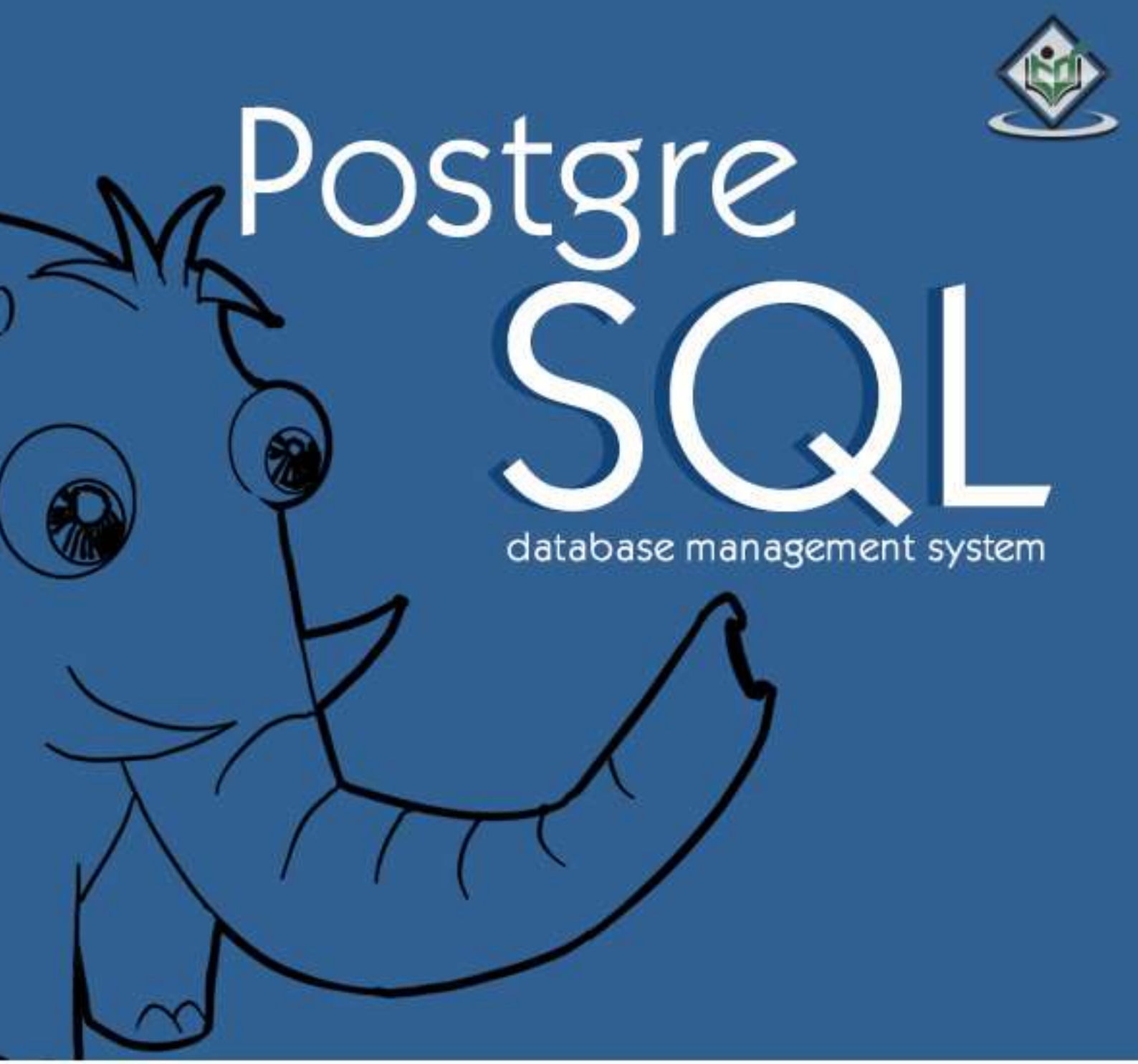


tutorialspoint

SIMPLYEASYLEARNING

www.tutorialspoint.com

About the Tutorial

PostgreSQL is a powerful, open source object-relational database system. It has more than 15 years of active development and a proven architecture that has earned it a strong reputation for reliability, data integrity, and correctness.

PostgreSQL runs on all major operating systems, including Linux, UNIX (AIX, BSD, HP-UX, SGI IRIX, Mac OS X, Solaris, Tru64), and Windows.

This tutorial will give you quick start with PostgreSQL and make you comfortable with PostgreSQL programming.

Audience

This tutorial has been prepared for the beginners to help them understand the basic to advanced concepts related to PostgreSQL Database.

Prerequisites

Before you start practicing with various types of examples given in this reference, I'm making an assumption that you are already aware about what is database, especially RDBMS and what is a computer programming language.

Copyright & Disclaimer

Table of Contents

1. PostgreSQL – Overview

PostgreSQL is a powerful, open source object-relational database system. It has more than 15 years of active development phase and a proven architecture that has earned it a strong reputation for reliability, data integrity, and correctness.

This tutorial will give you a quick start with PostgreSQL and make you comfortable with PostgreSQL programming.

What is PostgreSQL?

PostgreSQL (pronounced as **post-gress-Q-L**) is an open source relational database management system (DBMS) developed by a worldwide team of volunteers. PostgreSQL is not controlled by any corporation or other private entity and the source code is available free of charge.

A Brief History of PostgreSQL

PostgreSQL, originally called Postgres, was created at UCB by a computer science professor named Michael Stonebraker. Stonebraker started Postgres in 1986 as a follow-up project to its predecessor, Ingres, now owned by Computer Associates.

1. 1977-1985: A project called INGRES was developed.

- Proof-of-concept for relational databases
- Established the company Ingres in 1980
- Bought by Computer Associates in 1994

2. 1986-1994: POSTGRES

- Development of the concepts in INGRES with a focus on object orientation and the query language - Quel
- The code base of INGRES was not used as a basis for POSTGRES
- Commercialized as Illustra (bought by Informix, bought by IBM)

3. 1994-1995: Postgres95

- Support for SQL was added in 1994
- Released as Postgres95 in 1995
- Re-released as PostgreSQL 6.0 in 1996
- Establishment of the PostgreSQL Global Development Team

Key Features of PostgreSQL

PostgreSQL runs on all major operating systems, including Linux, UNIX (AIX, BSD, HP-UX, SGI IRIX, Mac OS X, Solaris, Tru64), and Windows. It supports text, images, sounds, and video, and includes programming interfaces for C / C++, Java, Perl, Python, Ruby, Tcl and Open Database Connectivity (ODBC).

PostgreSQL supports a large part of the SQL standard and offers many modern features including the following:

- Complex SQL queries
- SQL Sub-selects
- Foreign keys
- Trigger
- Views
- Transactions
- Multiversion concurrency control (MVCC)
- Streaming Replication (as of 9.0)
- Hot Standby (as of 9.0)

You can check official documentation of PostgreSQL to understand the above-mentioned features. PostgreSQL can be extended by the user in many ways. For example by adding new:

- Data types
- Functions
- Operators
- Aggregate functions
- Index methods

Procedural Languages Support

PostgreSQL supports four standard procedural languages, which allows the users to write their own code in any of the languages and it can be executed by PostgreSQL database server. These procedural languages are - PL/pgSQL, PL/Tcl, PL/Perl and PL/Python. Besides, other non-standard procedural languages like PL/PHP, PL/V8, PL/Ruby, PL/Java, etc., are also supported.

2. PostgreSQL – Environment Setup

To start understanding the PostgreSQL basics, first let us install the PostgreSQL. This chapter explains about installing the PostgreSQL on Linux, Windows and Mac OS platforms.

Installing PostgreSQL on Linux/Unix

Follow the given steps to install PostgreSQL on your Linux machine. Make sure you are logged in as **root** before you proceed for the installation.

- Pick the version number of PostgreSQL you want and, as exactly as possible, the platform you want from EnterpriseDB

- I downloaded **postgresql-9.2.4-1-linux-x64.run** for my 64-bit CentOS-6 machine. Now, let us execute it as follows:

```
[root@host]# chmod +x postgresql-9.2.4-1-linux-x64.run
[root@host]# ./postgresql-9.2.4-1-linux-x64.run
-----------------------------------------------------------------------
Welcome to the PostgreSQL Setup Wizard.
-----------------------------------------------------------------------
Please specify the directory where PostgreSQL will be installed.
Installation Directory [/opt/PostgreSQL/9.2]:
```

- Once you launch the installer, it asks you a few basic questions like location of the installation, password of the user who will use database, port number, etc. So keep all of them at their default values except password, which you can provide password as per your choice. It will install PostgreSQL at your Linux machine and will display the following message:

```
Please wait while Setup installs PostgreSQL on your computer.
 Installing
 0% ______________ 50% ______________ 100%
 ########################################
-----------------------------------------------------------------------
Setup has finished installing PostgreSQL on your computer.
```

- Follow the following post-installation steps to create your database:

```
[root@host]# su - postgres
Password:
bash-4.1$ createdb testdb
bash-4.1$ psql testdb
psql (8.4.13, server 9.2.4)
test=#
```

- You can start/restart postgres server in case it is not running, using the following command:

```
[root@host]# service postgresql restart
Stopping postgresql service:                               [  OK  ]
Starting postgresql service:                               [  OK  ]
```

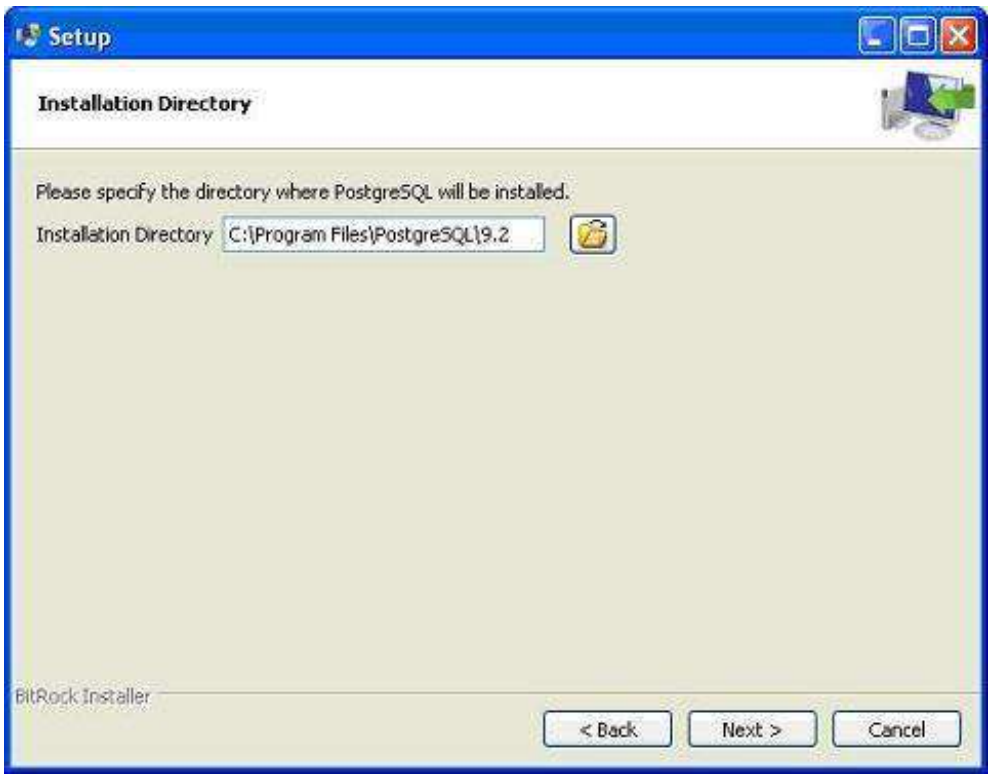
- If your installation was correct, you will have PotsgreSQL prompt **test=#** as shown above.

Installing PostgreSQL on Windows

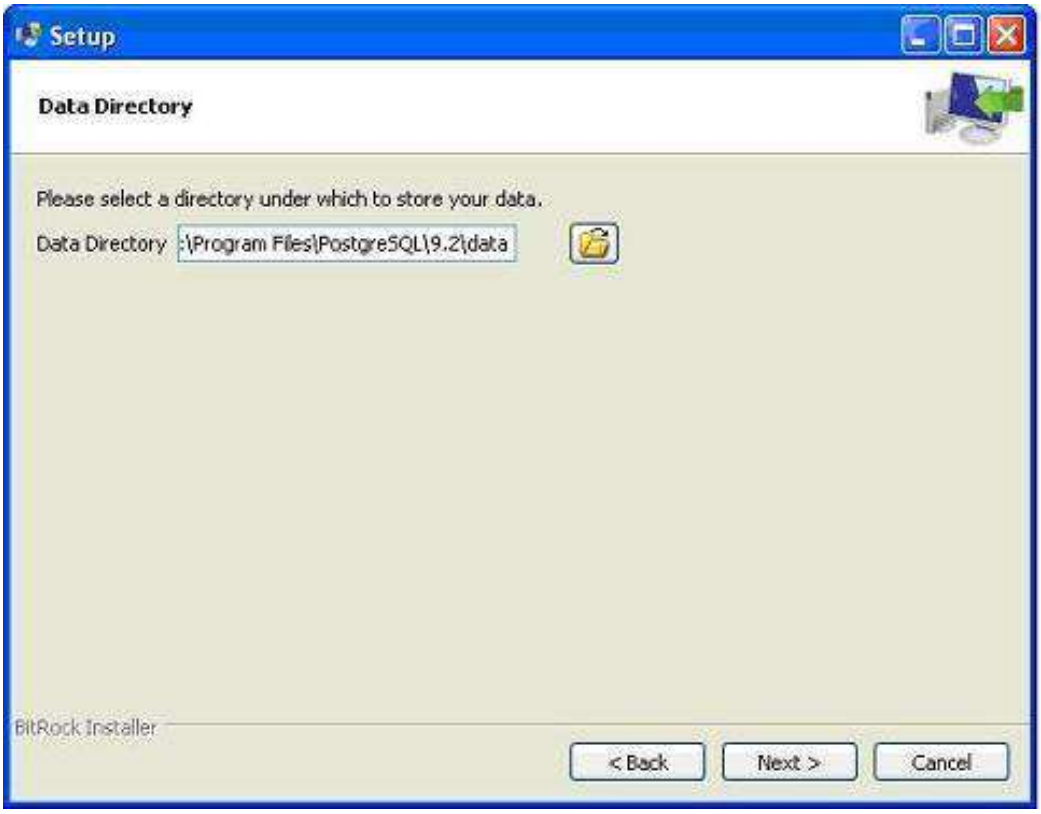
Follow the given steps to install PostgreSQL on your Windows machine. Make sure you have turned Third Party Antivirus off while installing.

- Pick the version number of PostgreSQL you want and, as exactly as possible, the platform you want from EnterpriseDB

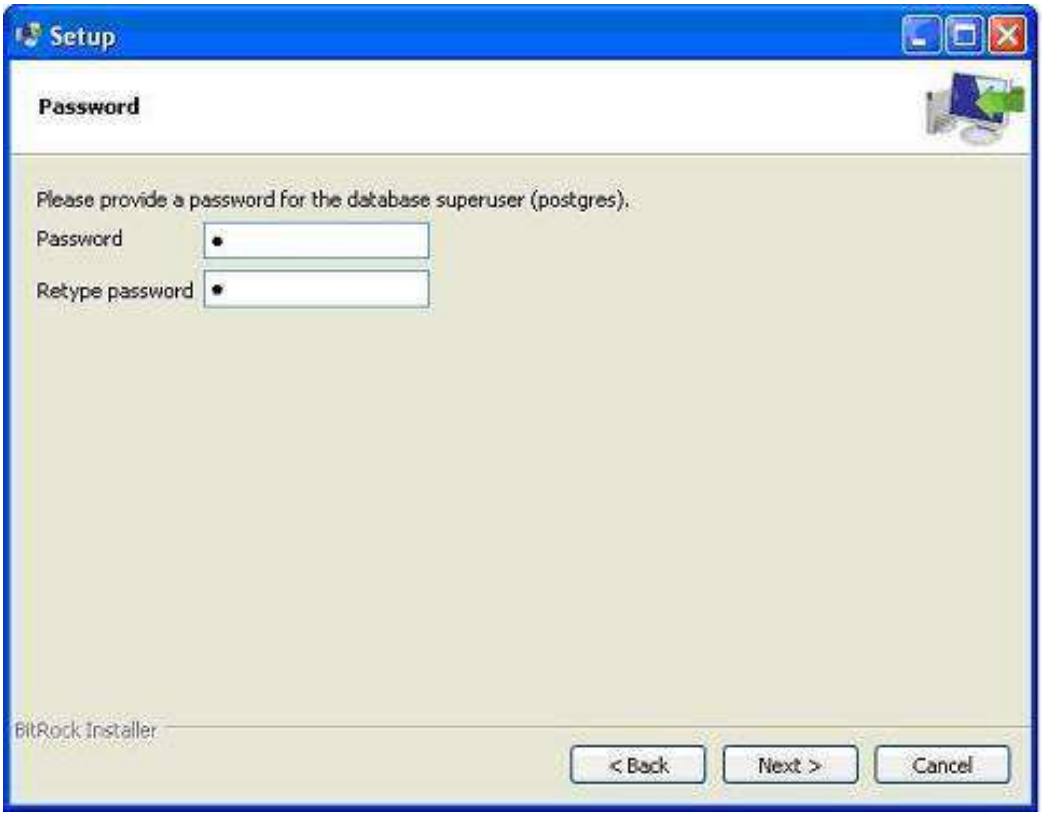
- I downloaded postgresql-9.2.4-1-windows.exe for my Windows PC running in 32-bit mode, so let us run **postgresql-9.2.4-1-windows.exe** as administrator to install PostgreSQL. Select the location where you want to install it. By default, it is installed within Program Files folder.



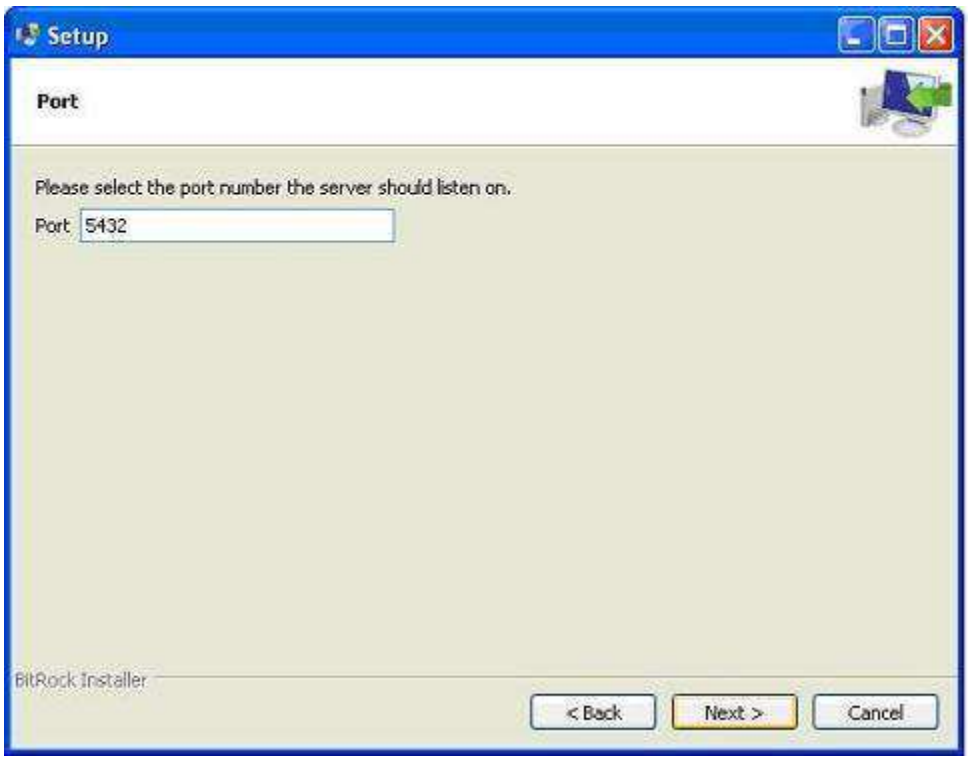
- The next step of the installation process would be to select the directory where your data would be stored. By default, it is stored under the "data" directory.



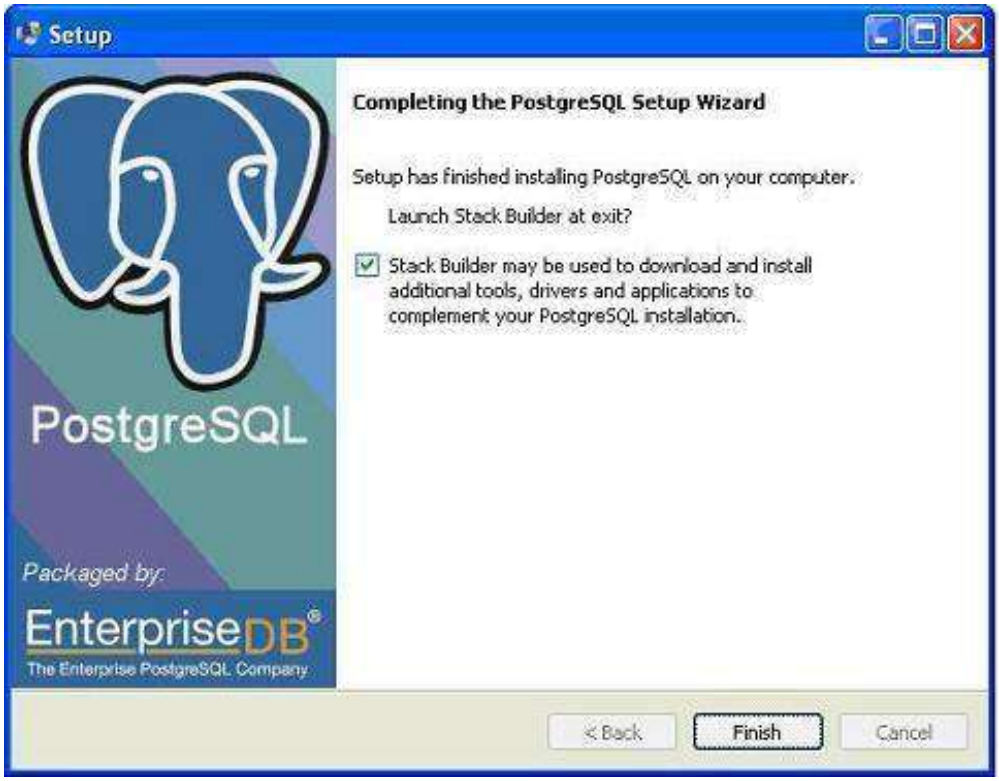
- Next, the setup asks for password, so you can use your favorite password.



- The next step; keep the port as default.



- In the next step, when asked for "Locale", I selected "English, United States".
- It takes a while to install PostgreSQL on your system. On completion of the installation process, you will get the following screen. Uncheck the checkbox and click the Finish button.



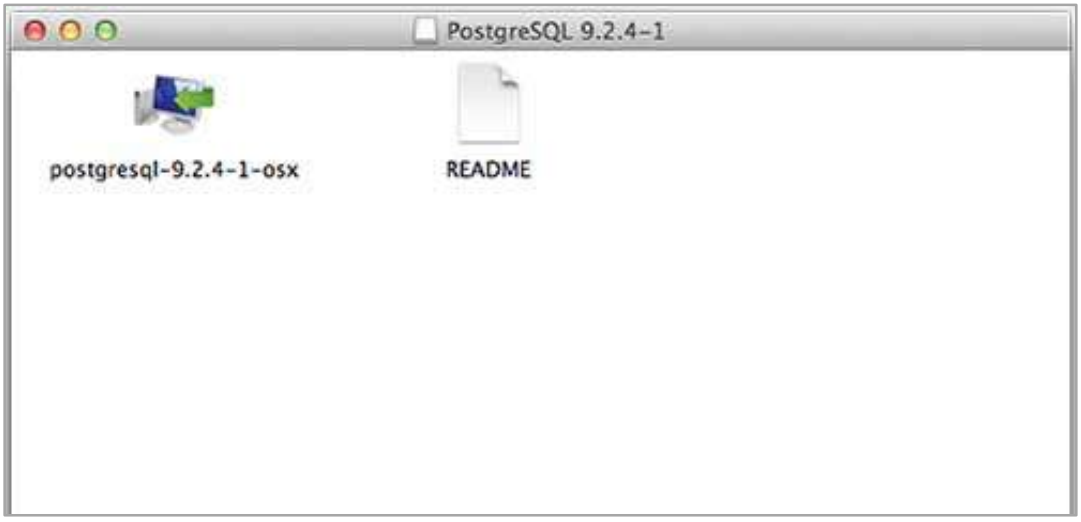
After the installation process is completed, you can access pgAdmin III, StackBuilder and PostgreSQL shell from your Program Menu under PostgreSQL 9.2.

Installing PostgreSQL on Mac

Follow the given steps to install PostgreSQL on your Mac machine. Make sure you are logged in as **administrator** before you proceed for the installation.

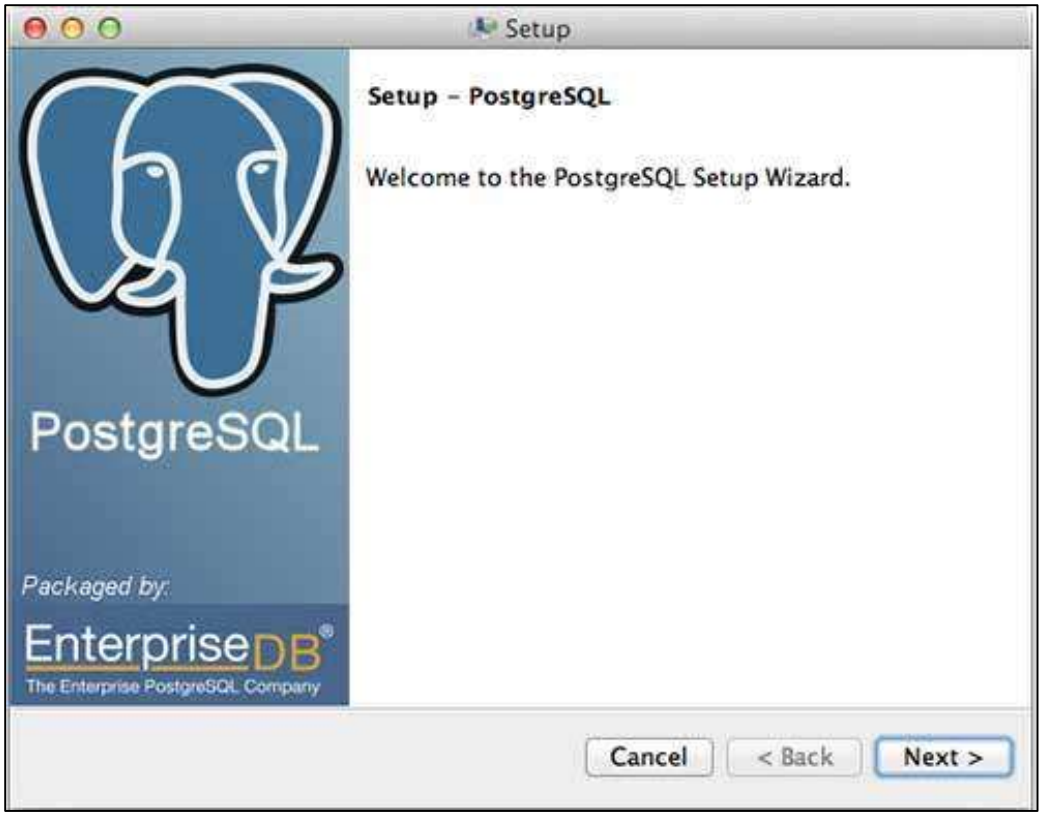
- Pick the latest version number of PostgreSQL for Mac OS available at EnterpriseDB

- I downloaded **postgresql-9.2.4-1-osx.dmg** for my Mac OS running with OS X version 10.8.3. Now, let us open the dmg image in finder and just double click it, which will give you PostgreSQL installer in the following window:



- Next, click the **postgres-9.2.4-1-osx** icon, which will give a warning message. Accept the warning and proceed for further installation. It will ask for the administrator password as seen in the following window:

- Enter the password, proceed for the installation, and after this step, restart your Mac machine. If you do not see the following window, start your installation once again.



- Once you launch the installer, it asks you a few basic questions like location of the installation, password of the user who will use database, port number etc. Therefore, keep all of them at their default values except the password, which you can provide as per your choice. It will install PostgreSQL in your Mac machine in the Application folder which you can check:

- Now, you can launch any of the program to start with. Let us start with SQL Shell. When you launch SQL Shell, just use all the default values it displays except, enter your password, which you had selected at the time of installation. If everything goes fine, then you will be inside postgres database and a **postgress#** prompt will be displayed as shown below:

Congratulations!!! Now you have your environment ready to start with PostgreSQL database programming.

3. PostgreSQL – Syntax

This chapter provides a list of the PostgreSQL SQL commands, followed by the precise syntax rules for each of these commands. This set of commands is taken from the psql command-line tool. Now that you have Postgres installed, open the psql as:

Program Files > PostgreSQL 9.2 > SQL Shell(psql).

Using psql, you can generate a complete list of commands by using the \help command. For the syntax of a specific command, use the following command:

```
postgres-# \help <command_name>
```

The SQL Statement

An SQL statement is comprised of tokens where each token can represent either a keyword, identifier, quoted identifier, constant, or special character symbol. The table given below uses a simple SELECT statement to illustrate a basic, but complete, SQL statement and its components.

	SELECT	id, name	FROM	states
Token Type	Keyword	Identifiers	Keyword	Identifier
Description	Command	Id and name columns	Clause	Table name

PostgreSQL SQL commands

ABORT

Abort the current transaction.

```
ABORT [ WORK | TRANSACTION ]
```

ALTER AGGREGATE

Change the definition of an aggregate function.

```
ALTER AGGREGATE name ( type ) RENAME TO new_name

ALTER AGGREGATE name ( type ) OWNER TO new_owner
```

ALTER CONVERSION

Change the definition of a conversion.

```
ALTER CONVERSION name RENAME TO new_name
ALTER CONVERSION name OWNER TO new_owner
```

ALTER DATABASE

Change a database specific parameter.

```
ALTER DATABASE name SET parameter { TO | = } { value | DEFAULT }
ALTER DATABASE name RESET parameter
ALTER DATABASE name RENAME TO new_name
ALTER DATABASE name OWNER TO new_owner
```

ALTER DOMAIN

Change the definition of a domain specific parameter.

```
ALTER DOMAIN name { SET DEFAULT expression | DROP DEFAULT }
ALTER DOMAIN name { SET | DROP } NOT NULL
ALTER DOMAIN name ADD domain_constraint
ALTER DOMAIN name DROP CONSTRAINT constraint_name [ RESTRICT | CASCADE ]
ALTER DOMAIN name OWNER TO new_owner
```

ALTER FUNCTION

Change the definition of a function.

```
ALTER FUNCTION name ( [ type [, ...] ] ) RENAME TO new_name
ALTER FUNCTION name ( [ type [, ...] ] ) OWNER TO new_owner
```

ALTER GROUP

Change a user group.

```
ALTER GROUP groupname ADD USER username [, ... ]
ALTER GROUP groupname DROP USER username [, ... ]
ALTER GROUP groupname RENAME TO new_name
```

ALTER INDEX

Change the definition of an index.

```
ALTER INDEX name OWNER TO new_owner
ALTER INDEX name SET TABLESPACE indexspace_name
ALTER INDEX name RENAME TO new_name
```

ALTER LANGUAGE

Change the definition of a procedural language.

```
ALTER LANGUAGE name RENAME TO new_name
```

ALTER OPERATOR

Change the definition of an operator.

```
ALTER OPERATOR name ( { lefttype | NONE } , { righttype | NONE } )
OWNER TO new_owner
```

ALTER OPERATOR CLASS

Change the definition of an operator class.

```
ALTER OPERATOR CLASS name USING index_method RENAME TO new_name
ALTER OPERATOR CLASS name USING index_method OWNER TO new_owner
```

ALTER SCHEMA

Change the definition of a schema.

```
ALTER SCHEMA name RENAME TO new_name
ALTER SCHEMA name OWNER TO new_owner
```

ALTER SEQUENCE

Change the definition of a sequence generator.

```
ALTER SEQUENCE name [ INCREMENT [ BY ] increment ]
[ MINVALUE minvalue | NO MINVALUE ]
[ MAXVALUE maxvalue | NO MAXVALUE ]
```

```
[ RESTART [ WITH ] start ] [ CACHE cache ] [ [ NO ] CYCLE ]
```

ALTER TABLE

Change the definition of a table.

```
ALTER TABLE [ ONLY ] name [ * ]
action [, ... ]
ALTER TABLE [ ONLY ] name [ * ]
RENAME [ COLUMN ] column TO new_column
ALTER TABLE name
RENAME TO new_name
```

Where *action* is one of the following lines:

```
ADD [ COLUMN ] column_type [ column_constraint [ ... ] ]
DROP [ COLUMN ] column [ RESTRICT | CASCADE ]
ALTER [ COLUMN ] column TYPE type [ USING expression ]
ALTER [ COLUMN ] column SET DEFAULT expression
ALTER [ COLUMN ] column DROP DEFAULT
ALTER [ COLUMN ] column { SET | DROP } NOT NULL
ALTER [ COLUMN ] column SET STATISTICS integer
ALTER [ COLUMN ] column SET STORAGE { PLAIN | EXTERNAL | EXTENDED | MAIN }
ADD table_constraint
DROP CONSTRAINT constraint_name [ RESTRICT | CASCADE ]
CLUSTER ON index_name
SET WITHOUT CLUSTER
SET WITHOUT OIDS
OWNER TO new_owner
SET TABLESPACE tablespace_name
```

ALTER TABLESPACE

Change the definition of a tablespace.

```
ALTER TABLESPACE name RENAME TO new_name
ALTER TABLESPACE name OWNER TO new_owner
```

ALTER TRIGGER

Change the definition of a trigger.

```
ALTER TRIGGER name ON table RENAME TO new_name
```

ALTER TYPE

Change the definition of a type.

```
ALTER TYPE name OWNER TO new_owner
```

ALTER USER

Change a database user account.

```
ALTER USER name [ [ WITH ] option [ ... ] ]
ALTER USER name RENAME TO new_name
ALTER USER name SET parameter { TO | = } { value | DEFAULT }
ALTER USER name RESET parameter
```

Where *option* can be:

```
[ ENCRYPTED | UNENCRYPTED ] PASSWORD 'password'
| CREATEDB | NOCREATEDB
| CREATEUSER | NOCREATEUSER
| VALID UNTIL 'abstime'
```

ANALYZE

Collect statistics about a database.

```
ANALYZE [ VERBOSE ] [ table [ (column [, ...] ) ] ]
```

BEGIN

Start a transaction block.

```
BEGIN [ WORK | TRANSACTION ] [ transaction_mode [, ...] ]
```

Where *transaction_mode* is one of:

```
ISOLATION LEVEL { SERIALIZABLE | REPEATABLE READ | READ COMMITTED
| READ UNCOMMITTED }
READ WRITE | READ ONLY
```

CHECKPOINT

Force a transaction log checkpoint.

```
CHECKPOINT
```

CLOSE

Close a cursor.

```
CLOSE name
```

CLUSTER

Cluster a table according to an index.

```
CLUSTER index_name ON table_name
CLUSTER table_name
CLUSTER
```

COMMENT

Define or change the comment of an object.

```
COMMENT ON
{
TABLE object_name |
COLUMN table_name.column_name |
AGGREGATE agg_name (agg_type) |
CAST (source_type AS target_type) |
CONSTRAINT constraint_name ON table_name |
CONVERSION object_name |
```

```
DATABASE object_name |
DOMAIN object_name |
FUNCTION func_name (arg1_type, arg2_type, ...) |
INDEX object_name |
LARGE OBJECT large_object_oid |
OPERATOR op (left_operand_type, right_operand_type) |
OPERATOR CLASS object_name USING index_method |
[ PROCEDURAL ] LANGUAGE object_name |
RULE rule_name ON table_name |
SCHEMA object_name |
SEQUENCE object_name |
TRIGGER trigger_name ON table_name |
TYPE object_name |
VIEW object_name
} IS 'text'
```

COMMIT

Commit the current transaction.

```
COMMIT [ WORK | TRANSACTION ]
```

COPY

Copy data between a file and a table.

```
COPY table_name [ ( column [, ...] ) ]
FROM { 'filename' | STDIN }
[ [ WITH ]
[ BINARY ]
[ OIDS ]
[ DELIMITER [ AS ] 'delimiter' ]
[ NULL [ AS ] 'null string' ]
[ CSV [ QUOTE [ AS ] 'quote' ]
[ ESCAPE [ AS ] 'escape' ]
[ FORCE NOT NULL column [, ...] ]
COPY table_name [ ( column [, ...] ) ]
```


```
TO { 'filename' | STDOUT }
[ [ WITH ]
[ BINARY ]
[ OIDS ]
[ DELIMITER [ AS ] 'delimiter' ]
[ NULL [ AS ] 'null string' ]
[ CSV [ QUOTE [ AS ] 'quote' ]
[ ESCAPE [ AS ] 'escape' ]
[ FORCE QUOTE column [, ...] ]
```

CREATE AGGREGATE

Define a new aggregate function.

```
CREATE AGGREGATE name (
BASETYPE = input_data_type,
SFUNC = sfunc,
STYPE = state_data_type
[ , FINALFUNC = ffunc ]
[ , INITCOND = initial_condition ]
)
```

CREATE CAST

Define a new cast.

```
CREATE CAST (source_type AS target_type)
```


```
WITH FUNCTION func_name (arg_types)
[ AS ASSIGNMENT | AS IMPLICIT ]
CREATE CAST (source_type AS target_type)
WITHOUT FUNCTION
[ AS ASSIGNMENT | AS IMPLICIT ]
```

CREATE CONSTRAINT TRIGGER

Define a new constraint trigger.

```
CREATE CONSTRAINT TRIGGER name
AFTER events ON
table_name constraint attributes
FOR EACH ROW EXECUTE PROCEDURE func_name ( args )
```

CREATE CONVERSION

Define a new conversion.

```
CREATE [DEFAULT] CONVERSION name
FOR source_encoding TO dest_encoding FROM func_name
```

CREATE DATABASE

Create a new database.

```
CREATE DATABASE name
[ [ WITH ] [ OWNER [=] db_owner ]
[ TEMPLATE [=] template ]
[ ENCODING [=] encoding ]
[ TABLESPACE [=] tablespace ] ]
```

CREATE DOMAIN

Define a new domain.

```
CREATE DOMAIN name [AS] data_type
[ DEFAULT expression ]
[ constraint [ ... ] ]
```

Where *constraint* is:

```
[ CONSTRAINT constraint_name ]
{ NOT NULL | NULL | CHECK (expression) }
```

CREATE FUNCTION

Define a new function.

```
CREATE [ OR REPLACE ] FUNCTION name ( [ [ arg_name ] arg_type [, ...] ] )
RETURNS ret_type
{ LANGUAGE lang_name
| IMMUTABLE | STABLE | VOLATILE
| CALLED ON NULL INPUT | RETURNS NULL ON NULL INPUT | STRICT
| [ EXTERNAL ] SECURITY INVOKER | [ EXTERNAL ] SECURITY DEFINER
| AS 'definition'
| AS 'obj_file', 'link_symbol'
} ...
[ WITH ( attribute [, ...] ) ]
```

CREATE GROUP

Define a new user group.

```
CREATE GROUP name [ [ WITH ] option [ ... ] ]
Where option can be:
SYSID gid
| USER username [, ...]
```

CREATE INDEX

Define a new index.

```
CREATE [ UNIQUE ] INDEX name ON table [ USING method ]
( { column | ( expression ) } [ opclass ] [, ...] )
[ TABLESPACE tablespace ]
[ WHERE predicate ]
```

CREATE LANGUAGE

Define a new procedural language.

```
CREATE [ TRUSTED ] [ PROCEDURAL ] LANGUAGE name
HANDLER call_handler [ VALIDATOR val_function ]
```

CREATE OPERATOR

Define a new operator.

```
CREATE OPERATOR name (
PROCEDURE = func_name
[, LEFTARG = left_type ] [, RIGHTARG = right_type ]
[, COMMUTATOR = com_op ] [, NEGATOR = neg_op ]
[, RESTRICT = res_proc ] [, JOIN = join_proc ]
[, HASHES ] [, MERGES ]
[, SORT1 = left_sort_op ] [, SORT2 = right_sort_op ]
[, LTCMP = less_than_op ] [, GTCMP = greater_than_op ]
)
```

CREATE OPERATOR CLASS

Define a new operator class.

```
CREATE OPERATOR CLASS name [ DEFAULT ] FOR TYPE data_type
USING index_method AS
{ OPERATOR strategy_number operator_name [ ( op_type, op_type ) ] [ RECHECK ]
| FUNCTION support_number func_name ( argument_type [, ...] )
| STORAGE storage_type
} [, ... ]
```

CREATE RULE

Define a new rewrite rule.

```
CREATE [ OR REPLACE ] RULE name AS ON event
TO table [ WHERE condition ]
DO [ ALSO | INSTEAD ] { NOTHING | command | ( command ; command ... ) }
```

CREATE SCHEMA

Define a new schema.

```
CREATE SCHEMA schema_name
[ AUTHORIZATION username ] [ schema_element [ ... ] ]
CREATE SCHEMA AUTHORIZATION username
[ schema_element [ ... ] ]
```

CREATE SEQUENCE

Define a new sequence generator.

```
CREATE [ TEMPORARY | TEMP ] SEQUENCE name
[ INCREMENT [ BY ] increment ]
[ MINVALUE minvalue | NO MINVALUE ]
[ MAXVALUE maxvalue | NO MAXVALUE ]
[ START [ WITH ] start ] [ CACHE cache ] [ [ NO ] CYCLE ]CREATE TABLE
```

Define a new table.

```
CREATE [ [ GLOBAL | LOCAL ] { TEMPORARY | TEMP } ] TABLE table_name (
{ column_name data_type [ DEFAULT default_expr ] [ column_constraint [ ... ] ]
| table_constraint
| LIKE parent_table [ { INCLUDING | EXCLUDING } DEFAULTS ] } [, ... ]
)
[ INHERITS ( parent_table [, ... ] ) ]
[ WITH OIDS | WITHOUT OIDS ]
[ ON COMMIT { PRESERVE ROWS | DELETE ROWS | DROP } ]
[ TABLESPACE tablespace ]
```

Where column_constraint is:

```
[ CONSTRAINT constraint_name ]
{ NOT NULL |
NULL |
UNIQUE [ USING INDEX TABLESPACE tablespace ] |
PRIMARY KEY [ USING INDEX TABLESPACE tablespace ] |
CHECK (expression) |
REFERENCES ref_table [ ( ref_column ) ]
[ MATCH FULL | MATCH PARTIAL | MATCH SIMPLE ]
[ ON DELETE action ] [ ON UPDATE action ] }
[ DEFERRABLE | NOT DEFERRABLE ] [ INITIALLY DEFERRED | INITIALLY IMMEDIATE ]
```

And table_constraint is:

```
[ CONSTRAINT constraint_name ]
{ UNIQUE ( column_name [, ... ] ) [ USING INDEX TABLESPACE tablespace ] |
PRIMARY KEY ( column_name [, ... ] ) [ USING INDEX TABLESPACE tablespace ] |
CHECK ( expression ) |
FOREIGN KEY ( column_name [, ... ] )
REFERENCES ref_table [ ( ref_column [, ... ] ) ]
[ MATCH FULL | MATCH PARTIAL | MATCH SIMPLE ]
[ ON DELETE action ] [ ON UPDATE action ] }
[ DEFERRABLE | NOT DEFERRABLE ] [ INITIALLY DEFERRED | INITIALLY IMMEDIATE ]
```

End of ebook preview
If you liked what you saw…
Buy it from our store @ **https://store.tutorialspoint.com**