# Install PostgreSQL

**Summary**: To **install PostgreSQL** on your local system for learning and practicing PostgreSQL.

PostgreSQL was developed for UNIX-like platforms, however, it was designed to be portable. It means that PostgreSQL can also run on other platforms such as Mac OS X, Solaris, and Windows.

Since version 8.0, PostgreSQL offers an installer for Windows systems that makes the installation process easier and faster. For development purpose, we will install PostgreSQL version 9.5  on Windows 10.

**NOTE**: If you are running the Windows 8 and 10, you need to create a Windows users with administrator role e.g., postgres and use this user to run the installation file.

There are three steps to complete the PostgreSQL installation:

1. Download PostgreSQL installer for Windows
2. Install PostgreSQL
3. Verify the installation

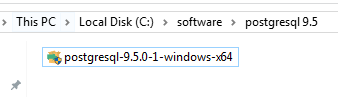
## Download PostgreSQL Installer for Windows

You need to download the installer from PostgreSQL Official website.

* Go to the PostgreSQL official website, download section for Windows http://www.postgresql.org/download/windows/
* Choose the latest version to download. It takes few minutes to complete the download.

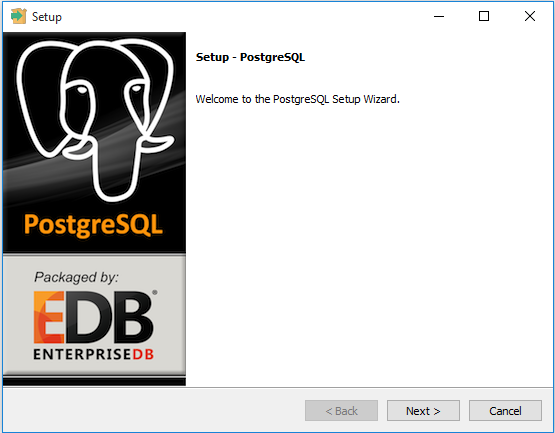
## Install PostgreSQL step by step

Double click on the installer file, an installation wizard will appear and guide you through multiple steps where you can choose different options that you would like to have in PostgreSQL.



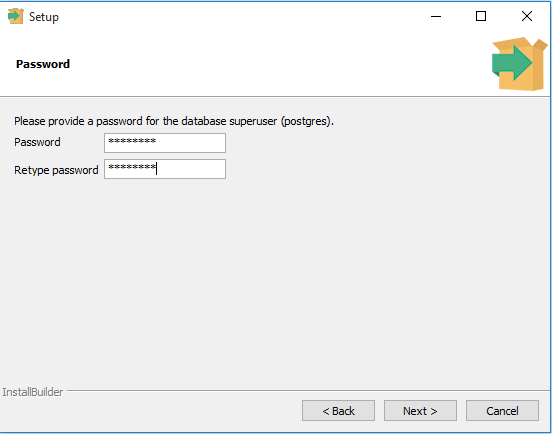
The following illustrates each step and its options for installation. If you install a different version, you may get additional steps.

**Start Installing PostgreSQL**

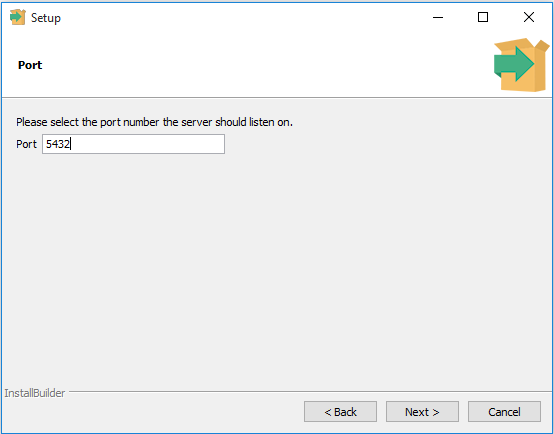


Specify installation folder, choose your own or keep the default folder suggested by PostgreSQL installer.

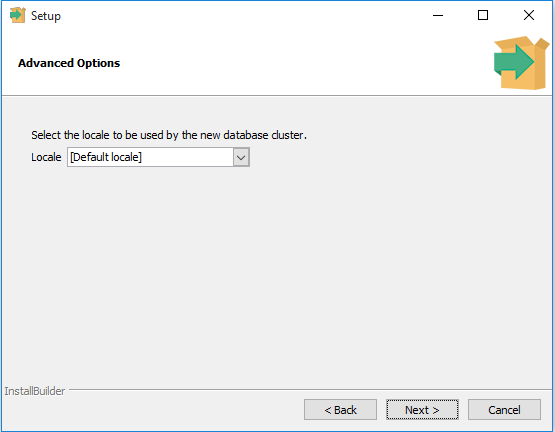
**Enter the password for the database superuser and service account.**



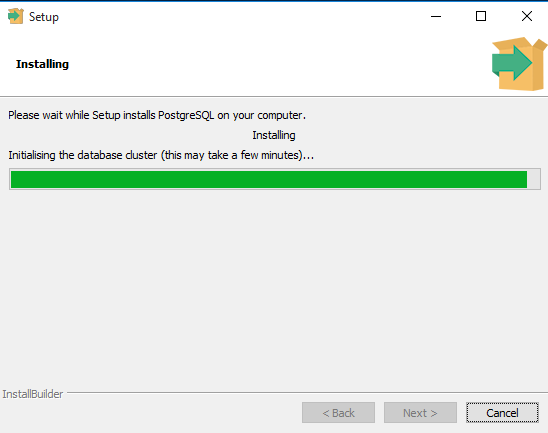
**Enter the port for PostgreSQL. Make sure that no other applications are using this port. Leave it as default if you are unsure.**



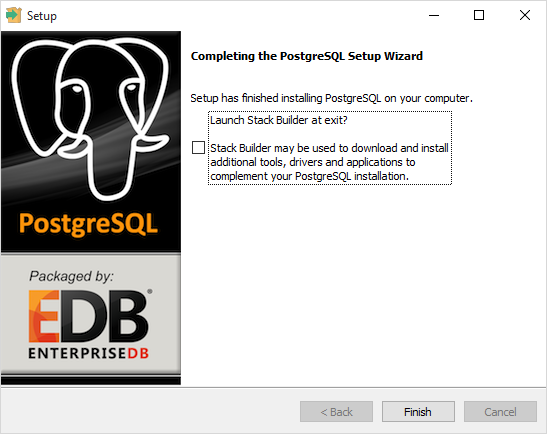
Choose the default locale used by the database.



You’ve completed providing information for the PostgreSQL installer. Click the Next button to install PostgreSQL



The installation may take few minutes to complete.



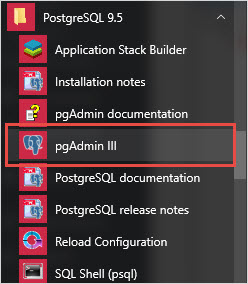
Click the Finish button to complete the PostgreSQL installation.

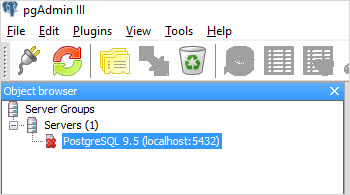
## Verify the Installation

There are several ways to verify the installation. You can try to connect to the PostgreSQL database server from any client application e.g.,  psql and pgAdmin.

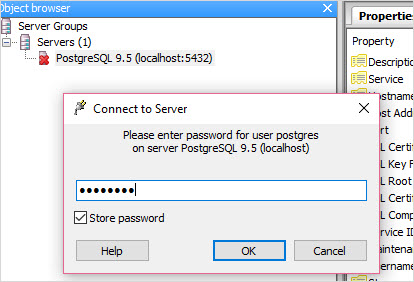
The quick way to verify the installation is through the pgAdmin application.

First, click on pgAdmin III to launch it. The pgAdmin III GUI will display.

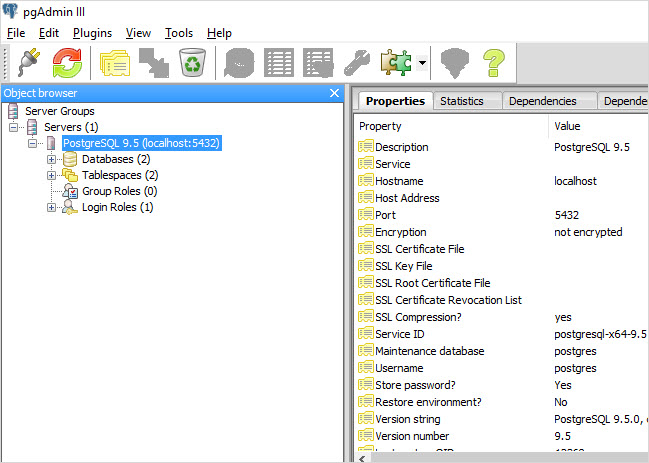




Second, double-click**PostgreSQL 9.5** on the object browser. It will ask you for the admin password. Just enter the password you’ve used in the installation step.



Third, if everything is fine, the pgAdmin will display all the objects that belong to the server.



# Install PostgreSQL on Linux (Ubuntu)

sudo apt-get update

sudo apt-get install postgresql postgresql-contrib

Now that our software is installed, we can go over how it works and how it may be different from similar database management systems you may have used.

## Using PostgreSQL Roles and Databases

By default, Postgres uses a concept called "roles" to handle in authentication and authorization. These are, in some ways, similar to regular Unix-style accounts, but Postgres does not distinguish between users and groups and instead prefers the more flexible term "role".

Upon installation Postgres is set up to use **ident** authentication, which means that it associates Postgres roles with a matching Unix/Linux system account. If a role exists within Postgres, a Unix/Linux username with the same name will be able to sign in as that role.

There are a few ways to utilize this account to access Postgres.

### Switching Over to the postgres Account

The installation procedure created a user account called postgres that is associated with the default Postgres role. In order to use Postgres, we can log into that account.

Switch over to the postgres account on your server by typing:

sudo -i -u postgres

You can now access a Postgres prompt immediately by typing:

psql

You will be logged in and able to interact with the database management system right away.

Exit out of the PostgreSQL prompt by typing:

\q

### Accessing a Postgres Prompt Without Switching Accounts

You can also run the command you'd like with the postgres account directly with sudo.

For instance, in the last example, we just wanted to get to a Postgres prompt. We could do this in one step by running the single command psql as the postgres user with sudo like this:

sudo -u postgres psql

This will log you directly into Postgres without the intermediary bash shell in between.

Again, you can exit the interactive Postgres session by typing:

\q

## Create a New Role

Currently, we just have the postgres role configured within the database. We can create new roles from the command line with the createrole command. The --interactive flag will prompt you for the necessary values.

If you are logged in as the postgres account, you can create a new user by typing:

createuser --interactive

If, instead, you prefer to use sudo for each command without switching from your normal account, you can type:

sudo -u postgres createuser --interactive

The script will prompt you with some choices and, based on your responses, execute the correct Postgres commands to create a user to your specifications.

Output

Enter name of role to add: sammy

Shall the new role be a superuser? (y/n) y

You can get more control by passing some additional flags. Check out the options by looking at the manpage:

man createuser

## Create a New Database

By default, another assumption that the Postgres authentication system makes is that there will be an database with the same name as the role being used to login, which the role has access to.

So if in the last section, we created a user called sammy, that role will attempt to connect to a database which is also called sammy by default. You can create the appropriate database with the createdbcommand.

If you are logged in as the postgres account, you would type something like:

createdb sammy

If, instead, you prefer to use sudo for each command without switching from your normal account, you would type:

sudo -u postgres createdb sammy

## Open a Postgres Prompt with the New Role

To log in with ident based authentication, you'll need a Linux user with the same name as your Postgres role and database.

If you don't have a matching Linux user available, you can create one with the adduser command. You will have to do this from an account with sudo privileges (not logged in as the postgres user):

sudo adduser sammy

Once you have the appropriate account available, you can either switch over and connect to the database by typing:

sudo -i -u sammy

psql

Or, you can do this inline:

sudo -u sammy psql

You will be logged in automatically assuming that all of the components have been properly configured.

If you want your user to connect to a different database, you can do so by specifying the database like this:

psql -d postgres

Once logged in, you can get check your current connection information by typing:

\conninfo

Output

You are connected to database "sammy" as user "sammy" via socket in "/var/run/postgresql" at port "5432".

This can be useful if you are connecting to non-default databases or with non-default users.

## Create and Delete Tables

Now that you know how to connect to the PostgreSQL database system, we can to go over how to complete some basic tasks.

First, we can create a table to store some data. Let's create a table that describes playground equipment.

The basic syntax for this command is something like this:

CREATE TABLE table\_name (

column\_name1 col\_type (field\_length) column\_constraints,

column\_name2 col\_type (field\_length),

column\_name3 col\_type (field\_length)

);

As you can see, we give the table a name, and then define the columns that we want, as well as the column type and the max length of the field data. We can also optionally add table constraints for each column.

For our purposes, we're going to create a simple table like this:

CREATE TABLE playground (

equip\_id serial PRIMARY KEY,

type varchar (50) NOT NULL,

color varchar (25) NOT NULL,

location varchar(25) check (location in ('north', 'south', 'west', 'east', 'northeast', 'southeast', 'southwest', 'northwest')),

install\_date date

);

We have made a playground table that inventories the equipment that we have. This starts with an equipment ID, which is of the serial type. This data type is an auto-incrementing integer. We have given this column the constraint of primary key which means that the values must be unique and not null.

For two of our columns (equip\_id and install\_date), we have not given a field length. This is because some column types don't require a set length because the length is implied by the type.

We then give columns for the equipment type and color, each of which cannot be empty. We create a location column and create a constraint that requires the value to be one of eight possible values. The last column is a date column that records the date that we installed the equipment.

We can see our new table by typing:

\d

Output

List of relations

Schema | Name | Type | Owner

--------+-------------------------+----------+-------

public | playground | table | sammy

public | playground\_equip\_id\_seq | sequence | sammy

(2 rows)

Our playground table is here, but we also have something called playground\_equip\_id\_seq that is of the type sequence. This is a representation of the serial type we gave our equip\_id column. This keeps track of the next number in the sequence and is created automatically for columns of this type.

If you want to see just the table without the sequence, you can type:

\dt

Output

List of relations

Schema | Name | Type | Owner

--------+------------+-------+-------

public | playground | table | sammy

(1 row)

## Add, Query, and Delete Data in a Table

Now that we have a table, we can insert some data into it.

Let's add a slide and a swing. We do this by calling the table we're wanting to add to, naming the columns and then providing data for each column. Our slide and swing could be added like this:

* INSERT INTO playground (type, color, location, install\_date) VALUES ('slide', 'blue', 'south', '2014-04-28');
* INSERT INTO playground (type, color, location, install\_date) VALUES ('swing', 'yellow', 'northwest', '2010-08-16');

You should take care when entering the data to avoid a few common hangups. First, keep in mind that the column names should not be quoted, but the column values that you're entering do need quotes.

Another thing to keep in mind is that we do not enter a value for the equip\_id column. This is because this is auto-generated whenever a new row in the table is created.

We can then get back the information we've added by typing:

* SELECT \* FROM playground;

Output

equip\_id | type | color | location | install\_date

----------+-------+--------+-----------+--------------

1 | slide | blue | south | 2014-04-28

2 | swing | yellow | northwest | 2010-08-16

(2 rows)

Here, you can see that our equip\_id has been filled in successfully and that all of our other data has been organized correctly.

If the slide on the playground breaks and we have to remove it, we can also remove the row from our table by typing:

* DELETE FROM playground WHERE type = 'slide';

If we query our table again, we will see our slide is no longer a part of the table:

* SELECT \* FROM playground;

Output

equip\_id | type | color | location | install\_date

----------+-------+--------+-----------+--------------

2 | swing | yellow | northwest | 2010-08-16

(1 row)

## How To Add and Delete Columns from a Table

If we want to modify a table after it has been created to add an additional column, we can do that easily.

We can add a column to show the last maintenance visit for each piece of equipment by typing:

* ALTER TABLE playground ADD last\_maint date;

If you view your table information again, you will see the new column has been added (but no data has been entered):

* SELECT \* FROM playground;

Output

equip\_id | type | color | location | install\_date | last\_maint

----------+-------+--------+-----------+--------------+------------

2 | swing | yellow | northwest | 2010-08-16 |

(1 row)

We can delete a column just as easily. If we find that our work crew uses a separate tool to keep track of maintenance history, we can get rid of the column here by typing:

* ALTER TABLE playground DROP last\_maint;

## How To Update Data in a Table

We know how to add records to a table and how to delete them, but we haven't covered how to modify existing entries yet.

You can update the values of an existing entry by querying for the record you want and setting the column to the value you wish to use. We can query for the "swing" record (this will match every swing in our table) and change its color to "red". This could be useful if we gave the swing set a paint job:

* UPDATE playground SET color = 'red' WHERE type = 'swing';

We can verify that the operation was successful by querying our data again:

* SELECT \* FROM playground;

Output

equip\_id | type | color | location | install\_date

----------+-------+-------+-----------+--------------

2 | swing | red | northwest | 2010-08-16

(1 row)