

Lab 1: First Steps

In this lab, we will cover the following topics:

- Start PostgreSQL server
- Connecting to the PostgreSQL server
- Using the pgAdmin GUI tool
- Using the `psql` query and scripting tool
- Changing your password securely

Start PostgreSQL server

Run the following command in the terminal to Start PostgreSQL server: `service postgresql start`

You can check the status of PostgreSQL server by running the following command: `service postgresql status`

```
root@625baed03682:~# service postgresql start
* Starting PostgreSQL 14 database server
root@625baed03682:~# service postgresql status
14/main (port 5432): online
root@625baed03682:~#
```

Connecting to the PostgreSQL server

You need to specify the following parameters to connect to PostgreSQL:

- A host or host address
- A port
- A database name
- A user
- A password (or other means of authentication, if any)

To connect, there must be a PostgreSQL server running on `host`, listening to port number `port`. On that server, a database named `dbname` and a user named `user` must also exist. The host must explicitly allow connections from your client and you must also pass authentication using the method the server specifies -- for example, specifying a password won't work if the server has requested a different form of authentication.

Almost all PostgreSQL interfaces use the `libpq` interface library. When using `libpq`, most of the connection parameter handling is identical, so we can discuss that just once.

If you don't specify the preceding parameters, PostgreSQL looks for values set through environment variables, which are as follows:

- `PGHOST` or `PGHOSTADDR`
- `PGPORT` (set this to `5432` if it is not set already)
- `PGDATABASE`
- `PGUSER`
- `PGPASSWORD` (this is definitely not recommended)

If you somehow specify the first four parameters but not the password, PostgreSQL looks for a password file, discussed in the *Avoiding hardcoding your password* topic.

Some PostgreSQL interfaces use the client-server protocol directly, so the ways in which the defaults are handled may differ. The information we need to supply won't vary significantly, so check the exact syntax for that interface.

Connection details can also be specified using a **Uniform Resource Identifier (URI)** format, as follows:

```
psql postgresql://postgres:postgres@localhost:5432/postgres
```

This specifies that we will connect the `psql` client application to the PostgreSQL server at the `localhost` host, on port `5432`, with the `postgres` database name, `postgres` user, and `postgres` password.

Note

If you do not set `postgres` in the preceding URI, you will be prompted to enter the password.

There's more...

If you are already connected to a database server with `psql` and you want to confirm that you've connected to the right place and in the right way, you can execute some, or all, of the following commands. Here is the command that shows the `current_database`:

```
SELECT current_database();
```

The following command shows the `current_user` ID:

```
SELECT current_user;
```

The next command shows the IP address and port of the current connection, unless you are using Unix sockets, in which case both values are `NULL`:

```
SELECT inet_server_addr(), inet_server_port();
```

A user's password is not accessible using general SQL, for obvious reasons.

You may also need the following:

```
SELECT version();
```

From PostgreSQL version 9.1 onward, you can also use the new `psql` meta-command, `\conninfo`. This displays most of the preceding information in a single line:

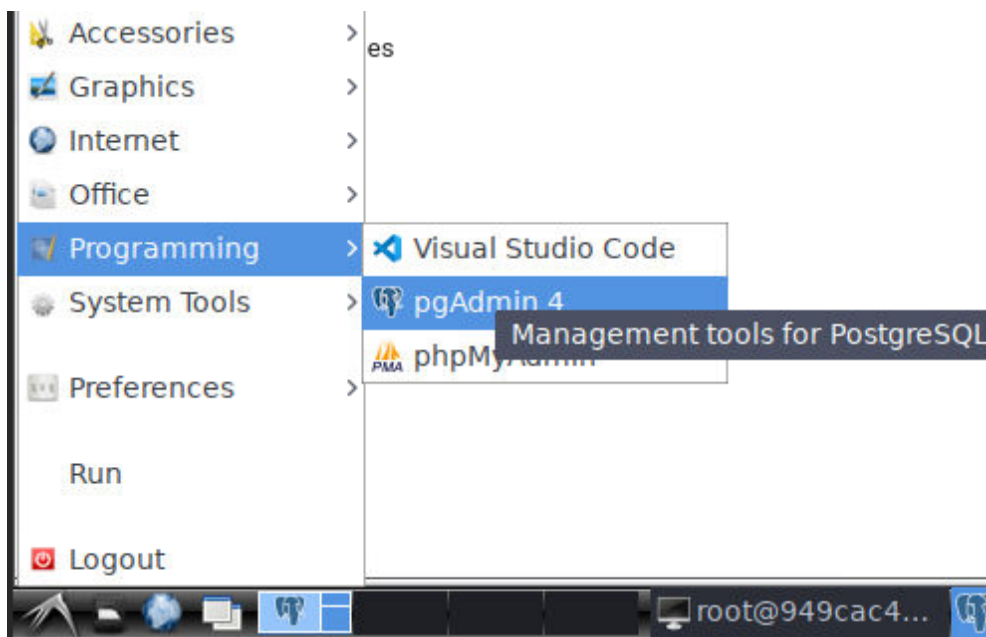
```
postgres=# \conninfo
You are connected to database postgres, as user postgres, via socket in
/var/run/postgresql, at port 5432.
```

Using the pgAdmin4 GUI tool

pgAdmin4 is a client application that sends and receives SQL to and from PostgreSQL, displaying the results for you. The admin client can access many databases servers, allowing you to manage a fleet of servers. The tool works in both standalone app mode and within web browsers.

Launch pgAdmin4

Launch pgAdmin4 desktop application from the menu as show below:



When you start pgAdmin, you will be prompted to register a new server.

Give your server a name on the **General** tab, and then click **Connection** and fill in the five basic connection parameters, as well as the other information. You should uncheck the **Save password?** box:

Note: Enter value `postgres` in username, password and database fields:

The screenshot shows a 'Register - Server' dialog box with the 'Connection' tab selected. The dialog has a title bar with a maximize icon and a close button. Below the title bar are five tabs: 'General', 'Connection', 'SSL', 'SSH Tunnel', and 'Advanced'. The 'Connection' tab contains the following fields and controls:

- Host name/addresses:** A text input field containing 'localhost'.
- Port:** A text input field containing '5432'.
- Maintenance database:** A text input field containing 'postgres'.
- Username:** A text input field containing 'postgres'.
- Kerberos authentication?** A toggle switch that is currently turned off.
- Password:** A password input field with masked characters (dots).
- Save password?** A toggle switch that is currently turned off.
- Role:** An empty text input field.
- Service:** An empty text input field.

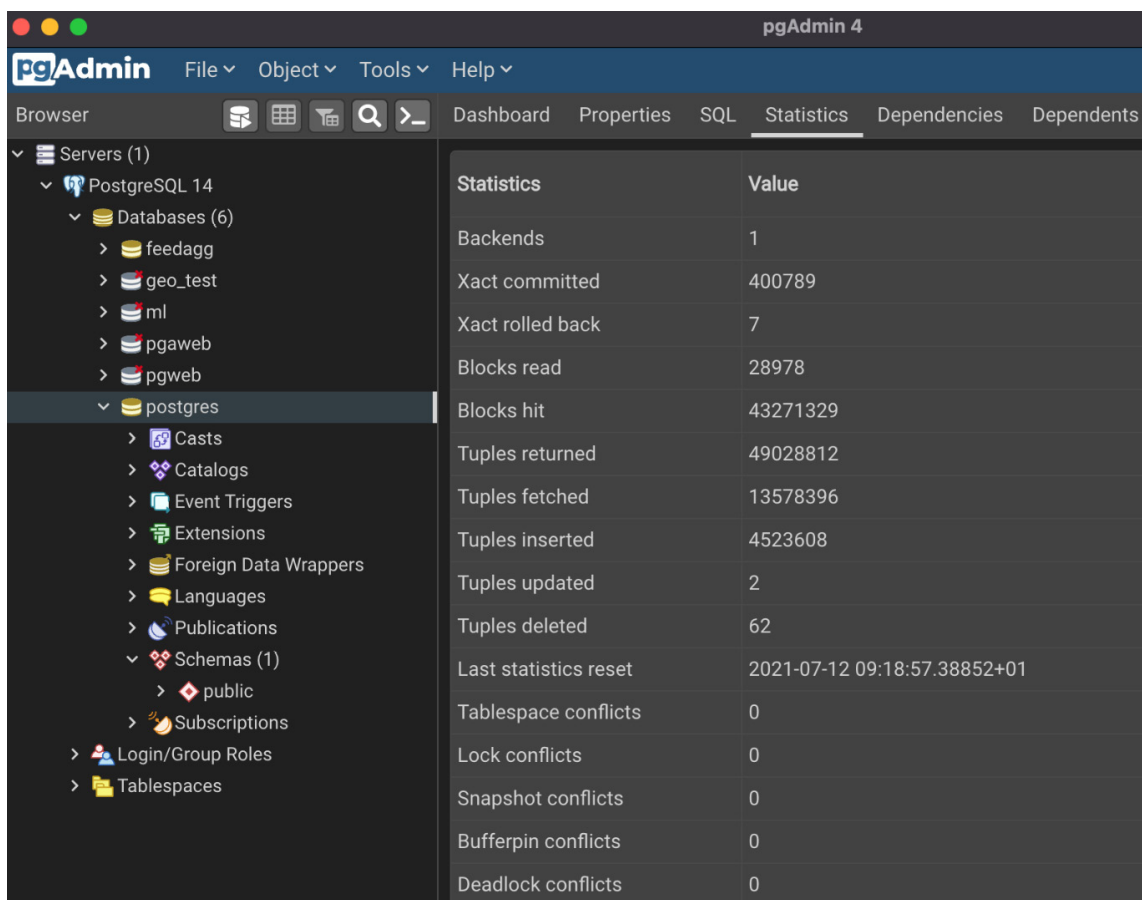
At the bottom of the dialog, there are three buttons: 'Close' (with a close icon), 'Reset' (with a circular arrow icon), and 'Save' (with a save icon). To the left of these buttons are two small icons: an information icon (i) and a help icon (?).

If you have many database servers, you can group them together. I suggest keeping any replicated servers together in the same server group. Give each server a sensible name.

Once you've added a server, you can connect to it and display information about it.

The default screen is **Dashboard**, which presents a few interesting graphs based on the data it polls from the server. That's not very useful, so click on the **Statistics** tab.

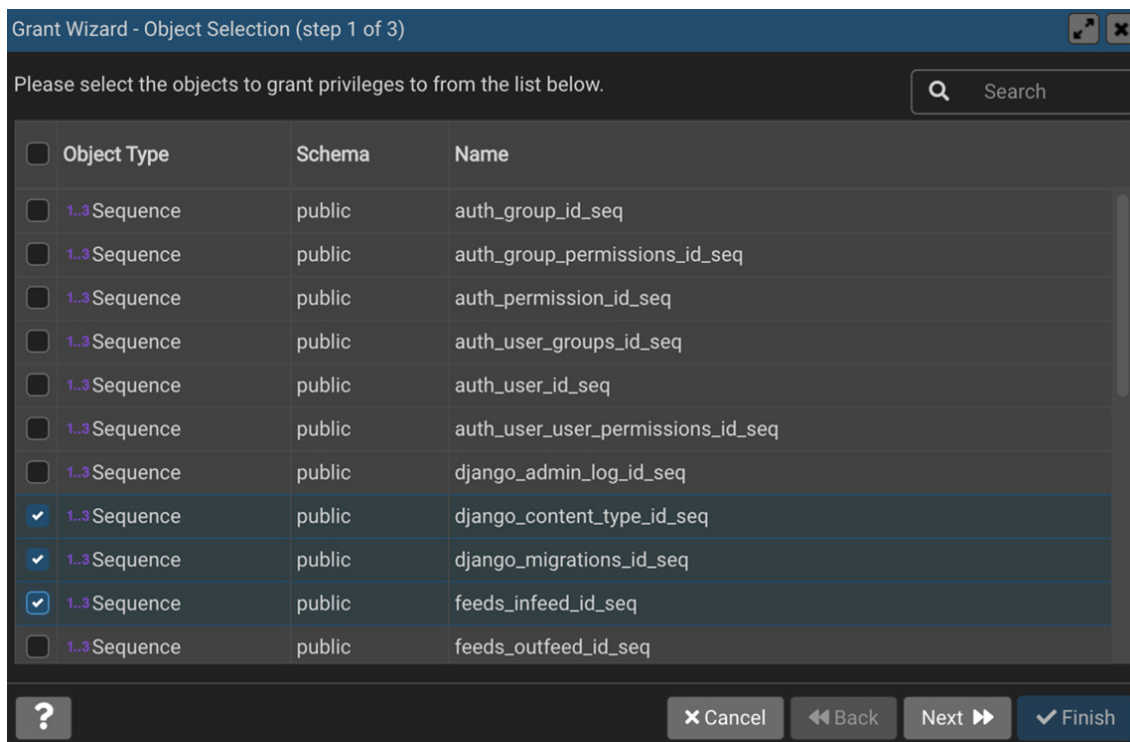
You will then get access to the main browser screen, with the object tree view on the left and statistics on the right, as shown in the following screenshot:



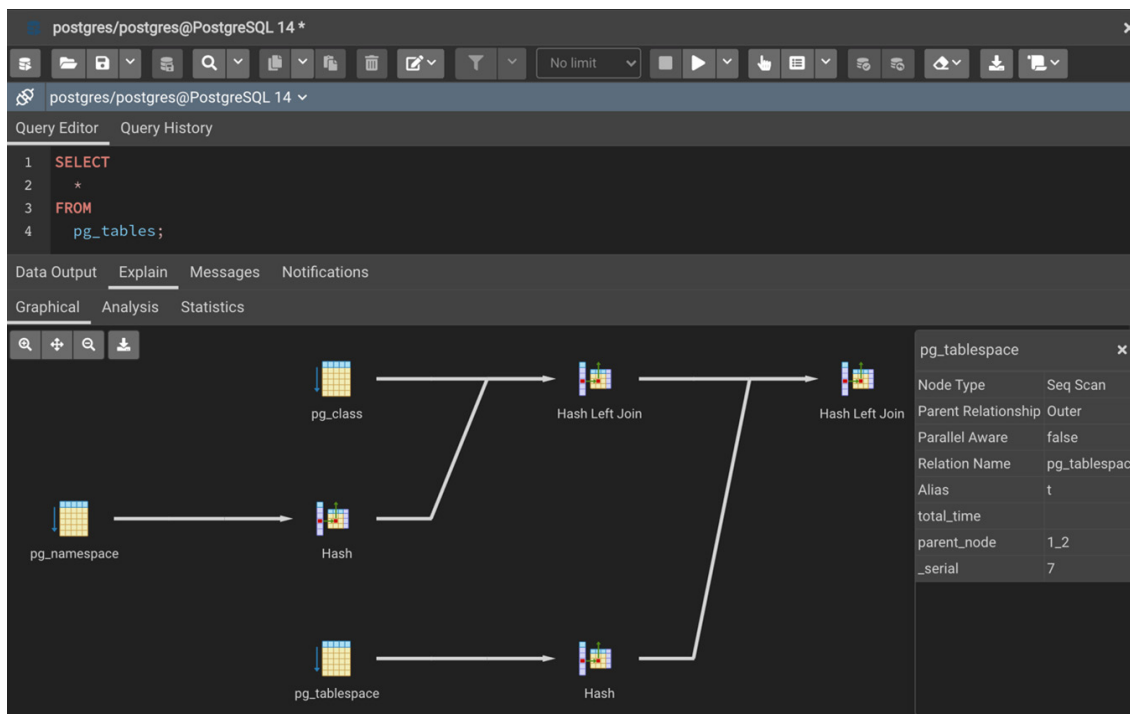
Statistics	Value
Backends	1
Xact committed	400789
Xact rolled back	7
Blocks read	28978
Blocks hit	43271329
Tuples returned	49028812
Tuples fetched	13578396
Tuples inserted	4523608
Tuples updated	2
Tuples deleted	62
Last statistics reset	2021-07-12 09:18:57.38852+01
Tablespace conflicts	0
Lock conflicts	0
Snapshot conflicts	0
Bufferpin conflicts	0
Deadlock conflicts	0

pgAdmin easily displays much of the data that is available from PostgreSQL. The information is context-sensitive, allowing you to navigate and see everything quickly and easily. The information is not dynamically updated; this will occur only when you click to refresh, so bear this in mind when using the application.

pgAdmin also provides **Grant Wizard**. This is useful for DBAs for review and immediate maintenance:



The pgAdmin query tool allows you to have multiple active sessions. The query tool has a good-looking visual **Explain** feature, which displays the `EXPLAIN` plan for your query:



Using the psql query and scripting tool

`psql` is the query tool supplied as a part of the core distribution of PostgreSQL, so it is available in all environments and works similarly in all of them. This makes it an ideal choice for developing portable applications and techniques.

`psql` provides features for use as both an interactive query tool and as a scripting tool.

Getting ready

From here on, we will assume that the `psql` command is enough to allow you access to the PostgreSQL server. This assumes that all your connection parameters are defaults, which may not be true.

Written in full, the connection parameters will be either of these options:

```
psql -h localhost -p 5432 -d postgres -U postgres
psql postgresql://postgres@localhost:5432/postgres
```

How to do it...

The command that executes a single SQL command and prints the output is the easiest, as shown here:

Note: Open new terminal and run following commands.

```
$ su - postgres
$ psql -c "SELECT current_time"
      timetz
-----
18:48:32.484+01
(1 row)
```

The `-c` command is non-interactive. If we want to execute multiple commands, we can write those commands in a text file and then execute them using the `-f` option. This command loads a very small and simple set of examples:

```
echo "SELECT current_database(); SELECT current_user;" >> examples.sql

psql -f examples.sql
```

```
postgres@SandboxHost-637997431697101465:~$ echo "SELECT current_database(); SELECT current_user;" >> examples.sql
postgres@SandboxHost-637997431697101465:~$ 
postgres@SandboxHost-637997431697101465:~$ psql -f examples.sql
current_database
-----
postgres
(1 row)

current_user
-----
postgres
(1 row)
postgres@SandboxHost-637997431697101465:~$ █
```

The `psql` tool can also be used with both the `-c` and `-f` modes together; each one can be used multiple times. In this case, it will execute all the commands consecutively:

```
$ psql -c "SELECT current_time" -f examples.sql -c "SELECT current_time"
```

```

      timetz
-----
18:52:15.287+01
(1 row)
...output removed for clarity...
      timetz
-----
18:58:23.554+01
(1 row)

```

The `psql` tool can also be used in interactive mode, which is the default, so it requires no option:

```
$ psql
postgres=#
```

The first interactive command you'll need is the following:

```
postgres=# help
```

You can then enter SQL or other commands. The following is the last interactive command you'll need:

```
postgres=# \quit
```

Unfortunately, you cannot type `quit` on its own, nor can you type `\exit` or other options. Sorry -- it's just `\quit`, or `\q` for short!

How it works...

In `psql`, you can enter the following two types of command:

- `psql` meta-commands
- SQL

A meta-command is a command for the `psql` client, whereas SQL is sent to the database server. An example of a meta-command is `\q`, which tells the client to disconnect. All lines that begin with `\` (a backslash) as the first non-blank character are presumed to be meta-commands of some kind.

If it isn't a meta-command, it's SQL. We keep reading SQL until we find a semicolon, so we can spread SQL across many lines and format it any way we find convenient.

The `help` command is the only exception. We provide this for people who are completely lost, which is a good thought; so let's start from there ourselves.

There are two types of `help` commands, which are as follows:

- `\?`: This provides help on `psql` meta-commands.
- `\h`: This provides help on specific SQL commands.

Consider the following snippet as an example:

```

postgres=# \h DELETE
Command: DELETE
Description: delete rows of a table
Syntax:
[ WITH [ RECURSIVE ] with_query [, ...] ]

```



```
DELETE FROM [ ONLY ] table [ [ AS ] alias ]
    [ USING usinglist ]
    [ WHERE condition | WHERE CURRENT OF cursor_name ]
    [ RETURNING * | output_expression [ AS output_name ] [,]]
```

I find this a great way to discover and remember options and syntax. You'll also appreciate having the ability to scroll back through the previous command history.

You'll get a lot of benefits from tab completion, which will fill in the next part of the syntax when you press the *Tab* key. This also works for object names, so you can type in just the first few letters and then press *Tab*; all the options will be displayed. Thus, you can type in just enough letters to make the object name unique and then hit *Tab* to get the rest of the name.

One-line comments begin with two dashes, as follows:

```
-- This is a single-line comment
```

Multiline comments are similar to those in C and Java:

```
/*
 * Multiline comment
 */
```

You'll probably agree that `psql` looks a little daunting at first, with strange backslash commands. I do hope you'll take a few moments to understand the interface and keep digging for more information. The `psql` tool is one of the most surprising parts of PostgreSQL, and it is incredibly useful for database administration tasks when used alongside other tools.

Changing your password securely

If you are using password authentication, then you may wish to change your password from time to time. This can be done from any interface. pgAdmin is a good choice, but here we show to do that from `psql`.

How to do it...

The most basic method is to use the `psql` tool. The `\password` command will prompt you once for a new password and again to confirm. Connect to the `psql` tool and type the following:

```
SET password_encryption = 'scram-sha-256';
\password
```

Enter a new password. This causes `psql` to send a SQL statement to the PostgreSQL server, which contains an already encrypted password string.

Make sure you use the `SCRAM-SHA-256` encryption, not the older and easily compromised MD5 encryption.

Make sure you don't forget your password either. It may prove difficult to maintain your database if you can't access it.

How it works...

As changing the password is just a SQL statement, any interface can do this.

If you don't use one of the main routes to change the password, you can still do it yourself, using SQL from any interface. Note that you need to encrypt your password because if you do submit one in plain text, such as the following, it will be shipped to the server in plaintext:

```
ALTER USER postgres PASSWORD 'postgres';
```

Luckily, the password in this case will still be stored in an encrypted form, but it will also be recorded in plaintext in the `psql` history file, as well as in any server and application logs, depending on the actual log-level settings.

Using a connection service file

As the number of connection options grows, you may want to consider using a connection service file.

The connection service file allows you to give a single name to a set of connection parameters. This can be accessed centrally to avoid the need for individual users to know the host and port of the database, and it is more resistant to future change.

You can set up a system-wide file as well as individual per-user files. The default file paths for these files are `/etc/pg_service.conf` and `~/.pg_service.conf` respectively.

A system-wide connection file controls service names for all users from a single place, while a per-user file applies only to that particular user. Keep in mind that the per-user file overrides the system-wide file -- if a service is defined in both the files, then the definition in the per-user file will prevail.

How to do it...

First, create a file named `pg_service.conf` with the following content:

```
[dbservice1]
host=localhost
port=5432
dbname=postgres
```

You can then copy it to either `/etc/pg_service.conf` or another agreed-upon central location. You can then set the `PGSYSCONFDIR` environment variable to that directory location.

Alternatively, you can copy it to `~/.pg_service.conf`. If you want to use a different name, set `PGSERVICEFILE`. Either way, you can then specify a connection string, such as the following:

```
service=dbservice1 user=sriggs
```

The service can also be set using an environment variable named `PGSERVICE`.