Chapter 4. Using SQL with PostgreSQL

Table of Contents

In this chapter we continue to discuss SQL, this time with a practical focus. We'll address creating tables, populating tables with data, and managing that data via SQL statements.

Like most network-capable database systems, PostgreSQL fits into a client-server paradigm. The heart of PostgreSQL is the server backend, or the *postmaster* process. It is called a "backend" because it is not meant to directly interface with a user; rather, it can be connected to with a variety of clients.

When you start the PostgreSQL service, the *postmaster* process starts running in the background, listening to a specific TCP/IP port for connections from clients. Unless explicitly configured, *postmaster* will bind to, and listen on, port 5432.

There are several interfaces available through which clients may connect to the *postmaster* process. The examples in this book use *psql*, the most portable and readily accessible client distributed with PostgreSQL.

This chapter covers *psql* basics, how to create and use tables, and how to retrieve and manage data within those tables. It also addresses SQL sub-queries and views.

Introduction to psql

The *psql* client is a command-line client distributed with PostgreSQL. It is often called the *interactive monitor* or *interactive terminal*. With *psql*, you get a simple yet powerful tool with which you can directly interface with the PostgreSQL server, and thereby begin exploring SQL.

Starting psql

Before starting *psql*, be sure that you have either copied the *psql* binary into a path in your system PATH variable (e.g., */usr/bin*), or that you have placed the PostgreSQL binary path (e.g., */usr/local/pgsql/bin*) within your list of paths in your PATH environment variable (as shown in Chapter 2).

How you set the appropriate PATH variable will depend on your system shell. An example in either bash or ksh might read:

```
$ export PATH=$PATH:/usr/local/pgsql/bin
```

An example in either csh or tcsh might read:

```
$ set path=($path /usr/local/pgsql/bin)
```

Example 4-1. Setting system path for psql

```
[user@host user]$ psql
bash: psql: command not found
[user@host user]$ echo $PATH
/bin:/usr/bin:/usr/local/bin:/usr/bin/X11:/usr/X11R6/bin
[user@host user]$ export PATH=$PATH:/usr/local/pgsql/bin
[user@host user]$ psql testdb
Welcome to psql, the PostgreSQL interactive terminal.

Type:  \copyright for distribution terms
       \h for help with SQL commands
       \? for help on internal slash commands
       \g or terminate with semicolon to execute query
       \q to quit

testdb=#
```

Note that Example 4-1 takes place within a bash shell.

Once you have appropriately set your `PATH` variable, you should be able to type *psql*, along with a database name, to start up the PostgreSQL interactive terminal.

Warning
Shell environment variables are erased after you have logged out. If you wish for your changes to the `PATH` variable to be retained upon logging in, you need to enter the appropriate `PATH` declaration into your shell-specific start-up scripts (e.g., *~/.bash_profile*).

Introduction to psql Syntax

Upon starting *psql*, you are greeted with a brief synopsis of four essential *psql slash commands* : `\h` for SQL help, `\?` for help on *psql*-specific commands, `\g` for executing queries and `\q` for actually exiting *psql* once you are done.

Every *psql*-specific command is prefixed by a backslash; hence the term "slash command" used earlier. For a complete list of slash commands and a brief description their functions, type `\?` into the *psql* command line, and press enter.

Example 4-2. Listing psql slash commands

```
booktown=# \?
 \a             toggle between unaligned and aligned mode
 \c[onnect] [dbname|- [user]]
                connect to new database (currently 'booktown')
 \C <title>     table title
 \copy ...      perform SQL COPY with data stream to the client machine
 \copyright     show PostgreSQL usage and distribution terms
 \d <table>     describe table (or view, index, sequence)
 \d{t|i|s|v}    list tables/indices/sequences/views
 \d{p|S|l}      list permissions/system tables/lobjects
 \da            list aggregates
 \dd [object]   list comment for table, type, function, or operator
 \df            list functions
 \do            list operators
```

```
\dT            list data types
\e [file]      edit the current query buffer or [file] with external editor
\echo <text>   write text to stdout
\encoding <encoding>  set client encoding
\f <sep>       change field separator
\g [file]      send query to backend (and results in [file] or |pipe)
\h [cmd]       help on syntax of sql commands, * for all commands
\H             toggle HTML mode (currently off)
\i <file>      read and execute queries from <file>
\l             list all databases
\lo_export, \lo_import, \lo_list, \lo_unlink
               large object operations
\o [file]      send all query results to [file], or |pipe
\p             show the content of the current query buffer
\pset <opt>    set table output  <opt> = {format|border|expanded|fieldsep|
               null|recordsep|tuples_only|title|tableattr|pager}
\q             quit psql
\qecho <text>  write text to query output stream (see \o)
\r             reset (clear) the query buffer
\s [file]      print history or save it in [file]
\set <var> <value>  set internal variable
\t             show only rows (currently off)
\T <tags>      HTML table tags
\unset <var>   unset (delete) internal variable
\w <file>      write current query buffer to a <file>
\x             toggle expanded output (currently off)
\z             list table access permissions
\! [cmd]       shell escape or command
```

Executing Queries

Entering and executing queries within *psql* can be done two different ways. When using the client in interactive mode, the normal method is to directly enter queries into the prompt (i.e., standard input, or `stdin`). However, through the use of *psql*'s `\i` slash command, you can have *psql* read and interpret a file on your local filesystem as the query data.

Entering queries at the psql prompt

To enter queries directly into the prompt, open *psql* and make sure you are connected to the correct database (and logged in as the correct user). You will be presented with a prompt that, by default, is set to display the name of the database you are currently connected to. The prompt will look like this: *psql*:

```
testdb=#
```

To pass SQL statements to PostgreSQL, simply type them into the prompt. Anything you type (barring a slash command) will be queued until you terminate the query with a semicolon. This is the case even if you start a new line of type, thus allowing you to spread query statements across multiple lines. Examine Example 4-3 to see how this is done.

Example 4-3. Entering statements into psql

```
testdb=# SELECT * FROM employees
testdb-#          WHERE firstname = 'Michael';
```

The query entered in Example 4-3 will return a table that consists of all employees whose first name is Michael. The query could be broken up over multiple lines to improve readability, and *psql* would not send it to the backend until the terminating semicolon was sent. The prompt will show the end-character of a previous line if the character requires a closing character, such as a parenthesis or a quote (this is not

shown in the example). If you were to issue a `CREATE TABLE` command to start a statement, and then hit enter to begin a new line for readability purposes, you would see a prompt similar to the one displayed in Example 4-4.

Example 4-4. Leaving end-characters open

```
testdb=# CREATE TABLE employees (
testdb(#
```

At this point you could continue the statement. The *psql* prompt is informing you of the open parenthesis by inserting an open parenthesis symbol into the prompt.

Editing the query buffer

Use the `\e` command to edit the current query buffer with the editor that your `EDITOR` environment variable is set to. Doing so can be very useful when entering queries and statements in *psql*, as you can easily view and modify all lines of your query or statement before it is committed. Example 4-5 shows how to set the `EDITOR` variable. The *vi* editor will be used if `EDITOR` is not set.

Example 4-5. Setting the EDITOR variable

```
$ set EDITOR='joe'
$ export EDITOR
```

You can also use this command to save your current buffer as a file. Issue the `\e` command to enter editing mode. This will open your editor and load the buffer as if it were a file. Complete whatever work you wish to do with the buffer, then use your editor's save function to save the buffer and return to *psql*. To save the query as a normal file, use your editor's save-as function and save it as a file other than the *.tmp* created by `\e`.
