SELECT A FROM T; <i>--sees the old state</i>	BEGIN TRANSACTION; SELECT A FROM T; <i>--sees the same old state</i>
INSERT INTO T VALUES (7); SELECT A FROM T; <i>--sees the old state + value 7</i>	SELECT A FROM T; <i>--sees the old state without 7</i>
COMMIT;	SELECT A FROM T; <i>--sees the old state without 7</i> COMMIT; <i>--new transaction starts</i> SELECT A FROM T; <i>--sees the new state including 7</i>



## Conflict Resolution Example

In this example, two transactions concurrently insert a record with the same value in a Primary Key attribute. This causes a conflict for the second committing transaction due to constraint violation.

Transaction A	Transaction B
<i>--Empty Table T has a PRIMARY KEY on attribute PK</i>	
BEGIN TRANSACTION;	
SELECT A FROM T; <i>--sees an empty table</i>	
INSERT INTO T VALUES(6);	
INSERT INTO T VALUES(7);	
SELECT A FROM T; <i>--sees 6 and 7</i>	BEGIN TRANSACTION;
	SELECT A FROM T; <i>--sees an empty table</i>
	INSERT INTO T VALUES(5);
	INSERT INTO T VALUES(7);
SELECT A FROM T; <i>--sees 6 and 7</i>	SELECT A FROM T; <i>--sees 5 and 7</i>
COMMIT; <i>--success</i>	SELECT A FROM T; <i>--sees 5 and 7</i>
	COMMIT; <i>--fails due to a primary key violation on 7</i>

## Transactions and DDL Operations

Snapshot isolation and optimistic concurrency control are applied to operations on data (DML).

For operations that change the database schema (DDL), a different isolation model is used. In this model, transactions only see DDL changes performed by other transactions after they committed. Still, if there are conflicting DDL operations (for example, two transactions creating a table with the same name), the system locks the second transaction until the first transaction either commits or aborts.



# Chapter 3: VectorWise Features

---

This section contains the following topics:

[VectorWise Storage Structures](#) (see page 27)

[Default Database Characteristics](#) (see page 28)

[Transaction Management](#) (see page 28)

[Update Restrictions](#) (see page 28)

[SQL Statements](#) (see page 29)

[SQL Data Types](#) (see page 30)

[SQL Functions](#) (see page 30)

[Command and Utility Summary](#) (see page 32)

[New Features in Version 1.5](#) (see page 32)

[Unsupported Features](#) (see page 34)

[Restrictions and Known Incompatibilities](#) (see page 35)

In principle, VectorWise is accessed through all the standard Ingres APIs and supports all Ingres features for SQL processing. (The legacy language QUEL is not supported.)

## VectorWise Storage Structures

A VectorWise table can be created with one of two storage types:

### **VECTORWISE**

(Default) Stores data in columns. Puts as few columns as possible in one disk block (in most cases only one).

### **VECTORWISE\_ROW**

Stores data in rows. Puts as many columns as possible in one disk block.

In one block, values are still kept in compressed vectors. Potentially, all columns in a row can be stored in a disk block. This type is useful for very small tables, especially those with a large number of columns; it saves disk space and disk cache memory.

The `result_structure` parameter in Ingres `config.dat` sets the default storage structure for the VectorWise instance. When a table is created, the default storage structure is assumed.

To override the default, specify `SET RESULT_STRUCTURE` for the session or use `WITH STRUCTURE=vw_structure` in the `CREATE TABLE` statement.

If you want to create a traditional Ingres table from a VectorWise instance, you can override the default storage setting on a table basis by using `WITH STRUCTURE = HEAP` in the `CREATE TABLE` statement.

## Default Database Characteristics

VectorWise uses UTF-8 character strings with UCS\_BASIC column collation. This applies to CHAR and VARCHAR data types, regardless of whether the table structure is a VectorWise or traditional Ingres structure. UCS\_BASIC collation is also applied to UCS2 Unicode data stored in NCHAR and NVARCHAR columns in VectorWise tables. By default, the createdb command creates databases that store UCS2 Unicode data using Normalization Form C (NFC). (You do not need to explicitly specify the -i flag on createdb).

## Transaction Management

By default, autocommit is turned off, which means each transaction consists of multiple statements ended by either a COMMIT or ROLLBACK statement.

If an error occurs during the execution of an SQL statement, the entire transaction is rolled back (that is, SET SESSION WITH ON\_ERROR = ROLLBACK TRANSACTION is enforced).

In version 1.5 alpha, the statement SET SESSION WITH ON\_ERROR = ROLLBACK STATEMENT is not available; attempts to issue the statement will produce an error.

To determine if a transaction was aborted as the result of a database statement error, issue the SELECT DBMSINFO('TRANSACTION\_STATE') statement. If the error aborted the transaction, this statement returns 0, indicating that the application is currently not in a transaction.

## Update Restrictions

VectorWise provides basic support for updates (INSERT, UPDATE, DELETE).

For performance reasons, users should avoid using UPDATE and DELETE operations to update a single record. Instead, we recommend using these operations in batch mode, where hundreds or more records are modified. INSERTs with very many records (for example, one million or more) should be implemented using the bulk operations COPY, CREATE AS SELECT, and INSERT SELECT. UPDATE and DELETE operations that involve very many records should be performed using a COMBINE statement.

### More information

Batch and Bulk Updates (see page 109)

## SQL Statements

The following SQL statements are partially supported:

Type	Statements
DDL	CREATE TABLE, CREATE INDEX, CREATE VIEW, DROP, COPY, ALTER TABLE, DECLARE GLOBAL TEMPORARY TABLE
DQL	SELECT
DML	INSERT, UPDATE, DELETE, COMMIT, ROLLBACK

On the CREATE TABLE statement, column definitions and CREATE TABLE ... AS SELECT are supported.

On the SELECT statement, the following clauses and features are supported:

- FIRST rowcount
- ALL, DISTINCT
- GROUP BY
- HAVING
- UNION
- UNION ALL
- ORDER BY
- WITH clause
- Derived tables in FROM clause
- Subqueries (with limitations)
- Joins between VectorWise tables, joins between Ingres tables
- ANSI joins

The CREATE INDEX statement is supported on VectorWise tables and creates a clustered index. This statement can be issued only after the table has been created and is still empty. A table can have one index only.

For syntax details, see SQL Reference (see page 125) and the Ingres *SQL Reference Guide*.

## SQL Data Types

VectorWise supports the following data types:

Category	Data Type
Numeric types	INTEGER1 (TINYINT)
	INTEGER2 (SMALLINT)
	INTEGER4 (INTEGER)
	INTEGER8 (BIGINT)
	DECIMAL
	FLOAT8
	FLOAT
	FLOAT4
	MONEY
String types	CHAR
	VARCHAR
Unicode types	NCHAR
	NVARCHAR
Date/time types	DATE
	ANSIDATE
	TIME
	TIMESTAMP
	INTERVAL
	<b>Note:</b> TIME and TIMESTAMP are supported only WITHOUT TIMEZONE.

## SQL Functions

VectorWise supports the following functions and elements of SQL:

Category	Supported SQL Element
Operators	arithmetic, comparison, logical
Numeric functions	ABS(), ACOS(), ASIN(), ATAN(), ATAN2(), CEIL(), CEILING(), COS(), EXP(), FLOOR(), LOG(), LN(), MOD(), PI(), POWER(), ROUND(), SIGN(), SIN(), SORT(), TAN(), TRUNC(), TRUNCATE()
String functions	CONCAT(), LEFT(), LENGTH(), LOWER(),

Category	Supported SQL Element
	LOWERCASE(), LPAD(), LTRIM(), RTRIM(), SHIFT(), SIZE(), SOUNDEX(), SQUEEZE(), SUBSTRING(), TRIM(), UPPER(), UPPERCASE()
Date/time functions	WEEK(), WEEK_ISO(), YEAR(), QUARTER(), MONTH(), DAY(), HOUR(), MINUTE(), SECOND(), EXTRACT()
Aggregate functions	ANY, AVG, COUNT, COUNT(*), MAX, MIN, SUM, STDDEV_POP, STDDEV_SAMP, VAR_POP, VAR_SAMP, CORR, REGR_R2, REGR_SLOPE, REGR_INTERCEPT, REGR_SXX, REGR_SYY, REGR_SXY, REGR_AVGX, REGR_AVGY
Constants	NULL, NOW, CURRENT_DATE, CURRENT_TIME, CURRENT_TIMESTAMP, LOCAL_TIME, LOCAL_TIMESTAMP, USER, CURRENT_USER, SYSTEM_USER, INITIAL_USER, SESSION_USER
Expressions	CASE, IF, NULLIF, COALESCE, CAST
Predicates	LIKE, BEGINNING, ENDING, CONTAINING, BETWEEN, IN, ANY/ALL, EXISTS, IS NULL, IS INTEGER, IS DECIMAL, IS FLOAT, NULL
Other	HASH, IFNULL, RANDOM, RANDOMF

The constants CURRENT\_TIME, LOCAL\_TIME, CURRENT\_TIMESTAMP, and LOCAL\_TIMESTAMP have a type WITH TIMEZONE. These can be used only when cast to TIME or TIMESTAMP. For example:

```
CAST(LOCAL_TIME AS TIME)
```

```
TIME(CURRENT_TIME)
```

```
CAST(CURRENT_TIMESTAMP AS TIMESTAMP)
```

```
TIMESTAMP(LOCAL_TIMESTAMP)
```

For more information, see the *Ingres SQL Reference Guide*.

## Command and Utility Summary

Commands and utilities for performing operations in VectorWise are listed in the following table.

**Note:** Lowercase = line-based command; uppercase = SQL statement

Task	Tools
Start or stop VectorWise	ingstart, ingstop
Create or destroy databases	createdb, destroydb
Create tables	CREATE TABLE
Load or unload data	copydb, unloaddb, iivwfastload
Back up or recover data	copydb, unloaddb, ckpdb, rollforwarddb
Query	sql
Maintain or tune databases	optimizedb, statdump
Configure installation	ingpreenv, ingsetenv, ingunset, iigetres, iisetres, iinitres, iiremres, iivalres, iigenres
Manage or monitor	infodb, iivwinfo

## New Features in Version 1.5

VectorWise version 1.5 contains the following new features:

### Transfer data between traditional Ingres and VectorWise

Data from traditional Ingres tables can be transferred into VectorWise tables (and the reverse) using INSERT...SELECT or CREATE TABLE AS SELECT. This feature lets you easily migrate a traditional Ingres-based application to VectorWise, easily share data between VectorWise-based tables and Ingres-based tables, and use OLTP and data warehousing applications in a single installation.

### Add and Drop Column

A column can be added to or dropped from a table. The ALTER TABLE...ADD COLUMN and the ALTER TABLE...DROP COLUMN statements are supported.



### Improved DML

- Transaction enhancements

Transaction management (see page 28) is supported through the COMMIT and ROLLBACK statements. Also, multiple concurrent transactions are supported.

**Note:** If you are changing the same rows in two different concurrent transactions, only one will be able to commit.

- Parallel CREATE TABLE AS SELECT

### Temporary tables

Temporary tables are supported through the DECLARE GLOBAL TEMPORARY TABLE statement. Temporary tables can be used to organize data derived from queries into new tables, which can then be queried repeatedly during a session. This feature improves query performance.

### Parallel query

Parallel query execution (using multiple cores to execute one query) is supported. You can enable it by modifying the [engine] max\_parallelism\_level option in the configuration file. This feature allows significant performance improvements on queries that access large volumes of data.

### Storage improvements

Storage is improved as follows:

- Database size is now unlimited. When the database is filled up (the default is 128 GB), the file grows. This feature makes the columnspace\_size configuration parameter obsolete.
- Performance is significantly improved when making multiple small updates to a table, using COPY, INSERT, DELETE, or UPDATE statements. Data is appended to the last previously written blocks, which reduces storage space and improves query performance.
- A table can use the VECTORWISE\_ROW storage structure, which stores data in rows rather than columns. Such storage is a more efficient use of disk space and is especially useful if you have thousands of tables or tables with a very large number (over 10,000) of columns.

### Support for Large Pages

The large pages (known as huge pages on Linux) feature of modern CPUs is supported. This feature limits memory fragmentation, which improves database performance.

### Fast loader

The iivwfastload utility allows faster loading of data from data files into VectorWise tables.

## Unsupported Features

VectorWise supports no collation sequences, other than the UCS\_BASIC column collation in tables stored with a VectorWise structure.

## Unsupported SQL Statements

Rules, auditing, location management, SQL procedures, and certain DDL statements are not supported.

Category	Statements
DDL	CREATE INTEGRITY, MODIFY, SAVE
Location Management	CREATE LOCATION, ALTER LOCATION, DROP LOCATION, SET WORK LOCATIONS
Auditing	ENABLE SECURITY_AUDIT, DISABLE SECURITY_AUDIT, CREATE SECURITY ALARM, DROP SECURITY ALARM, HELP SECURITY ALARM, REGISTER TABLE, REMOVE TABLE
SET	SESSION ISOLATION_LEVEL, TRANSACTION ISOLATION_LEVEL, JOURNALING, LOGGING, LOG_TRACE, SESSION WITH ON_ERROR, SESSION WITH ON_LOGFULL, LOCKMODE, LOCK_TRACE
Rules	CREATE RULE, DROP RULE, SET RULES
DML	SAVEPOINT

## Unsupported Data Types

The following Ingres Database SQL data types are not supported:

Category	Unsupported Data Type
QUEL data types	TEXT, C
Binary	BYTE, VARBYTE

Category	Unsupported Data Type
Unicode data types	LONG NVARCHAR
Long data types	LONG VARCHAR (CLOB), LONG VARBYTE (BLOB)
Date/time types	INGRESDATE Time zones: TIME WITH TIMEZONE, TIMESTAMP WITH TIMEZONE
Boolean	BOOLEAN
Other	UUID, TABLE_KEY, OBJECT_KEY

## Unsupported SQL Functions

The following Ingres Database SQL functions are not supported:

Category	Unsupported SQL Function
String functions	COLLATION_WEIGHT, II_IPADDR
Date/time functions	MICROSECOND, NANOSECOND
Constants	TODAY
Expressions	Sequence expressions: NEXT VALUE FOR, CURRENT VALUE FOR, LAST_IDENTITY
Predicates	SIMILAR TO
Other	HEX, UNHEX, INTExtract

## Restrictions and Known Incompatibilities

Certain features of VectorWise have restrictions or known incompatibilities with traditional Ingres.

### SQL Cursors

VectorWise supports forward read-only cursors only. Cursors on VectorWise queries cannot be scrolled, nor used for update.

## Some Nested Queries Do Not Work

Some forms of subselects (also known as subqueries or nested queries) that are supported in traditional Ingres do not yet work. Cases where the optimizer would fall back to an iterative query execution model (such queries tend to be slow) are not supported:

- ORed subqueries
- NOT IN/EXISTS subqueries that contain nested NOT IN/EXISTS subqueries
- Single value subqueries with count as result may not be supported; other aggregates are supported.

Subqueries that do not successfully flatten and compile to a valid VectorWise syntax produce the E\_OP08C2 error. Try to rewrite these queries.

## Cross-system Queries

SELECT queries that access both VectorWise and traditional Ingres tables cannot execute. To cross join data in traditional Ingres and VectorWise you have to move the data either into traditional Ingres or into VectorWise using INSERT...SELECT or CREATE TABLE AS SELECT.

## Host Variables in REPEATED Queries

REPEATED queries (<http://docs.ingres.com/ingres/10.0/sql-reference-guide/3135-repeated-queries>) do not work if they contain host variables.

## SQL Data Type Incompatibilities

The following incompatibilities between traditional Ingres and VectorWise are known:

- VectorWise currently does not check for overflows on operations on floating point types (integer operations and decimal operations with less than 15 digits are checked for overflow).
- TIMESTAMP and INTERVAL DAY TO SECOND can only store 7 decimal places (fractions of seconds), compared to 9 in traditional Ingres.
- VectorWise supports only the YYYY-MM-DD date format.
- Conversion from floating point types to character types can yield slightly different results than in traditional Ingres.
- The MONEY currency symbol is fixed to '\$'.

## SQL Function Incompatibilities

The following incompatibility between traditional Ingres and VectorWise is known: Due to a different hash computation algorithm, the HASH function returns different results.

## Differences in Constraint Checking

Constraint checking differs from traditional Ingres in the following ways:

- When loading data with COPY, if a constraint violation is detected, the entire dataset is rejected.
- When loading data with COPY, VectorWise detects foreign key (referential) constraint violations, unlike traditional Ingres.
- VectorWise does not handle cyclic constraints.
- Because of VectorWise's optimistic concurrency control model, it is possible that a constraint violation is only detected at commit time.

## Data Size Limitation for Some Queries

For some types of queries, VectorWise may not be able to perform a requested operation due to very high memory usage. The following scenarios are the most typical:

- Performing an aggregation that produces a very high number of records.  
For example:

```
SELECT l_orderkey, COUNT(*)  
FROM lineitem  
GROUP BY l_orderkey
```

If the number of distinct values of "l\_orderkey" is very high (tens of millions or more), the query may fail, depending on available memory.

Increasing available memory can be a partial solution for this problem. Also, if the "lineitem" table is indexed on "l\_orderkey", a faster and more memory-efficient solution is used.

- Performing a join between two very large tables without any restrictions.  
For example:

```
SELECT o_orderdate, l_receiptdate  
FROM lineitem, orders  
WHERE l_orderkey = o_orderkey
```

If the number of records in "orders" is very high (tens of millions or more), the query may fail, depending on available memory.

Increasing available memory can be a partial solution for this problem. Also, if both "lineitem" and "orders" tables are indexed on the join key, a faster and more memory-efficient solution is used.

- Performing a duplicate-eliminating UNION where the input relations have a high number of records. For example:

```
SELECT o_orderkey, o_orderdate  
FROM orders1
```

```
UNION
```

```
SELECT o_orderkey, o_orderdate  
FROM orders2
```

If the number of values in orders1 and orders2 is very high (tens of millions or more), the query may fail, depending on available memory. Increasing available memory can be a partial solution for this problem. Also, you can consider using a UNION ALL approach if you know there are no duplicates, or if you accept duplicate entries.

# Chapter 4: Developing Applications

---

This section contains the following topics:

[Application Programming Languages](#) (see page 39)

[Application Programming Interfaces](#) (see page 39)

[SQL Overview](#) (see page 40)

[Connectivity Drivers](#) (see page 41)

## Application Programming Languages

VectorWise developers can use various programming languages to develop database applications, including: C, C++, Java, COBOL, Visual Basic, C#, Python, PHP, and Perl. For details about how to connect to VectorWise when using these languages, see the Ingres *Quick Start Guide* and the Ingres *Connectivity Guide*.

Precompilers are provided that let you embed SQL in your C, COBOL, Fortran, Basic, Ada, and Pascal application programs. For details about embedded SQL, see the Ingres *Embedded SQL Companion Guide*.

## Application Programming Interfaces

Application Programming Interfaces (APIs) that can be used with VectorWise include:

- JDBC (Java Database Connectivity) is a standardized API that allows database connectivity. It defines a set of function calls, error codes, and data types that can be used to develop database-independent applications using Java.

**Note:** VectorWise does not support the INGRESDATE data type, so by default its `date_alias` parameter is set to ANSIDATE. The JDBC driver, however, overrides this setting by default. If you plan to use the DATE alias in your JDBC application, then you must set **`date_alias=ansidate`** in your JDBC connection.

- ODBC (Open Database Connectivity) is a standardized API that allows database connectivity. It defines a set of function calls, error codes, and data types that can be used to develop database independent applications using Structured Query Language (SQL).

- The Ingres .NET Data Provider is a Microsoft .NET component that provides native .NET connectivity to VectorWise databases to deliver VectorWise data to the Microsoft .NET Framework. It uses the Data Access Server to access VectorWise data sources.
- OpenAPI (Open Application Programming Interface) is a set of C language functions that let you create applications for accessing VectorWise, Ingres, and non-Ingres databases. It is an alternative to using embedded SQL, which requires a preprocessor in addition to a C compiler. With OpenAPI, these C functions are called directly with normal function call facilities.

For details, see the Ingres *Connectivity Guide* and *OpenAPI User Guide*.

## SQL Overview

Structured Query Language (SQL) lets you manipulate data and database objects, and perform database management functions such as copying data between tables and files.

SQL statements are categorized according to the task performed:

### **Data Definition Language (DDL)**

Creates or deletes objects such as tables and views

### **Data Manipulation Language (DML)**

Selects, inserts, updates, and deletes data in tables

SQL statements can be interactive or embedded. You enter interactive SQL statements from a Terminal Monitor, and query results display on the screen. Invoke the Terminal Monitor by typing **sql** at the operating system prompt. Embedded SQL statements can be included in procedural (3GL) programming languages such as C or Fortran.

For more information, see the Ingres *SQL Reference Guide*.



## Connectivity Drivers

A variety of connectivity drivers, data adapters, and dialects can be used with VectorWise, including the following:

- Ingres ODBC Driver
- Ingres JDBC Driver
- Ingres .NET Data Provider
- Ingres Python DBI Driver
- Ingres PHP Driver
- Ingres Perl DBI Extension
- Ingres Torque Database Adapter
- Ingres Hibernate Dialect

For a list of latest solutions and details about each, see the downloads page of the Ingres web site.

For more information about the Ingres ODBC Driver, JDBC Driver, and .NET Data Provider, see the Ingres *Connectivity Guide*.



# Chapter 5: Understanding Usage Scenarios

---

This section contains the following topics:

[Typical Use Scenarios](#) (see page 43)

[Reporting on DBT-3 Database Benchmark](#) (see page 43)

[How to Run the Benchmark](#) (see page 44)

## Typical Use Scenarios

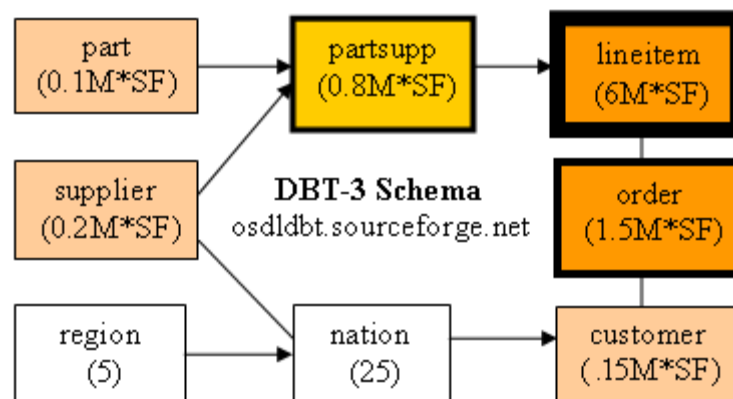
Typical scenarios where VectorWise demonstrates great performance are those where large quantities of data must be analyzed, such as found in data warehousing, reporting, and data mining.

## Reporting on DBT-3 Database Benchmark

An example scenario is the one modeled by the DBT-3 database benchmark, available in open source at <http://osdl.dbt.sourceforge.net/>.

This benchmark defines a data warehouse with customers placing orders consisting of multiple line items. Each line item is a number of parts; and parts are supplied by multiple suppliers. The benchmark comes with a data generator (dbgen) and set of queries (22 reporting queries, and 2 update streams). The size of the database can be scaled using a scaling factor (SF). The default SF is 1, which represents 1 GB.

The schema contains the following table and foreign key relationships. The number of tuples in each table is shown in parentheses:



The largest table (as indicated by the thickness of the boxes in the drawing) is "lineitem".

In this benchmark, we see a number of business intelligence (BI) features tested, including:

- Joins between a large fact table (lineitem) and smaller dimension tables (part, supplier, nation, region), followed by groupings
- Large joins (between lineitem, orders, and partsupp)
- Correlated subqueries
- Complex string predicates (such as LIKE)
- Top-N processing (getting the first N tuples of an ORDER BY)

## How to Run the Benchmark

Running the benchmark involves the following steps:

1. Download the DBT-3 package.
2. Make the database generator tool, dbgen.
3. Generate the data using the dbgen tool (see page 45).
4. Start VectorWise (see page 45).
5. Create a database (see page 46).
6. Create a VectorWise table (see page 46).
7. Bulk load the data into the table created in Step 6 (see page 47).
8. Create statistics using the traditional Ingres utility optimizedb (see page 48).
9. Run a reporting query (see page 49).

## Generate Data

The first step in running the benchmark is to download the DBT-3 package, make the database generator tool "dbgen", and generate the data.

### To download the DBT-3 package and make the dbgen tool

1. Download the DBT-3 package from here:

`http://downloads.sourceforge.net/project/osdl/dbt/dbt3/1.9/dbt3-1.9.tar.gz`  
(<http://downloads.sourceforge.net/project/osdl/dbt/dbt3/1.9/dbt3-1.9.tar.gz>)

2. Decompress and unarchive the downloaded file:

```
tar xvfz dbt3-1.9.tar.gz
```

3. Make the database generator tool, dbgen:

```
cd dbt3-1.9/src/dbgen
make
```

### To generate the 30 GB version of the benchmark

Issue the dbgen command with the following flags at the operating system prompt:

```
dbgen -vfF -s 30
```

In this example, we will use only the lineitem table; you therefore can add **-T L** to the command line to generate only the "lineitem.tbl" text file.

The system responds with:

```
DBT-3 Population Generator (Version 1.3.0)
Copyright Transaction Processing Performance Council 1994 - 2000
Generating data for lineitem table [pid: 7552] done.
```

## Start VectorWise

If VectorWise is not running, start it using the ingstart command.

### To start VectorWise

Issue the following command at the command line:

```
ingstart
```

VectorWise is started.

## Create a Database

The next step is to create a database using the `createdb` command.

### To create a database named "dbtest"

Issue the following command at the command line:

```
createdb dbtest
```

The database is created.

## Create a VectorWise Table

The next step is to create the `lineitem` table. The `CREATE TABLE` statement by default creates a table with `VECTORWISE` storage.

### To create the `lineitem` table with `VECTORWISE` storage

1. Connect to the database created in the previous step by issuing the `sql` command at the operating system prompt, as follows:

```
sql dbtest
```

The Ingres Terminal Monitor starts.

2. Type the following SQL statement at the terminal monitor prompt:

```
CREATE TABLE lineitem (  
    l_orderkey INTEGER NOT NULL,  
    l_partkey INTEGER NOT NULL,  
    l_suppkey INTEGER NOT NULL,  
    l_linenummer INTEGER NOT NULL,  
    l_quantity DECIMAL(2,0) NOT NULL,  
    l_extendedprice DECIMAL(8,2) NOT NULL,  
    l_discount DECIMAL(2,2) NOT NULL,  
    l_tax DECIMAL(2,2) NOT NULL,  
    l_returnflag CHAR(1) NOT NULL,  
    l_linestatus CHAR(1) NOT NULL,  
    l_shipdate ANSIDATE NOT NULL,  
    l_commitdate ANSIDATE NOT NULL,  
    l_receiptdate ANSIDATE NOT NULL,  
    l_shipinstruct CHAR(25) NOT NULL,  
    l_shipmode CHAR(10) NOT NULL,  
    l_comment VARCHAR(44) NOT NULL);\g
```

**Note:** We recommend declaring key constraints in VectorWise. (We did not do so here for the sake of brevity.) If a column always contains a value then you should define the column as `NOT NULL`.

## Bulk Load the Data into the Table

The next step is to use the SQL statement COPY to load the lineitem.tbl text file generated by the dbgen utility for the 30 GB data size (SF=30). This step assumes this file is available in the local directory.

### To load the data into the lineitem table

Type the following SQL statement at the Ingres Terminal Monitor prompt:

```
COPY TABLE lineitem (  
    l_orderkey = 'c0|',  
    l_partkey = 'c0|',  
    l_supkey = 'c0|',  
    l_linenum = 'c0|',  
    l_quantity = 'c0|',  
    l_extendedprice = 'c0|',  
    l_discount = 'c0|',  
    l_tax = 'c0|',  
    l_returnflag = 'c0|',  
    l_linestatus = 'c0|',  
    l_shipdate = 'c0|',  
    l_commitdate = 'c0|',  
    l_receiptdate = 'c0|',  
    l_shipinstruct = 'c0|',  
    l_shipmode = 'c0|',  
    l_comment = 'c0n1'  
) FROM 'lineitem.tbl' \g
```

The system responds with:

```
Executing . . .
```

```
(179998372 rows)
```

It may take up to an hour to load the 180 million rows, depending on the speed of your hardware and CPU.

**Note:** Bulk-loading goes through the SQL client (the sql utility) over a network connection to VectorWise. The bottleneck is client parsing and network handling, rather than the VectorWise server.

## Create Statistics

The next step is to run **optimizedb**, which creates statistics that the query optimizer uses to create query plans.

**Note:** We highly recommend this step. Skipping this step may lead to query plans that are inefficient; in extreme cases, it may lead to inability to execute a query.

By default, the `optimizedb` utility optimizes all tables in the database by looking at all rows.

### To run `optimizedb` on the database "`dbtest`"

Issue the following command from the command line:

```
optimizedb -zfq dbtest
```

The `-zfq` flag speeds the building of the histogram.



## Run a Reporting Query

The final step in running the benchmark is to run a reporting query.

This example reporting query (Q1) asks for the amount of business that was billed, shipped, and returned in the last quarter of 1998:

```
SELECT  l_returnflag, l_linestatus, sum(l_quantity) as sum_qty,
        sum(l_extendedprice) as sum_base_price,
        sum(l_extendedprice * (1 - l_discount)) as sum_disc_price,
        sum(l_extendedprice * (1 - l_discount) * (1 + l_tax)) as sum_charge,
        avg(l_quantity) as avg_qty, avg(l_extendedprice) as avg_price,
        avg(l_discount) as avg_disc, count(*) as count_order
FROM    lineitem
WHERE   l_shipdate <= date '1998-12-01' - interval '90' day
GROUP BY l_returnflag, l_linestatus
ORDER BY l_returnflag, l_linestatus;\g
```

Executing . . .

returnflag	linestatus	sum_qty	sum_base_price	sum_disc_price
A	F	1132676958	1698421699729.99	1613502554023.1240
N	F	29581022	44353456023.61	42135855268.9589
N	O	2230403478	3344516955134.66	3177286529933.9344
R	F	1132872903	1698719719123.67	1613789096673.0512

sum_charge	avg_qty	avg_price	avg_dis	count_order
1678049479312.961024	25.4	38236.375	0.050	44419004
43821835543.370384	25.5	38270.478	0.049	1158947
3304387310370.192384	25.4	38235.736	0.049	87470970
1678345906004.148224	25.5	38243.421	0.049	44418612

(4 rows)



## **PART 2: Installation**

---



# Chapter 6: Recommended Platforms

---

This section contains the following topics:

[RAID File System](#) (see page 53)

[Supported CPUs](#) (see page 55)

[Disk Subsystems](#) (see page 55)

[Guidelines for a Balanced Platform](#) (see page 56)

## RAID File System

A RAID file system is strongly recommended.

Currently VectorWise uses storage in a single file system only.

The single file system limitation somewhat complicates the use of multiple disks, which has a performance benefit, as it forces the system administrator to create a RAID volume over all available disks.

### Linux:

On Linux various file systems are known to work, including ext2, ext3, and xfs.

**Note:** There is a known data corruption bug in the ext4 implementation used in most versions of Linux. Do not use the ext4 file system for your data storage.

**Note:** VectorWise uses one file for all the data in a single database, whereas traditional Ingres uses many files.

## Hardware versus Software RAID

In principle, both hardware and software RAIDs can be used. Software RAID is the more flexible option because hardware RAID performance for a single volume is limited by the capabilities of a single RAID controller. The highest performance I/O subsystems for VectorWise have been created by using multiple high-end (PCIe) disk controllers (not necessarily RAID controllers), and uniting all disks in a software RAID volume, typically in RAID5.

## Database Location and Linux Symbolic Link

If VectorWise is not installed on the RAID volume itself, you can also use symbolic links from the `vectorwise` or `vectorwise/dbname` directories to the RAID volume.

### VectorWise data directory

`II_VWDATA/ingres/data/vectorwise`

### VectorWise per-database subdirectory

`II_VWDATA/ingres/data/vectorwise/dbname`

`II_VWDATA` can be defined during installation. By default, it is the same as `II_DATABASE` (VectorWise data location), which in turn is by default the same as `II_SYSTEM` (VectorWise installation location).

While the server is stopped, you can move an existing `vectorwise` or `vectorwise/dbname` directory to a different location, and replace it with a symbolic link. For example:

```
II_VWDATA=`ingprens II_VWDATA`  
  
mv $II_VWDATA/ingres/data/vectorwise /mnt/fastraid/  
  
ln -s /mnt/fastraid/vectorwise $II_VWDATA/ingres/data/
```

In this example, the entire `vectorwise` directory is moved to a different location, so the data of all databases you created or will create will reside on `/mnt/fastraid`. (This example assumes that your fast RAID drive is mounted at `/mnt/fastraid`.) You can perform the same operation for individual databases, located in subdirectories of `II_VWDATA/ingres/data/vectorwise`.

**Important!** The database system must be stopped during these operations; otherwise, irreversible data corruption can occur.

## Supported CPUs

VectorWise is designed to run on all modern hardware.

The 64-bit Linux distribution can run on all 64-bit x86 CPUs from both Intel and AMD.

**Note:** CHAR and VARCHAR operations can gain performance benefits from SSE4.2 instructions. Check with your hardware vendor whether SSE4.2 instructions are supported.

VectorWise uses multiple cores for handling concurrent queries and if you enable parallel execution, a single statement can use multiple CPU cores.

## Disk Subsystems

For processing outside-of-memory datasets, VectorWise needs, above all, a high performance sequential read disk subsystem. This can be achieved using multiple magnetic disks in RAID, using any of these technologies:

- SCSI
- SAS
- SATA

Because random lookups are not as important in a data warehousing context, using SAS hardware tends to be the cost effective option.

Solid State Drives (SSD), which typically use SATA, are more expensive per gigabyte, but high-end models deliver more than 2.5 times the sequential throughput of a single magnetic disk. They also come in a 2.5-inch form factor, allowing higher "bandwidth density". Therefore, SSDs are the high-performance storage method of choice for fast VectorWise warehousing.

Be sure to balance SSDs with enough disk controllers: at least one controller is needed per four drives. Because VectorWise uses advanced differential techniques for handling updates efficiently, the amount of write operations is significantly reduced, so using cost-effective "MLC" memory SSDs is an option.

## Guidelines for a Balanced Platform

A "balanced" hardware configuration is one that has no clear performance bottleneck. In a balanced configuration, CPUs can process at maximum performance while there is little to no surplus capacity in other resources. A balanced configuration gives you maximum return for your investment. Benchmarks are always run using a balanced platform.

Balanced hardware configurations on which VectorWise could be used are shown in the following table. Consider these configurations only as guidelines, because the characteristics of your particular data and query workload can influence hardware choices.

Objective	CPU	RAM	Disk Storage	Raw Data Size
Main memory resident	2-16 64-bit cores	8-144 GB	A single file system that can hold 1.5x the raw data size	Roughly equal to RAM size
Typical disk-based	2-8 cores, 64-bit	2-144 GB	As above, with at least two SAS disk drives (or one high-throughput SSD) per core	5-10x RAM size
Querying a TB	2 socket quad-core Nehalem 3.2 GHz	144 GB	16 high-throughput SSDs on four high-end disk controllers in software RAID5	1000 GB

The configurations described in the table are intended to achieve a balanced system in terms of CPU power versus disk throughput. If your data size falls outside the range of raw data size shown in the table, you can extrapolate based on these configurations. Storage size, because of the VectorWise compression, tends to be smaller than the raw data size; however, when dimensioning a system, additional space must be allocated (for data bulk-load staging and database backup, for example), hence the 1.5x recommendation.

A typical disk based configuration uses internal server disk storage (usually limited in common rack machines to 12x3.5-inch or 16x2.5-inch disk drives). Current software limitations in VectorWise require you to combine multiple disks into a single file system using RAID. (For a good balance between performance and security, we recommend RAID levels 5 or 6.)

In general, VectorWise can benefit from systems that are optimized for sequential throughput rather than expensive disk subsystems that can handle many requests per second.



Because VectorWise uses data compression within the database but also for data inside its RAM buffer pool (decompressing it with fast vectorised methods only when data flows to the CPU cache for query processing), the system can be optimized to work with most of its data cached in RAM. This compression keeps memory access bandwidth limited, which helps in the case of systems with multiple cores, where the memory bus quickly becomes a bottleneck when all cores are busy. Therefore, an alternative server dimensioning strategy is to aim for mostly in-memory query execution. Following this strategy, it is easier to keep many CPU cores busy, because available RAM bandwidth is much higher than disk bandwidth. A server designed with this goal can use a simpler disk subsystem.

The amount of memory required when designing a system for main memory operation tends to be less than the raw data size, thanks to compression and the fact that a typical analytical workload does not require all columns to be cached. Because each query needs additional memory for query processing, the number of concurrent queries must be taken into account. In the previous table, we keep things simple by using the "equal to raw data size" guideline.

**Note:** If your configuration falls short of the suggested "balanced system" values shown above, we still recommend that you consider VectorWise. Experience has shown that its performance advantage, compared to traditional database technology, tends to be largest in performance-constrained hardware.



# Chapter 7: Installing VectorWise

---

This section contains the following topics:

[Download VectorWise](#) (see page 59)

[Installation Considerations](#) (see page 59)

[Installing VectorWise for Linux](#) (see page 61)

[Response File—Define Configuration for VectorWise Instance](#) (see page 72)

[Upgrade to VectorWise 1.5](#) (see page 73)

[Post-installation Tasks](#) (see page 73)

[How You Safely Uninstall VectorWise](#) (see page 74)

[View Installation Environment Settings](#) (see page 75)

## Download VectorWise

The latest version of VectorWise is available on Ingres Corporation's electronic software delivery site.

### To download VectorWise

Go to <http://esd.ingres.com/> (<http://esd.ingres.com/>).

## Installation Considerations

During installation, your VectorWise instance will be configured according to the settings you provide. Such settings include:

Setting	Stored in Environment Variable	Default Value
Instance ID	II_INSTALLATION	VW
System files location	II_SYSTEM	/opt/Ingres/IngresVW
Data location	II_DATABASE	Value of II_SYSTEM
VectorWise data location (see page 60)	II_VWDATA	Value of II_DATABASE
Backup location	II_CHECKPOINT	Value of II_SYSTEM
Journal files location	II_JOURNAL	Value of II_CHECKPOINT
Dump files location	II_DUMP	Value of II_CHECKPOINT

Setting	Stored in Environment Variable	Default Value
Temporary files location	II_WORK	Value of II_SYSTEM
Transaction log file location	Not applicable	Value of II_SYSTEM
Time zone	II_TIMEZONE_NAME	NA-PACIFIC

For details, see Understanding Installation Considerations (<http://docs.ingres.com/ingres/10.0/installation-guide/1394-understanding-installation-considerations>) in the Ingres *Installation Guide*. Some VectorWise defaults may differ from those of traditional Ingres.

The easiest way to install VectorWise is to accept all the default values during the installation process.

## VectorWise Data Location (II\_VWDATA)

You can specify a dedicated location for storing VECTORWISE tables by setting the VectorWise Data location option during installation. This location is stored in the VectorWise environment variable II\_VWDATA. By default, it is the same as the VectorWise data location (II\_DATABASE).

Storing VECTORWISE tables in a different location from standard Ingres tables is useful for the following reasons:

- VectorWise tables often contain larger data volumes than standard Ingres tables due to their analytical orientation.
- VectorWise benefits more from storage with fast sequential disk access.

If II\_VWDATA is set, the default vectorwise.conf configuration file is also installed in *II\_VWDATA/ingres/data/vectorwise/*.

## Character Set

All VectorWise instances use the UTF8 character set. This setting cannot be changed.

**Note:** All client instances connecting to a VectorWise Server instance must also use the UTF8 character set.

## Installing VectorWise for Linux

VectorWise for Linux is distributed in RPM and non-RPM format.

You can install the product using any of these methods:

Distribution File Format	Installation Method
RPM	Installation Wizard (recommended) (see page 63)
	Ingres_express_install command (see page 66)
	RPM command (see page 68)
non-RPM	Interactive forms-based utility (ingbuild) (see page 70) <b>Note:</b> You must use this method if your Linux does not have RPM Package Manager (such as Ubuntu).

**Note:** Installation of VectorWise requires root access. If you are logged in as another user, you will need the root password.

## Packages That Can Be Installed

VectorWise RPM packages that can be installed include the following:

- "Core" – Prerequisite of all other packages
- DBMS – DBMS Server
- NET – Communications servers

Functionality	Packages Required
Client	Core and Net
SQL	Core and DBMS
JDBC	Core, DBMS, Net
ODBC	Core, DBMS, Net

## Install libaio

The Linux Asynchronous I/O library, libaio, must be installed before installing VectorWise.

Libaio has a huge performance benefit over the standard POSIX asynchronous I/O facility, because the operations are performed in the Linux kernel instead of as a separate user process.

### To install libaio

Using RPM:

```
rpm -ivh libaio*.rpm
```

Using the yum system update tool:


```
yum install libaio
```

Using Ubuntu:

```
apt-get install libaio1
```

## Install VectorWise Using the Installation Wizard on Linux

The installation wizard easily installs VectorWise without your having to know RPM commands. The wizard uses RPM to install VectorWise.

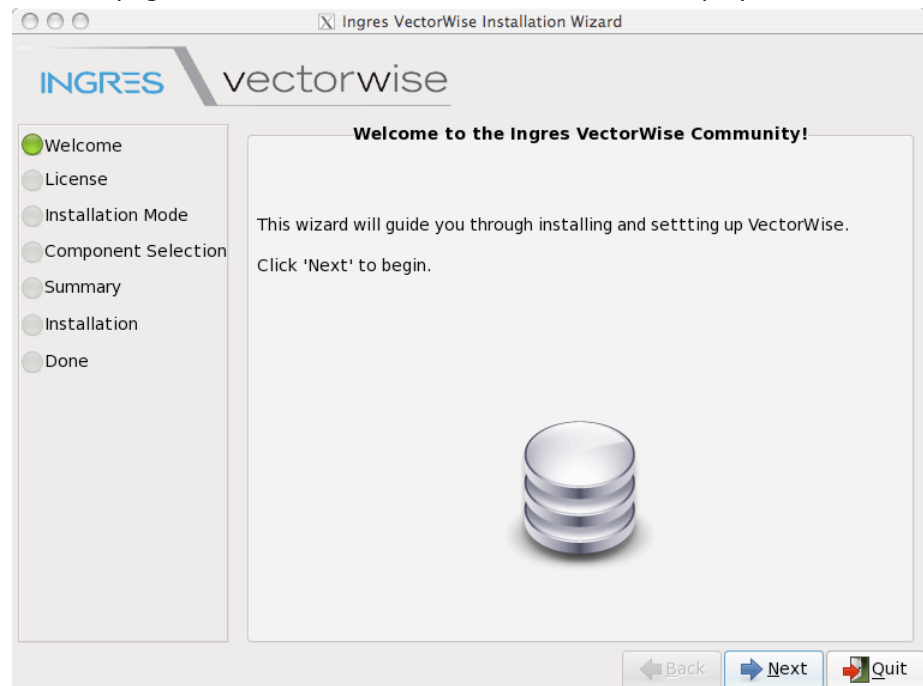
During the installation process, if you need more information about a specific option, click its information button  in the wizard.

### To install VectorWise using the Installation Wizard

1. Run the following script located in the root directory of the VectorWise distribution:

```
./ingres_install
```

The first page of the VectorWise Installation Wizard is displayed.

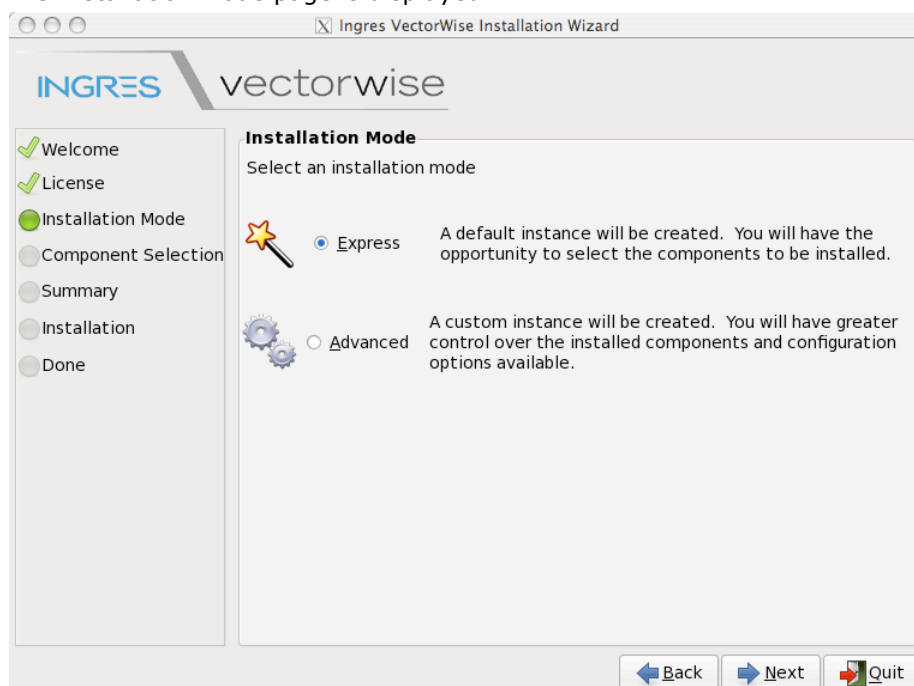


2. Click Next.

The license dialog is displayed.

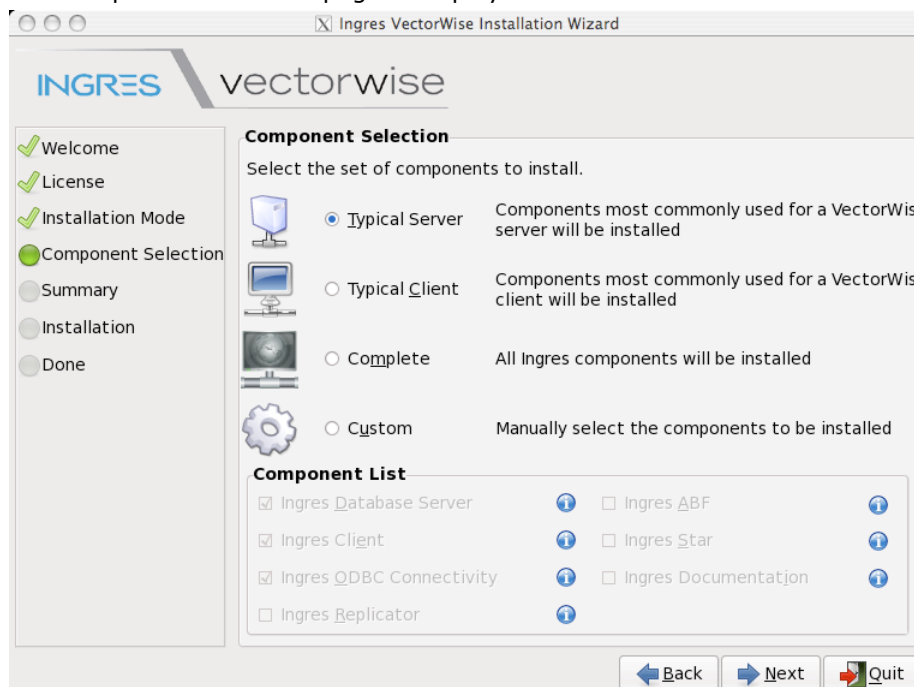
3. Select "I accept the terms of this license agreement," and then click Next.

The Installation Mode page is displayed.



4. Select Express, and then click Next.

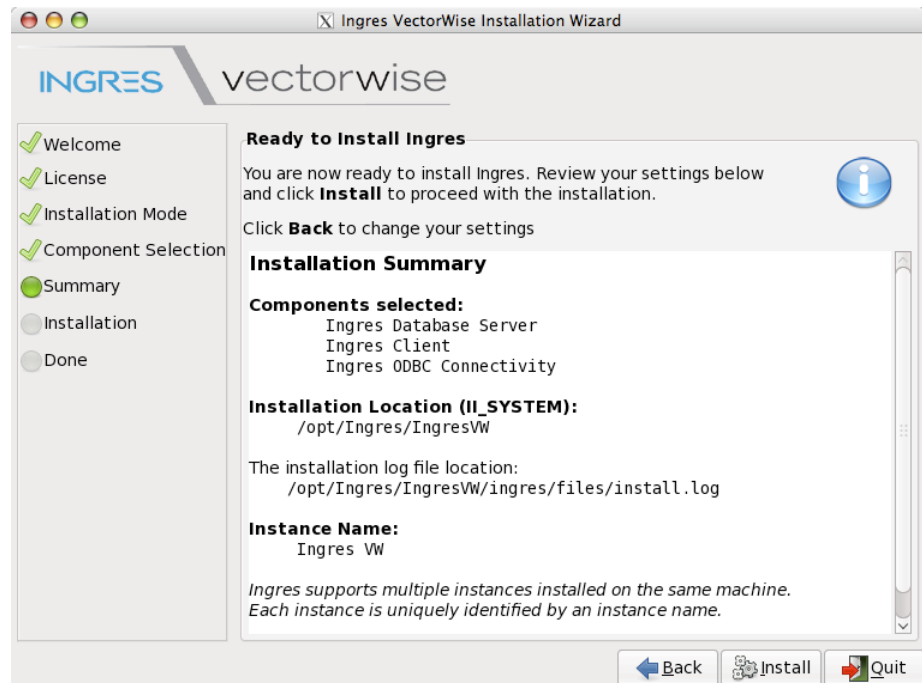
The Component Selection page is displayed.





5. Select Typical Server, and then click Next.

The Summary page is displayed.



6. Click Install.

The installation program installs VectorWise.

The installation Complete page is displayed.

7. Click Finish.

You are exited from the installation program.

## ingres\_express\_install Command—Install VectorWise Quickly

The `ingres_express_install.sh` command quickly installs all VectorWise packages.

This command has the following format:

```
ingres_express_install.sh [-tar] [instance_PATH] [instance_ID]
```

### **-tar**

Installs from `ingres.tar` instead of RPMs. (Linux only)

### ***instance\_PATH***

Identifies the full path to the location where VectorWise is to be installed (II\_SYSTEM).

Default: `/opt/Ingres/IngresVW`

### ***instance\_ID***

Defines a two-character string where the first character must be an uppercase letter and the second character must be an uppercase letter or a number from 0 to 9.

Default: `VW`

### **Notes:**

1. `ingres_express_install.sh` will not upgrade an instance.
2. The `ingres_express_install.sh` command must be run as the "root" system user.
3. `ingres_express_install.sh` will fail if:
  - An "ingres" system user does not exist.
  - The "ingres" system user does not have read, write, and execute permissions on the II\_SYSTEM directory.
  - Another instance of VectorWise exists with the same *instance ID* as the one we are installing.
4. If an alternate *instance ID* (for example, V2) is specified:
  - The *instance PATH* must be specified (for example, `/opt/Ingres/IngresV2`).
  - If *instance PATH* is not specified, `ingres_express_install.sh` tries to install into `/opt/Ingres/IngresVW`, which can corrupt an existing VW installation.
5. After `ingres_express_install.sh` completes, do the following:
  - a. Shut down the instance using the `ingstop` command.
  - b. Log off from the "root" system user.

- c. Log on to the "ingres" system user and execute `.ingXXsh` or `.ingXXcsh` (where XX is the instance ID), which are found in the home directory (\$HOME) of the "ingres" system user.
- d. Start VectorWise using the `ingstart` command.

**Examples:**

This command installs all RPM packages in the current working directory with the default configuration:

```
ingres_express_install.sh
```

This command installs all RPM packages with the default configuration into `/opt/Ingres/IngresVW`, but with an instance ID of V1:

```
ingres_express_install.sh V1
```

This command installs all RPM packages with the default configuration into `/opt/Ingres/IngresV2`, with an instance ID of V2:

```
ingres_express_install.sh /opt/Ingres/IngresV2 V2
```

## Install VectorWise Using RPM Commands

You can install VectorWise by invoking RPM directly.

### To install a single package

Invoke RPM with the appropriate installation flags, including the path of the package you want to install, as follows:

```
rpm -ivh path_to_directory/Ingres_package.rpm
```

### To install more than one package at a time

Pass multiple file names, specifying the path to each package, as follows:

```
rpm -ivh path_to_directory/Ingres_package.rpm  
path_to_directory/Ingres_package2.rpm path_to_directory/Ingres_package3.rpm
```

### To install all packages in the same directory

Specify the following:

```
rpm -ivh path_to_directory/*.rpm
```

### To install a package into a non-default location (that is, with an **II\_SYSTEM** value other than the default)

Use the `--prefix` flag when invoking RPM. The following command installs the specified package with `II_SYSTEM=/home/ingres/IngresVZ`:

```
rpm -ivh --prefix=/home/ingres/IngresVZ path_to_directory/Ingres_package.rpm
```

**Note:** All packages installed in a single instance must have the same value of `II_SYSTEM`. So if you use the `--prefix` flag to install the base package, you must install all subsequent packages with the same `--prefix` value.

**Note:** After you have installed VectorWise using the RPM command, you must run the service script to configure the instance (see page 69).

## Configure and Start the Instance

When installing VectorWise using the RPM command, the setup portion of the installation process must be completed by running the service script. The service script configures and starts the instance.

### **To run the setup programs and start the instance**

After the installation has completed, run the service script as follows:

```
/sbin/service ingresXX start
```

where XX is the instance ID.

The instance is configured and started.

### **To run the setup without starting the instance**

After the installation has completed, run the service script as follows:

```
/sbin/service ingresXX configure
```

The instance is configured but not started.

## Install VectorWise Interactively Using ingbuild

You can install VectorWise interactively using a character-based utility, ingbuild.

**Note:** Use the ingbuild installation utility if your Linux does not have RPM Package Manager.

### To install VectorWise using the character-based Installation Utility

**Note:** The downloaded tar file must have read permissions given to the installation owner (the user who installs VectorWise).

1. Log in as the root user.
2. Extract to a local directory all files from the saveset that was downloaded from the Ingres web site.
3. Change to the root directory from the directory where you extracted the saveset files, and enter the following command to run the installation utility:

```
install.sh
```

The Installation Utility (ingbuild) is started.

4. Respond to the installer dialogs.

VectorWise components are installed on your system.

The VectorWise instance is started automatically.

## How You Access the Instance on Linux

When the installation is complete, the instance is running.

During installation, an environment file is written to the home directory of the operating system user ID that was defined during installation (the default is "ingres"). The name of the environment file depends on the value of `II_INSTALLATION`:

For RPM installations: `ingXXbash` or `ingXXtsch`

For non-RPM installations: `ingXXsh` or `ingXXcsh`

where `XX` is the instance ID.

To access your instance, you must source this environment file.

### **To source the environment file created during installation**

Issue the following command.

**Note:** The following example assumes an operating system user ID of `ingres`.

```
source ~ingres/.ingXXtsch
```

For other users to have access to the instance and the VectorWise tools, they must have access to the `.ingXXbash` and `.ingXXtsch` scripts. The scripts can be copied to the home directory of any user.

## Response File—Define Configuration for VectorWise Instance

A response file contains parameters that define how a VectorWise instance is to be installed and configured. A response file can be used to install a custom configuration of VectorWise from the command line.

For more information, see How You Install VectorWise with a Custom Configuration at the Command Line (<http://docs.ingres.com/ingres/10.0/installation-guide/1426-how-you-install-ingres-with-a-custom-configuration-at-the-command-line>) in the Ingres *Installation Guide*.

Here is a sample response file for VectorWise for Linux:

```
# Ingres Response File
# Generated by the Ingres Response File API
# Created: Wed Jun 2 07:59:09 2010

# [ Ingres Locations ]
II_SYSTEM=/opt/Ingres/IngresV1
II_DATABASE=/opt/Ingres/IngresV1
II_CHECKPOINT=/opt/Ingres/IngresV1
II_JOURNAL=/opt/Ingres/IngresV1
II_WORK=/opt/Ingres/IngresV1
II_DUMP=/opt/Ingres/IngresV1
II_VWDATA=/opt/Ingres/IngresV1
II_LOG_FILE=/opt/Ingres/IngresV1

# [ Ingres Configuration ]
II_INSTALLATION=V1
II_CHARSET=ISO88591
II_TIMEZONE_NAME=NA-PACIFIC

# [ VectorWise Configuration ]
II_VWCFG_MAX_MEMORY=245M
II_VWCFG_BUFFERPOOL=122M
II_VWCFG_COLUMNSPACE=512G
II_VWCFG_BLOCK_SIZE=512K
II_VWCFG_GROUP_SIZE=8

# [ Ingres Installation Options ]
II_USERID=ingres
II_GROUPID=ingres
II_LOG_FILE_SIZE_MB=256
II_START_ON_BOOT=YES
```



## Upgrade to VectorWise 1.5

Upgrading to VectorWise 1.5 requires that you install a new instance and then fully reload all data. Do not select Modify or Upgrade options when running the installer.

## Post-installation Tasks

**Note:** Post-installation tasks may not be required for a basic installation.

Post-installation tasks include the following:

- (Linux) (Optional) Create and configure the transaction log file as a raw device
- (Linux) Allow access on systems using shadow passwords
- Establish user access to tools and databases

For more information on post-installation tasks on Linux, see Post-Installation Tasks (<http://docs.ingres.com/ingres/10.0/installation-guide/1256-post-installation-tasks-unix>) in the Ingres *Installation Guide*.

For information about establishing user access to tools and databases, see How User Access Is Established in the Ingres *Connectivity Guide*.

## How You Safely Uninstall VectorWise

Removing VectorWise is an irreversible event with pervasive effects. Any products or applications that share the removed DBMS Server are affected, as follows:

- Any future attempt to connect to this database will fail.
- If you re-install VectorWise, you may not be able to reference the data files.

If you know you want to remove a VectorWise instance, follow this process:

1. If you want to keep the data files for later use, you must run `unloaddb` against each database before removing VectorWise. Doing so will allow you to reference the data files easily if you re-install VectorWise later. For details on the `unloaddb` command, see the *Database Administrator Guide* and the *Command Reference Guide*.
2. If you do *not* want to keep the data files, you can run `destroydb` against each database before removing VectorWise. The `destroydb` command locates and deletes all data files. For details on the `destroydb` command, see the *Command Reference Guide*.
3. Uninstall VectorWise.

### Linux

For Linuxes that have RPM Package Manager, you can uninstall VectorWise using the instructions in the Ingres *Installation Guide* (<http://docs.ingres.com/ingres/10.0/installation-guide/1322-how-you-safely-uninstall-ingres>).

For Linuxes without RPM Package Manager, you can use the `uninstall-ingres.sh` script described in Knowledge Base document "Uninstalling Ingres on UNIX" ([http://servicedesk.ingres.com/CAisd/pdmweb.ingres?OP=SHOW\\_DETAIL+PERSID=KD:415714+HTMPL=kt\\_document\\_view.htmlplcom](http://servicedesk.ingres.com/CAisd/pdmweb.ingres?OP=SHOW_DETAIL+PERSID=KD:415714+HTMPL=kt_document_view.htmlplcom)).

## View Installation Environment Settings

VectorWise environment variables that are defined at the VectorWise system (installation) level affect all users in an installation. These are typically defined when installing the product. Most VectorWise environment variables are set in the symbol table (symbol.tbl).

### **To view all installation-wide environment variables**

Enter the `ingprens` command at the operating system prompt:

```
ingprens
```

To manually register or to remove an environment variable from the symbol table, use the `ingsetenv` and `ingunset` utilities. Never edit the `symbol.tbl` file directly.



# Chapter 8: Accessing VectorWise

---

This section contains the following topics:

[Startup and Shutdown](#) (see page 77)

[Start VectorWise as a Service](#) (see page 78)

[Network Connections](#) (see page 78)

## Startup and Shutdown

### To start the VectorWise instance

Log on to your system through the system administrator account for the instance.

Enter the following command:

```
ingstart
```

The `ingstart` command checks whether you have sufficient operating system resources to run the product components. If so, `ingstart` starts all servers that are part of your instance.

### To stop the VectorWise instance

Enter the following command:

```
ingstop
```

The instance is stopped.

## Start VectorWise as a Service

The script to start VectorWise for Linux as a service is available only for instances installed using RPM. The script is installed under `/etc/init.d`.

The service has the name `ingresXX`, where `XX` is the instance ID (value for `II_INSTALLATION`) specified during installation (default is `VW`).

### To start the VectorWise instance as a service

Run the following as root:

```
/sbin/service ingresXX start
```

where `XX` is the instance ID.

### To stop and restart the service

Use the following commands:

```
/sbin/service ingresXX stop
```

```
/sbin/service ingresXX restart
```

where `XX` is the instance ID.

## Network Connections

Ingres Net must be installed to allow a VectorWise client to access databases on another instance. With Ingres Net on each VectorWise instance in your network, you can access databases on remote nodes as well as on your own local node.

Any installation configuration in which the client and server processes do not reside on the same machine or in the same instance must use Ingres Net.

The following components are automatically installed with Ingres Net:

- Data Access Server  
Provides network access to the DBMS Server for Ingres JDBC drivers and .NET Data Providers.
- Ingres JDBC Driver

Concepts related to Ingres Net include virtual nodes, connection data, remote user authorizations, and global and private definitions. For more information, see the *Ingres Connectivity Guide*.

## **PART 3: Administration**

---





# Chapter 9: Configuring and Managing VectorWise

---

This section contains the following topics:

[Configuration File \(vectorwise.conf\)](#) (see page 81)

[Configuration Options](#) (see page 82)

[When to Change the Default Configuration Values](#) (see page 87)

[Memory Settings](#) (see page 88)

[I/O Settings](#) (see page 89)

[View Information about a Database with VectorWise Tables](#) (see page 90)

[Using Large Pages](#) (see page 94)

[Using Multiple Databases](#) (see page 96)

[Error Reporting—Database Log File](#) (see page 96)

[VectorWise SQL Settings](#) (see page 97)

[Performance Tips](#) (see page 98)

## Configuration File (vectorwise.conf)

The VectorWise engine uses its own configuration file called vectorwise.conf.

A configuration file is not required. If no configuration file exists, default values are used.

The system first uses the default values, which are overridden by parameters in the per-user configuration file in the home directory, then by the per-installation configuration file, and finally by a per-database configuration file.

To make it easy to supply a configuration file for a specific database, the file vectorwise.dbname.conf in the dbfarm directory overrides general dbfarm and general configuration files, but is overridden by a configuration file in the dbname directory.

### Per-user configuration file

`$HOME/.vectorwise.conf`

This file is not created by default.

### Per-installation configuration file

`II_VWDATA/ingres/data/vectorwise/vectorwise.conf`

This file is installed by default. It contains examples of available configuration options.

### Per-database configuration file

`II_VWDATA/ingres/data/vectorwise/dbname/vectorwise.conf`

where *dbname* is the name of the database.

To configure a single database, you must create this file manually.

**Important!** If you want to configure settings for a single database that influence the database creation process (for example: `[cbm] block_size`), you must create this directory and file before creating the database.

If you want to configure a database that has not yet been created, an alternative location for the per-database configuration file is:

`II_VWDATA/ingres/data/vectorwise/vectorwise.dbname.conf`

This configuration file is read after the per-installation configuration file, but is overridden by a configuration file in the database directory.

## Configuration Options

The configuration file, `vectorwise.conf` (see page 81), is a text file that consists of sections, such as `memory`, `system`, `cbm`. The section name is in brackets. Each section contains key/value pairs.

For example:

```
[memory]
max_memory_size=2G
min_mem_per_transaction=20M
[cbm]
bufferpool_size=1G
```

When setting byte values, use a suffix of K, M, G, or T, representing kilobytes, megabytes, gigabytes, and terabytes, respectively.

### [memory] Settings

Memory settings in `vectorwise.conf` affect the memory used for query processing (not for data caching in the buffer pool).

#### **max\_memory\_size**

Specifies the amount (in bytes) of the total memory used for query execution.

#### **Notes:**

- `Max_memory_size` applies to all queries executing across the system, but a single query can consume almost all available memory.

- Some queries will fail if they consume too much memory; increasing available memory can help.
- If `max_memory_size` is not specified or is not larger than zero, a default setting of 50% of the physical system memory is used.
- Memory size defined with this option does not include `bufferpool_size`.

For more information, see Memory Settings (see page 88).

Default: 0 (use 50% of the physical system memory)

#### **min\_mem\_per\_transaction**

Specifies the minimal memory reserved for each running transaction.

Default: 10M

#### **use\_huge\_tlb**

Tells VectorWise whether to use the "huge pages" feature (see page 94) of the CPU. Valid values are true and false.

Default: true

## **[system] Settings**

System settings in `vectorwise.conf` affect the VectorWise system.

#### **max\_transactions**

Specifies the maximum number of active transactions in the system. New incoming transaction requests after reaching this limit are queued.

**Note:** For each transaction a minimum amount of memory is reserved (10 MB by default). This means that 200 MB of system memory is reserved initially by default.

Default: 20

#### **num\_cores**

Specifies the number of processing units in the system. This value is used to calculate the amount of memory available for each transaction.

`Num_cores` is used to calculate the maximum parallelism level for a newly issued query if there are many queries already being executed. The goal is not to deteriorate the total throughput of the system. Generally, the higher the `num_cores` value, the higher parallelism levels are granted for queries running concurrently (where each parallelism level is not higher than `max_parallelism_level` (see page 87)).

`Num_cores` is a database-wide limit on the number of concurrent threads, but it is a soft limit. After `num_cores` is exceeded, new queries will always get one core and will be more likely to run out of memory since the `num_cores` value is used to reserve memory for possible other threads.

Default: Number of processors on the machine

### **use\_sse42**

Uses SSE4.2 for accelerated string processing on CPU architectures that support it.

Default: YES

### **PDT Parameters**

The Positional Delta Tree (PDT) is a special highly-optimized differential data structure used for batch updates residing in memory. For more information, see Management of In-memory Updates (see page 22).

#### **max\_global\_update\_memory**

Specifies an upper bound on total PDT memory consumption as a fraction of available RAM pool size. Upon reaching this upper bound, the system propagates PDT resident updates to disk.

Default: 0.5

#### **max\_table\_update\_ratio**

Specifies a per-table maximum fraction of PDT resident updates (that is, the number of PDT updates divided by the number of stable tuples). Upon reaching this limit, the system propagates PDT resident updates for this table to disk.

Default: 0.05

#### **min\_propagate\_table\_count**

Specifies the minimum number of tuples in a table above which max\_table\_update\_ratio is checked. This is to avoid frequent disk propagations on relatively small tables and update loads.

Default: 500K

#### **max\_table\_snapshot\_copy\_mem**

Specifies a per table upper limit on the amount of memory used for PDT copies in the snapshot isolation layer.

Default: 1M

### **vectorsize**

Specifies the number of records processed together.

Default: 1024

## [cbm] Settings

CBM (Column Buffer Manager) settings in `vectorwise.conf` affect the buffer manager for VectorWise.

CBM options determine the format of the database files when the database is created. If you want the **per-database configuration file** to be applied during database creation, the file must be created **before using the `createdb` command**. For more information, see Configuration File (`vectorwise.conf`) (see page 81).

Note especially these CBM options:

- `block_size` (see page 85)
- `bufferpool_size` (see page 85)
- `preload_mintuples` (see page 86)

### **block\_size**

Specifies the minimum I/O granularity.

This option is the most important for I/O performance.

**Note:** This setting cannot be changed after database creation!

**Note:** The system rounds up this setting to the closest power of 2.

Default: 512K

### **bufferpool\_size**

Specifies the buffer pool size in bytes (that is, disk cache). The setting can be changed without reloading data.

#### **Notes:**

- Increasing `bufferpool_size` makes more data reside in memory and may reduce I/O.
- If `bufferpool_size` is not specified or is not larger than zero, a default setting of 25% of the physical system memory is used.
- Memory size defined with this option does not include `max_memory_size`.

Default: 0 (use 25% of the physical system memory)

### **group\_size**

Specifies the number of blocks that are grouped together on disk to improve data locality for sequential scans.

**Note:** The system rounds up this setting to the closest power of 2.

Default: 8

### **io\_prefetch\_blocks**

Specifies the maximum number of blocks to prefetch on each read.

Default: 16

### **minmax\_maxsize**

Specifies the granularity of the (automatically created) min-max indexes on all columns. Min-max indexes are used automatically to derive scan ranges based on selection predicates on columns that correlate to tuple order.

The value of this parameter influences the precision of the range restrictions applied when performing a table scan. With a default setting (1024), up to ~0.1% of a table might be scanned extra for each found scan range. Increasing this value can improve precision, but may slightly decrease the update performance and increase the memory usage. We recommend increasing this parameter only if you issue queries that scan very small fractions of a table.

This parameter should be set before database creation.

Default: 1024

### **preload\_maxcolsize**

Excludes from cache large columns exceeding this threshold. This threshold is the size of a single column, in bytes. The column size is the record size times the number of rows.

Default: 280M

### **preload\_mintuples**

Keeps column data cached for tables smaller than this size.

In star schemas, you may want to ensure that all dimension tables are cached, but not the fact tables. To do this, set `preload_mintuples` to be considerably larger than the largest dimension table, but smaller than any of the fact tables.

Default: 100M

### **queue\_depth**

Specifies the length of the asynchronous I/O queue.

Default: 16

## **[engine] Settings**

Engine settings in `vectorwise.conf` affect the execution engine.

### **max\_parallelism\_level**

Defines the maximum number of threads used by a single query. This is a hard limit.

A value of 1 disables parallel query execution. A value larger than 1 enables parallel query execution.

Max\_parallelism\_level is related to the num\_cores parameter (see page 83). The execution engine tries not to use more than the max\_parallelism\_level number of cores at a time for any query.

Default: Number of processors on the machine

### **sort\_intmemory**

Maximum size in bytes allocated in the first phase of sorting operations (quicksort and saving to single external table)

Default: 256M (32M on 32-bit system)

### **sort\_extmemory**

Maximum size in bytes allocated in the second phase of sorting operations (merging external runs).

Default: 32M

## **When to Change the Default Configuration Values**

The VectorWise default configuration is suitable for a machine designated to run a database system, and optimized to exploit most of the available system resources.

The default configuration assumes a single VectorWise instance per server and a single active database with multiple concurrent queries.

If you need to use multiple instances and databases simultaneously on one server, manual configuration is needed. (For more information, see Using Multiple Databases (see page 96).) Reconfiguration can also be useful on machines with very little memory or if you want to achieve maximum performance for single-stream queries.

## Memory Settings

The main resource used by VectorWise is system memory. The system uses two memory pools:

### Disk page buffer pool

Configure this by setting the `bufferpool_size` (see page 85) option in the `[cbm]` configuration group. The value must be an exact number with a suffix K, M, or G representing kilobytes, megabytes, and gigabytes, respectively.

Default: 0 (use 25% of physical memory)

### Query execution memory

Configure this by setting the `max_memory_size` (see page 82) option in the `[memory]` configuration group. The number provided is the available physical memory VectorWise can claim.

Default: 0 (use 50% of available memory)

**Note:** These memory settings are cumulative. For example: On an 8 GB machine, the default configuration will consume 6 GB of memory (2 GB buffer pool + 4 GB query memory).

For example: On a 16 GB machine, the user may want to configure the system to use 4 GB on the buffer pool and the remaining 10 GB on query memory (assuming 2 GB is reserved for other programs). To do that, the configuration file should have these entries:

```
[cbm]
bufferpool_size = 4G
[memory]
max_memory_size = 10G
```



## Memory Configuration Guidelines

How should you determine how much memory should be set aside for the buffer pool versus query execution?

If you use rather simple queries that do not require a lot of memory (for example, large scans that produce relatively few aggregates), you do not need a lot of query execution memory.

If you have complex queries (for example, joins between large tables, or aggregations that produce many results), you need more query memory.

If your disk system is very fast or if your set of frequently accessed data is rather small, large disk buffer is not that crucial. Similarly, if you have a large disk-resident data set and a slow disk, increasing the buffer pool size can help.

A rule of thumb is to start with query memory to about 50% and buffer pool to about 25% of your memory (the default), and adjust as needed.

## I/O Settings

The effective disk block access unit is:

```
group_size * block_size
```

This value needs to be sufficiently large; even a single magnetic disk now needs a value of 2 MB to get reasonably good sequential throughput. If you have a multi-disk (RAID) system, you need to multiply this efficiency target by the number of disks, because the requests will be spread across all disks in equal chunks.

Finally, current SSDs need somewhat smaller sizes to be efficient. Group\_size\*block\_size should be at least 512 KB for SSDs.

For example, in a disk configuration consisting of 8 SAS drives in RAID5, you could opt for 16 MB disk transfers, using a group\_size of 8 and a block\_size of 2 MB:

```
[cbm]
bufferpool_size = 4G
group_size = 8
block_size = 2M
[memory]
max_memory_size = 12G
```

In a RAID 0,5 system, group\_size\*block\_size should ideally be an exact multiple of the RAID stripe size times the number of disks. In a RAID 1,10,01 system, group\_size\*block\_size should be stripe size times half the number of disks.

## View Information about a Database with VectorWise Tables

Use the `iivwinfo` command (see page 138) to display information about a database that uses VectorWise tables.

By default, `iivwinfo` displays various statistics about the database. You can supply options on the command to display other information. Issue **`iivwinfo -h`** to see the list of available options.

### To display database statistics

Issue any one of the following commands at the operating system prompt. All these commands produce the same output:

```
iivwinfo dbname
iivwinfo -s dbname
iivwinfo --stats dbname
```

where *dbname* is the name of the database.

Here is an example of the statistics output:

stat	value
varchar(36)	varchar(44)
memory.query_allocated	30920304
memory.query_maximum	214748364
memory.query_virtual_allocated	30920304
memory.query_virtual_maximum	70368744177664
memory.update_allocated	0
memory.update_maximum	53687091
memory.bufferpool_allocated	0
memory.bufferpool_maximum	3148349440
bm.block_size	524288
bm.group_size	8
bm.columnspace_total_blocks	262144
bm.columnspace_free_blocks	262144
bm.bufferpool_total_blocks	24197
bm.bufferpool_free_blocks	24197
bm.bufferpool_used_blocks	0
bm.bufferpool_cached_blocks	0
bm.columnspace_location	/opt/IngresVW/ingres/data/vectorwise/dbname/CBM/default/0
system.active_sessions	0
system.log_file_size	33239111
system.threshold_log_condense	33554432

Fields are as follows:

**memory.query\_allocated**

The amount of physical memory allocated by the currently running queries

**memory.query\_maximum**

The maximum allowed amount of physical memory. If this threshold is reached, some queries will stop working. It is configured with the [memory] max\_memory\_size configuration parameter.

**memory.query\_virtual\_allocated**

The amount of virtual memory allocated by the currently running queries. This amount can be higher than that of physical memory in some cases.

**memory.query\_virtual\_maximum**

The maximum allowed amount of allocated virtual memory

**memory.update\_allocated**

The amount of memory currently occupied by the batch (memory-cached) updates. To free this memory, you can use the COMBINE command to propagate these updates to disk.

**memory.update\_maximum**

The maximum amount of memory that can be used by memory-cached updates. If memory.update\_allocated reaches this limit, propagation (see page 113) (automatic COMBINE) is triggered. If propagation is not successful, and the limit is reached, the memory-cached updates may stop being accepted.

**memory.bufferpool\_maximum**

The total size of the buffer pool (disk block cache) in bytes

**memory.bufferpool\_allocated**

The used size of the buffer pool (disk block cache) in bytes

**bm.block\_size**

The disk block size used by this database. It is configured with the [cbm] block\_size configuration parameter.

**Note:** Changing this option is possible only before creating a database. Later changes are ignored.

**bm.group\_size**

The group size used by this database. It is configured with the [cbm] group\_size configuration parameter.

**Note:** Changing this option is possible only before creating a database. Later changes are ignored.

**bm.columnspace\_total\_blocks**

The current maximum size of a column space, in blocks. You can compute the total allowed physical size by multiplying this value by `bm.block_size`.

**bm.columnspace\_free\_blocks**

The number of free disk blocks among the current maximum defined by `columnspace_total_blocks`. By default the total number of blocks grows as necessary, so this value does not directly determine remaining data capacity.

**bm.bufferpool\_total\_blocks**

The size of a buffer pool (disk data cache), in blocks. You can compute the total allowed physical size by multiplying this value by `bm.block_size`. This value is a sum of `bm.bufferpool_free_blocks` and `bm.bufferpool_used_blocks`.

**bm.bufferpool\_free\_blocks**

The number of blocks in the buffer pool that can be used for new data

**bm.bufferpool\_used\_blocks**

The number of blocks in the buffer pool that are currently locked by queries

**bm.bufferpool\_cached\_blocks**

The number of blocks in the buffer pool that contain cached data from disk

**bm.columnspace\_location**

The location of a data file holding this database

**system.active\_sessions**

The number of currently running queries

**system.log\_file\_size**

Current size of the transaction log file

**system.threshold\_log\_condense**

Size of the transaction log file at which the system tries to compact it (drop unused information)

**To display the active configuration for the database**

Issue either of these commands, which produce the same output:

```
iivwinfo -c dbname
iivwinfo --config dbname
```

The current configuration settings for the specified database are displayed. Here is an excerpt from a typical output:

config	value
cbm.block_size	524288
cbm.group_size	8

**Note:** Both user-configurable options and internal system configuration are displayed. User-configurable options are discussed in Configuration Options (see page 82).

**To display the disk usage of tables**

Issue either of these commands, which produce the same output:

```
iivwinfo -tbu dbname
iivwinfo --table_block_use dbname
```

To limit the output to a specified table, use the -t (or --table) option, as follows:

```
iivwinfo -t tablename -tbu dbname
```

Here is an excerpt from a typical output:

table_name	block_count
part	30
partsupp	208

**Note:** Disk usage is shown in the number of disk blocks.

## Using Large Pages

**Note:** This feature requires a good understanding of memory management.

To reduce TLB (translation lookaside buffer) misses, modern CPUs offer a feature called "large pages" ("huge pages" on Linux). Large pages allow a page size of 2 MB (on some CPUs even 1 GB) instead of 4 KB. Using a larger page size is especially beneficial when accessing large amounts of memory with a random pattern. VectorWise supports the use of large pages for certain data structures that would benefit the most.

Consult the documentation for your operating system for details on how to enable large page support.

### Configuration

You must designate an amount of memory for large pages before starting VectorWise. This memory is used for large page allocations only, not for normal allocations. Because this division of memory is static, you should wisely choose the amount of memory available in large pages.

**Note:** The large pages feature applies to query memory only, not to buffer memory. Do not assign a significantly higher amount of memory to large pages than the VectorWise [memory] max\_memory\_size parameter. Doing so may not leave enough "normal" memory for the buffer pool and other processes in the system.

If you encounter problems with large pages, you can switch it off in the vectorwise.conf file by setting [memory] use\_huge\_tlb (see page 83) to FALSE.

## Requirements for Huge Pages in Linux

To enable the use of huge pages in VectorWise the following is needed:

- Kernel support. (Most distributions enable this in the standard kernel.)
- libhugetlbfs library must be installed.

For easier administration of this feature, extra tools are recommended, often found in a package called libhugetlbfs-utils or similar.

## Designate Memory for Huge Pages on Linux

The commands and amounts provided here is an example of designating memory for huge pages on Linux. Before issuing these commands, understand what they do, and adapt the examples, as needed. Reserving pages for use in huge page allocations is system wide, so make sure you are not interfering with other users.

To make 2 GB available for 2 MB huge pages, issue the following commands as root on the command line before starting VectorWise:

```
hugeadm --create-global-mounts
```

```
hugeadm --pool-pages-min 2M:1024
```

To switch off, enter the following command as root on the command line:

```
hugeadm --pool-pages-min 2M:0
```

To check if memory is allocated for huge pages and how much of it is in use, type at the command line:

```
cat /proc/meminfo
```

The information about huge pages is shown in the following example lines:

```
HugePages_Total: 1024
HugePages_Free: 1024
HugePages_Rsvd: 0
HugePages_Surp: 0
```

Making memory available for huge pages requires defragmenting the specified amount of memory, so it can take a while. Typically, it is fastest to do this immediately after system startup, when memory is not as fragmented.

For more in-depth information, see the man page of hugeadm, <http://linux-mm.org/HugePages>, and `vm/hugetlbpage.txt` in the Linux kernel documentation ([http://git.kernel.org/?p=linux/kernel/git/torvalds/linux-2.6.git;a=blob\\_plain;f=Documentation/vm/hugetlbpage.txt;hb=HEAD](http://git.kernel.org/?p=linux/kernel/git/torvalds/linux-2.6.git;a=blob_plain;f=Documentation/vm/hugetlbpage.txt;hb=HEAD)).

## Using Multiple Databases

VectorWise fully isolates activities from various databases. Each database has its own storage system, buffer pool, and query memory pool. As a result, multiple databases running at the same time can lead to very high resource consumption. For optimal performance, we strongly recommend that only a single database be used on a machine.

If you plan to use multiple databases on one machine, we recommend that you reduce their resource consumption, mainly memory usage (`max_memory_size` and `bufferpool_size` options).

The current version of VectorWise has a hard limit of eight databases running at the same time. If eight databases are running that contain VectorWise tables, and you want to use another database, you must close one of the active databases.

### To close an active database

Issue this statement in a SQL client connected to the database you want to close:

```
call vectorwise (terminate)
```

If you then try to access VectorWise tables in this same database, the system will attempt to start the database automatically.

**Note:** Running this statement can interrupt working queries.

Also, you can simply run the **ingstop** command, which will terminate all open VectorWise databases.

## Error Reporting—Database Log File

For information about errors, look in the `vectorwise.log` file for the installation. The log is located in:

```
II_SYSTEM/ingres/files/vectorwise.log
```



## VectorWise SQL Settings

The following SQL commands affect the current session only:

### **SET RESULT\_STRUCTURE VECTORWISE**

Forces WITH STRUCTURE=VECTORWISE on all CREATE TABLE, CREATE TABLE AS SELECT, and DECLARE GLOBAL TEMPORARY TABLE statements.

The default for the installation is taken from the result\_structure parameter in Ingres config.dat.

### **SET RESULT\_STRUCTURE VECTORWISE\_ROW**

Forces WITH STRUCTURE=VECTORWISE\_ROW on all CREATE TABLE, CREATE TABLE AS SELECT, and DECLARE GLOBAL TEMPORARY TABLE statements.

The default for the installation is taken from the result\_structure parameter in Ingres config.dat.

### **SET [NO]VECTORWISE\_DEFAULT**

(Deprecated) Forces (or disables forcing) WITH STRUCTURE=VECTORWISE on all CREATE TABLE statements.

**Note:** This setting is deprecated. Use SET RESULT\_STRUCTURE instead.

### **SET QEP**

Displays a graphical version of the query plan.

### **SET TRACE POINT OP150**

Displays the contents of the compiled query.

### **SET TRACE POINT QE82**

Disables compression for tables subsequently created in the session.

VectorWise by default uses automatic table compression. If the dataset is small and fits into memory, disabling compression may slightly increase performance by eliminating the overhead associated with decompression.

For example:

```
SET TRACE POINT QE82;  
CREATE TABLE...
```

## Performance Tips

Following these tips will help to increase VectorWise performance:

- Always use `optimizedb` (see page 107).  
Without `optimizedb`, the SQL optimizer can order joins incorrectly.
- Use a large buffer pool.  
If possible, set `bufferpool_size` (see page 85) to fit in memory the data you need for most queries; otherwise, be prepared for some I/O. This is not as important if you have a high-end disk subsystem.
- Do not use compression on tables residing in memory.  
If your data fits in main memory uncompressed, and you want to maximize performance, you should consider disabling automatic compression, because decompression causes some delays. (Use **set trace point qe82** before creating tables.)
- Prefer NOT NULL fields over nullable fields for lower disk consumption and faster processing.  
Processing nulls always introduces overhead. Make sure you define a column as NOT NULL when it always contains a value.
- Prefer numerical attributes over character data types for higher data compression ratios.  
Processing string data is more expensive than numerical data (integer, floating point, decimal, date/time). In some cases, where a limited number of strings are used as codes, consider using integer codes (or, if possible, single character strings) for these.
- Prefer DECIMAL and INTEGER data types over FLOAT data types for higher data compression ratios.  
**Note:** DECIMAL values with a precision greater than 18 are not compressed.
- Try clustered indexes.  
VectorWise supports clustered indexes (a table organized as an index) using the CREATE INDEX syntax. Specify columns that are used often in selections (for example, a date column in a typical data warehouse). For details and restrictions, see Create an Index (see page 105).

- When designing a database scheme for large and growing data warehouse tables, you should carefully take into account the query workload and update workload:
  - As a general rule, indexed tables may speed up query access, but slow down bulk load and updates.
  - Batch updates should be avoided in extremely intensive (“firehose”) update workloads.
  - When working with very many small tables (thousands) or having very frequent small appends, reduce the [cbm] groupsize parameter for lower disk consumption.
- Try parallel query execution.

Parallel query execution (using multiple cores to execute a single query) works in a wide number of scenarios. For example, it improves the performance of the query described in Run a Reporting Query (see page 49). Parallel query execution can be enabled by setting the `max_parallelism_level` option (see page 87) in the configuration file.

**Note:** The terms parallel query execution or parallelism in this guide mean using multiple cores to execute one query (as compared to running multiple queries at the same time, which is always possible).



# Chapter 10: Creating a Database

---

This section contains the following topics:

[How to Create a Database](#) (see page 101)

[Create a Database](#) (see page 102)

[Connect to a Database](#) (see page 103)

[Create a Table](#) (see page 104)

[Create an Index](#) (see page 105)

[Methods for Loading Data](#) (see page 105)

[Create Statistics with Optimizedb](#) (see page 107)

## How to Create a Database

The process of creating a database and populating its tables includes the following steps:

1. Create a database.
2. Create tables.
3. (Optional) Create indexes.
4. Load the tables with data.
5. Generate statistics on the database.

## Create a Database

To create a database, you must have the `createdb` privilege, which is granted by default to the installation owner.

### To create a database

Issue this command at the command line:

```
createdb dbname
```

The database is created in the default locations for the installation.

By default the VectorWise database is UTF-8 based. By default the `createdb` command creates a Unicode-enabled database using Normalization Form C (NFC). If Normalization Form D (NFD) is required instead, you can create such a database using the `-n` option.

### To create a Unicode-enabled database with Normalization Form D (NFD):

Issue this command at the command line:

```
createdb -n dbname
```

## Database Default Configuration

Unless you changed the defaults during the installation, the database will configure 50% of the total system memory for processing and 25% of the total system memory for the data buffer.

By default the VectorWise data file is stored in `II_VWDATA/ingres/data/vectorwise/`. As more data is loaded into the database or operations spill to disk, the file will grow in size. Available data blocks inside the file are first reused before the file is extended. However, once extended the file cannot be shrunk without recreating the database.

For more information, see the chapter "Configuring and Managing VectorWise."

## Connect to a Database

To connect to a database and enter SQL commands, you can use the line-based Terminal Monitor.

### To connect to the database

1. Issue the **sql** command, as follows:

```
sql dbname
```

where *dbname* is the name of the database.

The Ingres Terminal Monitor starts.

2. Type your SQL statements at the terminal monitor prompt.
3. Enter **\g** to execute the statements.

### To disconnect from the database and exit the Terminal Monitor

Enter **\q**.

### More information

Terminal Monitor Commands (<http://docs.ingres.com/ingres/10.0/sql-reference-guide/1224-terminal-monitor-commands>)

## Create a Table

The CREATE TABLE statement automatically creates a table with VECTORWISE storage structure.

The new table is owned by the user who creates it. Only the owner (or a user with the security privilege impersonating the owner) can alter or drop a table. When you create a table, it is placed in the default location designated for the database's data files.

### To create a table with VECTORWISE storage

Enter your CREATE TABLE statement at the terminal monitor prompt. For example:

```
CREATE TABLE lineitem (  
    l_orderkey INTEGER NOT NULL,  
    l_partkey INTEGER NOT NULL,  
    l_suppkey INTEGER NOT NULL,  
    l_linenummer INTEGER NOT NULL,  
    l_quantity DECIMAL(2,0) NOT NULL,  
    l_extendedprice DECIMAL(8,2) NOT NULL,  
    l_discount DECIMAL(2,2) NOT NULL,  
    l_tax DECIMAL(2,2) NOT NULL,  
    l_returnflag CHAR(1) NOT NULL,  
    l_linestatus CHAR(1) NOT NULL,  
    l_shipdate ANSIDATE NOT NULL,  
    l_commitdate ANSIDATE NOT NULL,  
    l_receiptdate ANSIDATE NOT NULL,  
    l_shipinstruct CHAR(25) NOT NULL,  
    l_shipmode CHAR(10) NOT NULL,  
    l_comment VARCHAR(44) NOT NULL  
); \g
```

**Note:** If a column always contains a value then define the column as NOT NULL.

If your table has a large number of columns and/or relatively few rows, we recommend the VECTORWISE\_ROW storage type, which saves disk space and increases performance.

### To create a table with VECTORWISE\_ROW storage

Either type the SET RESULT\_STRUCTURE VECTORWISE\_ROW statement before the CREATE TABLE statement or use the WITH STRUCTURE=VECTORWISE\_ROW on the CREATE TABLE statement:

```
CREATE TABLE ...  
WITH STRUCTURE = VECTORWISE_ROW
```



## Create an Index

The CREATE INDEX statement in VectorWise creates a clustered index, which is a table organized as an index.

Use an index only if the table is predominantly accessed through the indexed columns. If the table is often filtered or joined only on non-indexed columns then the index will likely slow rather than improve query performance.

Restrictions are as follows:

- Create an index after creating the table, and only when the table is still empty.
- Only one index is allowed per table.
- The table can be populated only once, using COPY. Subsequent updates must be performed using the COMBINE command (see page 113).

### To create an index

Enter your CREATE INDEX statement at the terminal monitor prompt. The following example creates an index named l\_idx on the l\_shipdate column of the lineitem table:

```
CREATE INDEX l_idx ON lineitem(l_shipdate);
```

## Methods for Loading Data

After a VectorWise table is created, you can load data into it. Available methods are:

- COPY statement
- iivwfastload utility

## Load Data with COPY Statement

The following COPY FROM example loads the lineitem.tbl text file into the lineitem table. It is assumed this file is available in the local directory.

### To load the data into the lineitem table using COPY FROM

Type the following SQL statements at the Ingres Terminal Monitor prompt:

```
COPY TABLE lineitem (  
    l_orderkey = 'c0|',  
    l_partkey = 'c0|',  
    l_supkey = 'c0|',  
    l_linenum = 'c0|',  
    l_quantity = 'c0|',  
    l_extendedprice = 'c0|',  
    l_discount = 'c0|',  
    l_tax = 'c0|',  
    l_returnflag = 'c0|',  
    l_linestatus = 'c0|',  
    l_shipdate = 'c0|',  
    l_commitdate = 'c0|',  
    l_receiptdate = 'c0|',  
    l_shipinstruct = 'c0|',  
    l_shipmode = 'c0|',  
    l_comment = 'c0n1'  
) FROM 'lineitem.tbl' \g
```

For details on this command, see COPY (see page 127).

## Load Data with iivwfastload Utility

The iivwfastload command (see page 137) can be used to load data into a VectorWise table. It is less flexible but easier to use than COPY and often faster.

For example, the following command loads the data in the lineitem.txt file into the lineitem table of the dbt3 database. In the lineitem table, fields are delimited by | and records are delimited by \n.

### To load the data into the lineitem table using iivwfastload

Enter the following command at the operating system prompt:

```
iivwfastload --database dbt3 --table lineitem \  
    --datafile /mnt/lineitem.txt --fdelim '|' --rdelim '\n'
```

## Create Statistics with Optimizedb

The `optimizedb` command generates statistics that tell the query optimizer what your data looks like. The query optimizer uses the statistics to generate a query execution plan (QEP) that shows how your query is executed. The QEP can be reused to execute the same query.

We highly recommend running the `optimizedb` command if a significant percentage of the data (ten percent or greater) in a table has changed since the last time you ran `optimizedb`.

### To run `optimizedb`

Enter the following command at the operating system prompt:

```
optimizedb dbname -zfq
```

To speed the building of the histogram, use the `-zfq` flag.

You can run `optimizedb` for certain tables or you can run it once for the entire database.

We recommend that you generate the statistics for all columns that appear in the qualification (WHERE clause) of a query. If statistics are missing or incorrect, the query will still execute, but the speed of query processing can be affected.

For more information, see `optimizedb` (<http://docs.ingres.com/ingres/10.0/command-reference-guide/1380-optimizedb-commandgenerate-statistics-for-the-query-optimizer>) in the Ingres *Command Reference Guide*.



# Chapter 11: Managing the Database

---

This section contains the following topics:

[Methods for Updating Data](#) (see page 109)

[Methods for Backing Up and Restoring Data](#) (see page 117)

## Methods for Updating Data

Data can be inserted, updated, and deleted using these methods:

- As a batch or bulk operation using SQL statements or the `iivwfastload` command
- As a bulk operation using the `COMBINE` command

### Batch and Bulk Updates

Updating data in VectorWise tables can be done as either a batch or bulk operation.

SQL statements and the `iivwfastload` command map to batch and bulk DML operations as follows:

- `INSERT` is a batch operation, which quickly adds a small number of records.
- `DELETE` is a batch operation, which quickly deletes a small number of records.
- `UPDATE` is a batch operation, which quickly modifies a small number of records.
- `CREATE TABLE AS SELECT` is a bulk operation, which adds a large number of records.
- `INSERT...SELECT` is a bulk operation, which adds a large number of records.

`INSERT...SELECT` is a batch operation if executed on a non-empty indexed table.

- `COPY FROM` is a bulk operation, which adds a large number of records.  
`COPY FROM` is a batch operation if executed on a non-empty indexed table.
- `iivwfastload` is a bulk operation, which adds a large number of records.  
`iivwfastload` is a batch operation if executed on a non-empty indexed table.

COPY FROM, INSERT...SELECT, and iivwfastload are batch operations if executed on non empty indexed tables. For these operations to be bulk, the target indexed table must meet all these conditions:

1. The table is empty.
2. There have been no batch updates performed on this table since the last COMBINE call for the table.

**Note:** The table can be seen as empty, but there can still be batch updates stored for it in memory. This can happen if a user issues, for example, an INSERT or COPY command followed by a DELETE.

3. If the index is based on a foreign key, there can be no batch updates on the referenced table.

The following table summarizes the various update methods and their behavior depending on the target table:

Operation	Raw Table	Indexed Table when Empty	Indexed Table when Not Empty
INSERT	Batch	Batch	Batch
DELETE	Batch	Batch	Batch
UPDATE	Batch	Batch	Batch
COPY FROM	Bulk	Bulk	Batch
CREATE TABLE AS SELECT	Bulk	Not applicable	Not applicable
INSERT...SELECT	Bulk	Bulk	Batch
iivwfastload	Bulk	Bulk	Batch

An alternative method of updating data is combining tables (see page 112).

## Transfer Data from Traditional Ingres Table

Data can be transferred in either direction between traditional Ingres tables and VectorWise tables using either INSERT...SELECT or CREATE TABLE AS SELECT.

### To transfer data from traditional Ingres using INSERT..SELECT

1. Create a destination table with VECTORWISE storage type:

```
CREATE TABLE dst (i INTEGER)
```

2. Issue INSERT..SELECT into that table:

```
INSERT INTO dst SELECT * FROM src
```

### To transfer data from traditional Ingres using CREATE TABLE AS SELECT

Use a statement similar to the following:

```
CREATE TABLE dst AS SELECT FROM src
```

## Combining Tables

Since DML operations can be costly in terms of resources or even impossible for some datasets if the batch update mode is used, an alternative way to apply data updates is to call the COMBINE command. This process merges the updates buffered in memory, and at the same time provides a way for performing bulk DML operations on any form of a table.

For example:

```
DELETE FROM tab WHERE x > y
```

can be implemented with these statements:

```
CREATE TABLE deletions AS
  SELECT key FROM tab WHERE x > y;
CALL VECTORWISE (COMBINE 'tab-deletions');
DROP TABLE deletions;
```

For example:

```
UPDATE tab SET x = x + 1 WHERE x > y
```

can be implemented with these statements:

```
CREATE TABLE updates AS
  SELECT key, CAST(x + 1 AS int) AS x, y WHERE x > y;
CALL VECTORWISE (COMBINE 'tab-updates+updates');
DROP TABLE updates;
```

### Notes:

- Table "tab" has a primary key on the "key" column.
- The COMBINE command works only if the column types and names of the new table match those of the original table. The example assumes that the type of column x is INT. When creating the updates table, the expression is casted so that it matches the type of the column in the original table. The expression "x+1" is remapped to the original column name ("AS x").

You can use similar solutions for other DML operations.

For details on this command, see [COMBINE Command—Merge and Update Data](#) (see page 113).



## How to Avoid Propagation

During batch operations VectorWise automatically propagates the changes buffered in memory to the disk-resident table. Because such propagation is costly, it is best to avoid frequent propagation to large tables by using one of the following approaches:

- For large tables with frequent inserts, consider using RAW (see page 18) storage format.
- For INDEXED tables, keep the volume of data changed below what will easily fit in RAM.

**Note:** The data buffered in memory is not compressed.

- If INDEXED format must be used for query performance, stage batches of inserts and deletes in one or more separate tables and use the COMBINE command (see page 113) to add them to the table.
- Enforce propagation at a convenient moment, as described in COMBINE Command—Merge and Update Data (see page 113).

## COMBINE Command—Merge and Update Data

For large updates, we recommend using a COPY command to initially load the data into staging tables, and then using explicit COMBINE commands.

The syntax of the CALL VECTORWISE COMBINE command is as follows:

```
CALL VECTORWISE( COMBINE 'PARAMETERS' )
```

where:

*PARAMETERS* can be a comma-separated list of merging commands. Each merging command has the form:

```
BASETABLE -DELETIONTABLE1 -DELETIONTABLE2... +INSERTIONTABLE1 +INSERTIONTABLE2...
```

The previous command tells the system that all tuples from deletion tables must be deleted from the base table, and then all tuples from insertion tables must be added to that table. This command generates a new copy of the base table.

### Usage Notes

- Deletion tables must contain attributes with names that match the Primary Key in the base table. These attributes do not need to be a Primary Key in the deletion table. If the base table does not contain a Primary Key, the format of the deletion table must match exactly the format of the base table, because the entire record is used to identify the tuple to be deleted.
- Insertion tables must match the columns of the base table: the names, orders and types (including NULLability) of columns should be identical. The insertion tables do not need to contain any primary- and foreign-key constraints (see below).
- The order of parameters (after the base table) does not matter: all deletions are applied before all insertions.
- Nothing should be assumed about the order of merging of various base tables. For example, in the following call:  

```
CALL VECTORWISE( COMBINE 'a+a1, x+a')
```

it is not guaranteed that tuples from 'a1' will be added to 'x'.
- This command can also be used to quickly delete all tuples from a given table by issuing, for example:  

```
CALL VECTORWISE( COMBINE 'a-a')
```

The system detects this as a special case and performs a very quick operation.
- You can replace all data in a given table with other data. For example:  

```
CALL VECTORWISE( COMBINE 'a-a+b')
```

replaces all tuples in 'a' with tuples from 'b', again with a faster operation.
- This command is a DML command, and as a result does not allow other concurrent updates.
- This command can return all standard errors reported by DML operations, for example, constraint violations.
- Spaces are ignored. You can use double quotes around complex table names. For example:  

```
CALL VECTORWISE( COMBINE 'tab + "tab insertions"')
```

- Specifying the same base table twice as a base table is an error. It is legal to use the same table multiple times as an insertion or deletion table. It is also legal to use it for self-deletion ('a-a').
- Adding the base table to itself (for example: "tab+tab") is illegal.
- The COMBINE process also applies batch ("small") updates buffered in memory to the new copy of a table—a process known as propagation (see page 22). You can enforce propagation as follows:

```
CALL VECTORWISE( COMBINE 'tab')
```

After the COMBINE, the memory used by the buffered updates is freed.

- The COMBINE command lets you perform large deletes. For example:  

```
CREATE TABLE deletions AS SELECT key FROM basetable WHERE orderdate < '2010-01-01';  
CALL VECTORWISE( COMBINE 'basetable-deletions');  
DROP TABLE deletions;
```

- The constraints and indexes in the staging tables are irrelevant for the merging process.

In particular, it is possible to have a non-indexed table and merge it (both as deletions and insertions) into an indexed table. Similarly, if the base table has a primary key or a foreign key, the staging tables do not have to have the same primary key defined (as long as the deletions tables contain columns matching the primary key in the base table and insertion tables match the column structure of the base table). If the staging tables are not used for querying, we highly recommend that they do not contain constraints (foreign or primary keys) or indexes, to improve their performance in loading data.

- The nullability of the columns in the insertion and deletion tables must match the base table.
- This command works on VectorWise tables only.

## Best Practices for Updates

The most efficient way to load data into VectorWise is to use bulk append (using COPY INTO or iivwfastload utility). You can use bulk append, however, only for tables in RAW format (that is, without an index) if the table is not empty. The performance cost of this method is roughly proportional to the volume of data appended.

When planning an append strategy, consider the granularity of appends (see page 21). To maximize efficiency, each append should use multiple disk blocks. The default block\_size (see page 85) is 512 KB. Because VectorWise stores data as compressed columns, during append, each column in a table will consume at least one group of blocks. For good disk usage, we recommend that each column receive a few megabytes of appended data. That means that the granularity of appends should be preferably in millions of records.

For small-cardinality data modifications, you can use the standard INSERT/DELETE/MODIFY commands, which work on both RAW and indexed tables. However, for large-cardinality deletions and updates, as well as for appends to an indexed table, you should use the explicit COMBINE method (see page 112), where you first create staging tables that contain deletions and updates, and then merge these into your tables in bulk fashion.

The performance cost of the COMBINE method can be high because it is roughly proportional to the total volume of data in the table, so it should be used only when modifying a significant percentage of a table. The benefit of this approach is that it allows very large modifications, results in lower memory consumption, and provides higher processing performance after the update.

## Methods for Backing Up and Restoring Data

VectorWise can only be backed up when there is no ongoing DML or DDL.

To back up VectorWise you can use the following methods:

- Checkpointing
- Copy the needed files to a backup location
- Copy the database using the copydb command

Automatic incremental backups are not currently supported. Develop a backup and recovery strategy that works for you. For example, if you load your VectorWise database through data files then you may take a weekly full backup on Saturday night and save all data files until the next full backup. If the database must be restored, then you restore the most recent full backup and apply all subsequent data loads to recover the database to the most current point in time.

### Backup and Recovery with Checkpoints

A checkpoint is a snapshot of your entire database.

You take a checkpoint (back up) your database by using the ckpdb command; you restore it from the checkpoint using the rollforwarddb command.

VectorWise does not support the use of journals to roll forward the database beyond the checkpoint. Whenever a database contains a VectorWise table, journaling is not an option.

Full database backup and recovery operations using the ckpdb and rollforwarddb commands apply to both traditional Ingres tables (if present) and VectorWise tables.

## Checkpoint (Back up) a Database with Ckpdb

A checkpoint is a snapshot of the database. Checkpoints in VectorWise can be taken when the database is either offline (no one is using the database) or online (other users are accessing the data for READONLY).

### To checkpoint a database offline

Issue the following command at the operating system prompt:

```
ckpdb -l dbname
```

A new offline checkpoint for the specified database is created. The -l flag causes the checkpoint to be taken offline. When using the -l flag, you can also use the "wait" flag (+w or -w):

#### **+w**

Waits for as long as necessary for the database to be free before taking the checkpoint.

#### **-w**

(Default) Returns an error message if the database is busy.

### To checkpoint a database online

Issue the following command at the operating system prompt:

```
ckpdb dbname
```

A new online checkpoint for the specified database is created. The checkpoint operation may stall for as long as required to quiesce the VectorWise Server into a READONLY state.

#### **Notes:**

1. New READ transactions will connect and be able to access data without affecting the checkpoint.
2. New WRITE transactions will fail to execute during the backup due to the READONLY state.

## Recover a Database with Rollforwarddb

Performing a roll forward (restore) of a database overwrites the current contents of the database being recovered. To perform a roll forward, you must be the DBA for the database or have the operator privilege.

The roll forward operation restores the database from the checkpoint location to the database location.

### To recover a database from the last checkpoint

Issue the following command at the operating system prompt:

```
rollforwarddb dbname
```

Upon restart, the database will be in the same state it was when you took the checkpoint.

### To recover a database from an old checkpoint

1. Issue the infodb command to see the list of valid checkpoints:

```
infodb dbname
```

2. Recover the database from the specific checkpoint:

Linux:

```
rollforwarddb '#cn' dbname
```

Windows:

```
rollforwarddb #cn dbname
```

where *n* is the checkpoint sequence number.

## Best Practices for Checkpoints

Keep as many checkpoints as is feasible. Doing so gives you more recovery options. You may want to delete older checkpoints. Use alterdb –delete\_oldest\_ckp rather than ckpdb –d (delete all previous checkpoints).

Make sure the checkpoints work; otherwise they are useless.

## Copy Files to a Backup Location

You can use operating system commands to copy the needed files to a backup location.

### To copy files

1. Issue **infodb dbname** to view all the directories related to the database.
2. Shut down the instance.

**Warning:** If you move files when VectorWise is running or without a clean shutdown, the database can become corrupted.

If you are unsure that VectorWise shut down cleanly, do the following:

- a. Start VectorWise again with the **ingstart** command.
- b. Ensure that you can access the database (for example: **sql dbname** then **select count(\*)** from both an Ingres table and a VectorWise table, and then **\q** to quit).

**Note:** Simply accessing the database does not start the VectorWise Server.

- c. Stop VectorWise with the **ingstop** command.
3. Copy the database file, journal, and dump files to the backup location. Create the work directories in the backup location, but you do not need to copy the work files.

### To restore the files

1. Stop VectorWise using the **ingstop** command.
2. Copy the files back to their original location.
3. Restart VectorWise using the **ingstart** command.



## Copy the Database with Copydb

The copydb command creates two scripts:

### **copy.out**

Contains query language statements to copy your tables to operating system files. The script contains a COPY statement for each table being copied.

You run the copy.out script (using the sql command) to copy tables out of the database.

### **copy.in**

Contains query language statements to recreate your tables, views, and associated indexes, permissions, and integrities, and copy the table's data from the operating system files into a database.

You run the copy.in script (using the sql command) to copy the tables into the same or another database.

You can run copydb without shutting down the database; however, copydb needs to acquire locks while reading system catalogs, so it will wait for any users that are running DML concurrently to finish.

### **To back up tables with copydb**

1. Use operating system commands to create a temporary working directory for the copy.in and copy.out scripts that will be created, and then move to this directory. For example:

#### **Linux:**

```
mkdir /tmp/mydir.backup  
cd /tmp/mydir.backup
```

#### **Windows:**

```
mkdir D:\tmp\mydir.backup  
D:  
cd \tmp\mydir.backup
```

2. Back up all the tables and views that you own in the database by issuing the following command at the operating system prompt:

```
copydb dbname
```

To back up specified tables, issue this command:

```
copydb dbname tablename {tablename}
```

Copy.out and copy.in scripts are created.

3. Copy the data out of the database by running the copy.out script. Issue the following command at the operating system prompt:

```
sql dbname <copy.out
```

A copy of the objects copied from the database is created. Store these files on disk or tape.

**To restore the data from a backup created by copydb**

Run the copy.in script as follows:

```
sql dbname <copy.in
```

The tables are copied into the specified database.

## PART 4: Reference

---



# Appendix A: SQL Reference

---

This section contains the following topics:

[ALTER TABLE](#) (see page 125)  
[CALL VECTORWISE](#) (see page 126)  
[COMMIT](#) (see page 126)  
[COPY](#) (see page 127)  
[CREATE INDEX](#) (see page 128)  
[CREATE TABLE](#) (see page 129)  
[CREATE VIEW](#) (see page 130)  
[DECLARE GLOBAL TEMPORARY TABLE](#) (see page 130)  
[DELETE](#) (see page 131)  
[DROP](#) (see page 131)  
[INSERT](#) (see page 131)  
[ROLLBACK](#) (see page 132)  
[SELECT \(Interactive\)](#) (see page 132)  
[SELECT \(Embedded\)](#) (see page 135)  
[UPDATE](#) (see page 136)

## ALTER TABLE

Valid in: SQL, ESQL, OpenAPI, ODBC, JDBC, .NET

The ALTER TABLE statement can be used to:

- Add or remove a column from a base table

**Note:** Adding or dropping a column is allowed only if there are no in-memory DML changes against the table. Use the CALL VECTORWISE command to write the in-memory changes to disk.

This statement has the following format:

```
[EXEC SQL] ALTER TABLE [schema.]table_name
    ADD [column] column_name format [default_clause]
    [null_clause] [column_constraint]
| DROP [COLUMN] column_name
```

Constraints at the table and column levels can be unique, primary key, or referential. For referential constraints, only the referential actions RESTRICT and NO ACTION are supported.

**Note:** Adding a constraint is allowed only if the table is empty.

## CALL VECTORWISE

Valid in: SQL, ESQL

The CALL VECTORWISE statement calls a VectorWise system command.

This statement has the following format:

```
CALL VECTORWISE( COMMAND ' PARAMETERS' )
```

where:

### **COMMAND**

Specifies the name of the system command.

Commands supported include COMBINE (see page 113).

### **'PARAMETERS'**

Specifies one or more parameters specific to the called system.

If *parameters* is a null, empty, or blank string, the statement transfers the user to the operating system and the user can execute any operating system command. Exiting or logging out of the operating system returns the user to the application.

**Note:** The syntax of this statement may change in the future.

## COMMIT

Valid in: SQL, ESQL, OpenAPI, ODBC, JDBC, .NET

The COMMIT statement commits the current transaction.

This statement has the following format:

```
[EXEC SQL] COMMIT [WORK]
```

**Note:** The optional keyword WORK is included for compliance with the ISO and ANSI standards for SQL.

## COPY

Valid in: SQL, ESQL, OpenAPI

The COPY statement copies the contents of a table to a data file (COPY INTO) or copies the contents of a file to a table (COPY FROM).

This statement has the following format:

```
[EXEC SQL] COPY [TABLE] [schema.]table_name
    ([column_name = format [WITH NULL [(value)]]
    {, column_name = format [WITH NULL [(value)]}])
    INTO | FROM 'filename[, type]'
    [WITH ON_ERROR=TERMINATE|CONTINUE]
    [WITH ERROR_COUNT=n] [WITH LOG='filename']
```

For information about specifying column formats, see Column Formats for COPY (<http://docs.ingres.com/ingres/10.0/sql-reference-guide/1685-column-formats-for-copy>) in the Ingres *SQL Reference Guide*.

## CSV and SSV Delimiters

Delimiters are characters in the data file that separate fields and mark the end of records.

Delimiters CSV and SSV allow COPY to read and write files that contain comma separated values (CSV).

The rules for a CSV delimited field are:

- The field is delimited by a comma, unless it is the last field in the COPY list; in that case, the field is delimited by a newline.
- COPY FROM: If the first non-blank character in the field is a double quote ("), the field extends until a closing double quote. Commas or newlines inside the quoted string are not delimiters and do not end the value. If a doubled double quote (") is seen while looking for the closing quote, it is translated to one double quote and the value continues. For example, the data file value:

"There is a double quote "" here"

is translated to the table value:

There is a double quote " here

Whitespace before the opening double quote, or between the closing double quote and the delimiter (comma or newline), is not part of the value and is discarded.

- COPY INTO: If the value to be written contains a comma, newline, or double quote, it is written enclosed in double quotes using quote doubling as described in the previous bullet item. If the value does not contain a comma, newline, or double quote, it is written as is.

The SSV delimiter works exactly the same as the CSV delimiter, with semicolon in place of comma.

CSV and SSV delimiters are only allowed with BYTE(0), C0, CHAR(0), and TEXT(0). They are not allowed with the "counted" formats (VARCHAR(0) and so on); the count defines the value exactly and there is no need for quoting. (If delimiting is desired, use the comma or nl delimiters on counted formats.)

COPY FROM: Some CSV file variants use quote escaping (\\") instead of quote doubling (""") to indicate a quote inside a quoted string. The C format handles escaping, so use the C0CSV format and delimiter to handle this type of file. (CSV with COPY INTO always writes quote doubling (never quote escaping) when needed.)

Other valid delimiters are listed in Delimiters in the Data File (<http://docs.ingres.com/ingres/10.0/sql-reference-guide/2236-delimiters-in-the-data-file>) in the Ingres *SQL Reference Guide*.

## CREATE INDEX

Valid in: SQL, ESQL, OpenAPI, ODBC, JDBC, .NET

The CREATE INDEX statement creates a clustered index on an existing table.

This statement has the following format:

```
[EXEC SQL] CREATE INDEX [schema.] index_name
                   ON [schema.] table_name
                   (column_name {, column_name})
```

### More information

Create an Index (see page 105)



## CREATE TABLE

Valid in: SQL, ESQL, OpenAPI, ODBC, JDBC, .NET

The CREATE TABLE statement creates a base table.

This statement has the following format:

```
[EXEC SQL] CREATE TABLE [schema.] table_name
    (column_specification {, column_specification }
    [table_constraint {, table_constraint}]
    [WITH STRUCTURE = VECTORWISE | VECTORWISE_ROW]
```

The *column\_specification* has the following syntax:

```
column_name datatype
[([WITH] DEFAULT default_spec | WITH DEFAULT | NOT DEFAULT]
[WITH NULL | NOT NULL]
[GENERATED ALWAYS AS [seq_name] IDENTITY [(seq_options)]
 | GENERATED BY DEFAULT AS [seq_name] IDENTITY [(seq_options)]
[column_constraint {, column_constraint}]
```

where *seq\_name* and *seq\_options* are optional Identity Column (<http://docs.ingres.com/ingres/10.0/sql-reference-guide/4813-identity-columns>) qualifiers.

Constraints at the table and column levels can be unique, primary key, or referential. For referential constraints, the only referential actions supported are RESTRICT and NO ACTION.

The CREATE TABLE...AS SELECT statement (which creates a table and loads rows from another table) has the following format:

```
[EXEC SQL] CREATE TABLE table_name
    (column_name {, column_name}) AS
        subselect
        {UNION [ALL]
        subselect}
    [WITH STRUCTURE = VECTORWISE | VECTORWISE_ROW]
```

**Note:** For VectorWise tables, defaults, nullability, and identity columns work only for INSERT...VALUE, not INSERT SELECTs.

## CREATE VIEW

Valid in: SQL, ESQL, OpenAPI, ODBC, JDBC, .NET

The CREATE VIEW statement defines a virtual table.

This statement has the following format:

```
[EXEC SQL] CREATE VIEW view_name
    [(column_name {, column_name} )]
    AS select_stmt
```

## DECLARE GLOBAL TEMPORARY TABLE

Valid in: SQL, ESQL, OpenAPI, ODBC, JDBC, .NET

The DECLARE GLOBAL TEMPORARY TABLE statement creates a temporary table.

This statement has the following format:

```
[EXEC SQL] DECLARE GLOBAL TEMPORARY TABLE [SESSION.] table_name
    (column_name format {, column_name format})
    ON COMMIT PRESERVE ROWS
    WITH NORECOVERY
    [with_clause]
```

To create a temporary table by selecting data from another table:

```
[EXEC SQL] DECLARE GLOBAL TEMPORARY TABLE [SESSION.] table_name
    (column_name {, column_name})
    AS subselect
    ON COMMIT PRESERVE ROWS
    WITH NORECOVERY
    [with_clause]
```

For more information, see Declare Global Temporary Table Syntax (<http://docs.ingres.com/ingres/10.0/sql-reference-guide/2208-declare-global-temporary-table-syntax>).

## DELETE

Valid in: SQL, ESQL, OpenAPI, ODBC, JDBC, .NET

The DELETE statement deletes rows from the specified table that satisfy the *search\_condition* in the WHERE clause. If the WHERE clause is omitted, the statement deletes all rows in the table. The result is a valid but empty table.

This statement has the following formats:

Interactive and database procedure version:

```
[EXEC SQL] DELETE FROM [schema.] table_name [corr_name]
                [WHERE search_condition];
```

Embedded non-cursor version:

```
[EXEC SQL] [REPEATED] DELETE FROM [schema.] table_name [corr_name]
                [WHERE search_condition];
```

**Note:** REPEATED queries do not work if they contain host variables.

## DROP

Valid in: SQL, ESQL, OpenAPI, ODBC, JDBC, .NET

The DROP statement destroys one or more tables or views.

This statement has the following format:

```
[EXEC SQL] DROP TABLE | VIEW [schema.] objectname {, [schema.] objectname};
```

## INSERT

Valid in: SQL, ESQL, OpenAPI, ODBC, JDBC, .NET

The INSERT statement inserts rows into a table.

This statement has the following format:

```
[EXEC SQL [REPEATED]] INSERT INTO [schema.] table_name
                [(column {, column})]
                [VALUES (expr{, expr})] | [subselect];
```

**Note:** REPEATED queries do not work if they contain host variables.

## ROLLBACK

Valid in: SQL, ESQL, OpenAPI, ODBC, JDBC, .NET

The ROLLBACK statement rolls back the current transaction.

This statement has the following format:

```
[EXEC SQL] ROLLBACK [WORK];
```

## SELECT (Interactive)

Valid in: SQL, OpenAPI, ODBC, JDBC, .NET

The SELECT (interactive) statement returns values from tables or views.

### Notes:

- SELECT queries that access both VectorWise and standard Ingres tables cannot execute.
- Similarly, INSERT...SELECT and CREATE AS SELECT do not work if the SELECT statement references both VectorWise and Ingres tables.

This statement has the following format:

```
SELECT [FIRST rowCount] [ALL | DISTINCT] * | expression [AS result_column]  
      {, expression [[AS] result_column]}  
      [FROM from_source {, from_source}]  
      [WHERE search_condition] WHERE (clause)  
      [GROUP BY expression{, expression}] GROUP BY (clause)  
      [HAVING search_condition] HAVING (clause)  
      {UNION [ALL]  
      (select)}  
      [ORDER BY ordering-expression [ASC | DESC]  
            {, ordering-expression [ASC | DESC]}};  
      [WITH options]
```

where:

The SELECT clause specifies which values are to be returned.

The FROM clause specifies the source tables and views from which data is to be read. The *from\_source* parameter can be:

- One or more tables or views, specified using the following syntax:

```
[schema.]table [[AS] corr_name]
```

where *table* is the name of a table, view, or synonym, and *schema* is the name of the user that owns the table.

- A join between two or more tables or views, specified using the following syntax:

*source join\_type JOIN source ON search\_condition*

or

*source join\_type JOIN source USING (column {, column})*

or

*source CROSS JOIN source*

where:

***source***

Specifies the table, view, or join where the data for the left or right side of the join originates.

***join\_type***

Specifies the type of join as one of the following:

**INNER**—(Default) Specifies an inner join.

**LEFT [OUTER]**—Specifies a left outer join, which returns all values from the left source.

**RIGHT [OUTER]**—Specifies a right outer join, which returns all values from the right source.

**FULL [OUTER]**—Specifies a full outer join, which returns all values from both left and right sources.

**Note:** RIGHT and LEFT joins are the mirror image of each other: (table1 RIGHT JOIN table2) returns the same results as (table2 LEFT JOIN table1).

***ON search\_condition***

Is a valid restriction, subject to the rules for the WHERE clause. The *search\_condition* must not include aggregate functions or subselects. Matching pairs of rows in the join result are those that satisfy the *search\_condition*.

***USING (column {,column})***

Is an alternate form of the *search\_condition*. Each column in the USING clause must exist unambiguously in each join source. An ON *search\_condition* is effectively generated in which the *search\_condition* compares the columns of the USING clause from each join source.

**CROSS JOIN**

Is a cross product join of all rows of the join sources.

**Note:** VectorWise and Ingres tables cannot be joined.

- A derived table specified using the following syntax:

```
(select_stmt) corr_name [(column_list)]
```

where *select\_stmt* is a SELECT statement with no ORDER BY clause, *corr\_name* is a mandatory correlation name, and *column\_list* is an optional list of override names for the columns in the SELECT list of the *select\_list*.

The WHERE clause specifies criteria that restrict the contents of the results table. Tests can be performed for simple relationships or, using subselects, for relationships between a column and a set of columns.

The GROUP BY clause combines the results for identical values in a column or expression. This clause is typically used with aggregate functions to generate a single figure for each unique value in a column or expression.

The HAVING clause filters the results of the GROUP BY clause in the same way the WHERE clause filters the results of the SELECT...FROM clauses. The HAVING clause uses the same restriction operators as the WHERE clause.

A subselect (also known as a subquery) is a SELECT statement placed in a WHERE or HAVING clause. The results returned by the subselect are used to evaluate the conditions specified in the WHERE or HAVING clause. Subselects must return a single column, and cannot include an ORDER BY or UNION clause.

The ORDER BY clause lets you specify the columns on which the results table is to be sorted.

The UNION clause combines the results of SELECT statements into a single result table. By default, the UNION clause eliminates any duplicate rows in the result table. To retain duplicates, specify UNION ALL.

The WITH clause consists of a comma-separated list of one or more options. For details, see WITH Clause for SELECT

(<http://docs.ingres.com/ingres/10.0/sql-reference-guide/3707-with-clause-for-select>) in the Ingres *SQL Reference Guide*.

## SELECT (Embedded)

Valid in: ESQL

The SELECT statement returns values from tables to host language variables in an embedded SQL program. For details about the various clauses of the SELECT statement, see SELECT (Interactive) (see page 132).

This statement has the following format:

Non-cursor version:

```
EXEC SQL [REPEATED] SELECT [FIRST rowCount] [ALL | DISTINCT]
      INTO variable[:indicator_var] {, variable[:indicator_var]}
      [FROM from_source {, from_source}
      [WHERE search_condition]
      [GROUP BY column {, column}]
      [HAVING search_condition]
      [UNION [ALL] full_select]
      [ORDER BY ordering-expression [ASC | DESC]
              {, ordering-expression [ASC | DESC]}]
      [WITH options]
[EXEC SQL BEGIN;
      program code;
EXEC SQL END;]
```

Cursor version (embedded in a DECLARE CURSOR statement):

```
SELECT [ALL|DISTINCT]
      SELECT [FIRST rowCount] [ALL | DISTINCT]
      [FROM from_source {, from_source}
      [WHERE search_condition]
      [GROUP BY column {, column}]
      [HAVING search_condition]
      [UNION [ALL] full_select]
      [ORDER BY result_column [ASC|DESC]
              {, result_column [ASC|DESC]}]
      [WITH options]
```

where *result\_expression* is:

*expression* | *result\_name* = *expression* | *expression* AS *result\_name*

**Note:** REPEAT queries do not work if they contain host variables.

## UPDATE

Valid in: SQL, ESQL, OpenAPI, ODBC, JDBC, .NET

The UPDATE statement updates column values in a table.

This statement has the following formats:

Interactive version:

```
UPDATE [schema.] table_name [corr_name]  
      [FROM [schema.] table_name [corr_name]  
      { , [schema.] table_name [corr_name] }]  
      SET column_name = expression {, column_name = expression}  
      [WHERE search_condition];
```

Embedded non-cursor version:

```
EXEC SQL [REPEATED] UPDATE [schema.] table_name [corr_name]  
      [FROM [schema.] table_name [corr_name]  
      { , [schema.] table_name [corr_name] }]  
      SET column = expression {, column = expression}  
      [WHERE search_condition];
```

**Note:** REPEATED queries do not work if they contain host variables.



# Appendix B: Command Reference

---

This section contains the following topics:

[iivwfastload Command—Load Data into VectorWise Tables](#) (see page 137)  
[iivwinfo Command—Display Information about a Database with VectorWise Tables](#) (see page 138)

## iivwfastload Command—Load Data into VectorWise Tables

The iivwfastload command loads data into a VectorWise table.

This command has the following format:

```
iivwfastload --database database --table tablename --datafile datafile  
[--fdelim 'fielddelim'] [--rdelim 'recorddelim'] [--attrs {attr1,attr2...}]  
[--ingreschar] [--nullvalue nullvalue] [--help]
```

where options can be any of the following:

**--database *database***

Specifies the name of the database

**--table *tablename***

Specifies table to load to.

**--datafile *datafile***

Specifies data file to use.

**--fdelim '*fielddelim*'**

Specifies field delimiter to use.

Default: '\\t'

**--rdelim '*recorddelim*'**

Specifies record delimiter to use.

Default: '\\n'

**--attrs *attr1,attr2...***

Specifies list of attributes to load.

Default: all

**--ingreschar**

Specifies that the data for CHAR columns comes from the copydb output, and contains a 5-byte length prefix for each character field.

**--nullvalue *nullvalue***

String identifying NULL values.

Default: ' '

**--help**

Displays syntax information.

**More information**

Load Data with iivwfastload Utility (see page 106)

## iivwinfo Command—Display Information about a Database with VectorWise Tables

The iivwinfo command displays information about a database that uses VectorWise tables, including:

- Various statistics
- Configuration settings
- Disk usage of tables

This command has the following format:

```
iivwinfo [options] [-h] dbname
```

***options***

Specify options to the command.

**-h**

Displays the options for the iivwinfo command. For example: **iivwinfo -h**.

***dbname***

Specifies the name of the database.

**More information**

View Information about a Database with VectorWise Tables (see page 90)

# Appendix C: VectorWise Limits

---

This section contains the following topics:

[VectorWise Limits](#) (see page 139)

## VectorWise Limits

Following is a summary of VectorWise limits:

Parameter	Limit
Maximum database size	Unlimited
Maximum tables in a database	1.1 billion
Maximum files per database	N/A
Maximum files per instance	N/A
Maximum page size	N/A
Maximum cache buffers	140 terabytes (2 <sup>47</sup> bytes)
Maximum rows per table	140 trillion (2 <sup>47</sup> )
Maximum row width per table	256 KB
Maximum fields (columns) per table	1024
Maximum indexes per table	1
Maximum file size per index	N/A
Maximum fields (columns) per index	32
Maximum integer size	64 bits (INTEGER8)
Maximum decimal precision	38 digits
Maximum float precision	64-bit IEEE (FLOAT8)
Maximum length of a character field	16,000 bytes
Maximum length of a varchar field	16,000 bytes
Maximum size of an SQL statement	Unlimited
Maximum members in an IN list	Unlimited
Maximum logical operators in a WHERE clause	Unlimited
Maximum join conditions	Unlimited

Parameter	Limit
Maximum tables in a join	126
Unicode support	UTF-8 and UCS-2 (UTF-16 without surrogate support)
XML support	Consume and publish

# Index

---

## 6

64-bit Linux • 55

## A

ALTER TABLE statement • 29, 125

autocommit • 22, 47

## B

backup • 117, 118, 120, 121

batch operations • 21, 22, 28, 84, 109, 113

benchmark • 43, 44

block\_size parameter • 18, 85, 89

buffer pool • 56, 85, 88, 89, 98

bufferpool\_size parameter • 85, 88, 98

bulk load • 44, 47

bulk operations • 21, 22, 28, 109, 112, 116

business intelligence • 43

## C

CALL VECTORWISE statement • 96, 112, 113, 126

checkpoint (backup) • 117, 119

ckpdb command • 117, 118, 119

clustered index • 29, 98, 128

ColumnBM • 85

COMBINE command • 22, 113, 116, 126

COMMIT statement • 29, 126

compression • 13, 19, 56, 98

configuration

    default • 87, 102

    file • 81

constraint checking • 37

COPY statement • 36, 47

CREATE INDEX statement • 128

CREATE TABLE statement • 14, 29, 32, 46, 129

CREATE VIEW statement • 130

## D

data analysis • 11

data directory • 53

date\_alias • 39

dbgen • 43, 44, 45, 47

DECLARE GLOBAL TEMPORARY TABLE  
    statement • 29, 130

decompression • 13, 98

DELETE statement • 29

disk

    bandwidth • 55, 56

    storage • 56

    subsystems • 55

DML operations • 21, 22, 28, 40, 109, 112

DROP statement • 29, 34, 131

## E

error log • 96

## F

file system • 53, 56

## G

group\_size parameter • 85, 89

## H

hardware configurations • 56

huge pages • 32, 83, 94

## I

I/O • 13, 53, 62, 85, 89, 98

II\_VWDATA • 59

iivwfastload command • 105, 137

iivwinfo command • 90, 138

indexed tables • 109, 113, 116

ingbuild utility • 70

ingprenv command • 75

ingres\_express\_install • 66

ingstart command • 77

ingstop command • 77

in-memory updates • 22, 84, 113

INSERT statement • 29, 36, 131

isolation • 22

---

## J

JDBC • 39, 41, 78

## L

large pages • 32, 83, 94  
libaio • 62  
loading data • 105, 106  
log file • 96

## M

max\_memory\_size parameter • 82, 85, 88, 89  
max\_parallelism\_level parameter • 87

## N

NOT NULL • 46, 98, 129  
nulls • 20

## O

optimizedb • 21, 32, 48, 98, 107

## P

parallel query • 32, 87, 98

## Q

query • 21

## R

RAID • 53, 55, 56, 89  
RAM • 12, 56  
raw table • 18, 28, 109, 113, 116  
recovery • 22, 117, 119, 120, 121  
response file • 66, 72  
restrictions • 35  
ROLLBACK statement • 29, 132  
rollforwarddb command • 117, 119

## S

SELECT statement • 29, 36, 37, 132, 135  
SET RESULT\_STRUCTURE statement • 27, 97, 104  
SSD • 55, 56, 89  
SSE • 12, 55, 84  
storage • 13, 14, 46, 53, 55, 56

formats • 17

VECTORWISE • 14, 27, 46, 97

VECTORWISE\_ROW • 27, 32, 97, 129

subqueries • 29, 36, 43

## T

trace point • 98

traditional Ingres • 11, 12, 14

## U

UCS\_BASIC collation • 28

UPDATE statement • 136

updating data

best practices • 116

methods • 109

UTF-8 • 28, 139

UTF8 character set • 60

## V

vectorised processing • 12

VECTORWISE storage structure • 14, 18, 27, 60, 97, 104, 111

vectorwise.conf • 81, 82, 83, 85

vectorwise.log • 96

VECTORWISE\_ROW storage structure • 14, 27, 97, 104