

How to Install, Configure, Backup, and Restore PostgreSQL on Ubuntu 20.04 LTS

Learn how to install, configure, backup, and restore PostgreSQL on Ubuntu 20.04 LTS with our comprehensive step-by-step guide for database administrators.

Contents

01	Introduction	3
02	Prerequisites	3
03	1. Install the PostgreSQL Server	3
04	2. Configure the PostgreSQL Server	5
05	3. Create a PostgreSQL Database	7
06	4. Create a PostgreSQL Table and Perform CRUD operations	8
07	5. Backup and Restore PostgreSQL Database	12
08	Conclusion	15

Introduction

PostgreSQL is an open-source Object-Relational Database Management System (ORDBMS) that supports most SQL standards, including foreign keys and transactions.

This community-driven database server is free to own, reliable, secure, and scalable. It powers dynamic websites and applications at Apple, Reddit, Spotify, and NASA.

In this tutorial, you'll install and configure PostgreSQL server on Ubuntu 20.04 LTS, then create a database and perform data manipulation. You'll learn how to safeguard the security of your database by creating and restoring backups.

Prerequisites

Before you begin, make sure you have the following:

- An Ubuntu 20.04 LTS server.
- A non-root sudo user.

1. Install the PostgreSQL Server

Start by updating your Ubuntu's server package information index.

```
$ sudo apt -y update
```

Then, install the PostgreSQL core database server, command-line client, and additional dependencies.

```
$ sudo apt install -y postgresql postgresql-contrib
```

Verify the PostgreSQL installation.

```
$ dpkg --status postgresql
```

Make sure your output is similar to the one shown below. You should have PostgreSQL version 12.

```
Package: postgresql
Status: install ok installed
...
Version: 12+214ubuntu0.1
...
```

- PostgreSQL runs on port 5432.
- The default configuration file is located here:

```
/etc/postgresql/12/main/postgresql.conf
```

- PostgreSQL creates all databases in this directory:

```
/var/lib/postgresql/12/main
```

The PostgreSQL database server runs as a service under the name `postgresql`. Manage the service by running the commands below.

- Stop PostgreSQL server:

```
$ sudo systemctl stop postgresql
```

- Start PostgreSQL server:

```
$ sudo systemctl start postgresql
```

- Restart PostgreSQL(e.g. after changing configuration settings) server:

```
$ sudo systemctl restart postgresql
```

- Reload PostgreSQL server:

```
$ sudo systemctl reload postgresql
```

- Check PostgreSQL status:

```
$ sudo systemctl status postgresql
```

You've successfully installed the PostgreSQL database server. Before you begin using it, you'll secure the root user with a password in the next step.

2. Configure the PostgreSQL Server

By default, PostgreSQL ships with `psql`. This is a command-line client that you can use to log in to the database server. The installation setup also creates a UNIX user named `postgres`. This is the PostgreSQL server **super-user** or **root** user.

To log in to the PostgreSQL server via the `psql` command-line client as the `postgres` user, use the command below.

```
$ sudo -u postgres psql
```

Under the hood, the statement above switches to the `postgres` **UNIX USER** and runs the `psql` command. Once you execute the command, you will get the PostgreSQL prompt below. This means your PostgreSQL server is ready to receive SQL commands.

```
postgres=#
```

When you install PostgreSQL for the first time, a password for the super-user is not set by default. To create the password, execute the statement below.

```
postgres=# \password postgres
```

You will get the prompt below. Enter a strong password and confirm it.

```
Enter new password: EXAMPLE_PASSWORD
Enter it again: EXAMPLE_PASSWORD
```

Output.

```
postgres=#
```

Then, exit from the `psql` command-line client.

```
postgres=# \q
```

To test the new password, edit the `/etc/postgresql/12/main/pg_hba.conf` configuration file using `nano`.

```
$ sudo nano /etc/postgresql/12/main/pg_hba.conf
```

Locate the line `local all postgres peer` as shown in the below content.

```
...

# Database administrative login by Unix domain socket
local  all             postgres              peer

# TYPE  DATABASE        USER            ADDRESS              METHOD

# "local" is for Unix domain socket connections only

...
```

Change the authentication method from `peer` to `md5` so that the line reads `local all postgres md5`.

```
...
```

```
# Database administrative login by Unix domain socket
local  all                postgres                md5

# TYPE  DATABASE      USER          ADDRESS            METHOD

# "local" is for Unix domain socket connections only

...
```

Save and close the file by pressing `Ctrl + X`, then `Y` and `Enter`. Then, restart the `postgresql` service to load the new settings.

```
$ sudo systemctl restart postgresql
```

Try to log in to the PostgreSQL server again; this time around, you should be prompted to enter a password.

```
$ sudo -u postgres psql
```

Enter the PostgreSQL server password that you created earlier and press `Enter` to continue. You should now be logged in to the PostgreSQL server. Make sure you get the command-line client prompt as shown below.

```
postgres=#
```

Once you've protected your PostgreSQL server with a password, you can now create a sample database and perform some data manipulation on it.

3. Create a PostgreSQL Database

Use the `CREATE DATABASE` command to create your first `test_db` database.

```
postgres=# CREATE DATABASE test_db;
```

Output.

```
CREATE DATABASE
```

To list the databases, use the `\l` command.

```
postgres=# \l
```

Your new `test_db` database should be listed as shown below.

```
   Name   | Owner   | Encoding | Collate | Ctype  | Access privileges
-----+-----+-----+-----+-----+-----
...
test_db  | postgres | UTF8     | C.UTF-8 | C.UTF-8 |
```

To switch to the `test_db` database, use the `\c` command.

```
postgres=# \c test_db;
```

The syntax below should also work.

```
postgres=# \connect test_db;
```

Ensure you get an output showing that you've been connected to the `test_db` database.

```
You are now connected to database "test_db" as user "postgres".
test_db=#
```

This means your `test_db` database is ready, and you can proceed to create a table in the next step.

4. Create a PostgreSQL Table and Perform CRUD operations

Create your first table named `customers` under the `test_db` database. Use the `SERIAL` statement to create an `auto-increment` column to store the `customer_id`'s.

```
test_db-# CREATE TABLE customers (  
    customer_id SERIAL PRIMARY KEY,  
    first_name VARCHAR (50),  
    last_name VARCHAR (50),  
    phone VARCHAR (10)  
);
```

Output.

```
CREATE TABLE
```

List PostgreSQL Tables

Run the `\dt` command to list PostgreSQL tables.

```
test_db-# \dt;
```

Your `customers` table should now be present on the list of relations, as shown below.

```
          List of relations  
Schema | Name      | Type  | Owner  
-----+-----+-----+-----  
public | customers | table | postgres  
(1 row)
```

Describe a PostgreSQL Table

To get a description of the `customers` table, use the `\d` command.

```
test_db=# \d customers;
```

You can now see the structure of your `customers` table as shown below.

```
   Column      |          Type          | Collation | Nullable |  
Default  
-----+-----+-----+-----  
+-----+-----+-----+-----
```

```
customer_id | integer          |          | not null |
nextval('customers_customer_id_seq'::regclass)
first_name  | character varying(50) |          |          |
last_name   | character varying(50) |          |          |
phone       | character varying(10) |          |          |
Indexes:
    "customers_pkey" PRIMARY KEY, btree (customer_id)
```

Press Q to get back to the `test_db=#` prompt.

Insert Data to the PostgreSQL Table

Run the SQL commands one by one to insert sample data into the `customers` table.

```
test_db=# INSERT INTO customers(first_name, last_name, phone) VALUES ('JOHN', 'DOE',
'11111');
test_db=# INSERT INTO customers(first_name, last_name, phone) VALUES ('MARY', 'ROE',
'33333');
test_db=# INSERT INTO customers(first_name, last_name, phone) VALUES ('JANE', 'SMITH',
'55555');
```

Ensure you get the output below after executing each `INSERT` statement.

```
INSERT 0 1
...
```

Display Data from a PostgreSQL Table

Run a `SELECT` statement against the `customers` table to display the records that you've inserted.

```
test_db=# SELECT * FROM customers;
```

You will get a list of customers as shown below.

```
customer_id | first_name | last_name | phone
-----+-----+-----+-----
          1 | JOHN      | DOE       | 11111
```

```
2 | MARY      | ROE      | 33333
3 | JANE      | SMITH    | 55555
(3 rows)
```

Update Data in a PostgreSQL Table

To edit the data in a PostgreSQL table, use the `UPDATE` and `WHERE` statements together. For instance, to update `JOHN DOE'S` phone to `88888` from `11111`, execute the command below.

```
test_db=# UPDATE customers SET phone = '88888' WHERE customer_id = 1;
```

Output.

```
UPDATE 1
```

Confirm if the `UPDATE` statement was executed successfully by running a `SELECT` statement against the record.

```
test_db=# SELECT * FROM customers WHERE customer_id = 1;
```

As you can see from the output below, `JOHN DOE'S` phone number has been updated to `88888`.

```
customer_id | first_name | last_name | phone
-----+-----+-----+-----
1 | JOHN      | DOE      | 88888
(1 row)
```

Delete Record From a PostgreSQL Table

Execute the `DELETE` statement to delete a record in a PostgreSQL database table. For instance, to delete `MARY ROE` from the `customers` table, use the command below.

```
test_db=# DELETE FROM customers WHERE customer_id = 2;
```

Output.

```
DELETE 1
```

To confirm the deletion, issue the `SELECT` statement against the `customers` table.

```
test_db=# SELECT * FROM customers;
```

You can now see that `MARY ROE's` record is missing from the table.

```
customer_id | first_name | last_name | phone
-----+-----+-----+-----
           3 | JANE      | SMITH    | 55555
           1 | JOHN     | DOE      | 88888
(2 rows)
```

Exit from the PostgreSQL command-line interface.

```
test_db=# \q
```

In the next step, you'll learn how to backup and restore a PostgreSQL database.

5. Backup and Restore PostgreSQL Database

PostgreSQL comes with useful tools for creating backups and restoring databases from dump files.

Backup PostgreSQL Database

To create a compressed backup for your `test_db`, use the `pg_dump` utility.

```
$ pg_dump -d test_db -U postgres | gzip > test_db_backup.sql.gz
```

When prompted, enter the super-user password for your PostgreSQL database server and press Enter to proceed. The `pg_dump` should finalize dumping and compressing the database.

Create a plain-text backup by executing the command below.

```
$ pg_dump -U postgres -f test_db_backup.sql test_db
```

Again, key in the password for your PostgreSQL server and press Enter to continue. A plain text SQL file should be created. You can confirm the creation of the above backup files by listing the current files from your working directory.

```
$ ls -lsa
```

Your backup files should be listed as shown below.

```
...  
... test_db_backup.sql  
... test_db_backup.sql.gz  
...
```

After creating the dump files, you will restore your database to its original state in the next step.

Restore PostgreSQL Database

Log in to the PostgreSQL database server as a super-user.

```
$ sudo -u postgres psql
```

When prompted, enter your password and press Enter to proceed. Next, switch to the `test_db` and drop the `customers` table that you created earlier. You'll try to recreate this table again from the backups.

```
postgres=# \c test_db;  
postgres=# DROP TABLE customers;
```

Then, exit from the PostgreSQL command-line interface by executing the `\q` command.

```
postgres=# \q
```

Next, issue either of the commands below to restore the database to its original state from the compressed backup file.

1. Restore from compressed backup:

```
$ gunzip -c test_db_backup.sql.gz | psql -U postgres -d test_db
```

2. Restore from plain text SQL file that you created earlier.

```
$ psql -U postgres -d test_db -f test_db_backup.sql
```

Key in your password and press Enter to continue. Your database should now be restored from either of the backup files. You can log in to the database server again to check if the `customers` table has been recreated.

```
$ sudo -u postgres psql
```

Enter your password and press Enter to proceed. Then switch to the `test_db`.

```
postgres=# \c test_db;
```

Try querying data from the `customers` table

```
test_db=# SELECT * FROM customers;
```

You will get a list of customers as shown below.

```
customer_id | first_name | last_name | phone
-----+-----+-----+-----
          3 | JANE      | SMITH    | 55555
          1 | JOHN     | DOE      | 88888
(2 rows)
```

This means the backup has been restored successfully.

Conclusion

In this guide, you've learned how to install and configure PostgreSQL on Ubuntu 20.04. You've also used the basic SQL syntax to create a PostgreSQL database and performed some CRUD operations. Towards the end, you've used data management solutions to backup files and restore your database. With this PostgreSQL step up in place, you can now use your favorite scripting language like PHP, Node.js, or Python to create your web application or website.



VULTR

