Pivotal

PRODUCT DOCUMENTATION

Pivotal™ Greenplum Database®

Version 4.3

Client Tools for Windows

Rev: A04

Notice

Copyright

Revised March 2015 (4.3.5.0)

Contents

Chapter 1

Installing the Greenplum Client Tools

This section contains information for installing the various client programs on your Windows machine and for enabling Greenplum Database to accept remote client connections:

- *Running the Client Tools Installer*
- *Configuring the Client Tools*
- *Enabling Greenplum Database for Remote Client Connections*
- *Configuring a Client System for Kerberos Authentication*

See the *Greenplum Database Release Notes* for the list of currently supported platforms for the Client Tools.

Running the Client Tools Installer

The Greenplum Database client tools installer installs `psql.exe`, the interactive command-line client interface to Greenplum Database.

To install the Greenplum Database client tools

1. Download the `greenplum-clients-4.3.x.x-WinXP-x86_32.msi` package from *Pivotal Network*.
2. Double-click on the `greenplum-clients-4.3.x.x-WinXP-x86_32.msi` package to launch the installer.
3. Click **Next** on the Welcome screen.
4. Click **I Agree** on the License Agreement screen.
5. By default, the Greenplum Database client tools will be installed into `greenplum-db-4.3.x.x`. Click **Browse** if you want to choose another location.
6. Click **Next** when you have chosen the install path you want.
7. Click **Install** to begin the installation.
8. Click **Finish** to exit the installer.

About Your Installation

Your Greenplum Database client tools installation contains the following files and directories:

- **bin** — client tools programs
- **greenplum_clients_path.bat** — sets environment variables
- **lib** — client tools library files

Configuring the Client Tools

Greenplum provides a batch program (`greenplum_clients_path.bat`) to set the required environment settings for Greenplum loader (located in `greenplum-db-4.3.x.x` by default).

To set the required environment settings

1. Open a Windows command prompt (**Start** > **Run** and type `cmd`).
2. At the command prompt, go to the directory where you installed Greenplum loader. For example:

```
cd \"Program Files"\Greenplum\greenplum-clients-4.3.x.x
dir
```

3. Execute the `greenplum_loaders_path.bat` program:

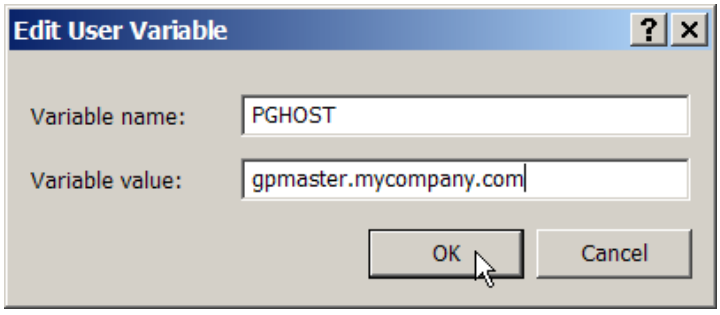
```
greenplum_clients_path.bat
```

The PostgreSQL command-line tools also require several connection parameters in order to be able to connect to a Greenplum database. In order to save some typing on the command-line, you can optionally create the following environment variables in your Windows Control Panel.

- `PGDATABASE` — The name of the default Greenplum database to connect to.
- `PGHOST` — The Greenplum Database master host name or IP address.
- `PGPORT` — The port number that the Greenplum master instance (postmaster process) is running on.
- `PGUSER` — The default database role name to use for login.

To add a new user environment variable on Windows XP

1. In Windows Explorer, go to `C:\\Control Panel`.
2. Double-click the **System** icon.
3. On the **Advanced** tab, click **Environment Variables** (bottom).
4. Click **New**.
5. Define the new environment variable. For example:



6. Click **OK**.

Enabling Greenplum Database for Remote Client Connections

In order for Greenplum Database to be able to accept remote client connections, you must configure your Greenplum Database master so that connections are allowed from the client hosts and database users that will be connecting to Greenplum Database.

To enable remote client connections

1. Make sure that the `pg_hba.conf` file of the Greenplum Database master is correctly configured to allow connections from the users to the database(s) using the authentication method you want. For details, see "Editing the pg_hba.conf File" in the *Greenplum Database Administration Guide*, and also see the *Greenplum Database Security Configuration Guide*.

   Make sure the authentication method you choose is supported by the client tool you are using.
2. If you edited `pg_hba.conf` file, the change requires a server reload (using the `gpstop -u` command) to take effect.
3. Make sure that the databases and roles you are using to connect exist in the system and that the roles have the correct privileges to the database objects.

Configuring a Client System for Kerberos Authentication

If you use Kerberos authentication to connect to your Greenplum Database with the `psql` utility, your client system must be configured to use Kerberos authentication. If you are not using Kerberos authentication to connect to a Greenplum Database, Kerberos is not needed on your client system.

For information about enabling Kerberos authentication with Greenplum Database, see the chapter "Kerberos Authentication" in the *Greenplum Database Administrator Guide*.

Requirements

The following are requirements to connect to a Greenplum Database that is enabled with Kerberos authentication from a client system with Greenplum Database client software.

Prerequisites

- Kerberos must be installed and configured on the Greenplum Database master host. See "*Enabling Greenplum Database for Remote Client Connections*."
- The client systems require the Kerberos configuration file `krb5.conf` from the Greenplum Database master.
- The client systems require a Kerberos keytab file that contains the authentication credentials for the Greenplum Database user that is used to log into the database.
- The client machines must be able to connect to Greenplum Database master host.

  If necessary, add the Greenplum Database master host name and IP address to the system `hosts` file. On Windows 7 systems, the `hosts` file is located in `C:\Windows\System32\drivers\etc\`.

Required Software on the Client Machine

- The Kerberos `kinit` utility. The `kinit.exe` utility is available with Kerberos for Windows. Greenplum Database supports Kerberos for Windows version 3.2.2. Kerberos for Windows is available from the Kerberos web site *http://web.mit.edu/kerberos/*.

  Note: When you install the Kerberos software, you can use other Kerberos utilities such as `klist` to display Kerberos ticket information.

Setting Up a Client System with Kerberos Authentication

To connect to Greenplum Database with Kerberos authentication requires a Kerberos ticket. On client systems, tickets are generated from Kerberos keytab files with the `kinit` utility and are stored in a cache file.

1. Install a copy of the Kerberos configuration file `krb5.conf` from the Greenplum Database master. The file is used by the Greenplum Database client software and the Kerberos utilities.

   Rename `krb5.conf` to `krb5.ini` and move it to the Windows directory. On Windows 7, the Windows directory is `C:\Windows`.

If needed, add the parameter `default_ccache_name` to the `[libdefaults]` section of the `krb5.ini` file and specify the location of the Kerberos ticket cache file on the client system.

2. Obtain a Kerberos keytab file that contains the authentication credentials for the Greenplum Database user.
3. Run `kinit` specifying the keytab file to create a ticket on the client machine. For the following example, the keytab file `gpdb-kerberos.keytab` is in the current directory. The ticket cache file is in the gpadmin user home directory.

```
> kinit -k -t gpdb-kerberos.keytab
   -c C:\Users\gpadmin\cache.txt
   gpadmin/kerberos-gpdb@KRB.GREENPLUM.COM
```

Accessing Greenplum Database with psql

From a remote system, you can access a Greenplum Database that has Kerberos authentication enabled.

1. As the gpadmin user, open a command window.
2. Run the client tool batch file from command window `greenplum_clients_path.bat`

   Change the current directory to the `bin` directory of the Kerberos for Windows installation. For example:

```
> cd C:\test\MIT\Kerberos\bin
```

3. Start `psql` from the command window and specify a connection to the Greenplum Database specifying the user that is configured with Kerberos authentication.

   The following example logs into the Greenplum Database on the machine `keberos-gpdb` as the `gpadmin` user with the Kerberos credentials `gpadmin/kerberos-gpdb`:

```
> psql -U "gpadmin/kerberos-gpdb" -h kerberos-gpdb template1
```

For information about running `psql`, see "*Running the Greenplum Client Tools*."

Chapter 2

Running the Greenplum Client Tools

This section contains information for connecting to Greenplum Database using the `psql.exe` command-line client tool.

Connecting to Greenplum Database

Users and administrators *always* connect to Greenplum Database through the master host. In order to establish a connection to the Greenplum Database master, you will need to know the following connection information and configure your client program accordingly. See *Configuring the Client Tools* for more information.

Table 1: Connection Parameters

Connection Parameter	Description	Environment Variable
Database name	The name of the database to which you want to connect. For a newly initialized system, use the `template1` database to connect for the first time.	`$PGDATABASE`
Host name	The host name of the Greenplum Database master. The default host is the local host.	`$PGHOST`
Port	The port number that the Greenplum Database master instance is running on. The default is 5432.	`$PGPORT`
User name	The database user (role) name to connect as. This is not necessarily the same as your OS user name. Check with your Greenplum administrator if you are not sure what you database user name is. Note that every Greenplum Database system has one superuser account that is created automatically at initialization time. This account has the same name as the OS name of the user who initialized the Greenplum system (typically `gpadmin`).	`$PGUSER`

Running psql.exe

The `psql.exe` program is invoked from a Windows command-line session. For complete command syntax and options for `psql.exe`, see *psql.exe*.

Depending on the default values used or the environment variables you have set, the following examples show how to access a database in Greenplum Database via `psql`:

```
$ psql.exe -d gpdatabase -h master_host -p 5432 -U gpadmin

$ psql.exe gpdatabase

$ psql.exe
```

If a user-defined database has not yet been created, you can access the system by connecting to the `template1` database. For example:

```
$ psql.exe template1
```

After connecting to a database, `psql` provides a prompt with the name of the database to which `psql` is currently connected, followed by the string `=>` (or `=#` if you are the database super user). For example:

```
gpdatabase=>
```

At the prompt, you may type in SQL commands. A SQL command must end with a `;` (semicolon) in order to be sent to the server and executed. For example:

```
=> SELECT count(*) FROM mytable;
```

For more information on using the `psql.exe` client application, see *psql.exe*. For more information on SQL commands and syntax, see *SQL Syntax Summary*.

Chapter 3

Client Tools Reference

This is a reference of the command-line client tools provided with this release. All of these tools can be run from a Windows console session (`cmd`) or a command-line utility such as Cygwin. They all require certain connection information such as the Greenplum master host name, port, database name, and role name. These can be configured using environment variables. For more information, see *Configuring the Client Tools*.

The following tools are provided:

- *`psql.exe`* (PostgreSQL interactive terminal)

psql.exe

Interactive command-line interface for Greenplum Database

Synopsis

```
psql.exe [option ... ] [dbname [username]]
```

Description

`psql` is a terminal-based front-end to Greenplum Database. It enables you to type in queries interactively, issue them to Greenplum Database, and see the query results. Alternatively, input can be from a file. In addition, it provides a number of meta-commands and various shell-like features to facilitate writing scripts and automating a wide variety of tasks.

Options

-a | --echo-all

Print all input lines to standard output as they are read. This is more useful for script processing rather than interactive mode.

-A | --no-align

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

-c 'command' | --command 'command'

Specifies that `psql` is to execute the specified command string, and then exit. This is useful in shell scripts. *command* must be either a command string that is completely parseable by the server, or a single backslash command. Thus you cannot mix SQL and `psql` meta-commands with this option. To achieve that, you could pipe the string into `psql`, like this:

```
echo '\x \\ SELECT * FROM foo;' | psql
```

(`\\` is the separator meta-command.)

If the command string contains multiple SQL commands, they are processed in a single transaction, unless there are explicit `BEGIN/COMMIT` commands included in the string to divide it into multiple transactions. This is different from the behavior when the same string is fed to `psql`'s standard input.

-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.

If this parameter contains an equals sign, it is treated as a `conninfo` string; for example you can pass `'dbname=postgres user=username password=mypass'` as `dbname`.

-e | --echo-queries

Copy all SQL commands sent to the server to standard output as well.

-E | --echo-hidden

Echo the actual queries generated by `\d` and other backslash commands. You can use this to study `psql`'s internal operations.

-f *filename* | --file *filename*

Use a file as the source of commands instead of reading commands interactively. After the file is processed, `psql` terminates. If *filename* is `-` (hyphen), then standard input is read. Using this option is subtly different from writing `psql <`*filename*. In general, both will do

what you expect, but using `-f` enables some nice features such as error messages with line numbers.

-F *separator* | --field-separator *separator*

Use the specified separator as the field separator for unaligned output.

-H | --html

Turn on HTML tabular output.

-l | --list

List all available databases, then exit. Other non-connection options are ignored.

-L *filename* | --log-file *filename*

Write all query output into the specified log file, in addition to the normal output destination.

-o *filename* | --output *filename*

Put all query output into the specified file.

-P *assignment* | --pset *assignment*

Allows you to specify printing options in the style of `\pset` on the command line. Note that here 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 `psql` should do its work quietly. By default, it prints welcome messages and various informational output. If this option is used, none of this happens. This is useful with the `-c` option.

-R *separator* | --record-separator *separator*

Use *separator* as the record separator for unaligned output.

-s | --single-step

Run in single-step mode. That means the user is prompted before each command is sent to the server, with the option to cancel execution as well. Use this to debug scripts.

-S | --single-line

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

-t | --tuples-only

Turn off printing of column names and result row count footers, etc. This command is equivalent to `\pset tuples_only` and is provided for convenience.

-T *table_options* | --table-attr *table_options*

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

-v *assignment* | --set *assignment* | --variable *assignment*

Perform a variable assignment, like the `\set` internal command. Note that you must separate name and value, if any, by an equal sign on the command line. To unset a variable, leave off the equal sign. To just set a variable without a value, use the equal sign but leave off the value. These assignments are done during a very early stage of start-up, so variables reserved for internal purposes might get overwritten later.

-V | --version

Print the `psql` version and exit.

-x | --expanded

Turn on the expanded table formatting mode.

-X | --no-psqlrc

Do not read the start-up file (neither the system-wide `psqlrc` file nor the user's `~/.psqlrc` file).

-1 | --single-transaction

When `psql` executes a script with the `-f` option, adding this option wraps `BEGIN/COMMIT` around the script to execute it as a single transaction. This ensures that either all the commands complete successfully, or no changes are applied.

If the script itself uses `BEGIN`, `COMMIT`, or `ROLLBACK`, this option will not have the desired effects. Also, if the script contains any command that cannot be executed inside a transaction block, specifying this option will cause that command (and hence the whole transaction) to fail.

-? | --help

Show help about `psql` command line arguments, and exit.

Connection Options
-h *host* | --host *host*

The host name of the machine on which the Greenplum master database server is running. If not specified, reads from the environment variable `PGHOST` or defaults to localhost.

-p *port* | --port *port*

The TCP port on which the Greenplum master database server is listening for connections. If not specified, reads from the environment variable `PGPORT` or defaults to 5432.

-U *username* | --username *username*

The database role name to connect as. If not specified, reads from the environment variable `PGUSER` or defaults to the current system role name.

-W | --password

Force a password prompt. `psql` should automatically prompt for a password whenever the server requests password authentication. However, currently password request detection is not totally reliable, hence this option to force a prompt. If no password prompt is issued and the server requires password authentication, the connection attempt will fail.

-w --no-password

Never issue a password prompt. If the server requires password authentication and a password is not available by other means such as a .pgpass file, the connection attempt will fail. This option can be useful in batch jobs and scripts where no user is present to enter a password.

Note: This option remains set for the entire session, and so it affects uses of the meta-command `\connect` as well as the initial connection attempt.

Exit Status

`psql` 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.

Usage

Connecting to a Database

`psql` is a client application for Greenplum Database. In order to connect to a database you need to know the name of your target database, the host name and port number of the Greenplum master server and what database user name you want to connect as. `psql` can be told about those parameters via command line options, namely `-d`, `-h`, `-p`, and `-U` respectively. If an argument is found that does not belong to any option it will be interpreted as the database name (or the user name, if the database name is already given). Not all these options are required; there are useful defaults. If you omit the host name, `psql` will connect via a UNIX-domain socket to a master server on the local host, or via TCP/IP to `localhost` on

machines that do not have UNIX-domain sockets. The default master port number is 5432. If you use a different port for the master, you must specify the port. The default database user name is your UNIX user name, as is the default database name. Note that you cannot just connect to any database under any user name. Your database administrator should have informed you about your access rights.

When the defaults are not right, you can save yourself some typing by setting any or all of the environment variables `PGAPPNAME`, `PGDATABASE`, `PGHOST`, `PGPORT`, and `PGUSER` to appropriate values.

It is also convenient to have a `~/.pgpass` file to avoid regularly having to type in passwords. This file should reside in your home directory and contain lines of the following format:

```
hostname:port:database:username:password
```

The permissions on `.pgpass` must disallow any access to world or group (for example: `chmod 0600 ~/.pgpass`). If the permissions are less strict than this, the file will be ignored. (The file permissions are not currently checked on Microsoft Windows clients, however.)

If the connection could not be made for any reason (insufficient privileges, server is not running, etc.), `psql` will return an error and terminate.

Entering SQL Commands

In normal operation, `psql` provides a prompt with the name of the database to which `psql` is currently connected, followed by the string **=>** for a regular user or **=#** for a superuser. For example:

```
testdb=>
testdb=#
```

At the prompt, the user may type in SQL commands. Ordinarily, input lines are sent to the server when a command-terminating semicolon is reached. An end of line does not terminate a command. Thus commands can be spread over several lines for clarity. If the command was sent and executed without error, the results of the command are displayed on the screen.

Meta-Commands

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

The format of a `psql` 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, use two single quotes. Anything contained in single quotes is furthermore subject to C-like substitutions for `\n` (new line), `\t` (tab), `\digits` (octal), and `\xdigits` (hexadecimal).

If an unquoted argument begins with a colon (`:`), it is taken as a `psql` 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 `psql` 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.

The following meta-commands are defined:

\a

If the current table output format is unaligned, it is switched to aligned. If it is not unaligned, it is set to unaligned. This command is kept for backwards compatibility. See `\pset` for a more general solution.

\cd [*directory*]

Changes the current working directory. Without argument, changes to the current user's home directory. To print your current working directory, use `\!pwd`.

\C [*title*]

Sets the title of any tables being printed as the result of a query or unset any such title. This command is equivalent to `\pset title`.

\c | \connect [*dbname* [*username*] [*host*] [*port*]]

Establishes a new connection. If the new connection is successfully made, the previous connection is closed. If any of dbname, username, host or port are omitted, the value of that parameter from the previous connection is used. If the connection attempt failed, the previous connection will only be kept if `psql` is in interactive mode. When executing a non-interactive script, processing will immediately stop with an error. This distinction was chosen as a user convenience against typos, and a safety mechanism that scripts are not accidentally acting on the wrong database.

\conninfo

Displays information about the current connection including the database name, the user name, the type of connection (UNIX domain socket, `TCP/IP`, etc.), the host, and the port.

\copy {*table* [(*column_list*)] | (*query*)} {from | to} {*filename* | stdin | stdout | pstdin | pstdout} [with] [binary] [oids] [delimiter [as] '*character*'] [null [as] '*string*'] [csv [header] [quote [as] 'character'] [escape [as] '*character*'] [force quote column_list] [force not null column_list]]

Performs a frontend (client) copy. This is an operation that runs an SQL `COPY` command, but instead of the server reading or writing the specified file, `psql` reads or writes the file and routes the data between the server and the local file system. This means that file accessibility and privileges are those of the local user, not the server, and no SQL superuser privileges are required.

The syntax of the command is similar to that of the SQL `COPY` command. Note that, because of this, special parsing rules apply to the `\copy` command. In particular, the variable substitution rules and backslash escapes do not apply.

`\copy ... from stdin | to stdout` reads/writes based on the command input and output respectively. All rows are read from the same source that issued the command, continuing until `\.` is read or the stream reaches `EOF`. Output is sent to the same place as command output. To read/write from `psql`'s standard input or output, use `pstdin` or `pstdout`. This option is useful for populating tables in-line within a SQL script file.

This operation is not as efficient as the SQL `COPY` command because all data must pass through the client/server connection.

\copyright

Shows the copyright and distribution terms of PostgreSQL on which Greenplum Database is based.

\d [*relation_pattern*] | \d+ [*relation_pattern*] | \dS [*relation_pattern*]

For each relation (table, external table, view, index, or sequence) matching the relation pattern, show all columns, their types, the tablespace (if not the default) and any special

attributes such as `NOT NULL` or defaults, if any. Associated indexes, constraints, rules, and triggers are also shown, as is the view definition if the relation is a view.

- The command form `\d+` is identical, except that more information is displayed: any comments associated with the columns of the table are shown, as is the presence of OIDs in the table.
- The command form `\dS` is identical, except that system information is displayed as well as user information.For example, `\dt` displays user tables, but not system tables; `\dtS` displays both user and system tables.Both these commands can take the `+` parameter to display additional information, as in `\dt+` and `\dtS+`.

  If `\d` is used without a pattern argument, it is equivalent to `\dtvs` which will show a list of all tables, views, and sequences.

\da [*aggregate_pattern*]

Lists all available aggregate functions, together with the data types they operate on. If a pattern is specified, only aggregates whose names match the pattern are shown.

\db [*tablespace_pattern*] | \db+ [*tablespace_pattern*]

Lists all available tablespaces and their corresponding filespace locations. If pattern is specified, only tablespaces whose names match the pattern are shown. If + is appended to the command name, each object is listed with its associated permissions.

\dc [*conversion_pattern*]

Lists all available conversions between character-set encodings. If pattern is specified, only conversions whose names match the pattern are listed.

\dC

Lists all available type casts.

\dd [*object_pattern*]

Lists all available objects. If pattern is specified, only matching objects are shown.

\dD [*domain_pattern*]

Lists all available domains. If pattern is specified, only matching domains are shown.

\df [*function_pattern*] | \df+ [*function_pattern*]

Lists available functions, together with their argument and return types. If pattern is specified, only functions whose names match the pattern are shown. If the form `\df+` is used, additional information about each function, including language and description, is shown. To reduce clutter, `\df` does not show data type I/O functions. This is implemented by ignoring functions that accept or return type `cstring`.

\dg [*role_pattern*]

Lists all database roles. If pattern is specified, only those roles whose names match the pattern are listed.

\distPvxS [*index* | *sequence* | *table* | *parent table* | *view* | *external_table* | *system_object*]

This is not the actual command name: the letters `i`, `s`, `t`, `P`, `v`, `x`, `S` stand for index, sequence, table, parent table, view, external table, and system table, respectively. You can specify any or all of these letters, in any order, to obtain a listing of all the matching objects. The letter `S` restricts the listing to system objects; without `S`, only non-system objects are shown. If + is appended to the command name, each object is listed with its associated description, if any. If a pattern is specified, only objects whose names match the pattern are listed.

\dl

This is an alias for `\lo_list`, which shows a list of large objects.

\dn [*schema_pattern*] | \dn+ [*schema_pattern*]

Lists all available schemas (namespaces). If pattern is specified, only schemas whose names match the pattern are listed. Non-local temporary schemas are suppressed. If `+` is appended to the command name, each object is listed with its associated permissions and description, if any.

\do [*operator_pattern*]

Lists available operators with their operand and return types. If pattern is specified, only operators whose names match the pattern are listed.

\dp [*relation_pattern_to_show_privileges*]

Produces a list of all available tables, views and sequences with their associated access privileges. If pattern is specified, only tables, views and sequences whose names match the pattern are listed. The `GRANT` and `REVOKE` commands are used to set access privileges.

\dT [*datatype_pattern*] | \dT+ [*datatype_pattern*]

Lists all data types or only those that match pattern. The command form `\dT+` shows extra information.

\du [*role_pattern*]

Lists all database roles, or only those that match pattern.

\e | \edit [*filename*]

If a file name is specified, the file is edited; after 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 `psql`, where the whole buffer is treated as a single line. (Thus you cannot make scripts this way. Use `\i` for that.) This means also that if the query ends with (or rather contains) a semicolon, it is immediately executed. In other cases it will merely wait in the query buffer.

`psql` searches the environment variables `PSQL_EDITOR`, `EDITOR`, and `VISUAL` (in that order) for an editor to use. If all of them are unset, `vi` is used on UNIX systems, `notepad.exe` on Windows systems.

\echotext [...]

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.

If you use the `\o` command to redirect your query output you may wish to use `'echo` instead of this command.

\encoding [*encoding*]

Sets the client character set encoding. Without an argument, this command shows the current encoding.

\f [*field_separator_string*]

Sets the field separator for unaligned query output. The default is the vertical bar (`|`). See also `\pset` 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 a file or pipes the output into a separate UNIX shell executing command. A bare `\g` is virtually equivalent to a semicolon. A `\g` with argument is a one-shot alternative to the `\o` command.

\h | \help [*sql_command*]

Gives syntax help on the specified SQL command. If a command is not specified, then `psql` will list all the commands for which syntax help is available. Use an asterisk (*) to show syntax help on all SQL commands. To simplify typing, commands that consists of several words do not have to be quoted.

\H

Turns on HTML query output format. If the HTML format is already on, it is switched back to the default aligned text format. This command is for compatibility and convenience, but see `\pset` about setting other output options.

\i *input_filename*

Reads input from a file and executes it as though it had been typed on the keyboard. If you want to see the lines on the screen as they are read you must set the variable `ECHO` to all.

\l | \list | \l+ | \list+

List the names, owners, and character set encodings of all the databases in the server. If `+` is appended to the command name, database descriptions are also displayed.

\lo_export *loid filename*

Reads the large object with OID *loid* from the database and writes it to *filename*. Note that this is subtly different from the server function `lo_export`, which acts with the permissions of the user that the database server runs as and on the server's file system. Use `\lo_list` to find out the large object's OID.

\lo_import *large_object_filename* [*comment*]

Stores the file into a large object. Optionally, it associates the given comment with the object. Example:

```
mydb=> \lo_import '/home/gpadmin/pictures/photo.xcf' 'a
picture of me'
lo_import 152801
```

The response indicates that the large object received object ID 152801 which one ought to remember if one wants to access the object ever again. For that reason it is recommended to always associate a human-readable comment with every object. Those can then be seen with the `\lo_list` command. Note that this command is subtly different from the server-side `lo_import` because it acts as the local user on the local file system, rather than the server's user and file system.

\lo_list

Shows a list of all large objects currently stored in the database, along with any comments provided for them.

\lo_unlink *largeobject_oid*

Deletes the large object of the specified OID from the database. Use `\lo_list` to find out the large object's OID.

\o [{*query_result_filename* | |*command*}]

Saves future query results to a file or pipes them into a UNIX shell command. If no arguments are specified, the query output will be reset to the standard output. Query results include all tables, command responses, and notices obtained from the database server, as well as output of various backslash commands that query the database (such as `\d`), but not error messages. To intersperse text output in between query results, use `'echo`.

\p

Print the current query buffer to the standard output.

\password [*username*]

Changes the password of the specified user (by default, the current user). This command prompts for the new password, encrypts it, and sends it to the server as an `ALTER ROLE` command. This makes sure that the new password does not appear in cleartext in the command history, the server log, or elsewhere.

\prompt [*text*] *name*

Prompts the user to set a variable *name*. Optionally, you can specify a prompt. Enclose prompts longer than one word in single quotes.

By default, \prompt uses the terminal for input and output. However, use the -f command line switch to specify standard input and standard output.

\pset *print_option* [*value*]

This command sets options affecting the output of query result tables. *print_option* describes which option is to be set. Adjustable printing options are:

- **format** – Sets the output format to one of **u**naligned, **a**ligned, **h**tml, **l**atex, **t**roff-ms, or **w**rapped. First letter abbreviations are allowed. 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. 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.)

  The wrapped option sets the output format like the `aligned` parameter , but wraps wide data values across lines to make the output fit in the target column width. The target width is set with the `columns` option. To specify the column width and select the wrapped format, use two \pset commands; for example, to set the with to 72 columns and specify wrapped format, use the commands `\pset columns 72` and then `\pset format wrapped`.

  > **Note:** Since `psql` does not attempt to wrap column header titles, the wrapped format behaves the same as aligned if the total width needed for column headers exceeds the target.

- **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.
- **columns** – Sets the target width for the `wrapped` format, and also the width limit for determining whether output is wide enough to require the pager. The default is *zero*. Zero causes the target width to be controlled by the environment variable `COLUMNS`, or the detected screen width if `COLUMNS` is not set. In addition, if `columns` is zero then the wrapped format affects screen output only. If columns is nonzero then file and pipe output is wrapped to that width as well.

  After setting the target width, use the command `\pset format wrapped` to enable the wrapped format.
- **expanded** | **x)** – Toggles between regular and expanded format. When expanded format is enabled, query results are displayed in two columns, with the column name on the left and the data on the right. This mode is useful if the data would not fit on the screen in the normal horizontal mode. Expanded mode is supported by all four output formats.
- **linestyle** [**unicode** | **ascii** | **old-ascii**] – Sets the border line drawing style to one of unicode, ascii, or old-ascii. Unique abbreviations, including one letter, are allowed for the three styles. The default setting is `ascii`. This option only affects the `aligned` and `wrapped` output formats.

  ascii – uses plain ASCII characters. Newlines in data are shown using a + symbol in the right-hand margin. When the wrapped format wraps data from one line to the next without a newline character, a dot (.) is shown in the right-hand margin of the first line, and again in the left-hand margin of the following line.

  old-ascii – style uses plain ASCII characters, using the formatting style used in PostgreSQL 8.4 and earlier. Newlines in data are shown using a : symbol in place of the left-hand column separator. When the data is wrapped from one line to the next without a newline character, a ; symbol is used in place of the left-hand column separator.

unicode – style uses Unicode box-drawing characters. Newlines in data are shown using a carriage return symbol in the right-hand margin. When the data is wrapped from one line to the next without a newline character, an ellipsis symbol is shown in the right-hand margin of the first line, and again in the left-hand margin of the following line.

When the `border` setting is greater than zero, this option also determines the characters with which the border lines are drawn. Plain ASCII characters work everywhere, but Unicode characters look nicer on displays that recognize them.

- **null 'string'** – The second argument is a string to print whenever a column is null. The default is not to print anything, which can easily be mistaken for an empty string. For example, the command `\psetnull '(empty)'` displays *(empty)* in null columns.
- **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).
- **numericlocale** – Toggles the display of a locale-aware character to separate groups of digits to the left of the decimal marker. It also enables a locale-aware decimal marker.
- **recordsep** – Specifies the record (line) separator to use in unaligned output mode. The default is a newline character.
- **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** | **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`.
- **tuples_only** | **t** [*novalue* | *on* | *off*] – The `\pset tuples_only` command by itselt toggles between tuples only and full display. The values *on* and *off* set the tuples display, regardless of the current setting. Full display may show extra information such as column headers, titles, and various footers. In tuples only mode, only actual table data is shown The `\t` command is equivalent to `\psettuples_only` and is provided for convenience.
- **pager** – Controls the use of a pager for query and `psql` help output. When `on`, 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 `off`, the pager is not used. When `on`, the pager is used only when appropriate. Pager can also be set to `always`, which causes the pager to be always used.

\q

Quits the `psql` program.

\qechotext [...]

This command is identical to `\echo` except that the output will be written to the query output channel, as set by `\o`.

\r

Resets (clears) the query buffer.

\s [*history_filename*]

Print or save the command line history to *filename*. If *filename* is omitted, the history is written to the standard output.

\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 just set with no value. To unset a variable, use the `\unset` command.

Valid variable names can contain characters, digits, and underscores. See "Variables" in *Advanced Features*. Variable names are case-sensitive.

Although you are welcome to set any variable to anything you want, `psql` treats several variables as special. They are documented in the topic about variables.

This command is totally separate from the SQL command `SET`.

\t [novalue | on | off]

The `\t` command by itself toggles a display of output column name headings and row count footer. The values `on` and `off` set the tuples display, regardless of the current setting.This command is equivalent to `\pset tuples_only` and is provided for convenience.

\T *table_options*

Allows you to specify attributes to be placed within the table tag in HTML tabular output mode.

\timing [novalue | on | off]

The `\timing` command by itself toggles a display of how long each SQL statement takes, in milliseconds. The values `on` and `off` set the time display, regardless of the current setting.

\w {*filename* | |*command*}

Outputs the current query buffer to a file or pipes it to a UNIX command.

\x

Toggles expanded table formatting mode.

\z [*relation_to_show_privileges*]

Produces a list of all available tables, views and sequences with their associated access privileges. If a pattern is specified, only tables, views and sequences whose names match the pattern are listed. This is an alias for `\dp`.

\! [*command*]

Escapes to a separate UNIX shell or executes the UNIX command. The arguments are not further interpreted, the shell will see them as is.

\?

Shows help information about the `psql` backslash commands.

Patterns

The various `\d` commands accept a pattern parameter to specify the object name(s) to be displayed. In the simplest case, a pattern is just the exact name of the object. The characters within a pattern are normally folded to lower case, just as in SQL names; for example, `\dt FOO` will display the table named `foo`. As in SQL names, placing double quotes around a pattern stops folding to lower case. Should you need to include an actual double quote character in a pattern, write it as a pair of double quotes within a double-quote sequence; again this is in accord with the rules for SQL quoted identifiers. For example, `\dt "FOO""BAR"` will display the table named `FOO"BAR` (not `foo"bar`). Unlike the normal rules for SQL names, you can put double quotes around just part of a pattern, for instance `\dt FOO"FOO"BAR` will display the table named `fooFOObar`.

Within a pattern, `*` matches any sequence of characters (including no characters) and `?` matches any single character. (This notation is comparable to UNIX shell file name patterns.) For example, `\dt int*` displays all tables whose names begin with `int`. But within double quotes, `*` and `?` lose these special meanings and are just matched literally.

A pattern that contains a dot (`.`) is interpreted as a schema name pattern followed by an object name pattern. For example, `\dt foo*.bar*` displays all tables whose table name starts with `bar` that are in schemas whose schema name starts with `foo`. When no dot appears, then the pattern matches only objects that are visible in the current schema search path. Again, a dot within double quotes loses its special meaning and is matched literally.

Advanced users can use regular-expression notations. All regular expression special characters work as specified in the *PostgreSQL documentation on regular expressions*, except for `.` which is taken as a separator as mentioned above, `*` which is translated to the regular-expression notation `.*`, and `?` which is translated to `.`. You can emulate these pattern characters at need by writing `?` for `.`, `(R+|)` for `R*`, or `(R|)` for `R?`. Remember that the pattern must match the whole name, unlike the usual interpretation of regular expressions; write `*` at the beginning and/or end if you don't wish the pattern to be anchored. Note that within double quotes, all regular expression special characters lose their special meanings and are matched literally. Also, the regular expression special characters are matched literally in operator name patterns (such as the argument of `\do`).

Whenever the pattern parameter is omitted completely, the `\d` commands display all objects that are visible in the current schema search path – this is equivalent to using the pattern `*`. To see all objects in the database, use the pattern `*.*`.

Advanced Features

Variables

`psql` provides variable substitution features similar to common UNIX command shells. Variables are simply name/value pairs, where the value can be any string of any length. To set variables, use the `psql` meta-command `\set`:

```
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. Thus you can construct interesting references such as `\set :foo 'something'` and get 'soft links' or 'variable variables' of Perl or PHP fame, respectively. Unfortunately, there is no way to do anything useful with these constructs. On the other hand, `\set bar :foo` is a perfectly 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`.

`psql`'s internal variable names can consist of letters, numbers, and underscores in any order and any number of them. A number of these variables are treated specially by `psql`. 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, as the program behavior might behave unexpectedly. 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. A list of all specially treated variables are as follows:

AUTOCOMMIT

When on (the default), each SQL command is automatically committed upon successful completion. To postpone commit in this mode, you must enter a `BEGIN` or `START TRANSACTION` SQL command. When off or unset, SQL commands are not committed until you explicitly issue `COMMIT` or `END`. The autocommit-on mode works by issuing an implicit `BEGIN` for you, just before any command that is not already in a transaction block and is not itself a `BEGIN` or other transaction-control command, nor a command that cannot be executed inside a transaction block (such as `VACUUM`).

In autocommit-off mode, you must explicitly abandon any failed transaction by entering `ABORT` or `ROLLBACK`. Also keep in mind that if you exit the session without committing, your work will be lost.

The autocommit-on mode is PostgreSQL's traditional behavior, but autocommit-off is closer to the SQL spec. If you prefer autocommit-off, you may wish to set it in your `~/.psqlrc` file.

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`. If set to queries, `psql` merely prints all queries as they are sent to the server. The switch for this is `-e`.

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 Greenplum Database internals and provide similar functionality in your own programs. (To select this behavior on program start-up, use the switch `-E`.) 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.

FETCH_COUNT

If this variable is set to an integer value > 0, the results of `SELECT` queries are fetched and displayed in groups of that many rows, rather than the default behavior of collecting the entire result set before display. Therefore only a limited amount of memory is used, regardless of the size of the result set. Settings of 100 to 1000 are commonly used when enabling this feature. Keep in mind that when using this feature, a query may fail after having already displayed some rows.

Although you can use any output format with this feature, the default aligned format tends to look bad because each group of `FETCH_COUNT` rows will be formatted separately, leading to varying column widths across the row groups. The other output formats work better.

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.

HISTFILE

The file name that will be used to store the history list. The default value is `~/.psql_history`. For example, putting

```
\set HISTFILE ~/.psql_history- :DBNAME
```

in `~/.psqlrc` will cause `psql` to maintain a separate history for each database.

HISTSIZE

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

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 `CTRL+D`) to an interactive session of `psql` 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`.

LASTOID

The value of the last affected OID, as returned from an `INSERT` or `lo_insert` command. This variable is only guaranteed to be valid until after the result of the next SQL command has been displayed.

ON_ERROR_ROLLBACK

When on, if a statement in a transaction block generates an error, the error is ignored and the transaction continues. When interactive, such errors are only ignored in interactive sessions, and not when reading script files. When off (the default), a statement in a transaction block that generates an error aborts the entire transaction. The on_error_rollback-on mode works by issuing an implicit `SAVEPOINT` for you, just before each command that is in a transaction block, and rolls back to the savepoint on error.

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 `psql` 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 `psql` session but rather using the `-f` option, `psql` 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 `psql` issues should look like. See "Prompting".

QUIET

This variable is equivalent to the command line option `-q`. It is not very useful in interactive mode.

SINGLELINE

This variable is equivalent to the command line option `-S`.

SINGLESTEP

This variable is equivalent to the command line option `-s`.

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

An additional useful feature of `psql` 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. You must make sure that it makes sense where you put it. Variable interpolation will not be performed into quoted SQL entities.

A popular application of this facility is to refer to the last inserted OID in subsequent statements to build a foreign key scenario. Another possible use of this mechanism is to copy the contents of a file into a table column. First load the file into a variable and then proceed as above.

```
testdb=> \set content '''' `cat my_file.txt` ''''
testdb=> INSERT INTO my_table VALUES (:content);
```

One problem with this approach is that `my_file.txt` might contain single quotes. These need to be escaped so that they don't cause a syntax error when the second line is processed. This could be done with the program `sed`:

```
testdb=> \set content '''' `sed -e "s/'/''/g" < my_file.txt`
''''
```

If you are using non-standard-conforming strings then you'll also need to double backslashes. This is a bit tricky:

```
testdb=> \set content '''' `sed -e "s/'/''/g" -e
's/\\/\\\\/g' < my_file.txt` ''''
```

Note the use of different shell quoting conventions so that neither the single quote marks nor the backslashes are special to the shell. Backslashes are still special to `sed`, however, so we need to double them.

Since colons may legally appear in SQL commands, the following rule applies: the character sequence `":name"` is not changed unless `"name"` is the name of a variable that is currently set. In any case you can escape a colon with a backslash to protect it from substitution. (The colon syntax for variables is standard SQL for embedded query languages, such as ECPG. The colon syntax for array slices and type casts are Greenplum Database extensions, hence the conflict.)

Prompting

The prompts `psql` 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 `psql` 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 UNIX domain socket, or `[local:/dir/name]`, if the UNIX domain socket is not at the compiled in default location.

%m

The host name of the database server, truncated at the first dot, or `[local]` if the connection is over a UNIX domain socket.

%>

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 `psql` 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 octal code is substituted.

%:name:

The value of the `psql` variable name. See "Variables" in *Advanced Features* 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 line editing 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.

Command-Line Editing

`psql` supports the NetBSD libedit library for convenient line editing and retrieval. The command history is automatically saved when `psql` exits and is reloaded when `psql` 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 psql
set disable-completion on
$endif
```

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
PGHOST
PGPORT
PGUSER

Default connection parameters.

PSQL_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, `psql` attempts to read and execute commands from the user's `~/.psqlrc` file.

The command-line history is stored in the file `~/.psql_history`.

Notes

`psql` only works smoothly with servers of the same version. That does not mean other combinations will fail outright, but subtle and not-so-subtle problems might come up. Backslash commands are particularly likely to fail if the server is of a different version.

Notes for Windows users

`psql` 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 `psql`. If `psql` 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 character encoding of the Latin alphabet, used by Microsoft Windows for English and some other Western languages. 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.

Examples

Start `psql` in interactive mode:

```
psql.exe -p 54321 -U sally mydatabase
```

In `psql` interactive mode, 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
```

Look at the table definition:

```
testdb=> \d my_table
             Table "my_table"
 Attribute |  Type   |      Modifier
-----------+---------+--------------------
 first     | integer | not null default 0
 second    | text    |
```

Run `psql` in non-interactive mode by passing in a file containing SQL commands:

```
psql.exe -f /home/gpadmin/test/myscript.sql
```

Chapter 4

SQL Syntax Summary

ABORT

Aborts the current transaction.

```
ABORT [WORK | TRANSACTION]
```

ALTER AGGREGATE

Changes the definition of an aggregate function

```
ALTER AGGREGATE name ( type [ , ... ] ) RENAME TO new_name

ALTER AGGREGATE name ( type [ , ... ] ) OWNER TO new_owner

ALTER AGGREGATE name ( type [ , ... ] ) SET SCHEMA new_schema
```

ALTER CONVERSION

Changes the definition of a conversion.

```
ALTER CONVERSION name RENAME TO newname

ALTER CONVERSION name OWNER TO newowner
```

ALTER DATABASE

Changes the attributes of a database.

```
ALTER DATABASE name [ WITH CONNECTION LIMIT connlimit ]

ALTER DATABASE name SET parameter { TO | = } { value | DEFAULT }

ALTER DATABASE name RESET parameter

ALTER DATABASE name RENAME TO newname

ALTER DATABASE name OWNER TO new_owner
```

ALTER DOMAIN

Changes the definition of a domain.

```
ALTER DOMAIN name { SET DEFAULT expression | DROP DEFAULT }

ALTER DOMAIN name { SET | DROP } NOT NULL

ALTER DOMAIN name ADD domain_constraint

ALTER DOMAIN name DROP CONSTRAINT constraint_name [RESTRICT | CASCADE]

ALTER DOMAIN name OWNER TO new_owner

ALTER DOMAIN name SET SCHEMA new_schema
```

ALTER EXTERNAL TABLE

Changes the definition of an external table.

```
ALTER EXTERNAL TABLE name RENAME [COLUMN] column TO new_column

ALTER EXTERNAL TABLE name RENAME TO new_name

ALTER EXTERNAL TABLE name SET SCHEMA new_schema

ALTER EXTERNAL TABLE name action [, ... ]
```

ALTER FILESPACE

Changes the definition of a filespace.

```
ALTER FILESPACE name RENAME TO newname

ALTER FILESPACE name OWNER TO newowner
```

ALTER FUNCTION

Changes the definition of a function.

```
ALTER FUNCTION name ( [ [argmode] [argname] argtype [, ...] ] )
   action [, ... ] [RESTRICT]

ALTER FUNCTION name ( [ [argmode] [argname] argtype [, ...] ] )
   RENAME TO new_name

ALTER FUNCTION name ( [ [argmode] [argname] argtype [, ...] ] )
   OWNER TO new_owner

ALTER FUNCTION name ( [ [argmode] [argname] argtype [, ...] ] )
   SET SCHEMA new_schema
```

ALTER GROUP

Changes a role name or membership.

```
ALTER GROUP groupname ADD USER username [, ... ]

ALTER GROUP groupname DROP USER username [, ... ]

ALTER GROUP groupname RENAME TO newname
```

ALTER INDEX

Changes the definition of an index.

```
ALTER INDEX name RENAME TO new_name

ALTER INDEX name SET TABLESPACE tablespace_name

ALTER INDEX name SET ( FILLFACTOR = value )

ALTER INDEX name RESET ( FILLFACTOR )
```

ALTER LANGUAGE

Changes the name of a procedural language.

```
ALTER LANGUAGE name RENAME TO newname
```

ALTER OPERATOR

Changes the definition of an operator.

```
ALTER OPERATOR name ( {lefttype | NONE} , {righttype | NONE} )
   OWNER TO newowner
```

ALTER OPERATOR CLASS

Changes the definition of an operator class.

```
ALTER OPERATOR CLASS name USING index_method RENAME TO newname

ALTER OPERATOR CLASS name USING index_method OWNER TO newowner
```

ALTER PROTOCOL

Changes the definition of a protocol.

```
ALTER PROTOCOL name RENAME TO newname

ALTER PROTOCOL name OWNER TO newowner
```

ALTER RESOURCE QUEUE

Changes the limits of a resource queue.

```
ALTER RESOURCE QUEUE name WITH ( queue_attribute=value [, ... ] )
```

ALTER ROLE

Changes a database role (user or group).

```
ALTER ROLE name RENAME TO newname

ALTER ROLE name SET config_parameter {TO | =} {value | DEFAULT}

ALTER ROLE name RESET config_parameter

ALTER ROLE name RESOURCE QUEUE {queue_name | NONE}

ALTER ROLE name [ [WITH] option [ ... ] ]
```

ALTER SCHEMA

Changes the definition of a schema.

```
ALTER SCHEMA name RENAME TO newname

ALTER SCHEMA name OWNER TO newowner
```

ALTER SEQUENCE

Changes the definition of a sequence generator.

```
ALTER SEQUENCE name [INCREMENT [ BY ] increment]
     [MINVALUE minvalue | NO MINVALUE]
     [MAXVALUE maxvalue | NO MAXVALUE]
     [RESTART [ WITH ] start]
     [CACHE cache] [[ NO ] CYCLE]
     [OWNED BY {table.column | NONE}]

ALTER SEQUENCE name SET SCHEMA new_schema
```

ALTER TABLE

Changes the definition of a table.

```
ALTER TABLE [ONLY] name RENAME [COLUMN] column TO new_column

ALTER TABLE name RENAME TO new_name

ALTER TABLE name SET SCHEMA new_schema

ALTER TABLE [ONLY] name SET
     DISTRIBUTED BY (column, [ ... ] )
   | DISTRIBUTED RANDOMLY
   | WITH (REORGANIZE=true|false)

ALTER TABLE [ONLY] name action [, ... ]

ALTER TABLE name
   [ ALTER PARTITION { partition_name | FOR (RANK(number))
   | FOR (value) } partition_action [...] ]
   partition_action
```

ALTER TABLESPACE

Changes the definition of a tablespace.

```
ALTER TABLESPACE name RENAME TO newname

ALTER TABLESPACE name OWNER TO newowner
```

ALTER TYPE

Changes the definition of a data type.

```
ALTER TYPE name
   OWNER TO new_owner | SET SCHEMA new_schema
```

ALTER USER

Changes the definition of a database role (user).

```
ALTER USER name RENAME TO newname

ALTER USER name SET config_parameter {TO | =} {value | DEFAULT}

ALTER USER name RESET config_parameter

ALTER USER name [ [WITH] option [ ... ] ]
```

ANALYZE

Collects statistics about a database.

```
ANALYZE [VERBOSE] [ROOTPARTITION [ALL] ]
   [table [ (column [, ...] ) ]]
```

BEGIN

Starts a transaction block.

```
BEGIN [WORK | TRANSACTION] [transaction_mode]
      [READ ONLY | READ WRITE]
```

CHECKPOINT

Forces a transaction log checkpoint.

```
CHECKPOINT
```

CLOSE

Closes a cursor.

```
CLOSE cursor_name
```

CLUSTER

Physically reorders a heap storage table on disk according to an index. Not a recommended operation in Greenplum Database.

```
CLUSTER indexname ON tablename

CLUSTER tablename

CLUSTER
```

COMMENT

Defines or change the comment of an object.

```
COMMENT ON
{ TABLE object_name |
  COLUMN table_name.column_name |
  AGGREGATE agg_name (agg_type [, ...]) |
  CAST (sourcetype AS targettype) |
  CONSTRAINT constraint_name ON table_name |
  CONVERSION object_name |
  DATABASE object_name |
  DOMAIN object_name |
  FILESPACE object_name |
  FUNCTION func_name ([[argmode] [argname] argtype [, ...]]) |
  INDEX object_name |
  LARGE OBJECT large_object_oid |
  OPERATOR op (leftoperand_type, rightoperand_type) |
  OPERATOR CLASS object_name USING index_method |
  [PROCEDURAL] LANGUAGE object_name |
  RESOURCE QUEUE object_name |
  ROLE object_name |
  RULE rule_name ON table_name |
  SCHEMA object_name |
  SEQUENCE object_name |
```

```
  TABLESPACE object_name |
  TRIGGER trigger_name ON table_name |
  TYPE object_name |
  VIEW object_name }
IS 'text'
```

COMMIT

Commits the current transaction.

```
COMMIT [WORK | TRANSACTION]
```

COPY

Copies data between a file and a table.

```
COPY table [(column [, ...])] FROM {'file' | STDIN}
     [ [WITH]
       [OIDS]
       [HEADER]
       [DELIMITER [ AS ] 'delimiter']
       [NULL [ AS ] 'null string']
       [ESCAPE [ AS ] 'escape' | 'OFF']
       [NEWLINE [ AS ] 'LF' | 'CR' | 'CRLF']
       [CSV [QUOTE [ AS ] 'quote']
            [FORCE NOT NULL column [, ...]]
       [FILL MISSING FIELDS]
       [[LOG ERRORS [INTO error_table] [KEEP]
       SEGMENT REJECT LIMIT count [ROWS | PERCENT] ]

COPY {table [(column [, ...])] | (query)} TO {'file' | STDOUT}
      [ [WITH]
        [OIDS]
        [HEADER]
        [DELIMITER [ AS ] 'delimiter']
        [NULL [ AS ] 'null string']
        [ESCAPE [ AS ] 'escape' | 'OFF']
        [CSV [QUOTE [ AS ] 'quote']
             [FORCE QUOTE column [, ...]] ]
```

CREATE AGGREGATE

Defines a new aggregate function.

```
CREATE [ORDERED] AGGREGATE name (input_data_type [ , ... ])
      ( SFUNC = sfunc,
        STYPE = state_data_type
        [, PREFUNC = prefunc]
        [, FINALFUNC = ffunc]
        [, INITCOND = initial_condition]
        [, SORTOP = sort_operator] )
```

CREATE CAST

Defines a new cast.

```
CREATE CAST (sourcetype AS targettype)
       WITH FUNCTION funcname (argtypes)
       [AS ASSIGNMENT | AS IMPLICIT]

CREATE CAST (sourcetype AS targettype) WITHOUT FUNCTION
       [AS ASSIGNMENT | AS IMPLICIT]
```

CREATE CONVERSION

Defines a new encoding conversion.

```
CREATE [DEFAULT] CONVERSION name FOR source_encoding TO
     dest_encoding FROM funcname
```

CREATE DATABASE

Creates a new database.

```
CREATE DATABASE name [ [WITH] [OWNER [=] dbowner]
                     [TEMPLATE [=] template]
                     [ENCODING [=] encoding]
                     [TABLESPACE [=] tablespace]
                     [CONNECTION LIMIT [=] connlimit ] ]
```

CREATE DOMAIN

Defines a new domain.

```
CREATE DOMAIN name [AS] data_type [DEFAULT expression]
       [CONSTRAINT constraint_name
       | NOT NULL | NULL
       | CHECK (expression) [...]]
```

CREATE EXTERNAL TABLE

Defines a new external table.

```
CREATE [READABLE] EXTERNAL TABLE table_name
    ( column_name data_type [, ...] | LIKE other_table )
      LOCATION ('file://seghost[:port]/path/file' [, ...])
        | ('gpfdist://filehost[:port]/file_pattern[#transform]'
        | ('gpfdists://filehost[:port]/file_pattern[#transform]'
            [, ...])
        | ('gphdfs://hdfs_host[:port]/path/file')
      FORMAT 'TEXT'
            [( [HEADER]
               [DELIMITER [AS] 'delimiter' | 'OFF']
               [NULL [AS] 'null string']
               [ESCAPE [AS] 'escape' | 'OFF']
               [NEWLINE [ AS ] 'LF' | 'CR' | 'CRLF']
               [FILL MISSING FIELDS] )]
          | 'CSV'
            [( [HEADER]
               [QUOTE [AS] 'quote']
               [DELIMITER [AS] 'delimiter']
               [NULL [AS] 'null string']
               [FORCE NOT NULL column [, ...]]
               [ESCAPE [AS] 'escape']
               [NEWLINE [ AS ] 'LF' | 'CR' | 'CRLF']
               [FILL MISSING FIELDS] )]
          | 'CUSTOM' (Formatter=<formatter specifications>)
     [ ENCODING 'encoding' ]
     [ [LOG ERRORS [INTO error_table]] SEGMENT REJECT LIMIT count
       [ROWS | PERCENT] ]

CREATE [READABLE] EXTERNAL WEB TABLE table_name
   ( column_name data_type [, ...] | LIKE other_table )
      LOCATION ('http://webhost[:port]/path/file' [, ...])
    | EXECUTE 'command' [ON ALL
                         | MASTER
                         | number_of_segments
                         | HOST ['segment_hostname']
```

```
                          | SEGMENT segment_id ]
      FORMAT 'TEXT'
            [( [HEADER]
               [DELIMITER [AS] 'delimiter' | 'OFF']
               [NULL [AS] 'null string']
               [ESCAPE [AS] 'escape' | 'OFF']
               [NEWLINE [ AS ] 'LF' | 'CR' | 'CRLF']
               [FILL MISSING FIELDS] )]
          | 'CSV'
            [( [HEADER]
               [QUOTE [AS] 'quote']
               [DELIMITER [AS] 'delimiter']
               [NULL [AS] 'null string']
               [FORCE NOT NULL column [, ...]]
               [ESCAPE [AS] 'escape']
               [NEWLINE [ AS ] 'LF' | 'CR' | 'CRLF']
               [FILL MISSING FIELDS] )]
          | 'CUSTOM' (Formatter=<formatter specifications>)
     [ ENCODING 'encoding' ]
     [ [LOG ERRORS [INTO error_table]] SEGMENT REJECT LIMIT count
       [ROWS | PERCENT] ]

CREATE WRITABLE EXTERNAL TABLE table_name
    ( column_name data_type [, ...] | LIKE other_table )
     LOCATION('gpfdist://outputhost[:port]/filename[#transform]'
      | ('gpfdists://outputhost[:port]/file_pattern[#transform]'
         [, ...])
      | ('gphdfs://hdfs_host[:port]/path')
      FORMAT 'TEXT'
               [( [DELIMITER [AS] 'delimiter']
               [NULL [AS] 'null string']
               [ESCAPE [AS] 'escape' | 'OFF'] )]
          | 'CSV'
               [([QUOTE [AS] 'quote']
               [DELIMITER [AS] 'delimiter']
               [NULL [AS] 'null string']
               [FORCE QUOTE column [, ...]] ]
               [ESCAPE [AS] 'escape'] )]
          | 'CUSTOM' (Formatter=<formatter specifications>)
    [ ENCODING 'write_encoding' ]
    [ DISTRIBUTED BY (column, [ ... ] ) | DISTRIBUTED RANDOMLY ]

CREATE WRITABLE EXTERNAL WEB TABLE table_name
    ( column_name data_type [, ...] | LIKE other_table )
    EXECUTE 'command' [ON ALL]
    FORMAT 'TEXT'
               [( [DELIMITER [AS] 'delimiter']
               [NULL [AS] 'null string']
               [ESCAPE [AS] 'escape' | 'OFF'] )]
          | 'CSV'
               [([QUOTE [AS] 'quote']
               [DELIMITER [AS] 'delimiter']
               [NULL [AS] 'null string']
               [FORCE QUOTE column [, ...]] ]
               [ESCAPE [AS] 'escape'] )]
          | 'CUSTOM' (Formatter=<formatter specifications>)
    [ ENCODING 'write_encoding' ]
    [ DISTRIBUTED BY (column, [ ... ] ) | DISTRIBUTED RANDOMLY ]
```

CREATE FUNCTION

Defines a new function.

```
CREATE [OR REPLACE] FUNCTION name
    ( [ [argmode] [argname] argtype [, ...] ] )
      [ RETURNS { [ SETOF ] rettype
        | TABLE ([{ argname argtype | LIKE other table }
          [, ...]])
        } ]
```

```
    { LANGUAGE langname
    | IMMUTABLE | STABLE | VOLATILE
    | CALLED ON NULL INPUT | RETURNS NULL ON NULL INPUT | STRICT
    | [EXTERNAL] SECURITY INVOKER | [EXTERNAL] SECURITY DEFINER
    | AS 'definition'
    | AS 'obj_file', 'link_symbol' } ...
    [ WITH ({ DESCRIBE = describe_function
           } [, ...] ) ]
```

CREATE GROUP

Defines a new database role.

```
CREATE GROUP name [ [WITH] option [ ... ] ]
```

CREATE INDEX

Defines a new index.

```
CREATE [UNIQUE] INDEX name ON table
       [USING btree|bitmap|gist]
       ( {column | (expression)} [opclass] [, ...] )
       [ WITH ( FILLFACTOR = value ) ]
       [TABLESPACE tablespace]
       [WHERE predicate]
```

CREATE LANGUAGE

Defines a new procedural language.

```
CREATE [PROCEDURAL] LANGUAGE name

CREATE [TRUSTED] [PROCEDURAL] LANGUAGE name
       HANDLER call_handler [VALIDATOR valfunction]
```

CREATE OPERATOR

Defines a new operator.

```
CREATE OPERATOR name (
       PROCEDURE = funcname
       [, LEFTARG = lefttype] [, RIGHTARG = righttype]
       [, COMMUTATOR = com_op] [, NEGATOR = neg_op]
       [, RESTRICT = res_proc] [, JOIN = join_proc]
       [, HASHES] [, MERGES]
       [, SORT1 = left_sort_op] [, SORT2 = right_sort_op]
       [, LTCMP = less_than_op] [, GTCMP = greater_than_op] )
```

CREATE OPERATOR CLASS

Defines a new operator class.

```
CREATE OPERATOR CLASS name [DEFAULT] FOR TYPE data_type
  USING index_method AS
  {
  OPERATOR strategy_number op_name [(op_type, op_type)] [RECHECK]
  | FUNCTION support_number funcname (argument_type [, ...] )
  | STORAGE storage_type
  } [, ... ]
```

CREATE RESOURCE QUEUE

Defines a new resource queue.

```
CREATE RESOURCE QUEUE name WITH (queue_attribute=value [, ... ])
```

CREATE ROLE

Defines a new database role (user or group).

```
CREATE ROLE name [[WITH] option [ ... ]]
```

CREATE RULE

Defines a new rewrite rule.

```
CREATE [OR REPLACE] RULE name AS ON event
  TO table [WHERE condition]
  DO [ALSO | INSTEAD] { NOTHING | command | (command; command
  ...) }
```

CREATE SCHEMA

Defines a new schema.

```
CREATE SCHEMA schema_name [AUTHORIZATION username]
   [schema_element [ ... ]]

CREATE SCHEMA AUTHORIZATION rolename [schema_element [ ... ]]
```

CREATE SEQUENCE

Defines a new sequence generator.

```
CREATE [TEMPORARY | TEMP] SEQUENCE name
       [INCREMENT [BY] value]
       [MINVALUE minvalue | NO MINVALUE]
       [MAXVALUE maxvalue | NO MAXVALUE]
       [START [ WITH ] start]
       [CACHE cache]
       [[NO] CYCLE]
       [OWNED BY { table.column | NONE }]
```

CREATE TABLE

Defines a new table.

```
CREATE [[GLOBAL | LOCAL] {TEMPORARY | TEMP}] TABLE table_name (
[ { column_name data_type [ DEFAULT default_expr ]
   [column_constraint [ ... ]
[ ENCODING ( storage_directive [,...] ) ]
]
   | table_constraint
   | LIKE other_table [{INCLUDING | EXCLUDING}
                      {DEFAULTS | CONSTRAINTS}] ...}
   [, ... ] ]
   )
   [ INHERITS ( parent_table [, ... ] ) ]
   [ WITH ( storage_parameter=value [, ... ] )
   [ ON COMMIT {PRESERVE ROWS | DELETE ROWS | DROP} ]
   [ TABLESPACE tablespace ]
   [ DISTRIBUTED BY (column, [ ... ] ) | DISTRIBUTED RANDOMLY ]
```

```
   [ PARTITION BY partition_type (column)
       [ SUBPARTITION BY partition_type (column) ]
          [ SUBPARTITION TEMPLATE ( template_spec ) ]
       [...]
    ( partition_spec )
        | [ SUBPARTITION BY partition_type (column) ]
          [...]
    ( partition_spec
      [ ( subpartition_spec
           [(...)]
       ) ]
    )
```

CREATE TABLE AS

Defines a new table from the results of a query.

```
CREATE [ [GLOBAL | LOCAL] {TEMPORARY | TEMP} ] TABLE table_name
   [(column_name [, ...] )]
   [ WITH ( storage_parameter=value [, ... ] ) ]
   [ON COMMIT {PRESERVE ROWS | DELETE ROWS | DROP}]
   [TABLESPACE tablespace]
   AS query
   [DISTRIBUTED BY (column, [ ... ] ) | DISTRIBUTED RANDOMLY]
```

CREATE TABLESPACE

Defines a new tablespace.

```
CREATE TABLESPACE tablespace_name [OWNER username]
       FILESPACE filespace_name
```

CREATE TYPE

Defines a new data type.

```
CREATE TYPE name AS ( attribute_name data_type [, ... ] )

CREATE TYPE name (
    INPUT = input_function,
    OUTPUT = output_function
    [, RECEIVE = receive_function]
    [, SEND = send_function]
    [, INTERNALLENGTH = {internallength | VARIABLE}]
    [, PASSEDBYVALUE]
    [, ALIGNMENT = alignment]
    [, STORAGE = storage]
    [, DEFAULT = default]
    [, ELEMENT = element]
    [, DELIMITER = delimiter] )

CREATE TYPE name
```

CREATE USER

Defines a new database role with the `LOGIN` privilege by default.

```
CREATE USER name [ [WITH] option [ ... ] ]
```

CREATE VIEW

Defines a new view.

```
CREATE [OR REPLACE] [TEMP | TEMPORARY] VIEW name
       [ ( column_name [, ...] ) ]
       AS query
```

DEALLOCATE

Deallocates a prepared statement.

```
DEALLOCATE [PREPARE] name
```

DECLARE

Defines a cursor.

```
DECLARE name [BINARY] [INSENSITIVE] [NO SCROLL] CURSOR
     [{WITH | WITHOUT} HOLD]
     FOR query [FOR READ ONLY]
```

DELETE

Deletes rows from a table.

```
DELETE FROM [ONLY] table [[AS] alias]
      [USING usinglist]
      [WHERE condition | WHERE CURRENT OF cursor_name ]
```

DROP AGGREGATE

Removes an aggregate function.

```
DROP AGGREGATE [IF EXISTS] name ( type [, ...] ) [CASCADE | RESTRICT]
```

DROP CAST

Removes a cast.

```
DROP CAST [IF EXISTS] (sourcetype AS targettype) [CASCADE | RESTRICT]
```

DROP CONVERSION

Removes a conversion.

```
DROP CONVERSION [IF EXISTS] name [CASCADE | RESTRICT]
```

DROP DATABASE

Removes a database.

```
DROP DATABASE [IF EXISTS] name
```

DROP DOMAIN

Removes a domain.

```
DROP DOMAIN [IF EXISTS] name [, ...]  [CASCADE | RESTRICT]
```

DROP EXTERNAL TABLE

Removes an external table definition.

```
DROP EXTERNAL [WEB] TABLE [IF EXISTS] name [CASCADE | RESTRICT]
```

DROP FILESPACE

Removes a filespace.

```
DROP FILESPACE [IF EXISTS] filespacename
```

DROP FUNCTION

Removes a function.

```
DROP FUNCTION [IF EXISTS] name ( [ [argmode] [argname] argtype
    [, ...] ] ) [CASCADE | RESTRICT]
```

DROP GROUP

Removes a database role.

```
DROP GROUP [IF EXISTS] name [, ...]
```

DROP INDEX

Removes an index.

```
DROP INDEX [IF EXISTS] name [, ...] [CASCADE | RESTRICT]
```

DROP LANGUAGE

Removes a procedural language.

```
DROP [PROCEDURAL] LANGUAGE [IF EXISTS] name [CASCADE | RESTRICT]
```

DROP OPERATOR

Removes an operator.

```
DROP OPERATOR [IF EXISTS] name ( {lefttype | NONE} ,
    {righttype | NONE} ) [CASCADE | RESTRICT]
```

DROP OPERATOR CLASS

Removes an operator class.

```
DROP OPERATOR CLASS [IF EXISTS] name USING index_method [CASCADE | RESTRICT]
```

DROP OWNED

Removes database objects owned by a database role.

```
DROP OWNED BY name [, ...] [CASCADE | RESTRICT]
```

DROP RESOURCE QUEUE

Removes a resource queue.

```
DROP RESOURCE QUEUE queue_name
```

DROP ROLE

Removes a database role.

```
DROP ROLE [IF EXISTS] name [, ...]
```

DROP RULE

Removes a rewrite rule.

```
DROP RULE [IF EXISTS] name ON relation [CASCADE | RESTRICT]
```

DROP SCHEMA

Removes a schema.

```
DROP SCHEMA [IF EXISTS] name [, ...] [CASCADE | RESTRICT]
```

DROP SEQUENCE

Removes a sequence.

```
DROP SEQUENCE [IF EXISTS] name [, ...] [CASCADE | RESTRICT]
```

DROP TABLE

Removes a table.

```
DROP TABLE [IF EXISTS] name [, ...] [CASCADE | RESTRICT]
```

DROP TABLESPACE

Removes a tablespace.

```
DROP TABLESPACE [IF EXISTS] tablespacename
```

DROP TYPE

Removes a data type.

```
DROP TYPE [IF EXISTS] name [, ...] [CASCADE | RESTRICT]
```

DROP USER

Removes a database role.

```
DROP USER [IF EXISTS] name [, ...]
```

DROP VIEW

Removes a view.

```
DROP VIEW [IF EXISTS] name [, ...] [CASCADE | RESTRICT]
```

END

Commits the current transaction.

```
END [WORK | TRANSACTION]
```

EXECUTE

Executes a prepared SQL statement.

```
EXECUTE name [ (parameter [, ...] ) ]
```

EXPLAIN

Shows the query plan of a statement.

```
EXPLAIN [ANALYZE] [VERBOSE] statement
```

FETCH

Retrieves rows from a query using a cursor.

```
FETCH [ forward_direction { FROM | IN } ] cursorname
```

GRANT

Defines access privileges.

```
GRANT { {SELECT | INSERT | UPDATE | DELETE | REFERENCES |
TRIGGER | TRUNCATE } [,...] | ALL [PRIVILEGES] }
    ON [TABLE] tablename [, ...]
    TO {rolename | PUBLIC} [, ...] [WITH GRANT OPTION]

GRANT { {USAGE | SELECT | UPDATE} [,...] | ALL [PRIVILEGES] }
    ON SEQUENCE sequencename [, ...]
    TO { rolename | PUBLIC } [, ...] [WITH GRANT OPTION]

GRANT { {CREATE | CONNECT | TEMPORARY | TEMP} [,...] | ALL
[PRIVILEGES] }
    ON DATABASE dbname [, ...]
    TO {rolename | PUBLIC} [, ...] [WITH GRANT OPTION]

GRANT { EXECUTE | ALL [PRIVILEGES] }
    ON FUNCTION funcname ( [ [argmode] [argname] argtype [, ...]
] ) [, ...]
    TO {rolename | PUBLIC} [, ...] [WITH GRANT OPTION]

GRANT { USAGE | ALL [PRIVILEGES] }
    ON LANGUAGE langname [, ...]
```

```
    TO {rolename | PUBLIC} [, ...] [WITH GRANT OPTION]

GRANT { {CREATE | USAGE} [,...] | ALL [PRIVILEGES] }
    ON SCHEMA schemaname [, ...]
    TO {rolename | PUBLIC} [, ...] [WITH GRANT OPTION]

GRANT { CREATE | ALL [PRIVILEGES] }
    ON TABLESPACE tablespacename [, ...]
    TO {rolename | PUBLIC} [, ...] [WITH GRANT OPTION]

GRANT parent_role [, ...]
    TO member_role [, ...] [WITH ADMIN OPTION]

GRANT { SELECT | INSERT | ALL [PRIVILEGES] }
    ON PROTOCOL protocolname
    TO username
```

INSERT

Creates new rows in a table.

```
INSERT INTO table [( column [, ...] )]
   {DEFAULT VALUES | VALUES ( {expression | DEFAULT} [, ...] )
   [, ...] | query}
```

LOAD

Loads or reloads a shared library file.

```
LOAD 'filename'
```

LOCK

Locks a table.

```
LOCK [TABLE] name [, ...] [IN lockmode MODE] [NOWAIT]
```

MOVE

Positions a cursor.

```
MOVE [ forward_direction {FROM | IN} ] cursorname
```

PREPARE

Prepare a statement for execution.

```
PREPARE name [ (datatype [, ...] ) ] AS statement
```

REASSIGN OWNED

Changes the ownership of database objects owned by a database role.

```
REASSIGN OWNED BY old_role [, ...] TO new_role
```

REINDEX

Rebuilds indexes.

```
REINDEX {INDEX | TABLE | DATABASE | SYSTEM} name
```

RELEASE SAVEPOINT

Destroys a previously defined savepoint.

```
RELEASE [SAVEPOINT] savepoint_name
```

RESET

Restores the value of a system configuration parameter to the default value.

```
RESET configuration_parameter

RESET ALL
```

REVOKE

Removes access privileges.

```
REVOKE [GRANT OPTION FOR] { {SELECT | INSERT | UPDATE | DELETE
       | REFERENCES | TRIGGER | TRUNCATE } [,...] | ALL [PRIVILEGES] }
       ON [TABLE] tablename [, ...]
       FROM {rolename | PUBLIC} [, ...]
       [CASCADE | RESTRICT]

REVOKE [GRANT OPTION FOR] { {USAGE | SELECT | UPDATE} [,...]
       | ALL [PRIVILEGES] }
       ON SEQUENCE sequencename [, ...]
       FROM { rolename | PUBLIC } [, ...]
       [CASCADE | RESTRICT]

REVOKE [GRANT OPTION FOR] { {CREATE | CONNECT
       | TEMPORARY | TEMP} [,...] | ALL [PRIVILEGES] }
       ON DATABASE dbname [, ...]
       FROM {rolename | PUBLIC} [, ...]
       [CASCADE | RESTRICT]

REVOKE [GRANT OPTION FOR] {EXECUTE | ALL [PRIVILEGES]}
       ON FUNCTION funcname ( [[argmode] [argname] argtype
                              [, ...]] ) [, ...]
       FROM {rolename | PUBLIC} [, ...]
       [CASCADE | RESTRICT]

REVOKE [GRANT OPTION FOR] {USAGE | ALL [PRIVILEGES]}
       ON LANGUAGE langname [, ...]
       FROM {rolename | PUBLIC} [, ...]
       [ CASCADE | RESTRICT ]

REVOKE [GRANT OPTION FOR] { {CREATE | USAGE} [,...]
       | ALL [PRIVILEGES] }
       ON SCHEMA schemaname [, ...]
       FROM {rolename | PUBLIC} [, ...]
       [CASCADE | RESTRICT]

REVOKE [GRANT OPTION FOR] { CREATE | ALL [PRIVILEGES] }
       ON TABLESPACE tablespacename [, ...]
       FROM { rolename | PUBLIC } [, ...]
       [CASCADE | RESTRICT]

REVOKE [ADMIN OPTION FOR] parent_role [, ...]
```

```
       FROM member_role [, ...]
       [CASCADE | RESTRICT]
```

ROLLBACK

Aborts the current transaction.

```
ROLLBACK [WORK | TRANSACTION]
```

ROLLBACK TO SAVEPOINT

Rolls back the current transaction to a savepoint.

```
ROLLBACK [WORK | TRANSACTION] TO [SAVEPOINT] savepoint_name
```

SAVEPOINT

Defines a new savepoint within the current transaction.

```
SAVEPOINT savepoint_name
```

SELECT

Retrieves rows from a table or view.

```
SELECT [ALL | DISTINCT [ON (expression [, ...])]]
  * | expression [[AS] output_name] [, ...]
  [FROM from_item [, ...]]
  [WHERE condition]
  [GROUP BY grouping_element [, ...]]
  [HAVING condition [, ...]]
  [WINDOW window_name AS (window_specification)]
  [{UNION | INTERSECT | EXCEPT} [ALL] select]
  [ORDER BY expression [ASC | DESC | USING operator] [, ...]]
  [LIMIT {count | ALL}]
  [OFFSET start]
  [FOR {UPDATE | SHARE} [OF table_name [, ...]] [NOWAIT] [...]]
```

SELECT INTO

Defines a new table from the results of a query.

```
SELECT [ALL | DISTINCT [ON ( expression [, ...] )]]
    * | expression [AS output_name] [, ...]
    INTO [TEMPORARY | TEMP] [TABLE] new_table
    [FROM from_item [, ...]]
    [WHERE condition]
    [GROUP BY expression [, ...]]
    [HAVING condition [, ...]]
    [{UNION | INTERSECT | EXCEPT} [ALL] select]
    [ORDER BY expression [ASC | DESC | USING operator] [, ...]]
    [LIMIT {count | ALL}]
    [OFFSET start]
    [FOR {UPDATE | SHARE} [OF table_name [, ...]] [NOWAIT]
    [...]]
```

SET

Changes the value of a Greenplum Database configuration parameter.

```
SET [SESSION | LOCAL] configuration_parameter {TO | =} value |
```

```
    'value' | DEFAULT}

SET [SESSION | LOCAL] TIME ZONE {timezone | LOCAL | DEFAULT}
```

SET ROLE

Sets the current role identifier of the current session.

```
SET [SESSION | LOCAL] ROLE rolename

SET [SESSION | LOCAL] ROLE NONE

RESET ROLE
```

SET SESSION AUTHORIZATION

Sets the session role identifier and the current role identifier of the current session.

```
SET [SESSION | LOCAL] SESSION AUTHORIZATION rolename

SET [SESSION | LOCAL] SESSION AUTHORIZATION DEFAULT

RESET SESSION AUTHORIZATION
```

SET TRANSACTION

Sets the characteristics of the current transaction.

```
SET TRANSACTION [transaction_mode] [READ ONLY | READ WRITE]

SET SESSION CHARACTERISTICS AS TRANSACTION transaction_mode
     [READ ONLY | READ WRITE]
```

SHOW

Shows the value of a system configuration parameter.

```
SHOW configuration_parameter

SHOW ALL
```

START TRANSACTION

Starts a transaction block.

```
START TRANSACTION [SERIALIZABLE | REPEATABLE READ | READ
                   COMMITTED | READ UNCOMMITTED]
                  [READ WRITE | READ ONLY]
```

TRUNCATE

Empties a table of all rows.

```
TRUNCATE [TABLE] name [, ...] [CASCADE | RESTRICT]
```

UPDATE

Updates rows of a table.

```
UPDATE [ONLY] table [[AS] alias]
   SET {column = {expression | DEFAULT} |
   (column [, ...]) = ({expression | DEFAULT} [, ...])} [, ...]
   [FROM fromlist]
   [WHERE condition | WHERE CURRENT OF cursor_name ]
```

VACUUM

Garbage-collects and optionally analyzes a database.

```
VACUUM [FULL] [FREEZE] [VERBOSE] [table]

VACUUM [FULL] [FREEZE] [VERBOSE] ANALYZE
              [table [(column [, ...] )]]
```

VALUES

Computes a set of rows.

```
VALUES ( expression [, ...] ) [, ...]
   [ORDER BY sort_expression [ASC | DESC | USING operator] [, ...]]
   [LIMIT {count | ALL}] [OFFSET start]
```