Vertica Database 2.0.5-0

SQL Programmer's Guide

Date of Publication: 1/8/2008

Copyright Notice

Vertica Systems, Inc.

Three Dundee Park Drive, Suite 102

Andover, MA 01810-3723

Phone: (978) 475-1070

Fax: (978) 475-6855

E-Mail: info@vertica.com

Web site: http://www.vertica.com
(http://www.vertica.com)

Trademarks

Open Source Software Acknowledgements

Boost

PostgreSQL

This product uses the PostgreSQL Database Management System(formerly known as Postgres, then as Postgres95)

Python Dialog

The Administration Tools part of this product uses Python Dialog,a Python module for doing console-mode user interaction.

Upstream Author:

Peter Astrand <peter@cendio.se>

Robb Shecter <robb@acm.org>

Spread

Although NOT REQUIRED, we at Spread Concepts would appreciate it if active users of Spread put a link on their web site to Spread's web site when possible. We also encourage users to let us know who they are, how they are using Spread, and any comments they have through either e-mail (spread@spread.org) or our web site at (http://www.spread.org/comments).

Contents

Technical Support

To submit problem reports, questions, comments, and suggestions, please use the Technical Support page on the Vertica Systems, Inc. Web site:

http://www.vertica.com/support (http://www.vertica.com/support)

You must be a registered user in order to access the page.

Before reporting a problem, please run the Diagnostics Utility described in the Troubleshooting Guide and attach the resulting .zip file.

About the Documentation

Where to Find the Vertica Documentation

Vertica Systems, Inc. recommends that you copy the Vertica documentation from the database server (any cluster host) to a client system on which you can use a browser and/or the Adobe Reader.

Database Server Systems

The Vertica Database Documentation Set is automatically installed in the **/opt/vertica/doc/** directory on all cluster hosts. If you have a browser and/or the Adobe Reader installed on a cluster host, you can access the documentation directly.

```
/opt/vertica/doc/HTML/Master/index.htm
/opt/vertica/doc/JDBC/index.htm
/opt/vertica/doc/PDF/book-name.pdf
```

Database Client Systems

To create a copy of the Vertica documentation on a client system, do one of the following:

- Download the documentation package (**.tar.gz** or **.zip**) from the Vertica Systems, Inc. Web site and extract the files to a directory on the client system, using the original pathnames
- Copy the documentation directories from a database server system to a convenient location in your client system. All cross-references within the HTML documentation are relative so there is no location dependency.

```
<your-location>HTML/Master/index.htm
<your-location>JDBC/index.htm
<your-location>PDF/book-name.htm
```

World Wide Web

You can read and/or download the Vertica documentation from the **Vertica Systems, Inc. Web site**:

http://www.vertica.com/v-zone/product_documentation http://www.vertica.com/v-zone/product_documentation.

You need a V-Zone login to access the Product Documentation page.

The documentation on the Vertica Systems, Inc. Web site is updated each time a new release is issued. If you are using an older version, refer to the documentation on your database server or client systems.

Reading the HTML Files

The Vertica documentation files are provided in HTML browser format for platform independence. The HTML files only require a browser that can **display frames** properly and has **JavaScript** enabled. The HTML files **do not require a Web (HTTP) server**.

The Vertica documentation has been tested on the following browsers:

- Internet Explorer 7
- FireFox
- Opera
- Safari

Please report any script, image rendering, or text formatting problems to ***Technical Support*** (on page 11).

The Vertica documentation may contain links to Web sites of other companies or organizations that Vertica does not own or control. If any of these links are broken, please inform us.

Printing the PDF Files

The documentation files are supplied in **Adobe Acrobat™ PDF** document format for the purpose of making printed copies as needed. The documents are designed to be printed on standard 8½ x 11 paper using full duplex (two sided printing).

You can open and print any of the PDF documents using the **Adobe Reader**. (You can download the latest version of the free Acrobat Reader from the ***Adobe Web site*** (http://www.adobe.com/products/acrobat/readstep2.html).)

HTML links to the PDF files are provided here for browser access.

- Database Administrator's Guide
- Database Administrator's Guide (Advanced)
- Glossary of Terms
- Installation Guide
- Product Overview
- Quick Start
- Release Notes
- SQL Programmer's Guide
- SQL Reference Manual
- Troubleshooting Guide

Suggested Reading Paths

This section provides a suggested reading path for various types of users. Read the manuals listed under All Users first.

All Users

- Product Overview (basic concepts critical to understanding Vertica)
- Quick Start (step-by-step guide to getting Vertica up and running)
- Glossary of Terms (glossary of terms)

System Administrators

- Installation Guide (platform configuration and software installation)
- Release Notes (release-specific information)
- Troubleshooting Guide (general troubleshooting information)

Database Administrators

- Installation Guide (platform configuration and software installation)
- Database Administrator's Guide (database configuration, loading, security, and maintenance)
- Troubleshooting Guide (general troubleshooting information)

Application Developers

- SQL Programmer's Guide (connecting to a database, queries, transactions, etc.)
- SQL Reference Manual (Vertica-specific language information)
- Troubleshooting Guide (general troubleshooting information)

Where to Find Additional Information

Visit the ***Vertica Systems, Inc. Web site*** (http://www.vertica.com) to keep up to date with:

- Downloads
- Frequently Asked Questions (FAQs)
- Discussion forums
- News, tips, and techniques

Typographical Conventions

It is important to understand the terms and typographical conventions used in this document.

General Convention	Description
colored bold text	introduces new terms defined either in the text, the glossary, or both.
normal italic text	indicates emphasis and the titles of other documents.
UPPERCASE TEXT	indicates the name of an SQL command or keyword.
`monospace text`	indicates literal interactive or programmatic input/output.
`italic monospace text`	indicates user-supplied information in interactive or programmatic input/output.
`bold monospace text`	indicates literal interactive user input
↵	indicates the Return/Enter key; implicit on all user input that includes text

SQL Syntax Convention	Description
indentation	is an attempt to maximize readability; SQL is a free-form language.
backslash \	continuation character used to indicate text that is too long to fit on a single line.
braces { }	indicate required items.
brackets [ ]	indicate optional items.
ellipses ...	indicate an optional sequence of similar items.
vertical ellipses ⋮	indicate an optional sequence of similar items or that part of the text has been omitted for readability.
vertical line |	within braces or brackets, indicates a choice .

Preface

This document describes how to connect to a Vertica database and execute SQL statements. For information about the SQL language, see the SQL Reference Manual.

Audience

This document is intended for anyone who retrieves information from a Vertica database. It assumes that you are familiar with the basic concepts and terminology of the SQL language and relational database management systems.

Prerequisites

This document assumes that you have installed and configured Vertica as described in the Installation Guide.

Overview

As a Vertica SQL programmer, most of your tasks will be very similar to those required by other relational database management systems.

Connecting to a Vertica Database

There is nothing unusual about connecting to a Vertica database. You can ***connect interactively*** (page 23) using vsql or connect from an application program using JDBC or ODBC.

Writing Queries

Vertica is designed to execute queries that are suitable for a star schema or snowflake schema. You may need to modify existing normalized schema queries in order to execute them againSt a Vertica database.

Connecting with vsql

psql is a character-based, interactive, front-end that is part of ***PostgreSQL*** (http://www.postgresql.org/) and used by other database management systems. It allows you to type in SQL statements and see the results. It also provides a number of meta-commands and various shell-like features to facilitate writing scripts and automating a wide variety of tasks. On a Vertica host, psql is actually a symbolic link to vsql.

You can connect to vsql from the:

- ***Administration Tools*** (page 23)
- ***Linux command line*** (page 24)

General Notes

- SQL statements can be spread over several lines for clarity.
- You can cancel SQL statements by typing Ctrl+C.
- When you disconnect a user session, any transactions in progress are automatically rolled back.
- To view wide result sets, you can the Linux less utility to truncate long lines.
  1. Before connecting to the database, specify that you want to use less for query output:

     ```
     $ export PAGER=less
     ```
  2. Connect to the database.
  3. Query a wide table:

     ```
     => select * from wide_table;
     ```
  4. At the less prompt, type:

     ```
     -S
     ```
- If a shell running vsql fails (crashes or freezes), the vsql processes continue to run even if you stop the database. In that case, log in as root on the machine on which the shell was running and manually kill the vsql process. For example:

  ```
  # ps -ef | grep Vertica
      ⋮
  fred   2401     1  0 06:02 pts/1    00:00:00 /opt/Vertica/bin/vsql -
     p 5433 -h test01_site01 quick_start_single
      ⋮
  # kill -9 2401
  ```

Connecting From the Administration Tools

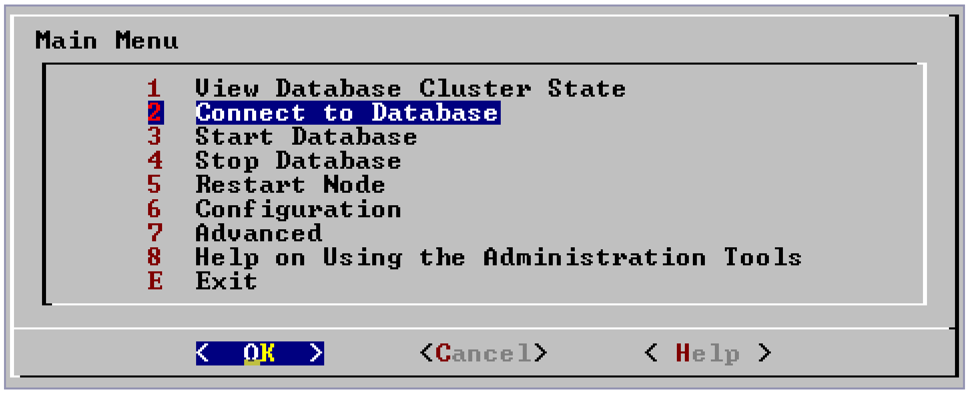
You can use the Administration Tools to connect to a database using vsql on any node in the cluster.

If you are not on the Administration Host, or if you are not logged into the Database Administrator account, you can use Connect to Database but **do not perform any other administrative tasks** such as configuration.

1. Log in as any user that does not have root privileges. (Vertica does not allow users with root privileges to connect to a database for security reasons).
2. Run the Administration Tools.

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

3. On the **Main Menu**, select **Connect to Database**.



4. **Supply the database password** if asked:

   ```
   Password:
   ```

   The Administration Tools maintain a session context with regard to password authentication. In other words, you are required to enter a password for a database once per Administration Tools session. If you enter it correctly, you will not be asked to enter it again until the next time you run the Administration Tools. If you enter an incorrect password you must exit the Administration Tools and run it again.

5. The Administration Tools connect to the database and transfer control to vsql.

   ```
   Welcome to the vsql, Vertica_Database v2.0.5-0 interactive terminal.
   Type:  \h for help with SQL commands
          \? for help with vsql commands
          \g or terminate with semicolon to execute query
          \q to quit
   Stock_Schema=>
   ```

Connecting From the Command Line

You can use vsql from the command line to connect to a database from any Linux machine, including ones that are not part of the cluster. Simply copy `/opt/vertica/bin/vsql` to your machine.

Syntax

```
/opt/vertica/bin/vsql [ option...] [ dbname [ username ] ]
```

Semantics

option	one or more of the vsql ***command line options*** (on page 25)
dbname	the name of the target database
username	the name of the user to connect as

Notes

- If the database is password protected, you must specify the ***-W or --password command line option*** (see "-W --password" on page 29).
- The default dbname and username is your Linux user name.
- You can set the ***environment variables*** (page 43) PGDATABASE, PGHOST, PGPORT and/or PGUSER to appropriate values.
- If the connection cannot be made for any reason (e.g., insufficient privileges, server is not running on the targeted host, etc.), vsql returns an error and terminates.
- vsql returns 0 to the shell if it finished normally, 1 if a fatal error of its own (out of memory, file not found) occurs, 2 if the connection to the server went bad and the session was not interactive, and 3 if an error occurred in a script and the variable ON_ERROR_STOP was set.
- Unrecognized words in the command line may be interpreted as database or user names.

Example

This example redirects vsql output and error messages into an output file:

```
vsql --echo-all < retail_queries.sql > retail_queries.out 2>&1
```

Command Line Options

-a --echo-all (on page 26)

-A --no-align (on page 26)

-c command --command command (on page 26)

-d dbname --dbname dbname (on page 27)

-e --echo-queries (on page 27)

-f filename --file filename (on page 27)

-F separator --field-separator separator (on page 27)

-h hostname --host hostname (on page 27)

-H --html (on page 27)

-o filename --output filename (on page 28)

-p port --port port (on page 28)

-P assignment --pset assignment (on page 28)

-q --quiet (on page 28)

-R separator --record-separator separator (on page 28)

-s --single-step (on page 28)

-S --single-line (on page 28)

-t --tuples-only (on page 28)

-T table_options --table-attr table_options (on page 28)

-U username --username username (on page 29)

-v assignment --set assignment --variable assignment (on page 29)

-V --version (on page 29)

-w password (on page 29)

-W --password (on page 29)

-x --expanded (on page 29)

-X, --no-vsqlrc (on page 29)

-? --help (on page 29)

-a --echo-all

Prints all input lines to standard output as they are read. This is more useful for script processing than interactive mode. It is equivalent to setting the variable `ECHO` (page 39) to `all`.

-A --no-align

Switches to unaligned output mode. (The default output mode is aligned.)

-c command --command command

Executes one command and exits. This is useful in shell scripts. The command must be either a command string that is completely parsable by the server (i.e., it contains no vsql specific features), or a single meta-command. In other words, you cannot mix SQL and vsql meta-commands. To achieve that, you can pipe the string into vsql like this: echo "\ x \\ select * from foo;" | vsql.

-d dbname --dbname dbname

Specifies the name of the database to connect to. This is equivalent to specifying *dbname* as the first non-option argument on the command line.

-e --echo-queries

Copies all SQL commands sent to the server to standard output as well. This is equivalent to setting the variable `ECHO` (page 39) to `queries`.

-E

Displays queries generated by internal commands.

-f filename --file filename

Uses the file *filename* as the source of commands instead of reading commands interactively. After the file is processed, vsql terminates. This is in many ways equivalent to the internal command `\i` (see "\i filename" on page 34).

If *filename* is `-` (hyphen), the standard input is read.

Using this option is subtly different from writing `vsql <` *filename*. In general, both will do what you expect, but using `-f` enables some nice features such as error messages with line numbers. There is also a slight chance that using this option will reduce the start-up overhead. On the other hand, the variant using the shell's input redirection is (in theory) guaranteed to yield exactly the same output that you would have gotten had you entered everything by hand.

-F separator --field-separator separator

Specifies the field separator. This is equivalent to `\pset (page 34) fieldsep` or `\f` (see "\f [string]" on page 33).

-h hostname --host hostname

Specifies the host name of the machine on which the server is running. If the value begins with a slash, it is used as the directory for the socket.

-H --html

Turn on HTML tabular output. This is equivalent to `\pset (page 34) format html` or the `\H` (on page 34) command.

-l --list

List all available databases, then exit. Other non-connection options are ignored. This is similar to the internal command `\list`.

-n

Disables command line editing.

-o filename --output filename

Writes all query output into file *filename*. This is equivalent to the command `\o` (page 34).

-p port --port port

Specifies the TCP port or the local socket file extension on which the server is listening for connections. Defaults to port 5433.

-P assignment --pset assignment

Allows you to specify printing options in the style of `\pset (page 34)` on the command line. Note that you have to separate name and value with an equal sign instead of a space. Thus to set the output format to LaTeX, you could write `-P format=latex`.

-q --quiet

Specifies that vsql should do its work quietly. By default, it prints welcome messages and various informational output. If this option is used, none of this appears. This is useful with the `-c` (page 26) option. Within vsql you can also set the `QUIET` (page 40) variable to achieve the same effect.

-R separator --record-separator separator

Use *separator* as the record separator. This is equivalent to the `\pset (page 34) recordsep` command.

-s --single-step

Run in single-step mode for debugging scripts. Forces vsql to prompt before each statement is sent to the database and allows you to cancel execution.

-S --single-line

Runs in single-line mode where a newline terminates an SQL command, as a semicolon does.

Note: This mode is provided for those who insist on it, but you are not necessarily encouraged to use it. In particular, if you mix SQL and meta-commands on a line the order of execution might not always be clear to the inexperienced user.

-t --tuples-only

Disables printing of column names and result row count footers, etc. This is equivalent to the `\t` (on page 37) command.

-T table_options --table-attr table_options

Allows you to specify options to be placed within the HTML `table` tag. See `\pset (page 34)` for details.

-U username --username username

Connects to the database as the user *username* instead of the default.

-v assignment --set assignment --variable assignment

Perform a variable assignment, like the `\set` (see "\set [name [value [...]]]" on page 37) internal command. You must separate name and value, if any, by an equal sign on the command line.

To unset a variable, omit the equal sign. To set a variable without a value, use the equal sign but omit the value. These assignments are done during a very early stage of start-up, so variables reserved for internal purposes can get overwritten later.

-V --version

Print the vsql version and exit.

-w password

Specifies the password for a database user.

Using this command line option displays the database password in plain text on the screen. Use it with care particularly if you are connecting as the Database Administrator.

-W --password

Forces vsql to prompt for a password before connecting to a database. The password is not displayed on the screen. This option remains set for the entire session, even if you change the database connection with the meta-command `\connect` (see "\c (or \connect) [dbname [username]]" on page 31).

-x --expanded

Enables extended table formatting mode. This is equivalent to the command `\ x` (on page 38).

-X, --no-vsqlrc

Prevents the start-up file from being read (the system-wide `vsqlrc` file or the user's `~/.vsqlrc` file).

-? --help

Displays help about vsql command line arguments and exits.

Connecting From a Non-Cluster Host

You can use the Vertica vsql executable image on a non-cluster host to connect to a Vertica database.

- If the non-cluster host is running the same operating system as the cluster, simply copy the image file to the remote system. For example:

  ```
  $ scp host01:/opt/vertica/bin/vsql .
  $ ./vsql
  ```

- If the non-cluster host is running a different operating system, you must log in as root and install the Vertica rpm package on that host. For example:

  ```
  # rpm -Uvh filename
  ```

  where *filename* is one of the following packages. **Replace the build number** (for example 5-0) with the actual build number of the release you are installing.

`vertica_2.0.5-0.i386.RHEL4.rpm`
`vertica_2.0.5-0.x86_64.RHEL4.rpm`
`vertica_2.0.5-0.i386.RHEL5.rpm`
`vertica_2.0.5-0.x86_64.RHEL5.rpm`
`vertica_2.0.5-0.i386.SUSE10.rpm`
`vertica_2.0.5-0.x86_64.SUSE10.rpm`
`vertica_2.0.5-0.i386-FC6_32.rpm`
`vertica_2.0.5-0.x86-FC6_64.rpm`
	Notes:
	• Fedora 4 Core packages are not supported.
	• 64-bit packages are required for production databases.
	• 32-bit packages are provided for testing and evaluation purposes only.

Notes

- Use the same ***Command Line Options*** (on page 25) that you would on a cluster host.
- Do you use more than two connections from the same host when bulk loading data.
- You cannot run vsql on a Cygwin bash shell (Windows). Use ssh to connect to a cluster host, then run vsql.

Meta-Commands

Anything you enter in vsql that begins with an unquoted backslash is a vsql meta-command that is processed by vsql itself. These commands help make vsql more useful for administration or scripting. Meta-commands are more commonly called slash or backslash commands.

The format of a vsql command is the backslash, followed immediately by a command verb, then any arguments. The arguments are separated from the command verb and each other by any number of whitespace characters.

To include whitespace into an argument you may quote it with a single quote. To include a single quote into such an argument, precede it by a backslash. Anything contained in single quotes is furthermore subject to C-like substitutions for `\n` (new line), `\t` (tab), `\`*digits*, `\0`*digits*, and `\0x`*digits* (the character with the given decimal, octal, or hexadecimal code).

If an unquoted argument begins with a colon (`:`), it is taken as a vsql variable and the value of the variable is used as the argument instead.

Arguments that are enclosed in backquotes (`` ` ``) are taken as a command line that is passed to the shell. The output of the command (with any trailing newline removed) is taken as the argument value. The above escape sequences also apply in backquotes.

Some commands take an SQL identifier (such as a table name) as argument. These arguments follow the syntax rules of SQL: Unquoted letters are forced to lowercase, while double quotes (`"`) protect letters from case conversion and allow incorporation of whitespace into the identifier. Within double quotes, paired double quotes reduce to a single double quote in the resulting name. For example, `FOO"BAR"BAZ` is interpreted as `fooBARbaz`, and `"A weird"" name"` becomes `A weird" name`.

Parsing for arguments stops when another unquoted backslash occurs. This is taken as the beginning of a new meta-command. The special sequence `\\` (two backslashes) marks the end of arguments and continues parsing SQL commands, if any. That way SQL and vsql commands can be freely mixed on a line. But in any case, the arguments of a meta-command cannot continue beyond the end of the line.

\! [command]

Escapes to a separate Linux shell (passing arguments as entered) or executes the Linux command *command*.

\?

Shows help information about the meta-commands.

\a

Toggles output format alignment. This command is kept for backwards compatibility. See `\pset (page 34)` for a more general solution. It is similar to the command line option ***-A --no-align*** (page 26), which only disables alignment.

\c (or \connect) [dbname [username]]

Establishes a connection to a new database and/or under a user name. The previous connection is closed. If dbname is - the current database name is assumed.

If username is omitted the current user name is assumed.

As a special rule, \connect without any arguments connects to the default database as the default user (as you would have gotten by starting vsql without any arguments).

If the connection attempt fails (wrong user name, access denied, etc.), the previous connection is kept if and only if vsql is in interactive mode. When executing a non-interactive script, processing immediately stops with an error. This distinction that avoids typos and a prevent scripts from accidentally acting on the wrong database.

\C [title]

Sets the title of any tables being printed as the result of a query or unsets any such title. This command is equivalent to `\pset (page 34) title` *title*. (The name of this command derives from "caption", as it was previously only used to set the caption in an HTML table.)

\cd [directory]

Changes the current working directory to *directory*. Without argument, changes to the current user's home directory.

Tip: To print your current working directory, use ***\!*** (see "\! [command]" on page 31)pwd.

\copyright

Shows the copyright and distribution terms of Vertica.

\d \dj \dn \dp \dt \du

\d

Lists all tables showing the schema, table name, columns, types, and sizes.

\dj

For all projections, show the schema, projection name, owner, and site (node).

\dn

Lists all schemas.

\dp

For all table access privileges, show the grantee, grantor, privilege, name, and type. Same as ***\l*** (page 34).

\dt

Lists all tables showing the schema, table name, and owner.

\du

Lists all database users and atributes.

\e \edit [filename]

Edits the specified file. When the editor exits, its content is copied back to the query buffer. If no argument is given, the current query buffer is copied to a temporary file which is then edited in the same fashion.

The new query buffer is then re-parsed according to the normal rules of vsql, where the whole buffer up to the first semicolon is treated as a single line. (Thus you cannot make scripts this way. Use `\i` (see "\i filename" on page 34) for that.) If there is no semicolon, vsql waits for one to be entered (it does not execute the query buffer).

Tip: vsql searches the environment variables VSQL_EDITOR, EDITOR, and VISUAL (in that order) for an editor to use. If all of them are unset, vi is used on Linux systems, notepad.exe on Windows systems.

\echo text [...]

Prints the arguments to the standard output, separated by one space and followed by a newline. This can be useful to intersperse information in the output of scripts. For example:

```
=> \echo `date` Tue Oct 26 21:40:57 CEST 1999
```

If the first argument is an unquoted `-n` the trailing newline is not written.

Tip: If you use the ***\o*** (page 34) command to redirect your query output you may wish to use ***\qecho*** (page 36) instead of this command.

\encoding [encoding]

Specifies the client character set encoding. With no argument, this command shows the current encoding.

\f [string]

Sets the field separator for unaligned query output. The default is the vertical bar (`|`). See also `\pset (page 34)` for a generic way of setting output options.

\g [{ filename | |command }]

Sends the current query input buffer to the server and optionally stores the query's output in *filename* or pipes the output into a separate Linux shell executing *command*. A bare `\g` is virtually equivalent to a semicolon. A `\g` with argument is a "one-shot" alternative to the `\o` (page 34) command.

\H

Toggles HTML query output format. This command is for compatibility and convenience, but see `\pset (page 34)` about setting other output options.

\h \help [command]

Gives syntax help on the specified SQL command. If *command* is not specified, vsql lists all the commands for which syntax help is available. If *command* is an asterisk (`*`), syntax help on all SQL commands is shown.

Note: To simplify typing, commands that consists of several words do not have to be quoted. For example:
`\help alter table`.

\i filename

Reads input from the file *filename* and executes it as though it had been typed on the keyboard.

Note: To see the lines on the screen as they are read, set the variable ***ECHO*** (page 39) to all.

\l

For all table access privileges, show the grantee, grantor, privilege, name, and type. Same as \dp.

\o filename   \o | command

Saves query results to the file *filename* or pipes query results into a separate shell to execute *command*. If no arguments are specified, the query output is reset to the standard output.

Query results includes all tables, command responses, and notices obtained from the database server.

Tip: To intersperse text output with query results, use ***\qecho*** (page 36).

\p

Prints the current query buffer to the standard output.

\pset parameter [value]

This command sets options affecting the output of query result tables. *parameter* describes which option is to be set. The semantics of *value* depend thereon.

Adjustable printing options are:

format

Sets the output format to one of `unaligned`, `aligned`, `html`, or `latex`. Unique abbreviations are allowed. (That would mean one letter is enough.)

"Unaligned" writes all columns of a row on a line, separated by the currently active field separator. This is intended to create output that might be intended to be read in by other programs (tab- separated, comma-separated). "Aligned" mode is the standard, human-readable, nicely formatted text output that is default. The "HTML" and "LaTeX" modes put out tables that are intended to be included in documents using the respective mark-up language. They are not complete documents! (This might not be so dramatic in HTML, but in LaTeX you must have a complete document wrapper.)

border

The second argument must be a number. In general, the higher the number the more borders and lines the tables will have, but this depends on the particular format. In HTML mode, this will translate directly into the `border=...` attribute, in the others only values 0 (no border), 1 (internal dividing lines), and 2 (table frame) make sense.

expanded (or x)

Toggles between regular and expanded format. When expanded format is enabled, all output has two columns with the column name on the left and the data on the right. This mode is useful if the data wouldn't fit on the screen in the normal "horizontal" mode.

Expanded mode is supported by all four output formats.

null

The second argument is a string that should be printed whenever a column is null. The default is not to print anything, which can easily be mistaken for, say, an empty string. Thus, one might choose to write `\pset null '(null)'`.

fieldsep

Specifies the field separator to be used in unaligned output mode. That way one can create, for example, tab- or comma-separated output, which other programs might prefer. To set a tab as field separator, type `\pset fieldsep '\t'`. The default field separator is `'|'` (a vertical bar).

footer

Toggles the display of the default footer `(x rows)`.

recordsep

Specifies the record (line) separator to use in unaligned output mode. The default is a newline character.

tuples_only (or t)

Toggles between tuples only and full display. Full display may show extra information such as column headers, titles, and various footers. In tuples only mode, only actual table data is shown.

title [text]

Sets the table title for any subsequently printed tables. This can be used to give your output descriptive tags. If no argument is given, the title is unset.

tableattr (or T) [text]

Allows you to specify any attributes to be placed inside the HTML `table` tag. This could for example be `cellpadding` or `bgcolor`. Note that you probably don't want to specify `border` here, as that is already taken care of by `\pset border`.

pager

Controls use of a pager for query and vsql help output. If the environment variable `PAGER` is set, the output is piped to the specified program. Otherwise a platform-dependent default (such as `more`) is used.

When the pager is off, the pager is not used. When the pager is on, the pager is used only when appropriate, i.e. the output is to a terminal and will not fit on the screen. (vsql does not do a perfect job of estimating when to use the pager.) `\pset pager` turns the pager on and off. Pager can also be set to `always`, which causes the pager to be always used.

Illustrations on how these different formats look can be seen in the ***Examples*** (page 44) section.

Tip: There are various shortcut commands for \pset. See ***a*** (on page 31), ***C*** (see "\C [title]" on page 32), ***H*** (on page 34), ***t*** (on page 37), ***T*** (see "\T table_options" on page 37), and **\ *x*** (on page 38).

Note: It is an error to call \pset without arguments. In the future this call might show the current status of all printing options.

\q

Quits the vsql program.

\qecho text [...]

This command is identical to `\echo` (see "\echo text [...]" on page 33) except that the output is written to the query output channel, as set by `\o` (page 34).

\r

Resets (clears) the query buffer.

\s [filename]

Prints or saves the command line history to *filename*. or if not specified writes the history to the standard output. This option is only available if vsql is configured to use the GNU Readline library.

\set [name [value [...]]]

Sets the internal variable *name* to *value* or, if more than one value is given, to the concatenation of all of them. If no second argument is given, the variable is set with no value. To unset a variable, use the `\ unset` (page 37) command.

Valid variable names can contain characters, digits, and underscores. See the section Variables below for details. Variable names are case-sensitive.

Note: vsql treats several variables as special. They are documented in Variables.

\t

Toggles the display of output column name headings and row count footer. This command is equivalent to `\pset (page 34) tuples_only` and is provided for convenience.

\T table_options

Specifies attributes to be placed within the `table` tag in HTML tabular output mode. This command is equivalent to `\pset (page 34) tableattr` *table_options*.

\timing

Toggles a display of how long each SQL statement takes, in milliseconds.

\ unset [name]

Removes the value of the internal variable *name* that was set using ***set*** (see "\set [name [value [...]]]" on page 37).

\w { filename | |command }

Outputs the current query buffer to the file *filename* or pipes it to the Linux command *command*.

\ x

Toggles extended table formatting mode. As such it is equivalent to `\pset (page 34) expanded`.

There is no space between the backslash and the x.

\z

For all table access privileges, show the grantee, grantor, privilege, name, and type. Same as \dp.

Variables

vsql provides variable substitution features similar to common Linux command shells. Variables are simply name/value pairs, where the value can be any string of any length. To set variables, use the vsql meta-command `\set` (see "\set [name [value [...]]]" on page 37):

```
testdb=> \set foo bar
```

sets the variable `foo` to the value `bar`. To retrieve the content of the variable, precede the name with a colon and use it as the argument of any slash command:

```
testdb=> \echo :foo
bar
```

Note: The arguments of \set are subject to the same substitution rules as with other commands. For example, \set bar :foo is a valid way to copy a variable.

If you call `\set` without a second argument, the variable is set, with an empty string as value. To unset (or delete) a variable, use the command `\unset` (page 37).

vsql's internal variable names can consist of letters, numbers, and underscores in any order and any number. Some of these variables are treated specially by vsql. They indicate certain option settings that can be changed at run time by altering the value of the variable or represent some state of the application. Although you can use these variables for any other purpose, this is not recommended. By convention, all specially treated variables consist of all upper-case letters (and possibly numbers and underscores). To ensure maximum compatibility in the future, avoid using such variable names for your own purposes.

SQL Interpolation

An additional useful feature of vsql variables is that you can substitute ("interpolate") them into regular SQL statements. The syntax for this is again to prepend the variable name with a colon (`:`).

```
testdb=> \set foo 'my_table'
testdb=> SELECT * FROM :foo;
```

would then query the table `my_table`. The value of the variable is copied literally, so it can even contain unbalanced quotes or backslash commands. Make sure that it makes sense where you put it. Variable interpolation is not performed into quoted SQL entities.

DBNAME

The name of the database you are currently connected to. This is set every time you connect to a database (including program start-up), but can be unset.

ECHO

If set to `all`, all lines entered from the keyboard or from a script are written to the standard output before they are parsed or executed. To select this behavior on program start-up, use the switch `-a` (page 26). If set to `queries`, vsql merely prints all queries as they are sent to the server. The switch for this is `-e` (page 27).

ECHO_HIDDEN

When this variable is set and a backslash command queries the database, the query is first shown. This way you can study the Vertica internals and provide similar functionality in your own programs. (To select this behavior on program start-up, use the switch `-E` (page 27).) If you set the variable to the value `noexec`, the queries are just shown but are not actually sent to the server and executed.

ENCODING

The current client character set encoding.

HISTCONTROL

If this variable is set to `ignorespace`, lines which begin with a space are not entered into the history list. If set to a value of `ignoredups`, lines matching the previous history line are not entered. A value of `ignoreboth` combines the two options. If unset, or if set to any other value than those above, all lines read in interactive mode are saved on the history list.

Note: This feature was shamelessly plagiarized from Bash.

HISTSIZE

The number of commands to store in the command history. The default value is 500.

Note: This feature was shamelessly plagiarized from Bash.

HOST

The database server host you are currently connected to. This is set every time you connect to a database (including program start-up), but can be unset.

IGNOREEOF

If unset, sending an EOF character (usually **Control**+**D**) to an interactive session of vsql will terminate the application. If set to a numeric value, that many EOF characters are ignored before the application terminates. If the variable is set but has no numeric value, the default is 10.

Note: This feature was shamelessly plagiarized from Bash.

ON_ERROR_STOP

By default, if non-interactive scripts encounter an error, such as a malformed SQL command or internal meta-command, processing continues. This has been the traditional behavior of vsql but it is sometimes not desirable. If this variable is set, script processing will immediately terminate. If the script was called from another script it will terminate in the same fashion. If the outermost script was not called from an interactive vsql session but rather using the `-f` (page 27) option, vsql will return error code 3, to distinguish this case from fatal error conditions (error code 1).

PORT

The database server port to which you are currently connected. This is set every time you connect to a database (including program start-up), but can be unset.

PROMPT1 PROMPT2 PROMPT3

These specify what the prompts vsql issues should look like. See ***Prompting*** (page 41) below.

QUIET

This variable is equivalent to the command line option `-q` (see "\q" on page 36). It is probably not too useful in interactive mode.

SINGLELINE

This variable is equivalent to the command line option `-S` (page 28).

SINGLESTEP

This variable is equivalent to the command line option `-s` (page 28).

USER

The database user you are currently connected as. This is set every time you connect to a database (including program start-up), but can be unset.

VERBOSITY

This variable can be set to the values `default`, `verbose`, or `terse` to control the verbosity of error reports.

SQL Interpolation

Prompting

The prompts vsql issues can be customized to your preference. The three variables `PROMPT1`, `PROMPT2`, and `PROMPT3` contain strings and special escape sequences that describe the appearance of the prompt. Prompt 1 is the normal prompt that is issued when vsql requests a new command. Prompt 2 is issued when more input is expected during command input because the command was not terminated with a semicolon or a quote was not closed. Prompt 3 is issued when you run an SQL `COPY` command and you are expected to type in the row values on the terminal.

The value of the selected prompt variable is printed literally, except where a percent sign (%) is encountered. Depending on the next character, certain other text is substituted instead. Defined substitutions are:

%M

The full host name (with domain name) of the database server, or `[local]` if the connection is over a socket, or `[local:`*/dir/name*`]`, if the socket is not at the compiled in default location.

%m

The host name of the database server, truncated at the first dot, or `[local]`.

%>

The port number at which the database server is listening.

%n

The database session user name. (The expansion of this value might change during a database session as the result of the command `SET SESSION AUTHORIZATION`.)

%/

The name of the current database.

%~

Like `%/`, but the output is ~ (tilde) if the database is your default database.

%#

If the session user is a database superuser, then a `#`, otherwise a `>`. (The expansion of this value might change during a database session as the result of the command `SET SESSION AUTHORIZATION`.)

%R

In prompt 1 normally `=`, but `^` if in single-line mode, and `!` if the session is disconnected from the database (which can happen if `\connect` fails). In prompt 2 the sequence is replaced by `-`, `*`, a single quote, a double quote, or a dollar sign, depending on whether vsql expects more input because the command wasn't terminated yet, because you are inside a `/* ... */` comment, or because you are inside a quoted or dollar-escaped string. In prompt 3 the sequence doesn't produce anything.

%x

Transaction status: an empty string when not in a transaction block, or `*` when in a transaction block, or `!` when in a failed transaction block, or `?` when the transaction state is indeterminate (for example, because there is no connection).

%digits

The character with the indicated numeric code is substituted. If *digits* starts with `0x` the rest of the characters are interpreted as hexadecimal; otherwise if the first digit is `0` the digits are interpreted as octal; otherwise the digits are read as a decimal number.

%:name:

The value of the vsql variable *name*. See the section Variables for details.

%`command`

The output of *command*, similar to ordinary "back- tick" substitution.

%[... %]

Prompts may contain terminal control characters which, for example, change the color, background, or style of the prompt text, or change the title of the terminal window. In order for the line editing features of Readline to work properly, these non-printing control characters must be designated as invisible by surrounding them with `%[` and `%]`. Multiple pairs of these may occur within the prompt. For example,

```
testdb=> \set PROMPT1 '%[%033[1;33;40m%]%n@%/%R%[%033[0m%#%] '
```

results in a boldfaced (`1;`) yellow-on-black (`33;40`) prompt on VT100-compatible, color-capable terminals.

To insert a percent sign into your prompt, write %%. The default prompts are `'%/%R%# '` for prompts 1 and 2, and `'>> '` for prompt 3.

Note: This feature was shamelessly plagiarized from tcsh.

Command-Line Editing

vsql supports the Readline library for convenient line editing and retrieval. The command history is automatically saved when vsql exits and is reloaded when vsql starts up. Tab-completion is also supported, although the completion logic makes no claim to be an SQL parser. If for some reason you do not like the tab completion, you can turn it off by putting this in a file named `.inputrc` in your home directory:

```
$if vsql
set disable-completion on
$endif
```

(This is not a vsql but a Readline feature. Read its documentation for further details.)

Environment

PAGER

If the query results do not fit on the screen, they are piped through this command. Typical values are `more` or `less`. The default is platform-dependent. The use of the pager can be disabled by using the `\pset` command.

PGDATABASE

Default connection database

PGHOST
PGPORT
PGUSER

Default connection parameters

VSQL_EDITOR
EDITOR
VISUAL

Editor used by the `\e` command. The variables are examined in the order listed; the first that is set is used.

SHELL

Command executed by the `\!` command.

TMPDIR

Directory for storing temporary files. The default is `/tmp`.

Files

- Before starting up, vsql attempts to read and execute commands from the system-wide vsqlrc file and the user's ~/.vsqlrc file. (On Windows, the user's startup file is named %APPDATA%\postgresql\vsqlrc.conf.)
- The command-line history is stored in the file ~/.vsql_history, or %APPDATA%\postgresql\vsql_history on Windows.

Notes for Windows users

vsql is built as a "console application". Since the Windows console windows use a different encoding than the rest of the system, you must take special care when using 8-bit characters within vsql. If vsql detects a problematic console code page, it will warn you at startup. To change the console code page, two things are necessary:

- Set the code page by entering cmd.exe /c chcp 1252. (1252 is a code page that is appropriate for German; replace it with your value.) If you are using Cygwin, you can put this command in /etc/profile.
- Set the console font to "Lucida Console", because the raster font does not work with the ANSI code page.

Output Formatting Examples

The first example shows how to spread a command over several lines of input. Notice the changing prompt:

```
testdb=> CREATE TABLE my_table (
testdb(> first integer not null default 0,
testdb(> second text) testdb-> ;
CREATE TABLE
```

Assume you have filled the table with data and want to take a look at it:

```
testdb=> SELECT * FROM my_table;
 first | second
-------+--------
     1 | one
     2 | two
     3 | three
     4 | four
(4 rows)
```

You can display tables in different ways by using the `\pset` command:

```
testdb=> \pset border 2
Border style is 2.
testdb=> SELECT * FROM my_table;
+-------+--------+
| first | second |
+-------+--------+
|     1 | one    |
|     2 | two    |
|     3 | three  |
|     4 | four   |
+-------+--------+
(4 rows)

testdb=> \pset border 0
Border style is 0.
testdb=> SELECT * FROM my_table;
first second
----- ------
    1 one
    2 two
    3 three
    4 four
(4 rows)

testdb=>
\pset border 1
Border style is 1.
testdb=> \pset format unaligned
Output format is unaligned.
testdb=> \pset fieldsep ","
Field separator is ",".
testdb=> \pset tuples_only
Showing only tuples.
testdb=> SELECT second, first FROM my_table; one,1
two,2
three,3
four,4
```

Alternatively, use the short commands:

```
testdb=> \a \t \ x
Output format is aligned.
Tuples only is off.
Expanded display is on.
testdb=> SELECT * FROM my_table;
-[ RECORD 1 ]-
first | 1
second | one
-[ RECORD 2 ]-
first | 2
second | two
-[ RECORD 3 ]-
first | 3
second | three
-[ RECORD 4 ]-
first | 4
second | four
```

Connecting with JDBC

Installing the JDBC Driver

The procedure for installing the Vertica JDBC driver are described in the Installation Guide under:

- Installing the JDBC Driver

For Information About JDBC

- ***JDBC Driver JavaDoc*** (../../JDBC/index.html) (Vertica extensions)
- ***An Introduction to JDBC, Part 1*** (http://www.onjava.com/pub/a/onjava/excerpt/javaentnut_2/index1.html)
- ***PostgreSQL JDBC Driver*** (http://jdbc.postgresql.org/)

Creating a Connection

To connect to a Vertica database:

1. In your Java code, import the SQL package:
   ```
   import java.sql.*;
   ```
2. Load the Vertica driver:
   ```
   Class.forName("com.vertica.Driver")
   ```
3. Make the connection:
   ```
   Connection db = DriverManager.getConnection
   ("jdbc:vertica://<server>:5433/<instance>",<user>, "");
   ```

Parameter	Description
<server>	A Vertica node (default localhost)
5433	5433
<instance>	database instance name
<user>	user (default is user who started Vertica)
""	empty password

Notes

- When you disconnect a user session, any transactions in progress are automatically rolled back.

Executing a Query

To execute a query:

1. Get a statement object.
   `Statement st = db.createStatement();`
2. Get a result set object.
   `ResultSet rs = st.executeQuery(<query>);`
   <*query*> is any standard SQL query (SELECT command)
3. Get the result set metadata.
   `ResultSetMetaData rsmeta = rs.getMetaData();`
4. Use a standard *forward only* ResultSet cursor
   ```
   while(rs.next())
   {
       ...
   }
   ```
5. Close objects.
   ```
   rs.close();
   st.close();
   db.close();
   ```

Sample Application

This sample application assumes that it is running on the same machine as the Vertica instance and that your username is `devel`.

```
import java.sql.*;
// Create a table, create a projection, insert a row, // query the table, and
drop the table (including the projection)
class jdbc_test
{
    // Static SQL statements
    static String create_table =
        "CREATE TABLE VTEST (COLUMN_1 CHAR(50));";
    static String create_projection =
        "CREATE PROJECTION VTEST_PROJ (COLUMN_1) AS SELECT COLUMN_1 FROM
VTEST;";
    static String insert_row =
        "INSERT INTO VTEST VALUES ('Testing Vertica');";
    static String select_row =
        "SELECT * FROM VTEST;";
    static String drop_table =
        "DROP TABLE VTEST CASCADE;";

    public static void main(String args[]) throws Exception
    {
```

```
        //try to load the class
        Class.forName("com.vertica.Driver");
        //get a connection to the database
        Connection db = DriverManager.getConnection
            ("jdbc:vertica://localhost:5433/testdb", "devel", "");
        //create a statement object
        Statement st = db.createStatement();
        //execute SQL statements
        st.executeUpdate(create_table);
        st.executeUpdate(create_projection);
        st.executeQuery(insert_row);

        // print out the result set
        ResultSet rs = st.executeQuery(select_row);
        while(rs.next())
        {
            System.out.println(rs.getObject(1));
        }

        //clean up
        st.executeUpdate(drop_table);
        rs.close();
        st.close();
        db.close();

    }
}
```

Connecting with ODBC

Installing the ODBC Driver

The procedure for installing the Vertica ODBC driver is described in the Installation Guide under:

- Installing the ODBC Driver

For Information About ODBC

- ***MSDN ODBC Start Page*** http://msdn.microsoft.com/library/default.asp?url=/library/en-us/odbc/htm/dasdkodbcoverview.asp
- ***Using Data Sources (ODBC)*** http://www.microsoft.com/resources/documentation/windows/xp/all/proddocs/en-us/app_mdac.mspx?mfr=true
- ***ODBC Developer Center*** http://www.datadirect.com/developer/odbc/index.ssp
- ***PostgreSQL ODBC Driver*** http://pgfoundry.org/projects/psqlodbc/

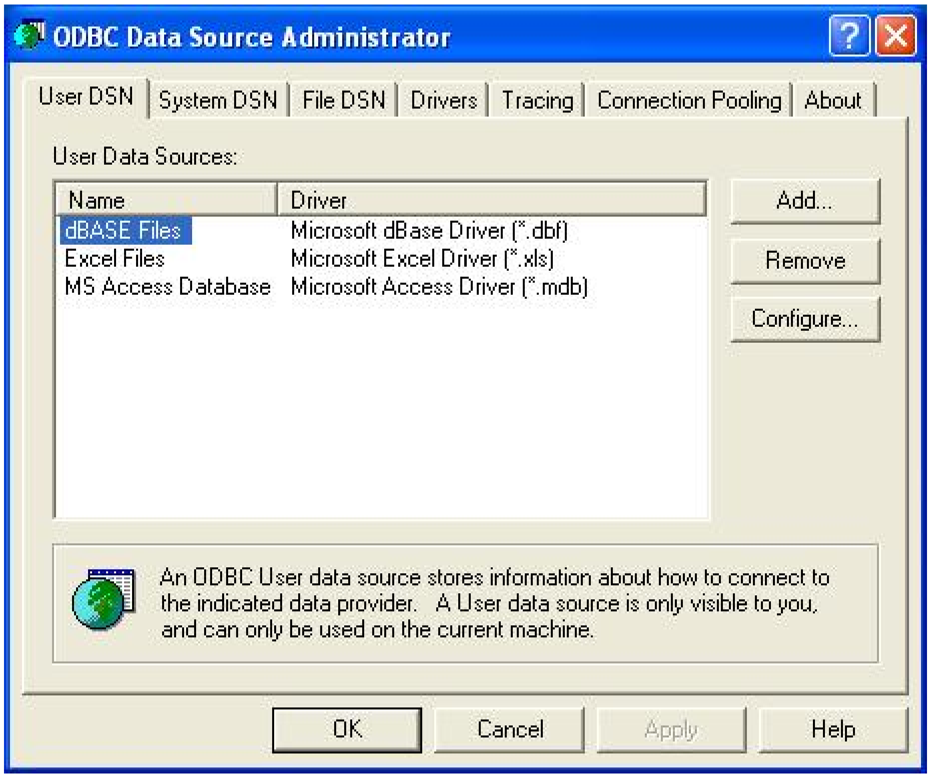
Setting Up a Data Source Name

This section describes how to set up an ODBC data source name using the Vertica ODBC Driver. It assumes that the driver has been installed as described in Installing the ODBC Driver in the Installation Guide.

1. Open the ODBC Administrator. For example, click **Start** > **Control Panel** > **Data Sources (ODBC)** or **Start** > **Control Panel** > **Administrative Tools** > **Data Sources (ODBC)** depending on which version of Windows you are using.

   Differences between Windows versions and Start Menu customizations may require a different action to open the ODBC Administrator.

2. In the **ODBC Data Source Administrator**, click the **System DSN** tab to create a system-wide data source name.

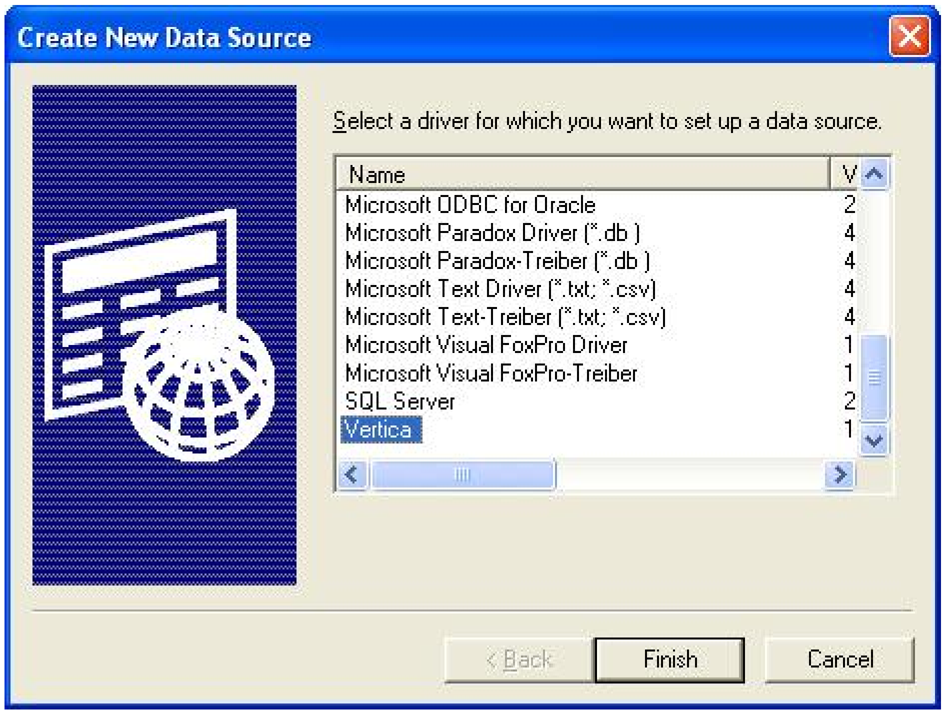


User Data Sources:

Name	Driver
dBASE Files	Microsoft dBase Driver (*.dbf)
Excel Files	Microsoft Excel Driver (*.xls)
MS Access Database	Microsoft Access Driver (*.mdb)

An ODBC User data source stores information about how to connect to the indicated data provider. A User data source is only visible to you, and can only be used on the current machine.

3. In the **Create a New Data Source** dialog, click the **Add...** button, scroll down far enough to select **Vertica**, and click **Finish**.

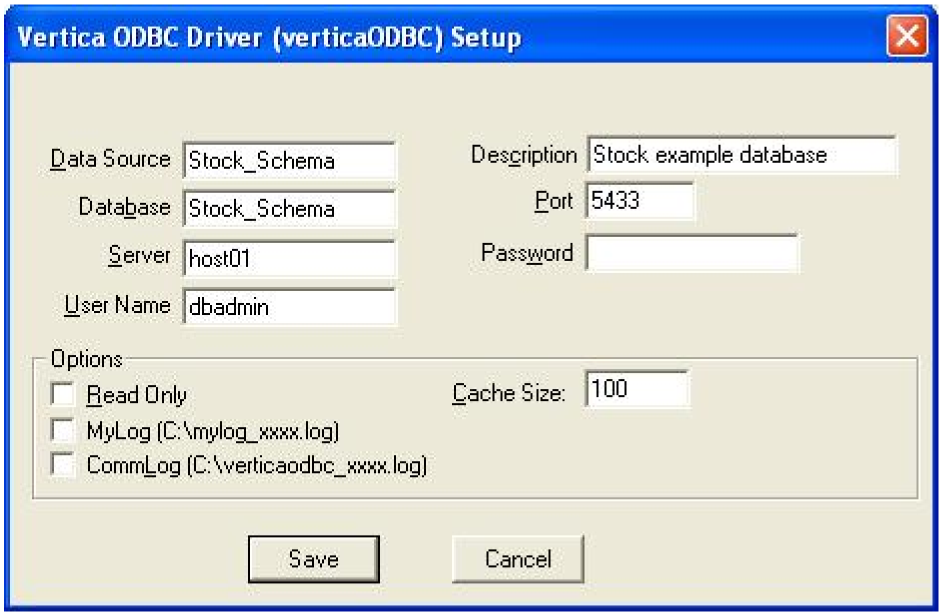


Create New Data Source

Select a driver for which you want to set up a data source.

Name	V
Microsoft ODBC for Oracle	2
Microsoft Paradox Driver (*.db)	4
Microsoft Paradox-Treiber (*.db)	4
Microsoft Text Driver (*.txt; *.csv)	4
Microsoft Text-Treiber (*.txt; *.csv)	4
Microsoft Visual FoxPro Driver	1
Microsoft Visual FoxPro-Treiber	1
SQL Server	2
Vertica	1

4. Enter your data source information in the **Vertica ODBC Driver Setup** dialog.



Vertica ODBC Driver (verticaODBC) Setup

Data Source: Stock_Schema
Database: Stock_Schema
Server: host01
User Name: dbadmin
Description: Stock example database
Port: 5433
Password:

Options
- [ ] Read Only
- [ ] MyLog (C:\mylog_xxxx.log)
- [ ] CommLog (C:\verticaodbc_xxxx.log)

Cache Size: 100

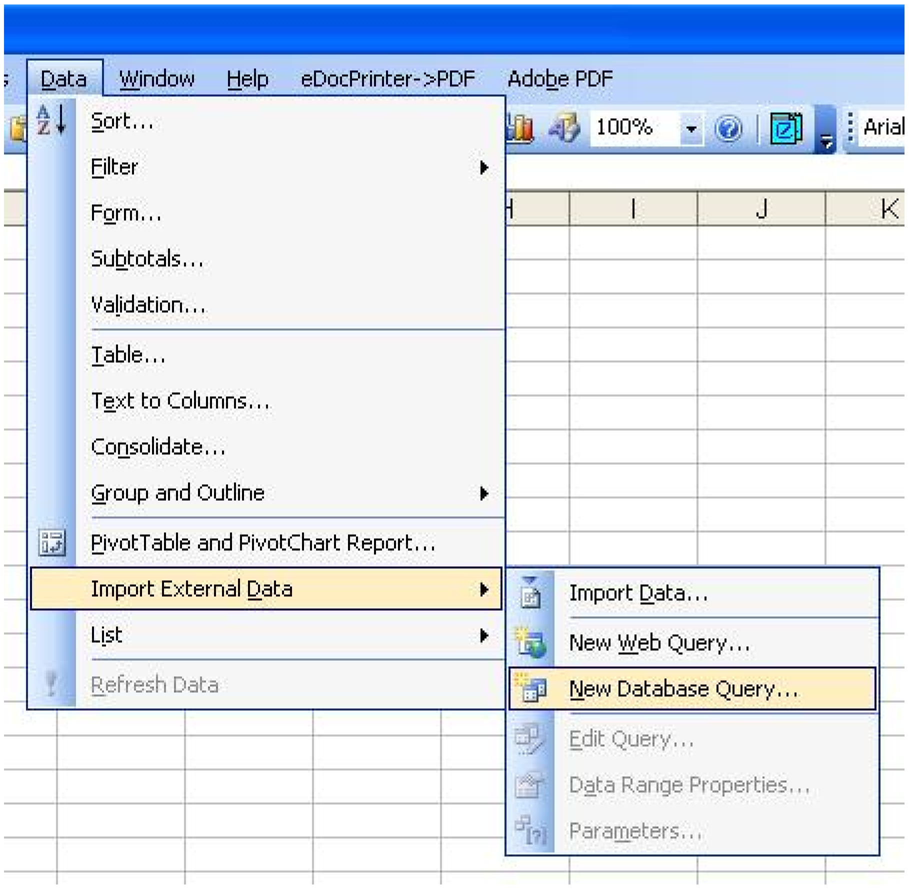
Data Source	any meaningful string. There is no special significance to the default name.
Database	the name of the database running on the Server.
Server	the hostname or the IP address of any active node within a Vertica database.

User Name	either the database superuser (same name as database administrator account) or a user that has been created by the superuser and granted privileges
Description	(optional) provides additional information about the Data Source.
Port	the port number (5433) on which Vertica listens for ODBC connections.
Password	the password for the specified user name.
Read Only	prevents users of this data source from writing to the database.
MyLog	logs only debug messages and is useful for debugging problems with the ODBC driver.
CommLog	logs all communications between the application and the Server and is useful for application debugging.
Cache Size	specifies the number of tuples for which to allocate memory at any given time. The default is 100 tuples.

Verifying a Data Source Name

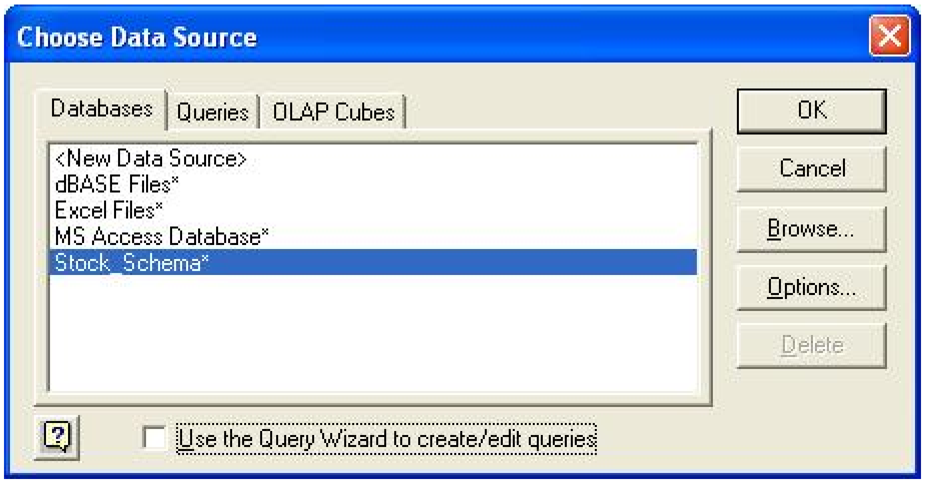
This section describes how to verify that an application can connect to an ODBC data source using **Microsoft Excel**. You can accomplish the same thing with any ODBC application.

1. Select **Data** > **Import External Data** > **New Database Query...**



If **Microsoft Query** is not installed, Excel offers to install it for you.

2. Select the **data source name** (Stock_Schema) in this example.
3. **Disable** "**Use the Query Wizard...**" and click **OK**.


Choose Data Source

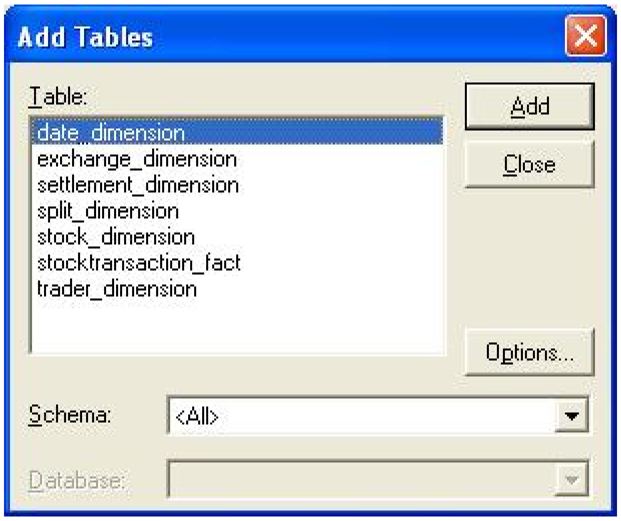
Databases | Queries | OLAP Cubes

<New Data Source>
dBASE Files*
Excel Files*
MS Access Database*
Stock_Schema*

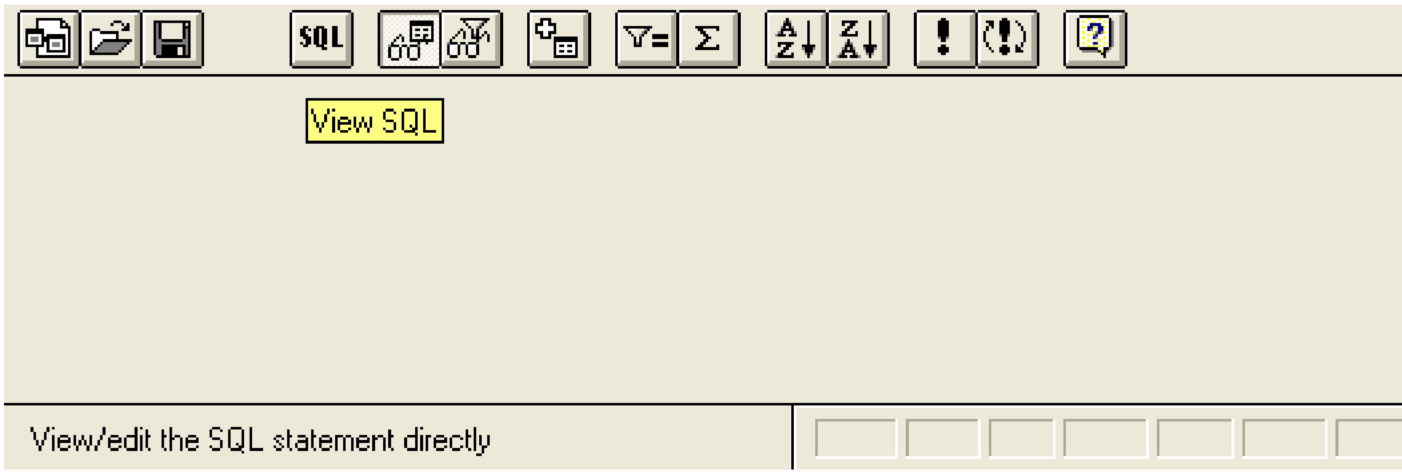
OK
Cancel
Browse...
Options...
Delete

[ ] Use the Query Wizard to create/edit queries

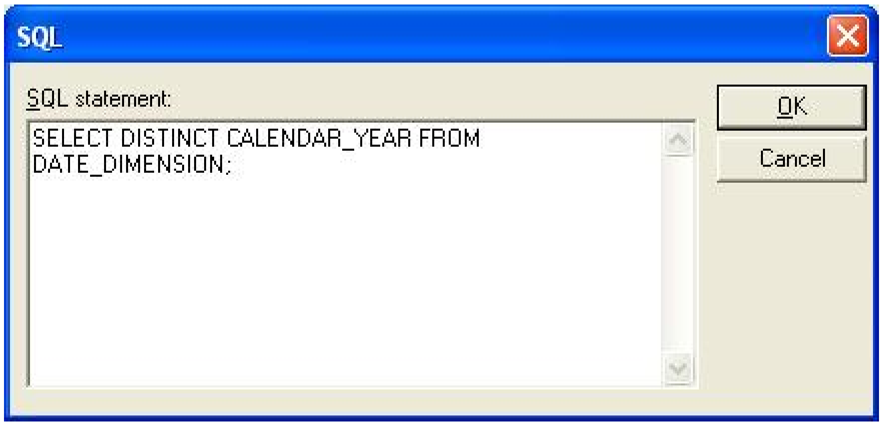
4. In the Add Tables dialog, clock **Close**.



5. Click the **SQL** button.

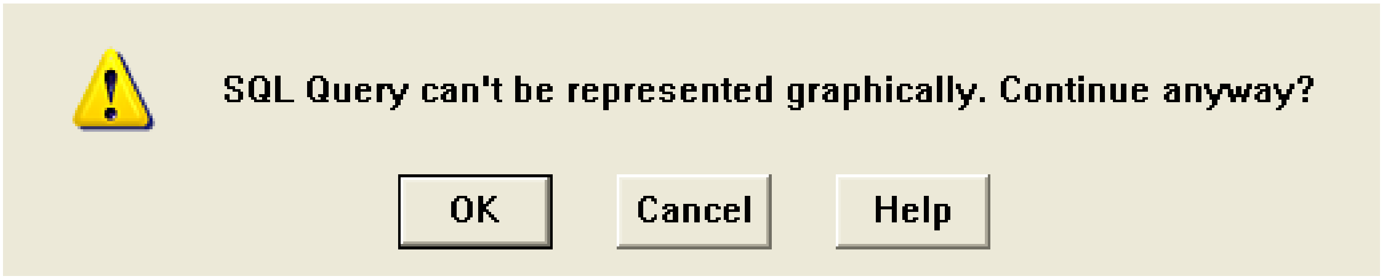


6. Enter a simple query such as **select distinct calendar_year from date_dimension** and click **OK**.



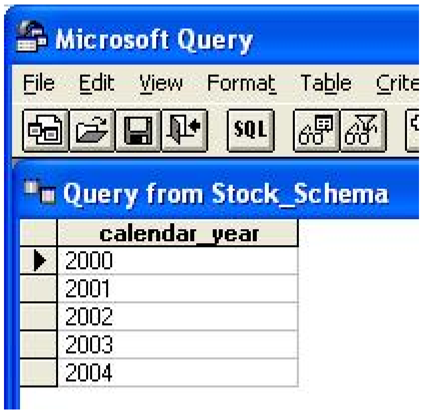
```
SELECT DISTINCT CALENDAR_YEAR FROM
DATE_DIMENSION;
```

7. Click **OK**.



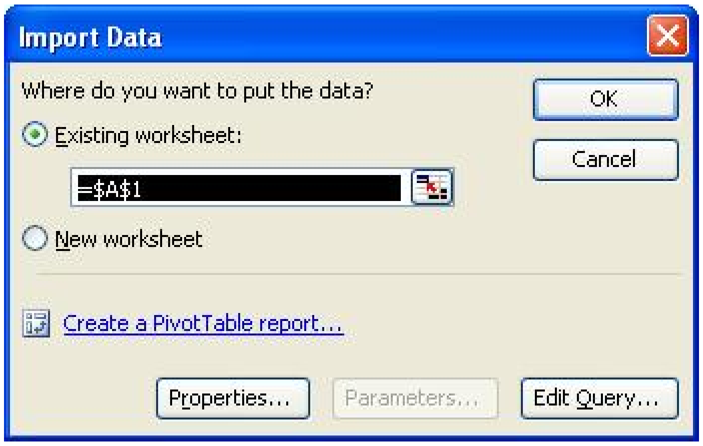
SQL Query can't be represented graphically. Continue anyway?

8. The data values **2000**, **2001**, **2002**, **2003**, **2004** indicate that you have successfully connected and executed a query through ODBC.



calendar_year
2000
2001
2002
2003
2004

9. Click **File** > **Return Data to Microsoft Office Excel**.
10. In the **Import Data** dialog, click **OK**.



Import Data

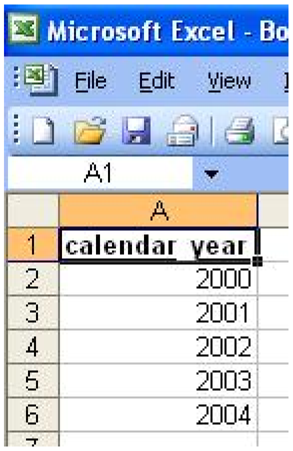
Where do you want to put the data?

(●) Existing worksheet:

=A1

( ) New worksheet

11. The data is now available to be used in an Excel worksheet.



	A
1	calendar_year
2	2000
3	2001
4	2002
5	2003
6	2004

Writing Queries

In general, Vertica is designed to execute the same SQL standard queries that run on other databases. However, there are several differences between Vertica queries and those used in other relational database management systems:

- Existing queries that were written for a **normalized database** may need modification in order to conform to the Star Schema or Snowflake Schema used in the Vertica database.
- Vertica does not support **SQL standard views** (stored queries). Existing queries that were written for standard SQL views may need to be rewritten to incorporate the query stored in the view. These queries are excellent candidates for pre-join Projections.
- The Vertica ***transaction model*** (page 63) is different from the SQL standard in a way that **has a profound effect on query performance** (see below). You can:
  - execute a query on a **static snapshot of the database** from any specific date and time. This avoids holding locks or blocking other database operations.
  - you can use a **subset of the standard SQL isolation levels and access modes** (read/write or read-only) for a user session.

IMPORTANT - QUERY PERFORMANCE

Explanation

By default, queries do not use snapshot isolation and thus hold locks on the projections associated with the tables in the SELECT list. This blocks other database operations (concurrent queries) until the transaction ends.

Workaround

Unless you need the result set of your query to include data from the current epoch, use:

```
AT EPOCH LATEST SELECT ...
```

If you need the result set of your query to include data from the current epoch, end the current transaction as soon as possible. This can be done by executing COMMIT or ROLLBACK or terminating the connection.

Explanation

Because Vertica does not support cursors, a query that produces a large result set can consume a great deal of client resources.

Workaround

- Add predicates to the query to reduce the size of the result set.

```
Use the LIMIT clause (described in the SQL Reference Manual) to reduce
the size of the result set.
```

Using Temporary Tables

You can use the CREATE TEMPORARY TABLE in the SQL Reference Manual to implement certain queries using multiple steps. In other words, you can:

1. create one or more temporary tables
2. execute queries and store the result sets in the temporary tables
3. execute the main query using the temporary tables as if they were a normal part of the logical schema

The name "temporary table" may seem like a misnomer. Only the data stored in the table is temporary. The metadata is persistent.

Notes

- Make sure to define all temporary tables used in a query before inserting data into any of them.
- You cannot query temporary tables using ***Snapshot Isolation*** (page 63). Set the isolation level (see SET SESSION CHARACTERISTICS) to READ COMMITTED. This may be necessary when running queries that use temporary tables while the database is being loaded.
- You cannot use ALTER TABLE ... ADD COLUMN on a temporary table.

Using Compound Predicate Joins

Vertica allows tables to have compound (multiple-column) primary and foreign keys. For example:

```
-- Create a pair of tables with multi-column keys

CREATE TABLE dimension
  (pk1 INTEGER NOT NULL,
   pk2 INTEGER NOT NULL);

ALTER TABLE dimension
  ADD PRIMARY KEY (pk1, pk2);
CREATE TABLE fact
  (fk1 INTEGER NOT NULL,
   fk2 INTEGER NOT NULL);

ALTER TABLE fact
  ADD FOREIGN KEY (fk1, fk2)
  REFERENCES dimension (pk1, pk2);
```

To join tables using compound keys, you must connect two join predicates with a Boolean AND operator. For example:

```
-- Query that joins the tables

SELECT *
FROM fact, dimension
WHERE fact.fk1 = dimension.pk1
      AND
      fact.fk2 = dimension.pk2;
```

Using Multiple Dimension Table References in the WHERE Clause

The same dimension table can appear in multiple predicates in the WHERE clause of a query. For example:

```
SELECT *
FROM   fact, dimension
```

```
WHERE  fact.fk = dimension.pk
       AND
       fact.name = dimension.name
```

Using Multiple Instances of Dimension Tables in the FROM Clause

The same dimension table can appear multiple times in the FROM clause of a query, using different aliases. For example:

```
SELECT *
FROM   fact, dimension d1, dimension d2
WHERE  fact.fk = d1.pk
       AND
       fact.name = d2.name
```

Using the Transaction Model

Vertica supports conventional SQL transactions with standard ACID properties. Specifically:

- Vertica supports ANSI SQL 92 style implicit transactions. You do not need to execute a BEGIN or START TRANSACTION command.
- Vertica does not use a redo/undo log or two-phase commit.
- The COPY command automatically commits itself and any current transaction. Vertica recommends that you COMMIT or ROLLBACK the current transaction before using COPY.

Session-Scoped Isolation Levels

Vertica supports a **subset of the standard SQL isolation levels and access modes** for a user session as described in ***SERIALIZABLE Isolation*** (page 64) and ***READ COMMITTED Isolation*** (page 65). These modes determine what data a transaction can access when other transactions are running concurrently. You can change the default isolation level for a user session using the SET SESSION CHARACTERISTICS command.

> Session-scoped isolation levels do not apply to ***temporary tables*** (page 60), which are scoped to the current transaction. They do not take table locks, are only visible to one user, and their contents are always visible regardless of advancing epochs and COMMITs.

Query-Scoped Isolation Levels

Snapshot Isolation (page 63) (historical queries) are part of the Vertica query syntax:

```
[ AT EPOCH LATEST ] | [ AT TIME 'timestamp' ] SELECT ...
```

AT EPOCH LATEST allows queries to access all historical data up to but not including the current epoch . It does not hold locks and does not block write operations. This provides a **profound query performance advantage**. It does not apply to temporary tables.

Automatic Rollback

When an error occurs or a user session is disconnected, the current transaction automatically rolls back. This behavior may be different from other databases.

Snapshot Isolation

Runs a SELECT query in snapshot isolation mode, which queries all data in the database up to and including the most recently closed (advanced) epoch without holding a lock or blocking write operations. This provides a substantial query performance improvement over the default ***SERIALIZABLE Isolation*** (page 64) level.

Syntax

```
AT EPOCH LATEST SELECT...
```

Semantics

To be precise, a query using snapshot isolation is actually an historical query. The syntax `AT EPOCH LATEST` is equivalent to the syntax `AT TIME '`*timestamp*`'` where *timestamp* is known to be within the most recently closed epoch.

Notes

- Snapshot isolation does not apply to temporary tables.
- The only disadvantage of using snapshot isolation mode is that queries cannot see data that has been commited within the current epoch (the epoch that has not yet been closed). Because the automatic tuple mover advances the epoch every three minutes by default, the query result set may be missing up to three minutes worth of newly loaded tuples.

SERIALIZABLE Isolation

The default transaction isolation level is SERIALIZABLE, which is the strictest level of SQL transaction isolation. This level emulates transactions executed one after another, serially, rather than concurrently. It holds locks, and blocks write operations. For normal query operations, you can gain a significant performance improvement by using ***Snapshot Isolation*** (page 63).

Syntax

```
SET SESSION CHARACTERISTICS AS TRANSACTION
    ISOLATION LEVEL SERIALIZABLE
SELECT ...
```

Semantics

When a transaction's isolation level is SERIALIZABLE, a non-historical SELECT query sees:

- all data that was committed before the current transaction began.
- uncommited data resulting from prior statements within the current transaction.

It cannot see data committed during the current transaction by other transactions. To illustrate, two separate transactions are shown executing concurrently.

Transaction A	Transaction B
`SELECT C1 FROM T1;`	`SELECT C1 FROM T1;`
`C1`	`C1`
`--`	`--`
`(0 rows)`	`(0 rows)`
`INSERT INTO T1 (C1) VALUES (1);`	

```
SELECT C1 FROM T1;                SELECT C1 FROM T1;
C1                                C1
--                                --
1                                 (0 rows)
(1 rows)
COMMIT;
SELECT C1 FROM T1;                SELECT C1 FROM T1;
C1                                C1
--                                --
1                                 1
(1 rows)                          (1 rows)
```

Only the owner of the transaction that executed the INSERT statement (A) can see the uncommited data.

READ COMMITTED Isolation

READ COMMITTED isolation allows concurrent transactions and is the default in PostgreSQL and other databases, but not in Vertica. Use READ COMMITED isolation or ***Snapshot Isolation*** (page 63) for normal query operations but be aware that there is a subtle difference between the two:

By itself, AT EPOCH LATEST produces purely historical query behavior. However, with READ COMMITTED, SELECT queries return the same result set as AT EPOCH LATEST plus any changes made by the current transaction.

This is standard ANSI SQL semantics for ACID transactions. Any SELECT query within a transaction should see the transactions's own changes regardless of isolation level.

Index