**Installation and Setup**

We will install postgresql package and pgAdmin4 from the internet and get them installed. During installation of postgresql we will be asked for a password. After the installation is complete postgresql should be automatically running. If not, we can run it using “services” and searching for postgres and starting it. After the installation is complete and server is running, we will include the path of our postgres installation directory up to bin and data into environmental path. After that we can confirm that postgres is installed by running command “postgres –version” in our terminal and we should see the version of postgres sql. We will then write psql -U postgres in our terminal to connect to the server and it'll ask for the password. We will enter the same password that was entered during postgress installation.

**Administration**

PostgreSQL can be administered from the command line using the psql utility, by running the command below:

psql postgres

This should get your psql utility running. psql is PostgreSQL’s command line tool.

To quit from the psql interface, you can type \q and you’re out.

If you need help, type \help on your psql terminal. This will list all the available help options. You can type in \help [Command Name], in case you need help with a particular command. For example, typing in \help UPDATE from within psql will show you the syntax of the update option.

Description: update rows of a table[ WITH [ RECURSIVE ] with\_query [, ...] ]UPDATE [ ONLY ] table\_name [ \* ] [ [ AS ] alias ] SET { column\_name = { expression | DEFAULT } | ( column\_name [, ...] ) = ( { expression | DEFAULT } [, ...] ) | ( column\_name [, ...] ) = ( sub-SELECT ) } [, ...] [ FROM from\_list ] [ WHERE condition | WHERE CURRENT OF cursor\_name ] [ RETURNING \* | output\_expression [ [ AS ] output\_name ] [, ...] ]

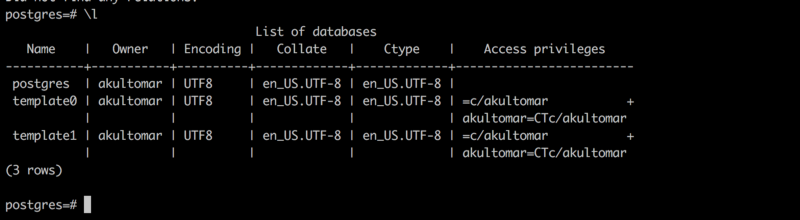
If you’re a beginner, you may still not understand. A quick Google search will provide you examples of its use or you can always search the official [psql documentation](https://www.postgresql.org/docs/current/static/sql-update.html) which will provide many examples.

**Starting up With Postgres**

When you first install PostgreSQL, there are a few common administrative tasks that you’ll frequently perform.

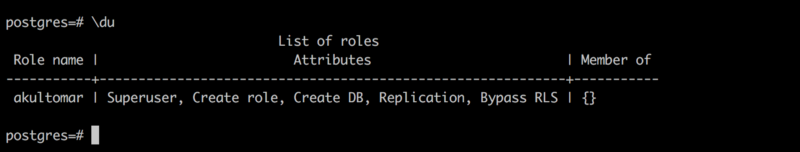
The first thing would be to check for existing users and databases. Run the command below to list all databases:

\list or \l



In the figure above, you can see **three** default databases and a superuser akultomar that get created when you install PostgreSQL.

To list all users, use the \du command. The attributes of the user tell us that they’re a Superuser.

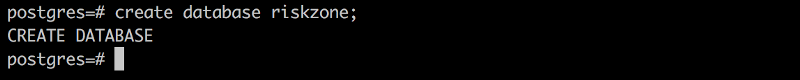


**Basic Database Operations**

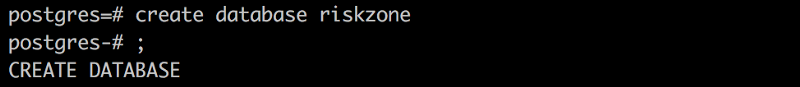
To perform basic database operations, you use the Structured Query Language (commonly known as SQL).

**Create a database**

To create a database, you use the create database command. In the example below, we’ll create a database named riskzone.

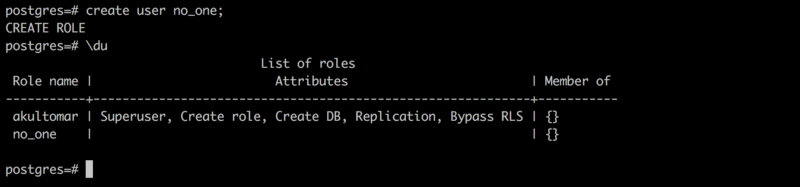


If you forget the semicolon at the end, the = sign at the postgres prompt is replaced with a - as in the figure below. This is basically an indication that you need to terminate your query. You’ll understand it’s significance when you actually start writing longer queries. For now, just put a semi-colon to complete the SQL statement and hit return.



**Create a user**

To create a user, you use the create user command. In the example below, we’ll create a user named no\_one.



When you create a user, the message shown is **CREATE ROLE**. Users are roles with login rights. I have used them interchangeably. You’ll also notice that the Attributes column is empty for the user no\_one. This means that the user no\_one has no administrative permissions. They can only read data and cannot create another user or database.

You can set a password for your user. To a set password for an existing user, you need use the \password command below:

postgres=#\password no\_one

To set a password when a user is created, the command below can be used:

postgres=#create user no\_two with login password 'qwerty';

**Delete a user or database**

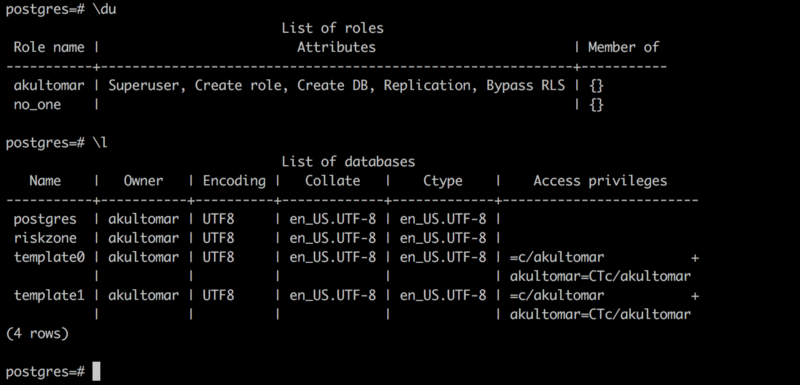
The drop command can be used to delete a database or user, as in the commands below.

drop database <database\_name>

drop user <user\_name>

This command needs to be used very carefully. Things dropped don’t come back unless you have a backup in place.

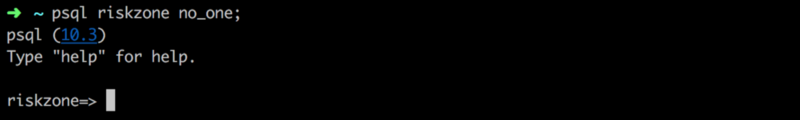
If we run the \du and \l that we learned about earlier to display the list of users and databases respectively, we can see that our newly created no\_one user and riskzone database.



When you specify psql postgres (without a username), it logs into the postgres database using the default superuser (akultomar in my case). To log into a database using a specific user, you can use the command below:

psql [database\_name] [user\_name]

Let’s login to the riskzone database with the no\_one user. Hit \q to quit from the earlier postgres database and then run the command below to log into riskzone with the user no\_one.



I hoped you like the short introduction to PostgreSQL. I’ll be writing another article to help you understand roles better. If you’re new to SQL, my advice would be to practice as much as you can. Get your hands dirty and create your own little tables and practice.

**Connecting Postgresql with Django:**

First after making sure that we are in our virtual environment we will install two packages through pip “psycopg2” and “psycopg2-binary”. Once that is done, we will open settings.py file and in the databases section we will enter following line of code under “default” key:

DATABASES = {

'default': {

'ENGINE': 'django.db.backends.postgresql',

'NAME': 'btredb',

'USER': 'postgres',

'PASSWORD': '421546',

'HOST': 'localhost'

}

}

**GETTING RID OF UNAPPLIED MIGRATIONS ERROR:**

From the beginning we are seeing the migration errors that says that we have unapplied migrations. Migration is basically a file that tells a database about setting up tables, data, columns, data types etc. Django has default migrations for admin area and the migrations files has been setup. They just haven’t been run and put into the database.

Later we will build our own migrations for listings and realtors but for now we want to run these migrations to get rid of these errors. For this we will just write “python manage.py migrate”. Now if something went wrong with our database during setting up, we will see error here.

Once we do that, then either from our terminal or from pgAdmin we will see that many tables have been created.