How to Install and Configure PostgreSQL in Django

PostgreSQL is one of the best choices for a secure hosting environment. Here's how to integrate PostgreSQL into Django.

Adding a database to the application ensures data integrity and security. PostgreSQL (Postgres) is a structured query language (SQL) database management system that you can try.

PostgreSQL supports most popular operating systems and is compatible with modern programming languages. PostgreSQL also handles different data and document types. In the trend of programming that requires more and more SQL, learning to configure and use PostgreSQL gives programmers an advantage.

The following article will show you how to install, configure, and use Postgres in a Django application. You will also test the functionality of the database by adding, storing, and retrieving data.

1. Install PostgreSQL on the system

The following tutorial explains how to install Postgres on Ubuntu operating system. Before installing Postgres, update package versions and their dependencies with the following command:

```
$ sudo apt-get update
```

Next, install PostgreSQL with the following command:

```
$ sudo apt-get install postgresql postgresql-contrib libpq-dev
```

When prompted to confirm the installation, press Y for Yes .

Connect the server with the following command:

```
$ sudo -i -u postgres
```

Then use the client database to determine the installed Postgres version.

Run the command psql --version .

```
postgres@nameofaccount:~$ psql --version
```

The result will show the Postgres version as below:

```
psql (PostgreSQL) 14.5 (Ubuntu 14.5-0ubuntu0.22.04.1)
```

Exit the Postgres account by running the exit command.

```
postgres@nameofaccount:~$ exit
```

2. Create database

You need to create a database to connect to your Django app. Navigate back to the Postgres shell and run the following commands consecutively.

```
sudo -i -u postgres
```

```
$ psql
```

Then use the client to create the database on the server.

```
postgres=# create database new_db;
```

This server returns the term CREATE DATABASE when it creates a database. You can also check by listing all the databases in the system with the l command .

```
postgres=# l
```

Structure of PostgreSQL

Like a typical SQL database, PostgreSQL stores data in tables. This table represents the different items/models in an application. This table has a fixed number of columns and rows.

Each table has a special column called a primary key, a unique identifier for each row in the table. A table can also have a foreign key that connects it to another table's primary key.

Foreign keys define the relationship between two tables.

Next, you need to create a Django app and link the database. However, first, install psycopg2 to connect the application and the database.

3. Install Django and the Psychcopg2 . library

To connect Postgres with your Django app, you need to install the psycopg2 library. This is a Postgres dependency that helps to connect and communicate with Django.

Run the following command to install psycopg2 and Django :

```
$ pipenv install psycopg2 Django
```

4. Create Django App

You need to create a Django app that will use the Postgres database. First, create a project named myboma that supports this application. Django projects automatically add the dependencies and application settings needed to run app.

Create a folder named Boma-watch and navigate to it with the following command:

```
$ mkdir Boma-watch
```

```
$ cd Boma-watch
```

Then create the Django project with the following command:

```
$ django-admin startproject myboma .
```

Next, create a new app named boma with the following command:

```
$ django startapp boma
```

After running the application in the browser to confirm it is working, you will connect it to the database in the next section.

5. Database connection with Django app

Now you will connect the Django app to the created database using the following steps:

Step 1: Change project settings to use Postgres

You must change your project settings to connect your Django app to Postgres. Navigate to the project's settings.py file . Then change the DATABASES settings to add your Postgres configuration.

Replace USER and PASSWORD with psql username and password .

```
#. DATABASES = { 'default': { 'ENGINE':'django.db.backends.postgresql', 'NAME':'
```

Step 2: Update time zone

Next, in the settings.py file , set the Timezone to represent your location. Django projects are preconfigured with UTC time zone.

```
TIME_ZONE = Africa/Nairobi
```

Step 3: Create a template

Create a Profile template in the application. You will use the model class to create a table that stores your application's name and biological data. in the database

```
class Profile(models.Model): name = models.CharField(max_length=30) bio = models
```

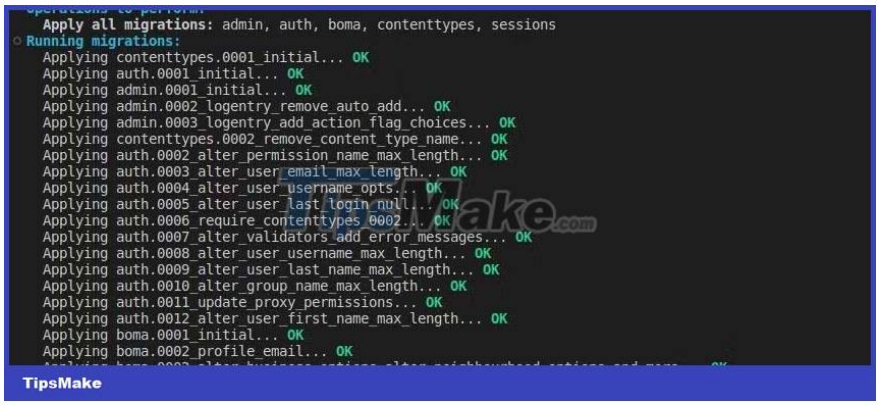
Step 4: Start moving

Run the following command to start the migration:

```
(virtual)$ python manage.py makemigrations boma
```

```
(virtual)$ python manage.py migrate
```

A successful migration will look like this:



The command python manage.py migrate selects the app from INSTALLED_APPS settings > models.py file and creates a table for each model. You have successfully connected Postgres to the application.

Now, you can test running CREATE, READ, UPDATE, and DELETE (CRUD) commands on the application.

6. Check the CRUD command on the app

Django's Python API will allow you to implement some CRUD database operations. The API connects functions to templates to allow you to manipulate the database.

Open the Python shell in the Django project with the following command:

```
(virtual)$ python manage.py shell
```

This command will open a console where you can check CRUD operations.

Creation activity

First, import the Profile model from the models module with the following command:

```
from boma.models import Profile
```

Then create an instance of the Profile class and pass your data in.

```
 prof1 = Profile.objects.create(name ='Ken',bio ='I am a Scorpio')
```

Next, save the data in the database.

```
prof1.save()
```

Reading activities

After creating the data in the database and saving it, you can query it to get the saved data.

Use Profile.objects.all() to access all data in the Profile table in the database.

```
Profile.objects.all() #outputs
```

You can also access an object by primary key or pk . They are numbers attached to each item stored in the database.

```
Profile.objects.get(pk = 1) #outputs
```

Update activity

You can update the saved data with the following command:

```
Profile.objects.filter(id = 1).update(name ='Kim';) #outputs 1
```

To check if its name has been updated, execute the following command:

```
Profile.objects.filter(id = 1) #outputs
```

Delete operation

You can delete a saved item with the following command:

```
Profile.objects.filter(id = 1).delete() #outputs (1, {app.Profile: 1})
```

To confirm deletion, run the following command:

```
Profile.objects.filter(id = 1) #outputs
```

You may see an empty query set, indicating that the data is no longer in the database.

Above is how to configure PostgreSQL in Django. Hope the article is useful to you.


```
Profile.objects.filter(id = 1) #outputs
```