Pivotal

PRODUCT DOCUMENTATION

Pivotal™ Greenplum Database®

Version 4.3

Load Tools for Windows

Rev: A10

Notice

Copyright

Revised October 2016 (4.3.10.0)

Contents

Chapter 1

Installing Greenplum Loader

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

- *Installing Python*
- *Running the Loader Installer*
- *Configuring Greenplum Loader*
- *Enabling Greenplum Database for Remote Client Connections*

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

Installing Python

The Greenplum loader program (`gpload.py`) for Windows requires Python 2.5 (32-bit version) to also be installed on your machine. If you do not have an installation of Python, you can get one from *www.python.org*.

> **Note:** The 64-bit version of Python is not compatible with the Greenplum loader program for Windows.

To install Python

1. Download the Python 2.5 installer for Windows from *https://www.python.org/downloads/*.
2. Locate, then double-click on the `python Load Tools for Windows-2.5.x.msi` package to launch the installer.
3. Select **Install for all users** and click **Next**.
4. By default, Python will be installed into `C:\Python`*xx*. Click **Up** or **New** if you want to choose another location. Click **Next**.
5. Click **Next** to install the selected Python components.
6. Click **Finish** to complete your installation.

Running the Loader Installer

The Greenplum loader installer installs the following loader programs:

- *gpload.py* (loader program)
- *gpfdist.exe* (parallel file distribution program used by gpload.py)

To install Greenplum loader

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, Greenplum loader will be installed into greenplum-db-4.3.*x.x*. Click **Browse** to choose another location.
6. Click **Next**.
7. Click **Install** to begin the installation.
8. Click **Finish** to exit the installer.

About Your Installation

Your Greenplum loader installation contains the following files and directories:

- **bin** — loader command-line utilities (gpload.py and gpfdist.exe)
- **bin/lib** — additional Python libraries needed by gpload.py
- **greenplum_loaders_path.bat** — sets the required environment variables

Configuring Greenplum Loader

Greenplum provides a batch program (`greenplum_loaders_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-loaders-4.3.x.x
dir
```

3. Execute the `greenplum_loaders_path.bat` program:

```
greenplum_loaders_path.bat
```

4. Verify that you can execute the `gpload.py` program:

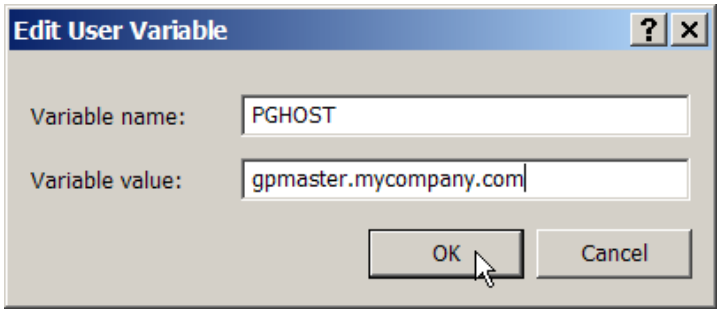
```
gpload.py -?
```

The loader 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 priviliges to the database objects.

Chapter 2

Running Greenplum Loader

This section contains information for defining a load job and running the Greenplum loader program (`gpload.py`).

Before You Begin

Before you can run Greenplum loader:

1. Make sure you have installed and configured Python and the Greenplum loader programs. See *Installing Greenplum Loader*.
2. Make sure that you have network access to and from all hosts in your Greenplum Database array (master and segments), and to and from the hosts where the data to be loaded resides (if not on the local machine).
3. Make sure that the ports you declare in your load control file are unblocked by your Windows firewall.
4. Make sure your Greenplum Database system is up and running and that you know all of the connection information (host name, port, role name, database name, etc.).
5. Create your database, schema, and table structures in Greenplum Database prior to loading data.
6. Prepare your data so that it is in a format acceptable by Greenplum loader. See *Formatting the Input Data*.
7. Write your control file. The control file specifies the source data, load rules, and target table for loading the data. See *Creating the Load Control File*.

Creating the Load Control File

Before you can run Greenplum loader (`gpload.py`), you must create a text file that contains the load specification information. This file must be in valid *YAML 1.1* document format and use the Greenplum schema for defining the various steps of a load operation. See *gpload.py* for details on the correct control file format and schema.

Here is an example of a load control file:

```
---
VERSION: 1.0.0.1
DATABASE: ops
USER: gpadmin
HOST: mdw-1
PORT: 5432
GPLOAD:
   INPUT:
    - SOURCE:
         LOCAL_HOSTNAME:
           - etl1-1
           - etl1-2
           - etl1-3
           - etl1-4
         PORT: 8081
         FILE:
           - /var/load/data/*
    - COLUMNS:
           - name: text
           - amount: float4
           - category: text
           - desc: text
           - date: date
    - FORMAT: text
    - DELIMITER: '|'
    - ERROR_LIMIT: 25
    - ERROR_TABLE: payables.err_expenses
   OUTPUT:
    - TABLE: payables.expenses
    - MODE: INSERT
   SQL:
   - BEFORE: "INSERT INTO audit VALUES('start', current_timestamp)"
   - AFTER: "INSERT INTO audit VALUES('end', current_timestamp)"
```

Formatting the Input Data

When you use Greenplum loader, you need to specify how your data is formatted. Data can be in either delimited text (`TEXT`) or comma separated values (`CSV`) format. External data must be formatted correctly in order to be read by Greenplum Database. This section explains the format of data files expected by Greenplum Database.

Formatting Rows

Greenplum Database expects rows of data to be separated by the `LF` character (Line feed, `0x0A`), `CR` (Carriage return, `0x0D`), or `CR` followed by `LF` (`CR+LF`, `0x0D 0x0A`). `LF` is the standard newline representation on UNIX or UNIX-like operating systems. Other operating systems (such as Windows or Mac OS 9) may use `CR` individually, or `CR+LF`. All of these representations of a newline are supported by Greenplum Database as a row delimiter.

Formatting Columns

The default column or field delimiter is the horizontal `TAB` character (`0x09`) for text files and the comma character (`0x2C`) for CSV files. However, it is possible to declare another single character delimiter using the `DELIMITER` attribute in the load configuration file. The delimiter character must only appear between any two data value fields. Do not place a delimiter at the beginning or end of a row. For example, if using the pipe character (|) as your delimiter:

```
data value 1 | data value 2 | data value 3
```

Representing NULL Values

`NULL` is the value used to represent an unknown piece of data in a column or field. Within your data files you can designate a string to represent null values. The default string is `\N` (backslash-N) in `TEXT` mode, or an empty value with no quotations in `CSV` mode. You can also declare a different string using the `NULL` attribute in the load configuration file. For example, you might prefer an empty string for cases where you do not want to distinguish nulls from empty strings. When using the Greenplum Database loading tools, any data item that matches the designated null string will be considered a null value.

Escaping

The data file has two reserved characters that have special meaning to Greenplum Database:

- The designated delimiter character, which is used to separate columns or fields in the data file.
- The newline character used to designate a new row in the data file.

If your data contains either of these characters, you must escape the character so Greenplum treats it as data and not as a field separator or new row. By default, the escape character is a backslash (`\`) for text-formatted files and a double quote (`"`) for csv-formatted files.

Escaping in Text Formatted Files

By default, the escape character is a backslash (`\`) for text-formatted files. If you want to use a different escape character, use the `ESCAPE` attribute in the load configuration file. In cases where your selected escape character is present in your data, you can use it to escape itself.

For example, suppose you have a table with three columns and you want to load the following three fields:

- `backslash = \`
- `vertical bar = |`
- `exclamation point = !`

Your designated delimiter character is pipe (`|`), and your designated escape character is backslash (`\`). The formatted row in your data file would look like this:

```
backslash = \\ | vertical bar = \| | exclamation point = !
```

Notice how the backslash character that is part of the data is escaped with another backslash character, and the pipe character that is part of the data is escaped with a backslash character.

The escape character can also be used to escape octal and hexidecimal sequences. When used in this way, the escaped value will get converted to the equivalent character when loaded into Greenplum Database. For example, to load the ampersand character (`&`), you could use the escape character to escape its equivalent hexidecimal (`\0x26`) or octal (`\046`) representation.

If there is no need to escape the data in text-formatted files, you can disable escaping using the `ESCAPE` clause of the `COPY` and `CREATE EXTERNALTABLE` commands or the `ESCAPE` attribute of the load control file for *`gpload.py`* as follows:

```
ESCAPE 'OFF'
```

This is useful for input data that contains a lot of backslash characters within the data itself (such as web log data).

Escaping in CSV Formatted Files

By default, the escape character is a double quote (`"`) for CSV-formatted files. If you want to use a different escape character, use the `ESCAPE` clause of `COPY` and `CREATE EXTERNALTABLE` commands or the `ESCAPE` attribute of the load control file for *`gpload.py`* to declare a different escape character. In cases where your selected escape character is present in your data, you can use it to escape itself.

For example, suppose you have a table with three columns and you want to load the following three fields:

- `Free trip to A,B`
- `5.89`
- `Special rate "1.79"`

Your designated delimiter character is comma (`,`), and your designated escape character is double quote (`"`). The formatted row in your data file would look like this:

```
"Free trip to A,B" ,"5.89" ,"Special rate ""1.79"""
```

Notice how that for the comma character that is part of the data, the entire data value is enclosed in double quotes. Also notice how the double quotes that are part of the data are also escaped with a double quote even though the field value is enclosed in double quotes.

Embedding the entire field inside a set of double quotes also guarantees preservation of leading and trailing whitespace characters:

```
"Free trip to A,B " ,"5.89 " ,"Special rate ""1.79"" "
```

Character Encoding

A character encoding system consists of a code that pairs each character from a given repertoire with something else, such as a sequence of numbers or octets, in order to facilitate the transmission and storage of data. The character set support in Greenplum Database allows you to store text in a variety of character sets, including single-byte character sets such as the ISO 8859 series and multiple-byte character sets such as EUC (Extended UNIX Code), UTF-8, and Mule internal code. All supported character sets can be used transparently by clients, but a few are not supported for use within the server (that is, as a server-side encoding).

Data files must be in a character encoding recognized by Greenplum Database. See the *Greenplum Database Reference Guide* for the supported character sets. Data files that contain invalid or unsupported encoding sequences will encounter errors when loading into Greenplum Database.

> **Note:** On data files generated on a Microsoft Windows operating system, try running the `dos2unix` system command to remove any Windows-only characters prior to loading into Greenplum Database.

Running Greenplum Loader

Greenplum loader is invoked by running the `gpload.py` program from a Windows command-line session. For complete command syntax and options, see *gpload.py*.

Greenplum Loader Log Files

By default, `gpload.py` creates a directory called `gpAdminLogs` in the same location from where you execute the program and writes its log files there. Alternatively, you can use the `-l` option when executing `gpload.py` to direct the log output to a different location. See *gpload.py* for the format of these log files.

Updating Database Statistics After Data Loads

After loading data, always run the `ANALYZE` SQL command to update the database statistics used by the query planner. `ANALYZE` collects statistics about the contents of tables in the database, and stores the results in the system table `pg_statistic`. The query planner uses these statistics to help determine the most efficient execution plans for queries. For example, to collect statistics on a newly loaded table, run the following on the Greenplum master host:

```
psql dbname -c 'ANALYZE mytable;'
```

Vacumming the Database After Load Errors

The Greenplum loader will stop a load operation if it encounters an error. When this happens, the target table may already have received earlier rows in the load operation. Although these rows will not be visible or accessible, they still occupy disk space. This may amount to a considerable amount of wasted disk space if the failure happened well into a large load operation. You may wish to invoke the `VACUUM` command to recover the wasted space. For example, run the following command on the master host after a load error:

```
vacummdb dbname [table_name]
```

`VACUUM` reclaims storage occupied by deleted tuples. In normal operation, tuples that are deleted or obsoleted by an update are not physically removed from their table; they remain present until a `VACUUM` is done. Therefore it's recommended to do `VACUUM` periodically, especially on frequently-updated tables.

Chapter 3

Loader Program Reference

This is a reference of the command-line loader programs. These programs 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 Greenplum Loader*.

The following loader programs are provided:

- *`gpload.py`* (loader program)
- *`gpfdist.exe`* (parallel file distribution program used by `gpload.py`)

gpload.py

Runs a load job as defined in a YAML formatted control file.

Synopsis

```
gpload.py -f control_file [-l log_file] [-h hostname] [-p port]
   [-U username] [-d database] [-W] [--gpfdist_timeout seconds]
   [--no_auto_trans] [[-v | -V] [-q]] [-D]

gpload.py -?

gpload.py --version
```

Prerequisites

The client machine where `gpload.py` is executed must have the following:

- Python 2.5.
- The *`gpfdist.exe`* parallel file distribution program installed and in your `PATH`. This program is installed with the Greenplum loaders package.
- Network access to and from all hosts in your Greenplum Database array (master and segments).
- Network access to and from the hosts where the data to be loaded resides (ETL servers).

Description

`gpload.py` is a data loading utility that acts as an interface to Greenplum Database's external table parallel loading feature. Using a load specification defined in a YAML formatted control file, `gpload.py` executes a load by invoking the Greenplum parallel file server (*`gpfdist.exe`*), creating an external table definition based on the source data defined, and executing an `INSERT`, `UPDATE` or `MERGE` operation to load the source data into the target table in the database.

The operation, including any SQL commands specified in the `SQL` collection of the YAML control file (see *Control File Format*), are performed as a single transaction to prevent inconsistent data when performing multiple, simultaneous load operations on a target table.

Options

-f *control_file*

Required. A YAML file that contains the load specification details. See *Control File Format*.

--gpfdist_timeout *seconds*

Sets the timeout for the `gpfdist` parallel file distribution program to send a response. Enter a value from `0` to `30` seconds (entering "`0`" to disables timeouts). Note that you might need to increase this value when operating on high-traffic networks.

-l *log_file*

Specifies where to write the log file. Defaults to `~/gpAdminLogs/gpload_`*`YYYYMMDD`*. For more information about the log file, see *Log File Format*.

--no_auto_trans

Specify `--no_auto_trans` to disable processing the load operation as a single transaction if you are performing a single load operation on the target table.

By default, `gpload.py` processes each load operation as a single transaction to prevent inconsistent data when performing multiple, simultaneous operations on a target table.

-q (no screen output)

Run in quiet mode. Command output is not displayed on the screen, but is still written to the log file.

-D (debug mode)

Check for error conditions, but do not execute the load.

-v (verbose mode)

Show verbose output of the load steps as they are executed.

-V (very verbose mode)

Shows very verbose output.

-? (show help)

Show help, then exit.

--version

Show the version of this utility, then exit.

Connection Options

-d *database*

The database to load into. If not specified, reads from the load control file, the environment variable `$PGDATABASE` or defaults to the current system user name.

-h *hostname*

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

-p *port*

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

-U *username*

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

-W (force password prompt)

Force a password prompt. If not specified, reads the password from the environment variable `$PGPASSWORD` or from a password file specified by `$PGPASSFILE` or in `~/.pgpass`. If these are not set, then `gpload.py` will prompt for a password even if `-W` is not supplied.

Control File Format

The `gpload.py` control file uses the *YAML 1.1* document format and then implements its own schema for defining the various steps of a Greenplum Database load operation. The control file must be a valid YAML document.

The `gpload.py` program processes the control file document in order and uses indentation (spaces) to determine the document hierarchy and the relationships of the sections to one another. The use of white space is significant. White space should not be used simply for formatting purposes, and tabs should not be used at all.

The basic structure of a load control file is:

```
---
VERSION: 1.0.0.1
DATABASE: db_name
USER: db_username
HOST: master_hostname
PORT: master_port
GPLOAD:
```

```
    INPUT:
     - SOURCE:
          LOCAL_HOSTNAME:
            - hostname_or_ip
          PORT: http_port
        | PORT_RANGE: [start_port_range, end_port_range]
          FILE:
            - /path/to/input_file
          SSL: true | false
          CERTIFICATES_PATH: /path/to/certificates
     - FULLY_QUALIFIED_DOMAIN_NAME: true | false
     - COLUMNS:
            - field_name: data_type
     - TRANSFORM: 'transformation'
     - TRANSFORM_CONFIG: 'configuration-file-path'
     - MAX_LINE_LENGTH: integer
     - FORMAT: text | csv
     - DELIMITER: 'delimiter_character'
     - ESCAPE: 'escape_character' | 'OFF'
     - NULL_AS: 'null_string'
     - FORCE_NOT_NULL: true | false
     - QUOTE: 'csv_quote_character'
     - HEADER: true | false
     - ENCODING: database_encoding
     - ERROR_LIMIT: integer
     - LOG_ERRORS: true | false
     - ERROR_TABLE: schema.table_name
    EXTERNAL:
       - SCHEMA: schema | '%'
    OUTPUT:
     - TABLE: schema.table_name
     - MODE: insert | update | merge
     - MATCH_COLUMNS:
            - target_column_name
     - UPDATE_COLUMNS:
            - target_column_name
     - UPDATE_CONDITION: 'boolean_condition'
     - MAPPING:
              target_column_name: source_column_name | 'expression'
    PRELOAD:
     - TRUNCATE: true | false
     - REUSE_TABLES: true | false
    SQL:
     - BEFORE: "sql_command"
     - AFTER: "sql_command"
```

VERSION

Optional. The version of the `gpload.py` control file schema. The current version is 1.0.0.1.

DATABASE

Optional. Specifies which database in the Greenplum Database system to connect to. If not specified, defaults to `$PGDATABASE` if set or the current system user name. You can also specify the database on the command line using the `-d` option.

USER

Optional. Specifies which database role to use to connect. If not specified, defaults to the current user or `$PGUSER` if set. You can also specify the database role on the command line using the `-U` option.

If the user running `gpload.py` is not a Greenplum Database superuser, then the server configuration parameter `gp_external_grant_privileges` must be set to `on` in order for the load to be processed. See the *Greenplum Database Reference Guide* for more information.

HOST

Optional. Specifies Greenplum Database master host name. If not specified, defaults to localhost or `$PGHOST` if set. You can also specify the master host name on the command line using the `-h` option.

PORT

Optional. Specifies Greenplum Database master port. If not specified, defaults to 5432 or `$PGPORT` if set. You can also specify the master port on the command line using the `-p` option.

GPLOAD

Required. Begins the load specification section. A `GPLOAD` specification must have an `INPUT` and an `OUTPUT` section defined.

INPUT

Required. Defines the location and the format of the input data to be loaded. `gpload.py` will start one or more instances of the *`gpfdist.exe`* file distribution program on the current host and create the required external table definition(s) in Greenplum Database that point to the source data. Note that the host from which you run `gpload.py` must be accessible over the network by all Greenplum Database hosts (master and segments).

SOURCE

Required. The `SOURCE` block of an `INPUT` specification defines the location of a source file. An `INPUT` section can have more than one `SOURCE` block defined. Each `SOURCE` block defined corresponds to one instance of the *`gpfdist.exe`* file distribution program that will be started on the local machine. Each `SOURCE` block defined must have a `FILE` specification. For more information about using the `gpfdist` parallel file server and single and multiple `gpfdist` instances, see "Loading and Unloading Data" in the *Greenplum Database Administrator Guide*.

LOCAL_HOSTNAME

Optional. Specifies the host name or IP address of the local machine on which `gpload.py` is running. If this machine is configured with multiple network interface cards (NICs), you can specify the host name or IP of each individual NIC to allow network traffic to use all NICs simultaneously. The default is to use the local machine's primary host name or IP only.

PORT

Optional. Specifies the specific port number that the *`gpfdist.exe`* file distribution program should use. You can also supply a `PORT_RANGE` to select an available port from the specified range. If both `PORT` and `PORT_RANGE` are defined, then `PORT` takes precedence. If neither `PORT` or `PORT_RANGE` are defined, the default is to select an available port between 8000 and 9000.

If multiple host names are declared in `LOCAL_HOSTNAME`, this port number is used for all hosts. This configuration is desired if you want to use all NICs to load the same file or set of files in a given directory location.

PORT_RANGE

Optional. Can be used instead of `PORT` to supply a range of port numbers from which `gpload.py` can choose an available port for this instance of the *`gpfdist.exe`* file distribution program.

FILE

Required. Specifies the location of a file, named pipe, or directory location on the local file system that contains data to be loaded. You can declare more than one file so long as the data is of the same format in all files specified.

If the files are compressed using `gzip` or `bzip2` (have a `.gz` or `.bz2` file extension), the files will be uncompressed automatically (provided that `gunzip` or `bunzip2` is in your path).

When specifying which source files to load, you can use the wildcard character (`*`) or other C-style pattern matching to denote multiple files. The files specified are assumed to be relative to the current directory from which `gpload.py` is executed (or you can declare an absolute path).

SSL

Optional. Specifies usage of SSL encryption. If `SSL` is set to true, `gpload` starts the `gpfdist` server with the `--ssl` option and uses the `gpfdists://` protocol.

CERTIFICATES_PATH

Required when SSL is `true`; cannot be specified when SSL is `false` or unspecified. The location specified in `CERTIFICATES_PATH` must contain the following files:

- The server certificate file, `server.crt`
- The server private key file, `server.key`
- The trusted certificate authorities, `root.crt`

The root directory (`/`) cannot be specified as `CERTIFICATES_PATH`.

FULLY_QUALIFIED_DOMAIN_NAME

Optional. Specifies whether `gpload` resolve hostnames to the fully qualified domain name (FQDN) or the local hostname. If the value is set to `true`, names are resolved to the FQDN. If the value is set to `false`, resolution is to the local hostname. The default is `false`.

A fully qualified domain name might be required in some situations. For example, if the Greenplum Database system is in a different domain than an ETL application that is being accessed by `gpload`.

COLUMNS

Optional. Specifies the schema of the source data file(s) in the format of *`field_name`*`:`*`data_type`*. The `DELIMITER` character in the source file is what separates two data value fields (columns). A row is determined by a line feed character (`0x0a`).

If the input `COLUMNS` are not specified, then the schema of the output `TABLE` is implied, meaning that the source data must have the same column order, number of columns, and data format as the target table.

The default source-to-target mapping is based on a match of column names as defined in this section and the column names in the target `TABLE`. This default mapping can be overridden using the `MAPPING` section.

TRANSFORM

Optional. Specifies the name of the input XML transformation passed to `gploadl.py`. For more information about XML transformations, see "Loading and Unloading Data" in the *Greenplum Database Administrator Guide*.

TRANSFORM_CONFIG

Optional. Specifies the location of the XML transformation configuration file that is specified in the `TRANSFORM` parameter, above.

MAX_LINE_LENGTH

Optional. An integer that specifies the maximum length of a line in the XML transformation data passed to `gpload.py`.

FORMAT

Optional. Specifies the format of the source data file(s) - either plain text (`TEXT`) or comma separated values (`CSV`) format. Defaults to `TEXT` if not specified. For more information about the format of the source data, see "Loading and Unloading Data" in the *Greenplum Database Administrator Guide*.

DELIMITER

Optional. Specifies a single ASCII character that separates columns within each row (line) of data. The default is a tab character in TEXT mode, a comma in CSV mode. You can also specify a non- printable ASCII character or a non-printable unicode character, for example: `"\x1B"` or `"\u001B"`. The escape string syntax, `E'`*`character-code`*`'`, is also supported for non-printable characters. The ASCII or unicode character must be enclosed in single quotes. For example: `E'\x1B'` or `E'\u001B'`.

ESCAPE

Specifies the single character that is used for C escape sequences (such as `\n`, `\t`, `\100`, and so on) and for escaping data characters that might otherwise be taken as row or column delimiters. Make sure to choose an escape character that is not used anywhere in your actual column data. The default escape character is a \ (backslash) for text-formatted files and a `"` (double quote) for csv-formatted files, however it is possible to specify another character to represent an escape. It is also possible to disable escaping in text-formatted files by specifying the value `'OFF'` as the escape value. This is very useful for data such as text-formatted web log data that has many embedded backslashes that are not intended to be escapes.

NULL_AS

Optional. Specifies the string that represents a null value. The default is `\N` (backslash-N) in `TEXT` mode, and an empty value with no quotations in `CSV` mode. You might prefer an empty string even in `TEXT` mode for cases where you do not want to distinguish nulls from empty strings. Any source data item that matches this string will be considered a null value.

FORCE_NOT_NULL

Optional. In CSV mode, processes each specified column as though it were quoted and hence not a NULL value. For the default null string in CSV mode (nothing between two delimiters), this causes missing values to be evaluated as zero-length strings.

QUOTE

Required when `FORMAT` is `CSV`. Specifies the quotation character for `CSV` mode. The default is double-quote (`"`).

HEADER

Optional. Specifies that the first line in the data file(s) is a header row (contains the names of the columns) and should not be included as data to be loaded. If using multiple data source files, all files must have a header row. The default is to assume that the input files do not have a header row.

ENCODING

Optional. Character set encoding of the source data. Specify a string constant (such as `'SQL_ASCII'`), an integer encoding number, or `'DEFAULT'` to use the default client encoding. If not specified, the default client encoding

is used. For information about supported character sets, see the *Greenplum Database Reference Guide*.

ERROR_LIMIT

Optional. Enables single row error isolation mode for this load operation. When enabled, input rows that have format errors will be discarded provided that the error limit count is not reached on any Greenplum Database segment instance during input processing. If the error limit is not reached, all good rows will be loaded and any error rows will either be discarded or logged to the table specified in `ERROR_TABLE`. The default is to abort the load operation on the first error encountered. Note that single row error isolation only applies to data rows with format errors; for example, extra or missing attributes, attributes of a wrong data type, or invalid client encoding sequences. Constraint errors, such as primary key violations, will still cause the load operation to abort if encountered. For information about handling load errors, see "Loading and Unloading Data" in the *Greenplum Database Administrator Guide.*

LOG_ERRORS

Optional when `ERROR_LIMIT` is declared. Value is either `true` or `false`. The default value is `false`. If the value is `true`, rows with formatting errors are logged internally when running in single row error isolation mode. You can examine formatting errors with the Greenplum Database built-in SQL function `gp_read_error_log('table_name')`. If formatting errors are detected when loading data, `gpload.py` generates a warning message with the name of the table that contains the error information similar to this message.

```
timestamp|WARN|1 bad row, please use GPDB built-in function
 gp_read_error_log('table-name')
   to access the detailed error row
```

If `LOG_ERRORS: true` is specified, `REUSE TABLES: true` must be specified to retain the formatting errors in Greenplum Database error logs. If `REUSE TABLES: true` is not specified, the error information is deleted after the `gpload` operation. You can delete the formatting errors from the error logs with the Greenplum Database function `gp_truncate_error_log()`.

Only `LOG_ERRORS` or `ERROR_TABLE` can be specified. If both are specified, an error message is returned.

For more information about handling load errors, see "Loading and Unloading Data" in the *Greenplum Database Administrator Guide*. For information about the `gp_read_error_log()` function, see the `CREATE EXTERNAL TABLE` command in the *Greenplum Database Reference Guide*

ERROR_TABLE

Optional when `ERROR_LIMIT` is declared. Specifies an error table where rows with formatting errors will be logged when running in single row error isolation mode. You can then examine this error table to see error rows that were not loaded (if any). If the `ERROR_TABLE` specified already exists, it will be used. If it does not exist, it will be automatically generated.

Only `LOG_ERRORS` or `ERROR_TABLE` can be specified. If both are specified, an error message is returned. Pivotal recommends using `LOG_ERRORS` to capture formatting errors.

For more information about handling load errors, see "Loading and Unloading Data" in the *Greenplum Database Administrator Guide*.

EXTERNAL

Optional. Defines the schema of the external table database objects created by `gpload.py`.

The default is to use the Greenplum Database `search_path`.

SCHEMA

Required when `EXTERNAL` is declared. The name of the schema of the external table. If the schema does not exist, an error is returned.

If `%` (percent character) is specified, the schema of the table name specified by `TABLE` in the `OUTPUT` section is used. If the table name does not specify a schema, the default schema is used.

OUTPUT

Required. Defines the target table and final data column values that are to be loaded into the database.

TABLE

Required. The name of the target table to load into.

MODE

Optional. Defaults to `INSERT` if not specified. There are three available load modes:

INSERT - Loads data into the target table using the following method:

```
INSERT INTO target_table SELECT * FROM input_data;
```

UPDATE - Updates the `UPDATE_COLUMNS` of the target table where the rows have `MATCH_COLUMNS` attribute values equal to those of the input data, and the optional `UPDATE_CONDITION` is true.

MERGE - Inserts new rows and updates the `UPDATE_COLUMNS` of existing rows where `FOOBAR` attribute values are equal to those of the input data, and the optional `MATCH_COLUMNS` is true. New rows are identified when the `MATCH_COLUMNS` value in the source data does not have a corresponding value in the existing data of the target table. In those cases, the **entire row** from the source file is inserted, not only the `MATCH` and `UPDATE` columns. If there are multiple new `MATCH_COLUMNS` values that are the same, only one new row for that value will be inserted. Use `UPDATE_CONDITION` to filter out the rows to discard.

MATCH_COLUMNS

Required if `MODE` is `UPDATE` or `MERGE`. Specifies the column(s) to use as the join condition for the update. The attribute value in the specified target column(s) must be equal to that of the corresponding source data column(s) in order for the row to be updated in the target table.

UPDATE_COLUMNS

Required if `MODE` is `UPDATE` or `MERGE`. Specifies the column(s) to update for the rows that meet the `MATCH_COLUMNS` criteria and the optional `UPDATE_CONDITION`.

UPDATE_CONDITION

Optional. Specifies a Boolean condition (similar to what you would declare in a `WHERE` clause) that must be met in order for a row in the target table to be updated (or inserted in the case of a `MERGE`).

MAPPING

Optional. If a mapping is specified, it overrides the default source-to-target column mapping. The default source-to-target mapping is based on a match

of column names as defined in the source `COLUMNS` section and the column names of the target `TABLE`. A mapping is specified as either:

target_column_name: *source_column_name*

or

target_column_name: '*expression*'

Where *expression* is any expression that you would specify in the `SELECT` list of a query, such as a constant value, a column reference, an operator invocation, a function call, and so on.

PRELOAD

Optional. Specifies operations to run prior to the load operation. Right now the only preload operation is `TRUNCATE`.

TRUNCATE

Optional. If set to true, `gpload.py` will remove all rows in the target table prior to loading it.

REUSE_TABLES

Optional. If set to true, `gpload.py` will not drop the external table objects and staging table objects it creates. These objects will be reused for future load operations that use the same load specifications. This improves performance of trickle loads (ongoing small loads to the same target table).

If `LOG_ERRORS: true` is specified, `REUSE TABLES: true` must be specified to retain the formatting errors in Greenplum Database error logs. If `REUSE TABLES: true` is not specified, formatting error information is deleted after the `gpload` operation.

SQL

Optional. Defines SQL commands to run before and/or after the load operation. You can specify multiple `BEFORE` and/or `AFTER` commands. List commands in the order of desired execution.

BEFORE

Optional. An SQL command to run before the load operation starts. Enclose commands in quotes.

AFTER

Optional. An SQL command to run after the load operation completes. Enclose commands in quotes.

Log File Format

Log files output by `gpload.py` have the following format:

```
timestamp|level|message
```

Where *timestamp* takes the form: `YYYY-MM-DD HH:MM:SS`, *level* is one of `DEBUG`, `LOG`, `INFO`, `ERROR`, and *message* is a normal text message.

Some `INFO` messages that may be of interest in the log files are (where # corresponds to the actual number of seconds, units of data, or failed rows):

```
INFO|running time: #.## seconds
INFO|transferred #.# kB of #.# kB.
INFO|gpload succeeded
INFO|gpload succeeded with warnings
INFO|gpload failed
INFO|1 bad row
INFO|# bad rows
```

Notes

If your database object names were created using a double-quoted identifier (delimited identifier), you must specify the delimited name within single quotes in the `gpload.py` control file. For example, if you create a table as follows:

```
CREATE TABLE "MyTable" ("MyColumn" text);
```

Your YAML-formatted `gpload.py` control file would refer to the above table and column names as follows:

```
- COLUMNS:
   - '"MyColumn"': text
OUTPUT:
   - TABLE: public.'"MyTable"'
```

Examples

Run a load job as defined in `my_load.yml`:

```
gpload.py -f my_load.yml
```

Example load control file:

```
---
VERSION: 1.0.0.1
DATABASE: ops
USER: gpadmin
HOST: mdw-1
PORT: 5432
GPLOAD:
   INPUT:
    - SOURCE:
         LOCAL_HOSTNAME:
           - etl1-1
           - etl1-2
           - etl1-3
           - etl1-4
         PORT: 8081
         FILE:
           - /var/load/data/*
    - COLUMNS:
           - name: text
           - amount: float4
           - category: text
           - desc: text
           - date: date
    - FORMAT: text
    - DELIMITER: '|'
    - ERROR_LIMIT: 25
    - ERROR_TABLE: payables.err_expenses
   OUTPUT:
    - TABLE: payables.expenses
    - MODE: INSERT
   SQL:
   - BEFORE: "INSERT INTO audit VALUES('start',
current_timestamp)"
   - AFTER: "INSERT INTO audit VALUES('end',
current_timestamp)"
```

See Also

`gpfdist.exe`, `CREATE EXTERNAL TABLE` in the *Greenplum Database Reference Guide*

gpfdist.exe

Serves data files to or writes data files out from Greenplum Database segments.

Synopsis

```
gpfdist.exe [-d directory] [-p http_port] [-l log_file] [-t timeout]
   [-S] [-w time] [-v | -V] [-s] [-m max_length] [--ssl certificate_path]

gpfdist.exe -? | --help

gpfdist.exe --version
```

Description

`gpfdist.exe` is Greenplum's parallel file distribution program. It is used by readable external tables and `gpload.py` to serve external table files to all Greenplum Database segments in parallel. It is used by writable external tables to accept output streams from Greenplum Database segments in parallel and write them out to a file.

In order for `gpfdist.exe` to be used by an external table, the `LOCATION` clause of the external table definition must specify the external table data using the `gpfdist://` protocol (see the Greenplum Database command `CREATE EXTERNAL TABLE`).

> **Note:** If the `--ssl` option is specified to enable SSL security, create the external table with the `gpfdists://` protocol.

The benefit of using `gpfdist.exe` is that you are guaranteed maximum parallelism while reading from or writing to external tables, thereby offering the best performance as well as easier administration of external tables.

For readable external tables, `gpfdist.exe` parses and serves data files evenly to all the segment instances in the Greenplum Database system when users `SELECT` from the external table. For writable external tables, `gpfdist.exe` accepts parallel output streams from the segments when users `INSERT` into the external table, and writes to an output file.

For readable external tables, if load files are compressed using `gzip` or `bzip2` (have a `.gz` or `.bz2` file extension), `gpfdist.exe` uncompresses the files automatically before loading provided that `gunzip` or `bunzip2` is in your path.

> **Note:** Currently, readable external tables do not support compression on Windows platforms, and writable external tables do not support compression on any platforms.

Most likely, you will want to run `gpfdist.exe` on your ETL machines rather than the hosts where Greenplum Database is installed. To install `gpfdist.exe` on another host, simply copy the utility over to that host and add `gpfdist.exe` to your `PATH`.

> **Note:** When using IPv6, always enclose the numeric IP address in brackets.

You can also run `gpfdist.exe` as a Windows Service. See *Running gpfdist as a Windows Service* for more details.

Options

-d *directory*

The directory from which `gpfdist.exe` will serve files for readable external tables or create output files for writable external tables. If not specified, defaults to the current directory.

-l *log_file*

The fully qualified path and log file name where standard output messages are to be logged.

-p *http_port*

The HTTP port on which `gpfdist.exe` will serve files. Defaults to 8080.

-t *timeout*

Sets the time allowed for Greenplum Database to establish a connection to a `gpfdist.exe` process. Default is 5 seconds. Allowed values are 2 to 7200 seconds (2 hours). May need to be increased on systems with a lot of network traffic.

-m *max_length*

Sets the maximum allowed data row length in bytes. Default is 32768. Should be used when user data includes very wide rows (or when `line too long` error message occurs). Should not be used otherwise as it increases resource allocation. Valid range is 32K to 256MB. (The upper limit is 1MB on Windows systems.)

> **Note:** Memory issues might occur if you specify a large maximum row length and run a large number of `gpfidst` concurrent connections. For example, setting this value to the maximum of 1MB with 96 concurrent `gpfdist` processes requires approximately 97GB of memory (`(96 + 1) x 1MB`).

-s

Enables simplified logging. When this option is specified, only messages with `WARN` level and higher are written to the `gpfdist` log file. `INFO` level messages are not written to the log file. If this option is not specified, all `gpfdist` messages are written to the log file.

You can specify this option to reduce the information written to the log file.

-S (use O_SYNC)

Opens the file for synchronous I/O with the `O_SYNC` flag. Any writes to the resulting file descriptor block `gpfdist.exe` until the data is physically written to the underlying hardware.

-w *time*

Sets the number of seconds that Greenplum Database delays before closing a target file such as a named pipe. The default value is 0, no delay. The maximum value is 7200 seconds (2 hours).

For a Greenplum Database with multiple segments, there might be a delay between segments when writing data from different segments to the file. You can specify a time to wait before Greenplum Database closes the file to ensure all the data is written to the file.

--ssl *certificate_path*

Adds SSL encryption to data transferred with `gpfdist.exe`. After executing `gpfdist.exe` with the `--ssl` *`certificate_path`* option, the only way to load data from this file server is with the `gpfdist://` protocol. For information on the `gpfdist://` protocol, see "Loading and Unloading Data" in the *Greenplum Database Administrator Guide*.

The location specified in *certificate_path* must contain the following files:

- The server certificate file, `server.crt`
- The server private key file, `server.key`
- The trusted certificate authorities, `root.crt`

The root directory (`/`) cannot be specified as *certificate_path*.

-v (verbose)

Verbose mode shows progress and status messages.

-V (very verbose)

Verbose mode shows all output messages generated by this utility.

-? (help)

Displays the online help.

--version

Displays the version of this utility.

Running gpfdist as a Windows Service

Greenplum Loaders allow `gpfdist.exe` to run as a Windows Service.

Follow the instructions below to download, register and activate `gpfdist.exe` as a service:

1. Update your Greenplum Loader package to the latest version. This package is available from *Pivotal Network*.
2. Register `gpfdist` as a Windows service:
   a. Open a Windows command window
   b. Run the following command:

   ```
   sc create gpfdist binpath= "path_to_gpfdist.exe -p 8081 -d External\load\files
   \path -l Log\file\path"
   ```

   You can create multiple instances of `gpfdist` by running the same command again, with a unique name and port number for each instance:

   ```
   sc create gpfdistN binpath= "path_to_gpfdist.exe -p 8082 -d External\load\files
   \path -l Log\file\path"
   ```
3. Activate the `gpfdist` service:
   a. Open the Windows Control Panel and select **Administrative Tools > Services**.
   b. Highlight then right-click on the `gpfdist` service in the list of services.
   c. Select **Properties** from the right-click menu, the Service Properties window opens.

   Note that you can also stop this service from the Service Properties window.
   d. Optional: Change the **Startup Type** to **Automatic** (after a system restart, this service will be running), then under **Service** status, click **Start**.
   e. Click **OK**.

Repeat the above steps for each instance of `gpfdist` that you created.

Examples

To serve files from a specified directory using port 8081 (and start `gpfdist.exe` in the background):

```
gpfdist.exe -d /var/load_files -p 8081 &
```

To start `gpfdist.exe` in the background and redirect output and errors to a log file:

```
gpfdist.exe -d /var/load_files -p 8081 -l /home/gpadmin/log &
```

See Also

gpload.py, `CREATE EXTERNAL TABLE` in the *Greenplum Database Reference Guide*

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 [, ...]] ]
      [IGNORE EXTERNAL PARTITIONS ]
```

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')
        | ('s3://S3_endpoint/bucket_name/[S3_prefix]
            [config=config_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] )]
           | 'AVRO'
           | 'PARQUET'

           | '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'] )]
           | 'AVRO'
           | 'PARQUET'

           | 'CUSTOM' (Formatter=<formatter specifications>)
    [ ENCODING 'write_encoding' ]
    [ DISTRIBUTED BY (column, [ ... ] ) | DISTRIBUTED RANDOMLY ]

CREATE WRITABLE EXTERNAL TABLE table_name
    ( column_name data_type [, ...] | LIKE other_table )
     LOCATION('s3://S3_endpoint/bucket_name/[S3_prefix]
            [config=config_file]')
      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'] )]

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 | 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]
```