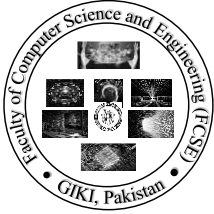CS232L – Database Management System

INTRO TO DBMS AND POSTGRESQL

Ghulam Ishaq Khan Institute of Engineering
Sciences

Faculty of Computer Science & Engineering

CS232L – Database Management system Lab

Lab 1 – Intro to DBMS and PostgreSQL

Objective

The objective of this session is to get basic introduction to DBMS and installation guide to PostgreSQL0. Also we will cover basic DDL and DML commands.

Instructions

- Open the handout/lab manual in front of a computer in the lab during the session.
- Practice each new command by completing the examples and exercise.
- Turn-in the answers for all the exercise problems as your lab report.
- When answering problems, indicate the commands you entered and the output displayed.

What is DBMS?

A database management system (DBMS) is system software for creating and managing databases. The DBMS provides users and programmers with a systematic way to create, retrieve, update and manage data. A DBMS makes it possible for end users to create, read, update and delete data in a database. The DBMS essentially serves as an interface between the database and end users or application programs, ensuring that data is consistently organized and remains easily accessible.

The DBMS manages three important things: the data, the database engine that allows data to be accessed, locked and modified -- and the database schema, which defines the database's logical structure. These three foundational elements help provide concurrency, security, data integrity and uniform administration procedures. Typical database administration tasks supported by the DBMS include change management, performance monitoring/tuning and backup and recovery. Many database management systems are also responsible for automated rollbacks, restarts and recovery as well as the logging and auditing of activity. The DBMS is perhaps most useful for providing a centralized view of data that can be accessed by multiple users, from multiple locations, in a controlled manner.

The DBMS can offer both logical and physical data independence. That means it can protect users and applications from needing to know where data is stored or having to be concerned about changes to the physical structure of data (storage and hardware). As long as programs use the application programming interface (API) for the database that is provided by the DBMS, developers won't have to modify programs just because changes have been made to the database.
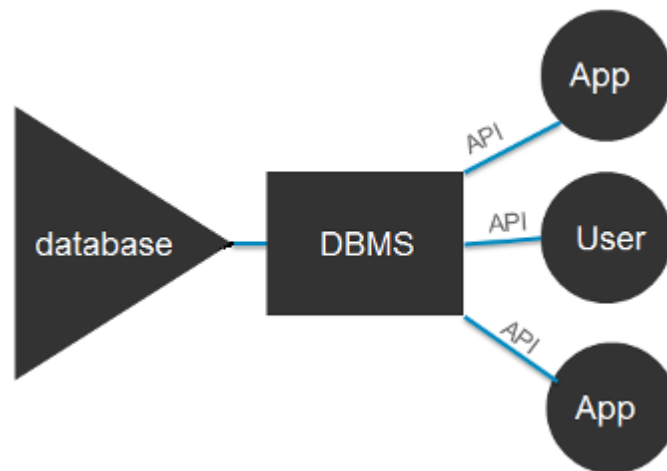

Figure 1. General Structure of DBMS

Some of the DBMS examples include:
- MySQL
- PostgreSQL
- SQL Server
- Oracle
- dBASE
- FoxPro

What is PostgreSQL

Let's start with a simple question: what is PostgreSQL?
PostgreSQL is an advanced, enterprise-class, and open-source relational database system.
PostgreSQL supports both SQL (relational) and JSON (non-relational) querying.
PostgreSQL is a highly stable database backed by more than 20 years of development by the open-source community.
PostgreSQL is used as a primary database for many web applications as well as mobile and analytics applications.

PostgreSQL's community pronounces PostgreSQL as /ˈpoʊstɡrɛs ˌkjuː ˈɛl/.

History of PostgreSQL

The PostgreSQL project started in 1986 at [Berkeley Computer Science Department](#), University of California.

The project was originally named POSTGRES, in reference to the older Ingres database which also developed at Berkeley. The goal of the POSTGRES project was to add the minimal features needed to support multiple data types.

In 1996, the POSTGRES project was renamed to PostgreSQL to clearly illustrate its support for SQL. Today, PostgreSQL is commonly abbreviated as Postgres.

Since then, the PostgreSQL Global Development Group, a dedicated community of contributors continues to make the releases of the open-source and free database project.

Originally, PostgreSQL was designed to run on UNIX-like platforms. And then, PostgreSQL was evolved run on various platforms such as Windows, macOS, and Solaris.

Common Use cases of PostgreSQL

The following are the common use cases of PostgreSQL.

1) A robust database in the LAPP stack

LAPP stands for **L**inux, **A**pache, **P**ostgreSQL, and **P**HP (or Python and Perl). PostgreSQL is primarily used as a robust back-end database that powers many dynamic websites and web applications.

2) General purpose transaction database

Large corporations and startups alike use PostgreSQL as primary databases to support their applications and products.

3) Geospatial database

PostgreSQL with the PostGIS extension supports geospatial databases for geographic information systems (GIS).

Language support

PostgreSQL support most popular programming languages:

- Python
- Java
- C#
- C/C+
- Ruby
- JavaScript (Node.js)
- Perl
- Go
- Tcl

PostgreSQL feature highlights

PostgreSQL has many advanced features that other enterprise-class database management systems offer, such as:

- User-defined types
- Table inheritance
- Sophisticated locking mechanism
- Foreign key referential integrity
- Views, rules, subquery
- Nested transactions (savepoints)
- Multi-version concurrency control (MVCC)
- Asynchronous replication

The recent versions of PostgreSQL support the following features:

- Native Microsoft Windows Server version
- Tablespaces
- Point-in-time recovery

And more new features are added in each new release.

PostgreSQL is designed to be extensible. PostgreSQL allows you to define your own data types, index types, functional languages, etc.

If you don't like any part of the system, you can always develop a custom plugin to enhance it to meet your requirements e.g., adding a new optimizer.

Who uses PostgreSQL

Many companies have built products and solutions based on PostgreSQL. Some featured companies are Apple, Fujitsu, Red Hat, Cisco, Juniper Network, Instagram, etc.

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 macOS, Solaris, and Windows.

Install PostgreSQL on Windows

Since version 8.0, PostgreSQL offers an installer for Windows systems that makes the installation process easier and faster. For development purposes, we will install PostgreSQL version 12 on Windows 10.

1) Download PostgreSQL Installer for Windows

First, you need to go to the download page of PostgreSQL installers on the EnterpriseDB.

Second, click the download link as shown below:

Version	Linux x86-64	Linux x86-32	Mac OS X	Windows x86-64	Windows x86-32
12.3	N/A	N/A	Download	Download	N/A
11.8	N/A	N/A	Download	Download	N/A
10.13	Download	Download	Download	Download	Download
9.6.18	Download	Download	Download	Download	Download
9.5.22	Download	Download	Download	Download	Download
9.4.26 (Not Supported)	Download	Download	Download	Download	Download
9.3.25 (Not Supported)	Download	Download	Download	Download	Download

It will take a few minutes to complete the download.

### 2) Install PostgreSQL on Window step by step

To install PostgreSQL on Windows, you need to have administrator privileges.

Step 1. 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.
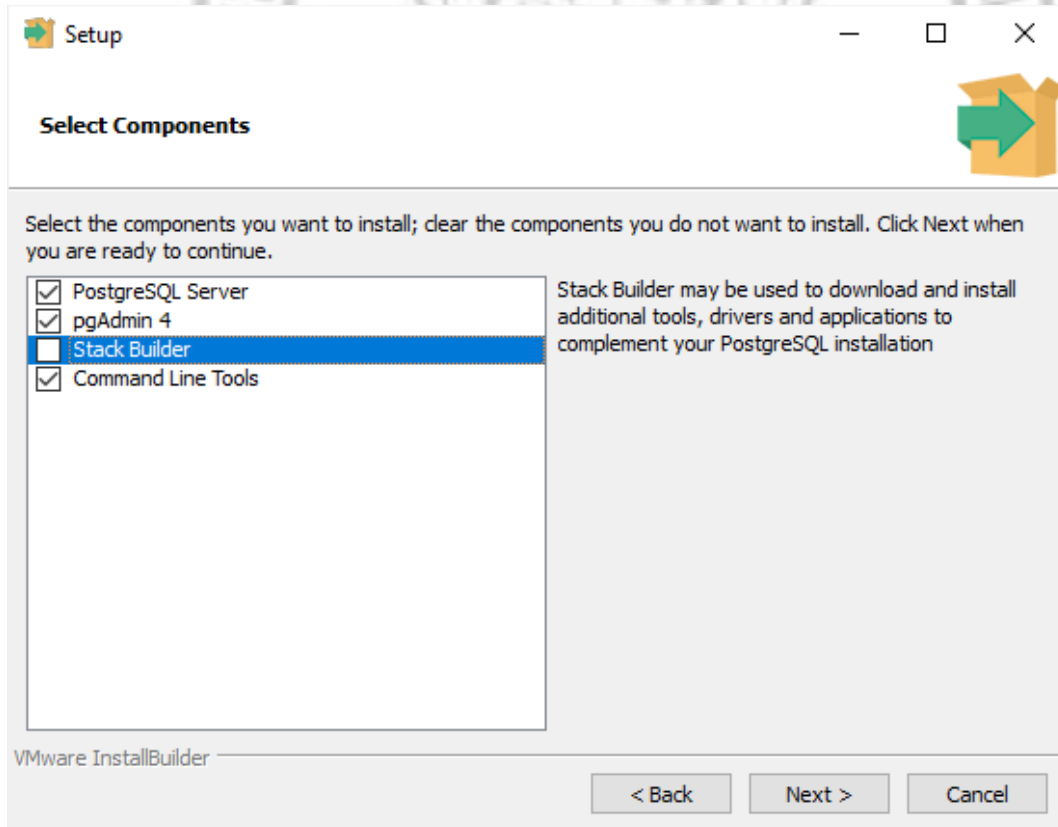
Step 2. Click the Next button

Step 3. Specify installation folder, choose your own or keep the default folder suggested by PostgreSQL installer and click the Next button


-
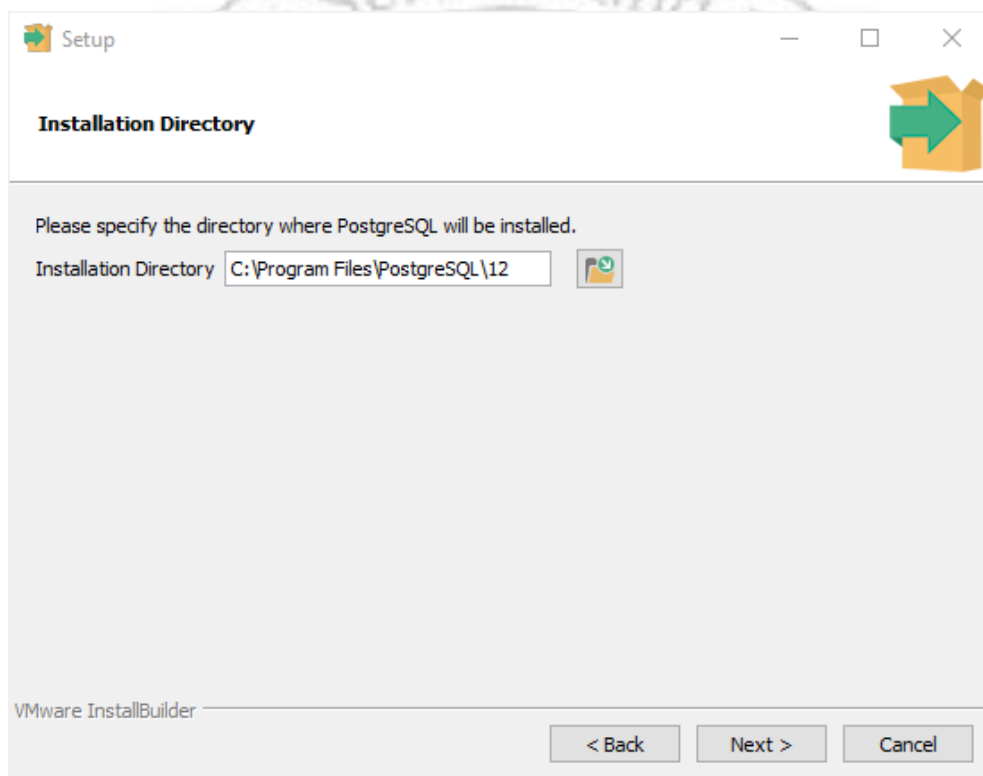
Step 4. Select software components to install:

- The PostgreSQL Server to install the PostgreSQL database server
- pgAdmin 4 to install the PostgreSQL database GUI management tool.
- Command Line Tools to install command-line tools such as psql, pg_restore, etc. These tools allow you to interact with the PostgreSQL database server using the command-line interface.
- Stack Builder provides a GUI that allows you to download and install drivers that work with PostgreSQL.

For the tutorial on this website, you don't need to install Stack Builder so feel free to uncheck it and click the Next button to select the data directory:


Step 5. Select the database directory to store the data or accept the default folder. And click the Next button to go to the next step:
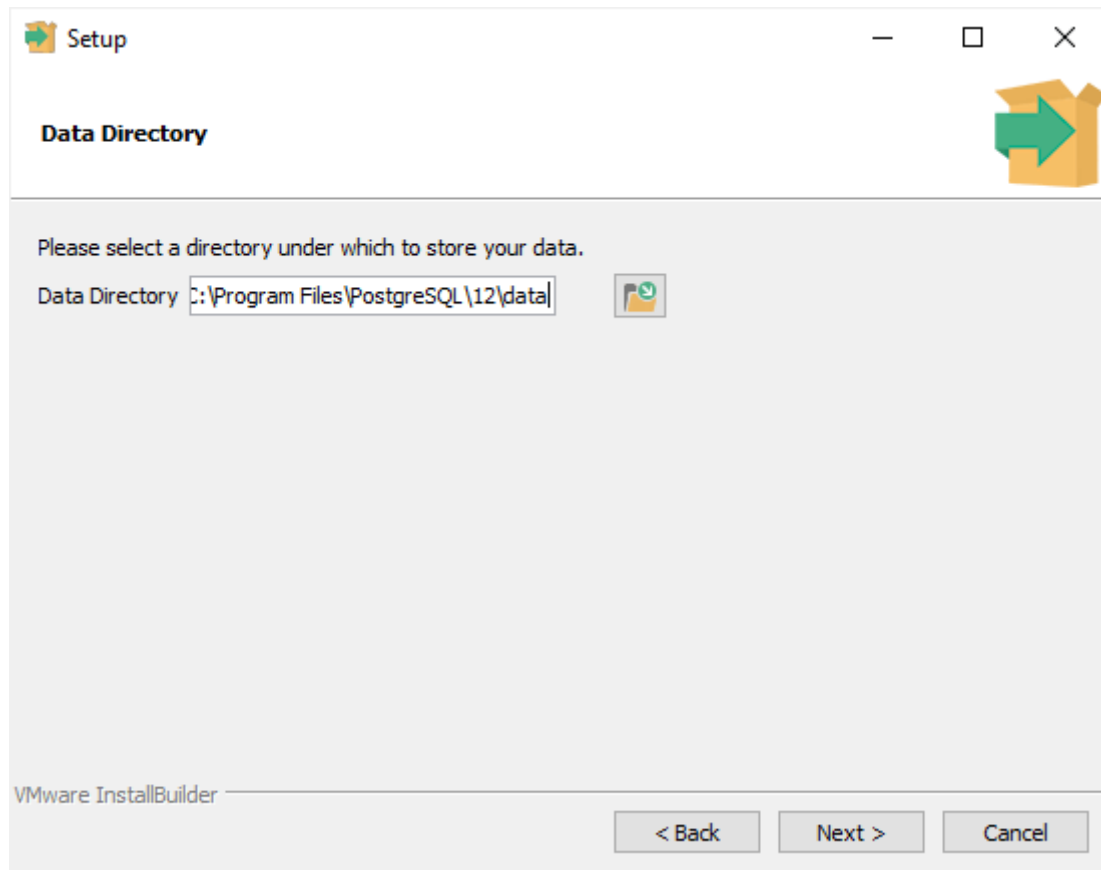
Step 6. Enter the password for the database superuser (postgres)

PostgreSQL runs as a service in the background under a service account named postgres. If you already created a service account with the name postgres, you need to provide the password of that account in the following window.

After entering the password, you need to retype it to confirm and click the Next button:
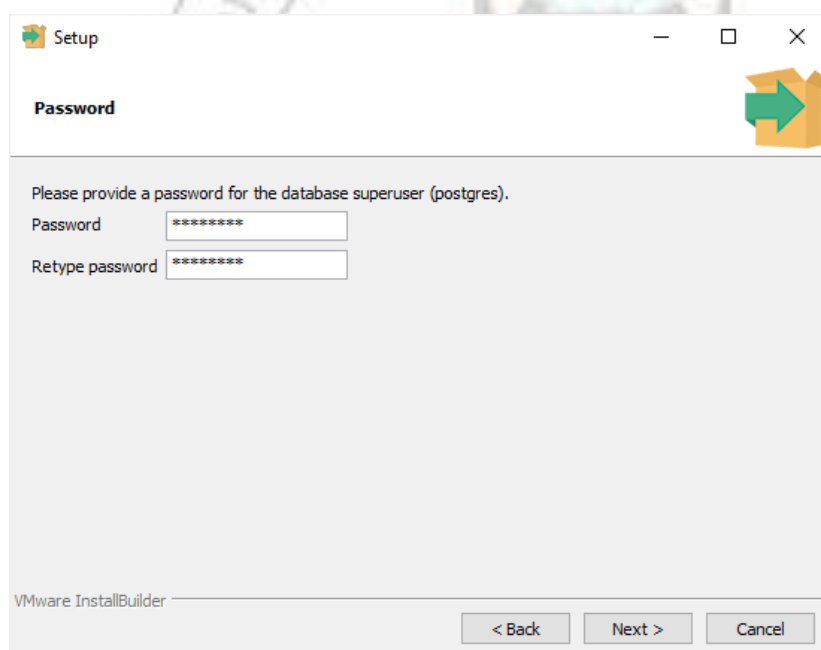
Step 7. Enter a port number on which the PostgreSQL database server will listen. The default port of PostgreSQL is 5432. You need to make sure that no other applications are using this port.
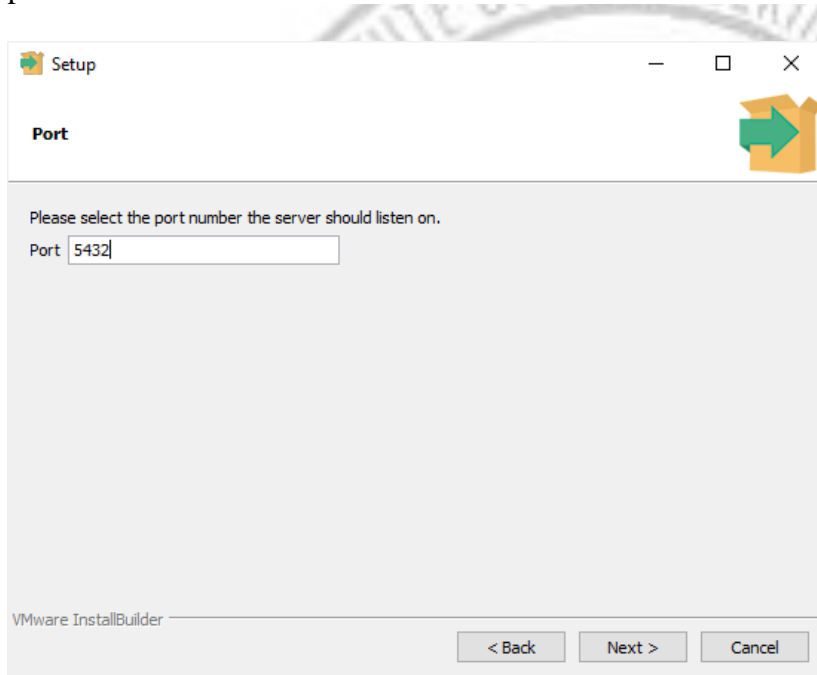

Step 8. Choose the default locale used by the PostgreSQL database. If you leave it as default locale, PostgreSQL will use the operating system locale. After that click the Next button.
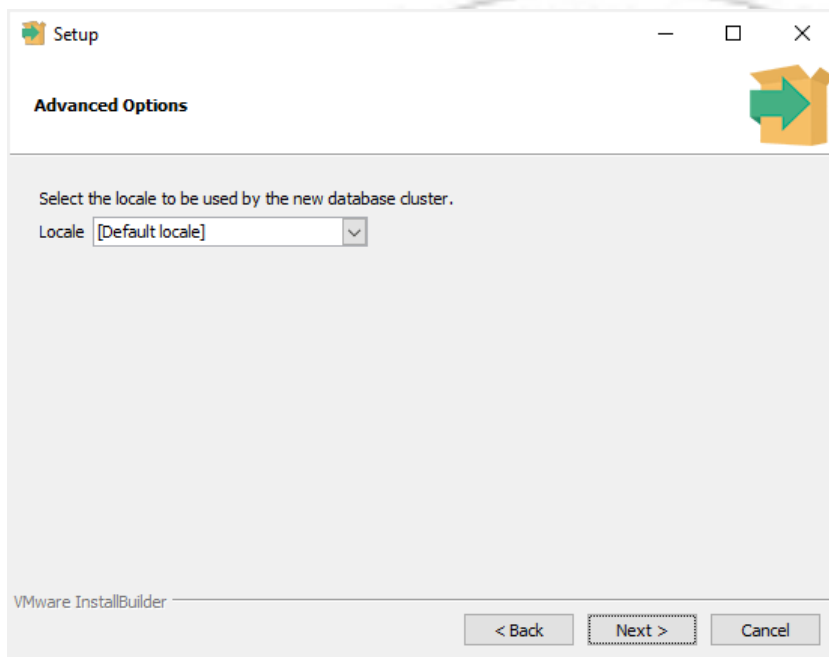
Step 9. The setup wizard will show the summary information of PostgreSQL. You need to review it and click the Next button if everything is correct. Otherwise, you need to click the Back button to change the configuration accordingly.
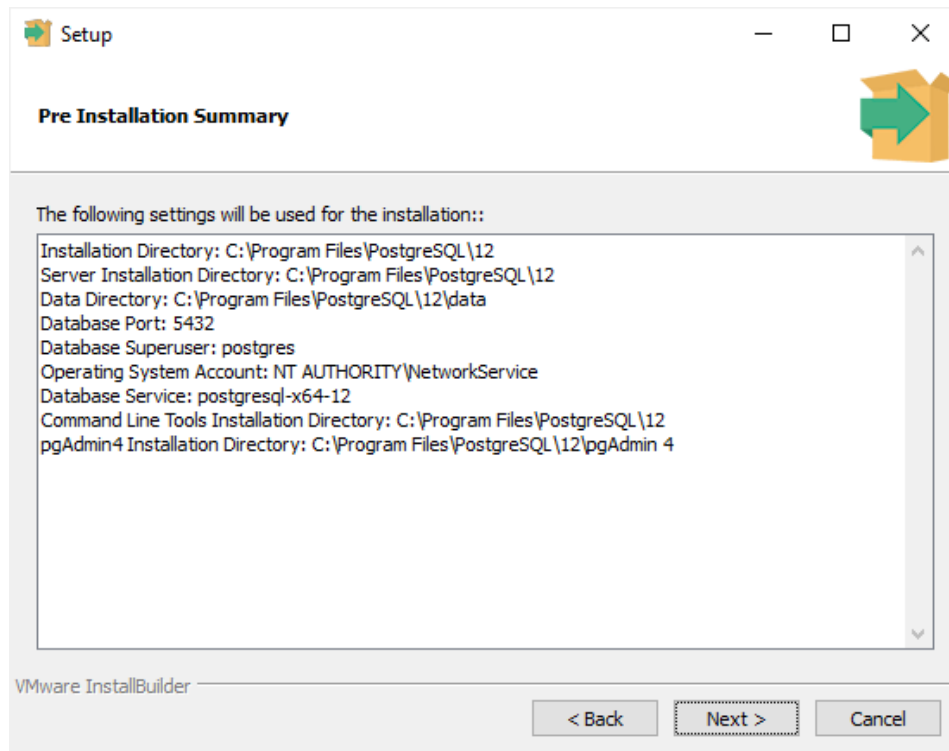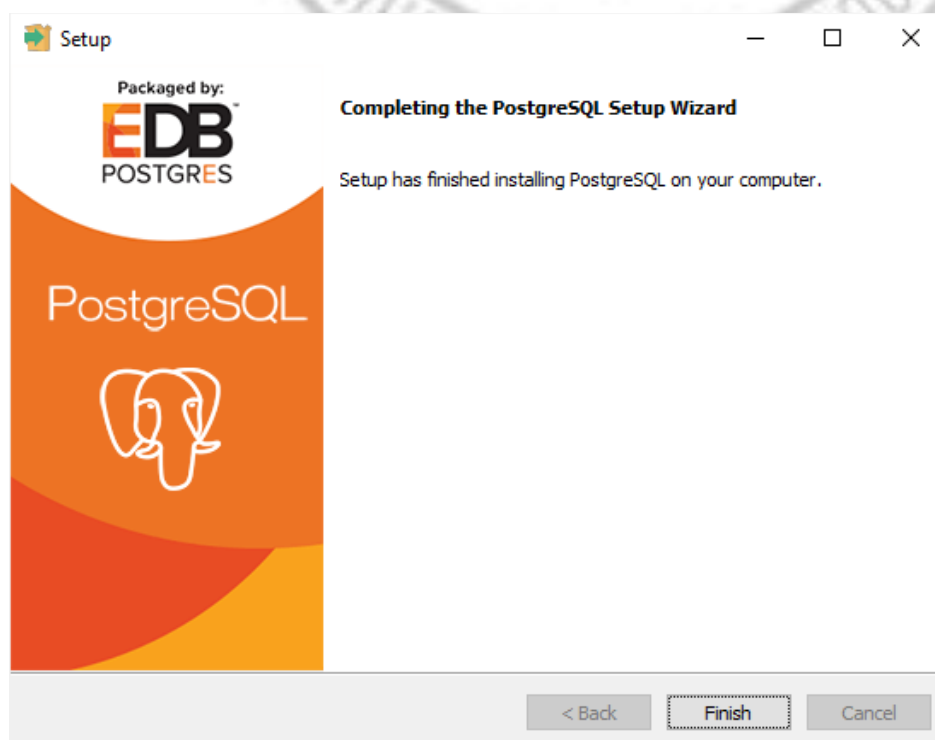
Now, you're ready to install PostgreSQL on your computer. Click the **Next** button to begin installing PostgreSQL.
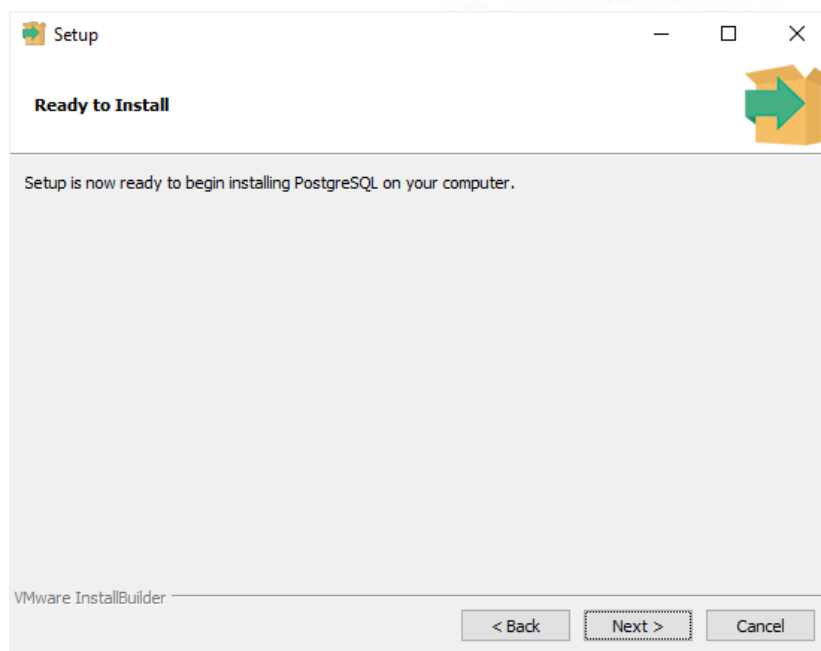
The installation may take a few minutes to complete.
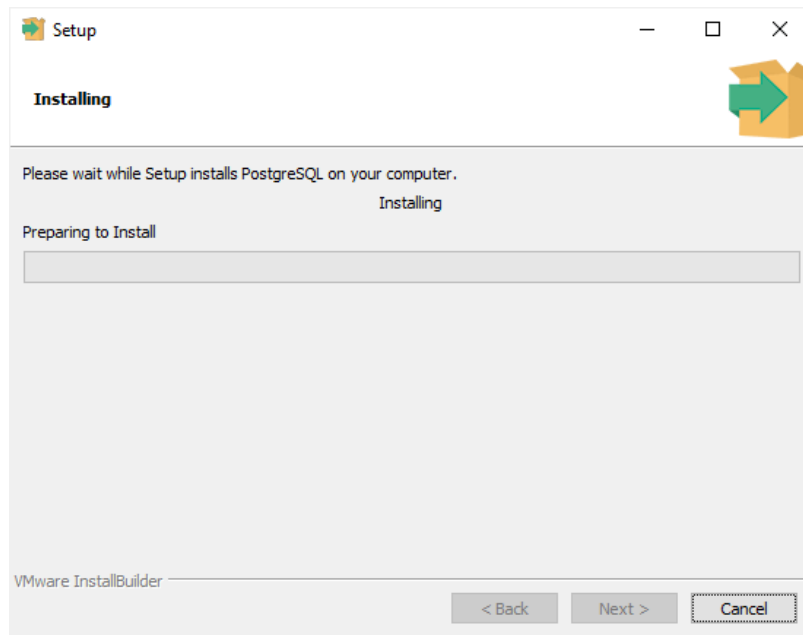
Step 10. Click the **Finish** button to complete the PostgreSQL installation.

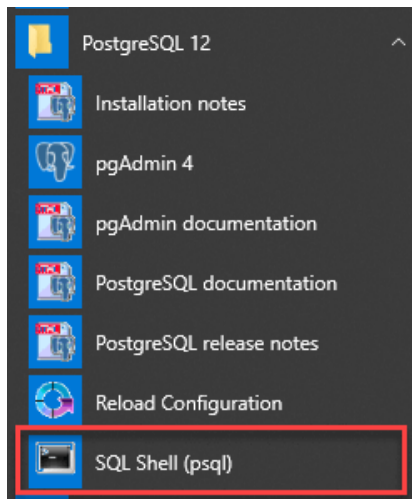**3) Verify the Installation**

There are several ways to verify the PostgreSQL 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 psql program.

First, click the psql application to launch it. The psql command-line program will display.

Second, enter all the necessary information such as the server, database, port, username, and password. To accept the default, you can press **Enter**. Note that you should provide the password that you entered during installing the PostgreSQL.

```
Server [localhost]:
Database [postgres]:
Port [5432]:
Username [postgres]:
Password for user postgres:
psql (12.3)
WARNING: Console code page (437) differs from Windows code page (1252)
        8-bit characters might not work correctly. See psql reference
        page "Notes for Windows users" for details.
Type "help" for help.

postgres=#
```

Code language: Shell Session (shell)

Third, issue the command SELECT version(); you will see the following output:

Congratulation! you've successfully installed PostgreSQL database server on your local system.

Connect To a PostgreSQL Database Server

When you installed the PostgreSQL database server, the PostgreSQL installer also installed some useful tools for working with the PostgreSQL database server. In this tutorial, you will learn how to connect to the PostgreSQL database server via the following tools:

- psql – a terminal-based front-end to PostgreSQL database server.
- pgAdmin – a web-based front-end to PostgreSQL database server.

1) Connect to PostgreSQL database server using psql
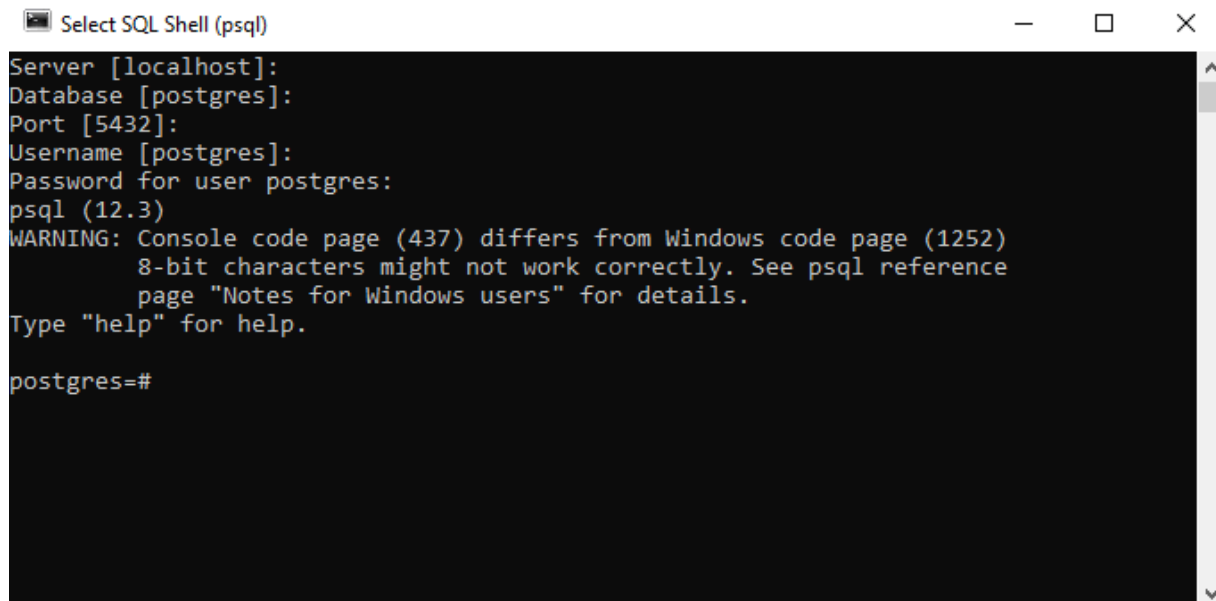
psql is an interactive terminal program provided by PostgreSQL. It allows you to interact with the PostgreSQL database server such as executing SQL statements and managing database objects.

The following steps show you how to connect to the PostgreSQL database server via the *psql* program:

First, launch the **psql** program and connect to the PostgreSQL Database Server using the **postgres** user:
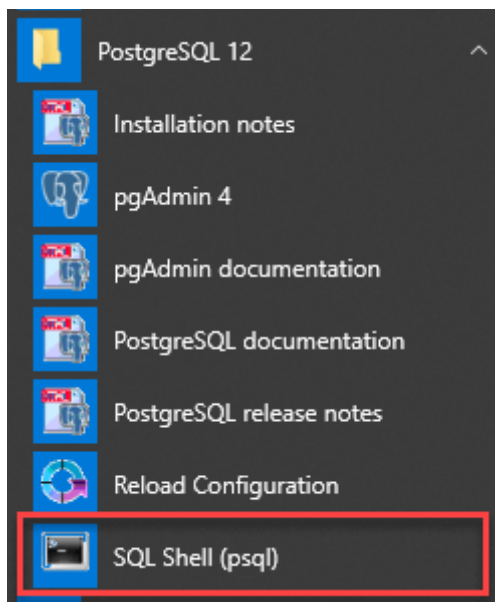
Second, enter all the information such as Server, Database, Port, Username, and Password. If you press Enter, the program will use the default value specified in the square bracket [] and move the cursor to the new line. For example, localhost is the default database server. In the step for entering the password for user postgres, you need to enter the password the user postgres that you chose during the PostgreSQL installation.

Third, interact with the PostgreSQL Database Server by issuing an SQL statement. The following statement returns the current version of PostgreSQL:

SELECT version();

Code language: SQL (Structured Query Language) (sql)

Please do not forget to end the statement with a semicolon (;). After pressing **Enter**, psql will return the current PostgreSQL version on your system.

2) Connect to PostgreSQL database server using pgAdmin

The second way to connect to a database is by using a pgAdmin application. The pgAdmin application allows you to interact with the PostgreSQL database server via an intuitive user interface.

The following illustrates how to connect to a database using pgAdmin GUI application:

First, launch the pgAdmin application.
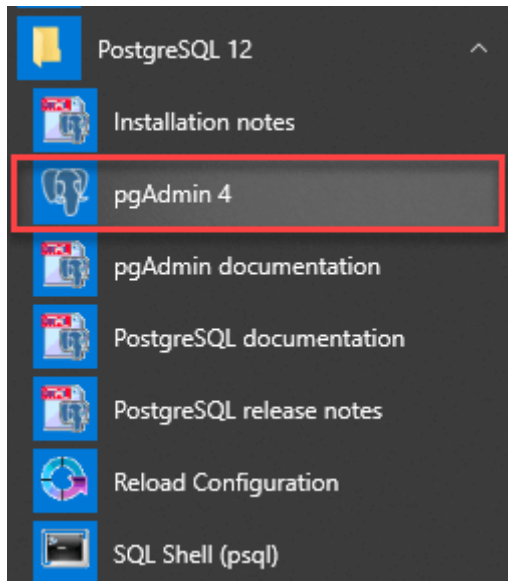
The pgAdmin application will launch on the web browser as shown in the following picture:
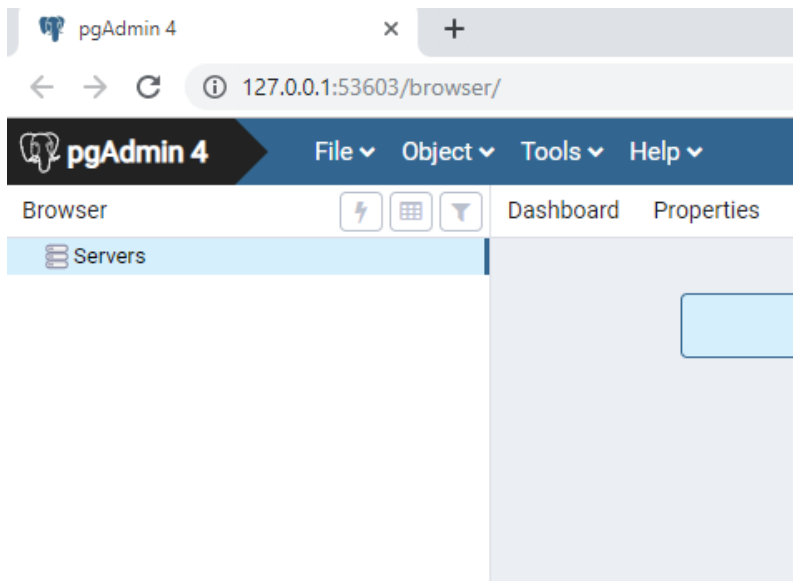
Second, right-click the Servers node and select **Create > Server…** menu to create a server
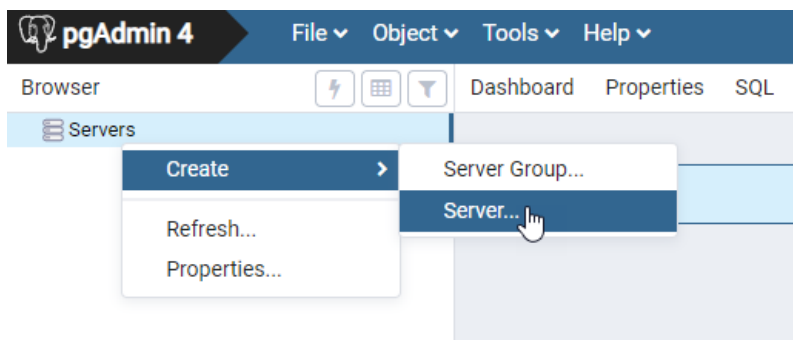
Third, enter the server name e.g., **PostgreSQL** and click the **Connection** tab:

Fourth, enter the host and password for the **postgres** user and click the **Save** button:

Fifth, click on the Servers node to expand the server. By default, PostgreSQL has a database named postgres as shown below:
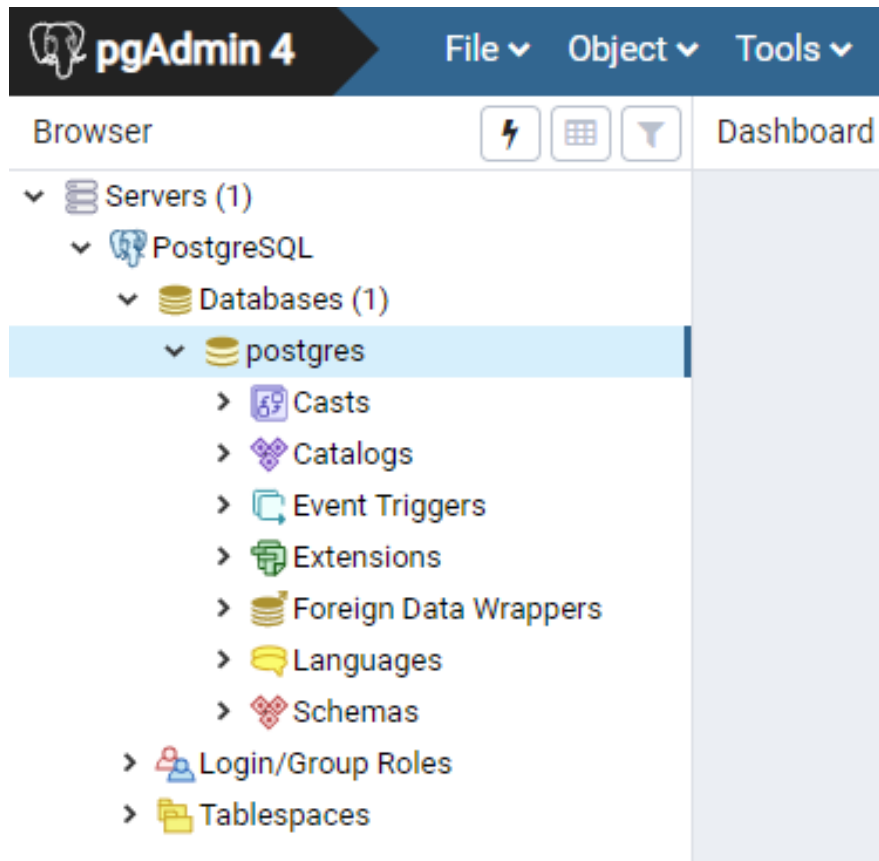
Sixth, open the query tool by choosing the menu item **Tool > Query Tool** or click the lightning icon.
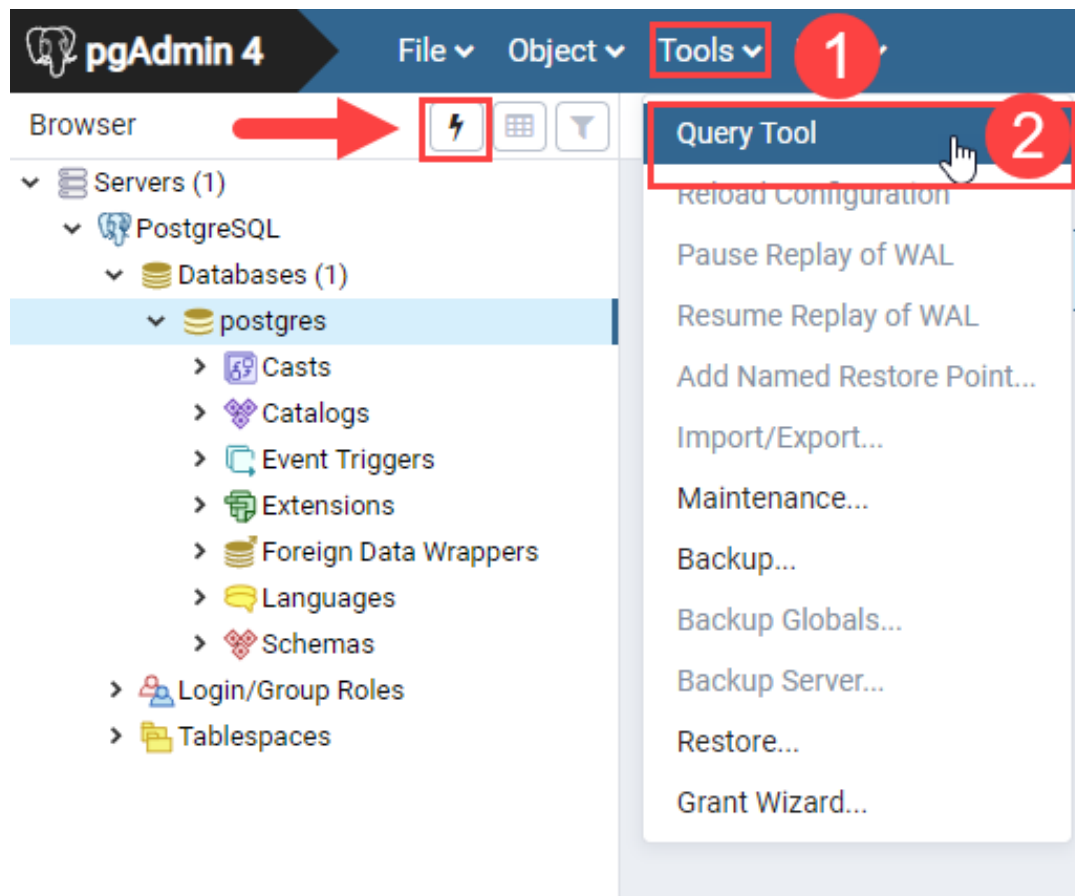
Seventh, enter the query in the **Query Editor**, click the **Execute** button, you will see the result of the query displaying in the **Data Output** tab:
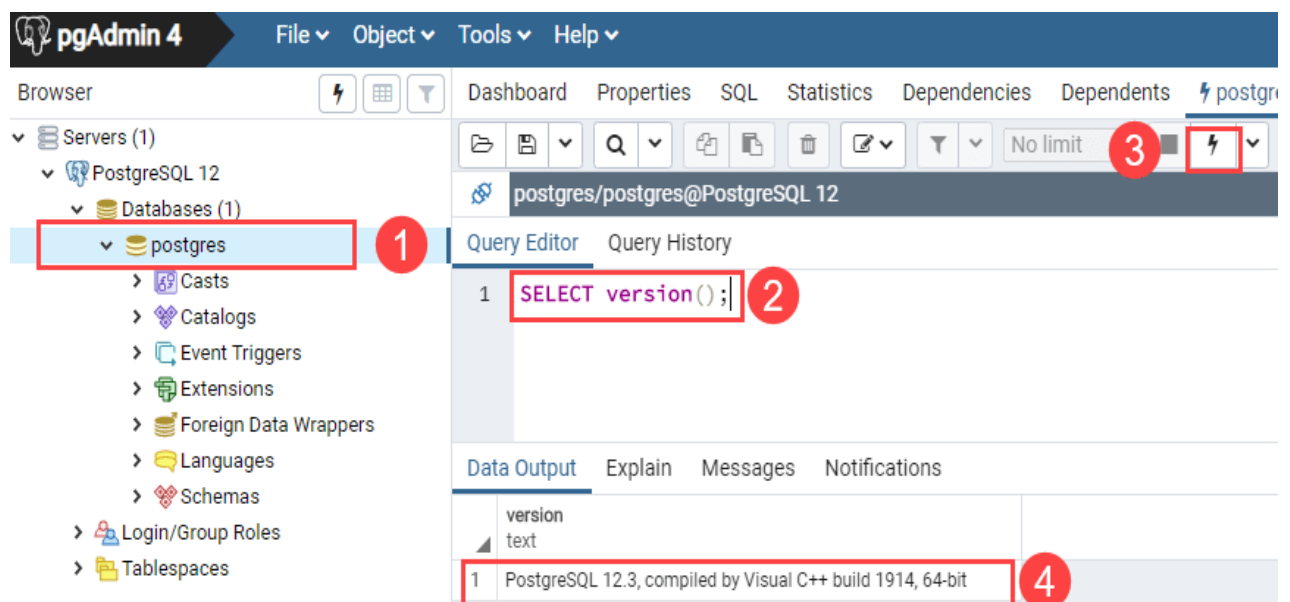
Load PostgreSQL Sample Database

Before going forward with this, you need to have:

- A PostgreSQL database server installed on your system.
- A PostgreSQL sample database called dvdrental.

The DVD rental database represents the business processes of a DVD rental store. The DVD rental database has many objects, including:
- 15 tables
- 1 trigger
- 7 views
- 8 functions
- 1 domain
- 13 sequences

PostgreSQL Sample Database Tables

There are 15 tables in the DVD Rental database:
- actor – stores actors data including first name and last name.
- film – stores film data such as title, release year, length, rating, etc.
- film_actor – stores the relationships between films and actors.
- category – stores film's categories data.
- film_category- stores the relationships between films and categories.
- store – contains the store data including manager staff and address.
- inventory – stores inventory data.
- rental – stores rental data.
- payment – stores customer's payments.
- staff – stores staff data.
- customer – stores customer data.
- address – stores address data for staff and customers
- city – stores city names.
- country – stores country names.

DVD Rental ER Model
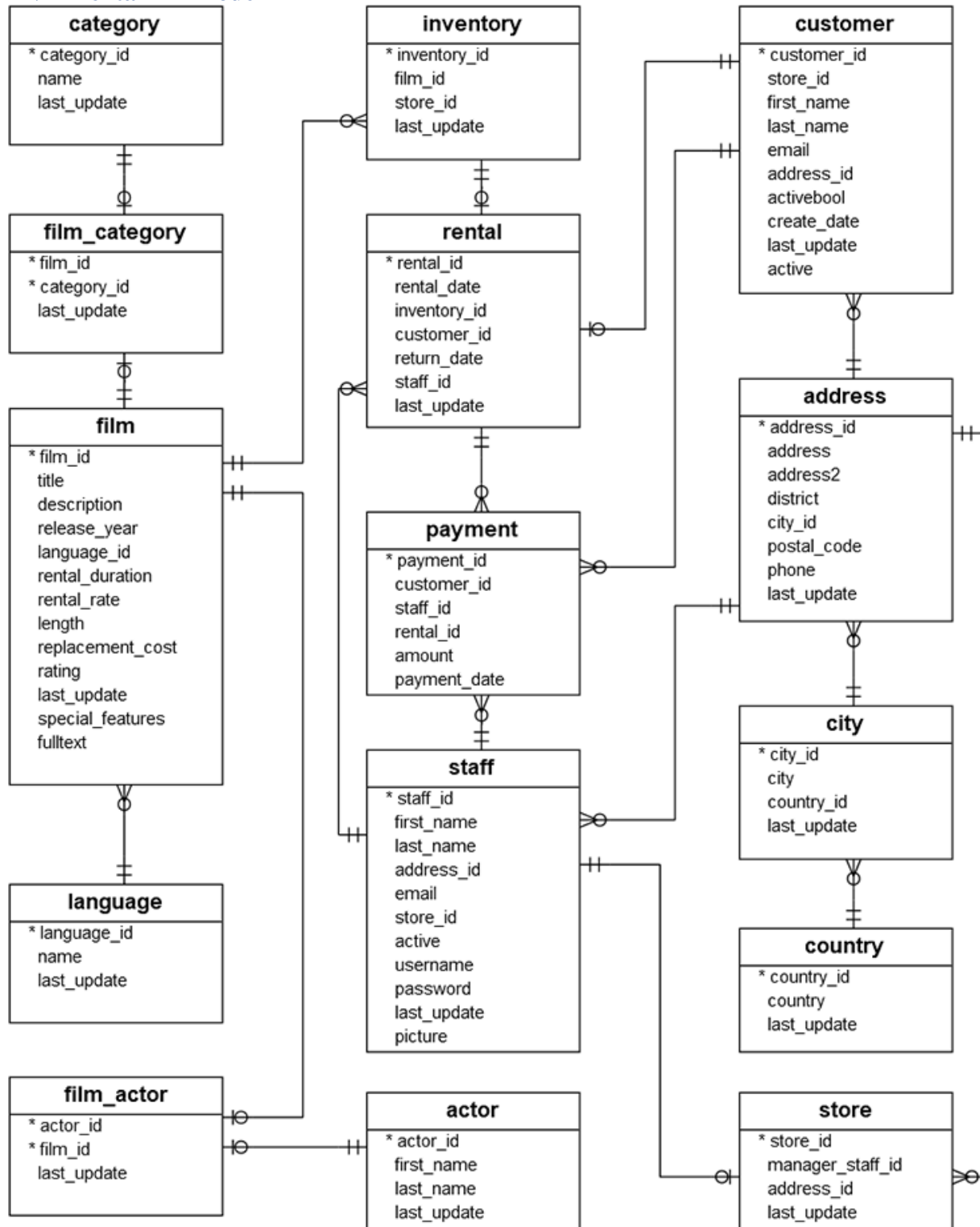
Load the sample database using psql tool

First, launch the **psql** tool.

```
>psql
```

Second, enter the account's information to log in to the PostgreSQL database server. You can use the default value provided by psql by pressing the **Enter** keyboard. However, for the password, you need to enter the one that you provided during PostgreSQL installation.

```
Server [localhost]:
Database [postgres]:
Port [5432]:
Username [postgres]:
Password for user postgres:
```
Code language: SQL (Structured Query Language) (sql)

Third, enter the following CREATE DATABASE statement to create a new **dvdrental** database.

```
postgres=# CREATE DATABASE dvdrental;
CREATE DATABASE
```

PostgreSQL will create a new database named dvdrental.

Finally, enter the exit command to quit psql:

```
postgres=# exit
```
Code language: PHP (php)

Then, navigate the **bin** folder of the PostgreSQL installation folder:

```
C:\>cd C:\Program Files\PostgreSQL\12\bin
```

After that, use the **pg_restore** tool to load data into the **dvdrental** database:

```
pg_restore -U postgres -d dvdrental C:\sampledb\dvdrental.tar
```
Code language: CSS (css)

In this command:

- The -U postgres specifies the postgresuser to login to the PostgreSQL database server.
- The -d dvdrental specifies the target database to load.

Finally, enter the password for the **postgres** user and press enter

```
Password:
```
Code language: SQL (Structured Query Language) (sql)

It takes about seconds to load data stored in the dvdrental.tar file into the dvdrentaldatabase.

Load the DVD Rental database using the pgAdmin

The following shows you step by step on how to use the pgAdmin tool to restore the sample database from the database file:
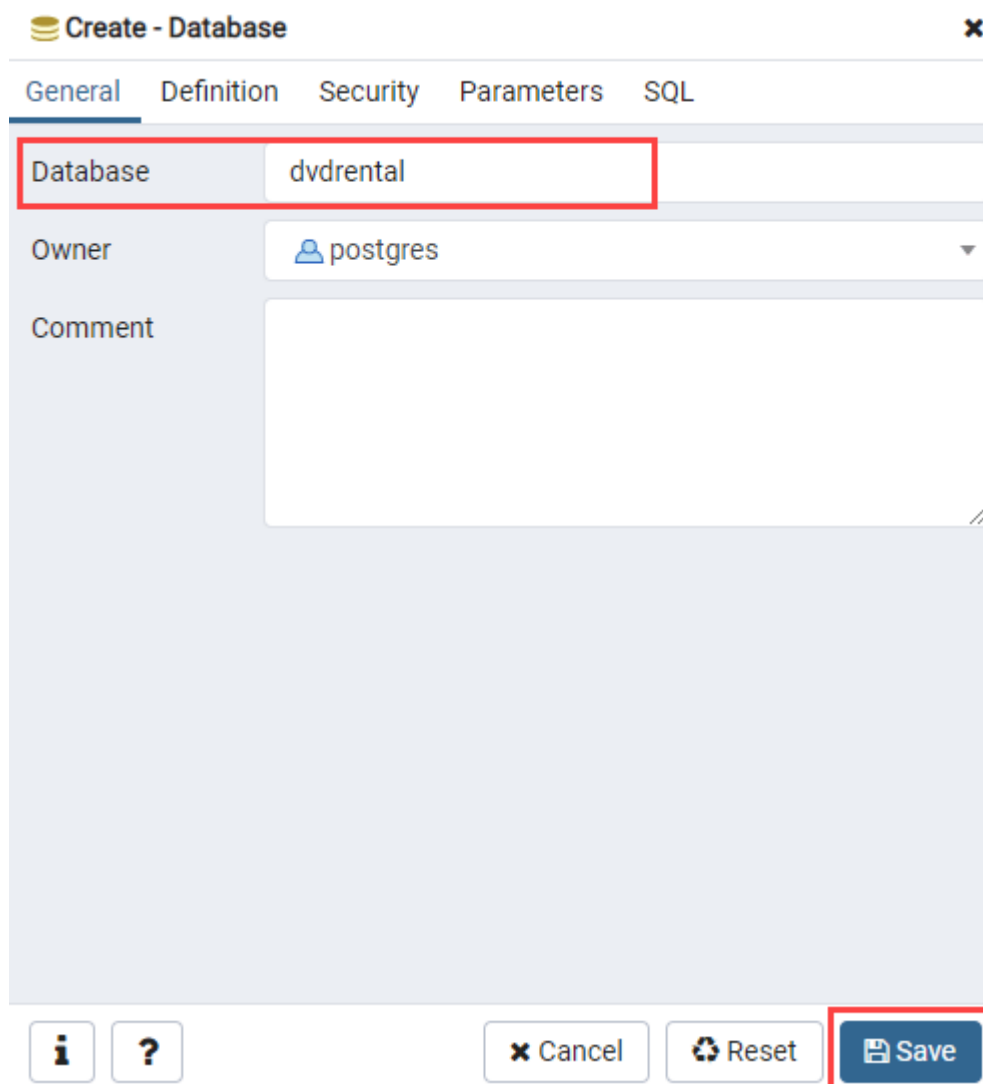
First, launch the **pgAdmin** tool and connect to the PostgreSQL server.

Second, right click the **Databases** and select the **Create > Database…** menu option:
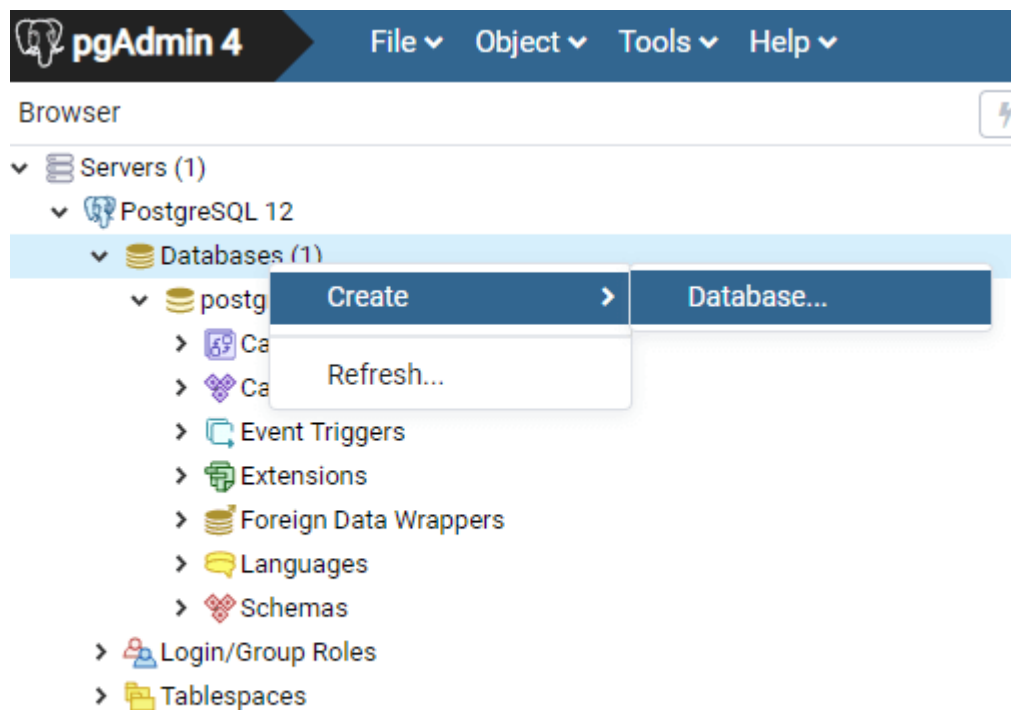
Third, enter the database name dvdrental and click the **Save** button:

Create - Database ✖

General   Definition   Security   Parameters   SQL

Database	dvdrental
Owner	👤 postgres ▾
Comment	

ℹ   ?   ✖ Cancel   ♻ Reset   💾 Save

You'll see the new empty database created under the Databases node:
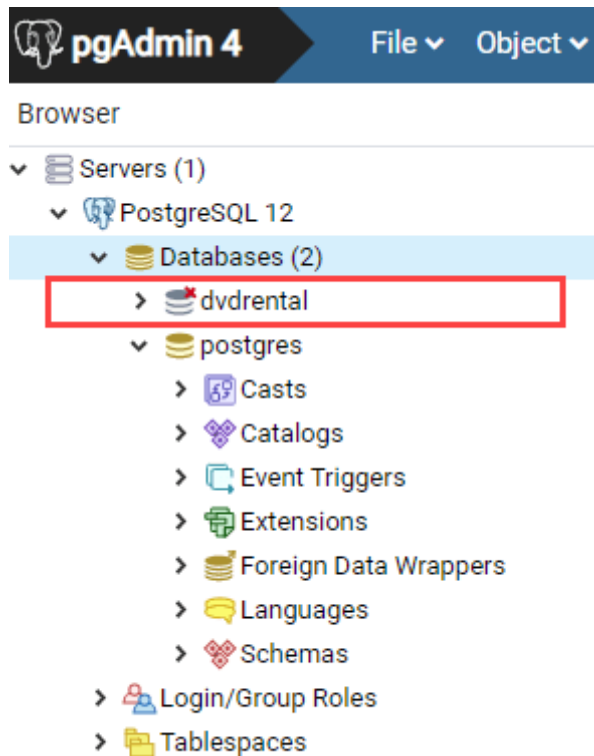
Fourth, right-click on the **dvdrental** database and choose **Restore…** menu item to restore the database from the downloaded database file:
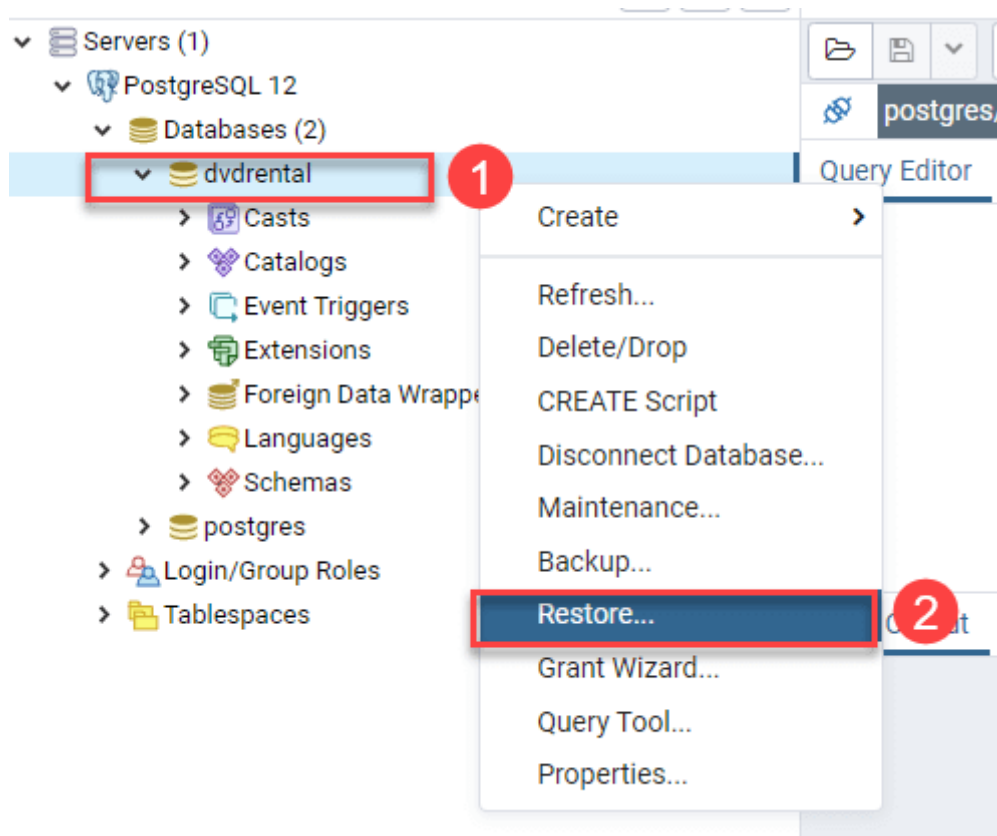
Fifth, enter the path to the sample database file e.g., **c:\sampledb\dvdrental.tar** and click the **Restore** button:

Sixth, the restoration process will complete in few seconds and shows the following dialog once it completes:

Restoring backup on the server ✖

Restoring backup on the server 'PostgreSQL 12 (localhost:5432)'

🕐 1.88 seconds   ❶ More details...   ⊗ Stop Process

✔ Successfully completed.

Finally, open the dvdrental database from object browser panel, you will find tables in the public schema and other database objects as shown in the following picture:

- ⌄ 🛢 dvdrental
  - ❯ Casts
  - ❯ Catalogs
  - ❯ Event Triggers
  - ❯ Extensions
  - ❯ Foreign Data Wrappers
  - ❯ Languages
  - ⌄ Schemas (1)
    - ⌄ ◈ public
      - ❯ Collations
      - ❯ Domains
      - ❯ FTS Configurations
      - ❯ FTS Dictionaries
      - ❯ Aa FTS Parsers
      - ❯ FTS Templates
      - ❯ Foreign Tables
      - ❯ Functions
      - ❯ Materialized Views
      - ❯ Procedures
      - ❯ Sequences
      - ⌄ Tables (15)
        - ❯ actor
        - ❯ address
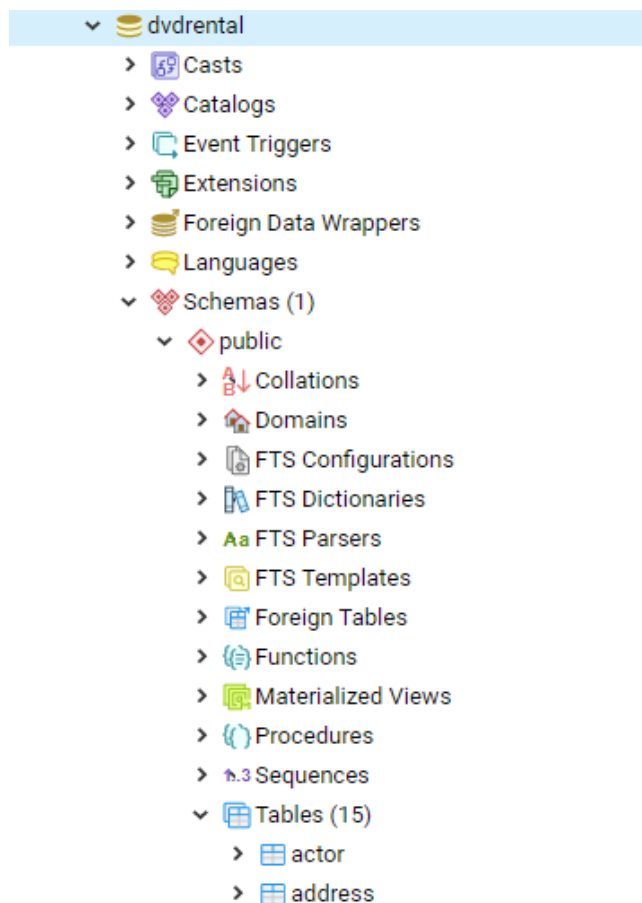
Overview of PostgreSQL data types

PostgreSQL supports the following data types:

- Boolean
- Character types such as char, varchar, and text.
- Numeric types such as integer and floating-point number.
- Temporal types such as date, time, timestamp, and interval
- UUID for storing Universally Unique Identifiers
- Array for storing array strings, numbers, etc.
- JSON stores JSON data
- hstore stores key-value pair
- Special types such as network address and geometric data.

Boolean

A Boolean data type can hold one of three possible values: true, false or null. You use boolean or bool keyword to declare a column with the Boolean data type.

When you insert data into a Boolean column, PostgreSQL converts it to a Boolean value

- 1, yes, y, t, true values are converted to true
- 0, no, false, f values are converted to false.

When you select data from a Boolean column, PostgreSQL converts the values back e.g., t to true, f to false and space to null.

Character

PostgreSQL provides three character data types: CHAR(n), VARCHAR(n), and TEXT

- CHAR(n) is the fixed-length character with space padded. If you insert a string that is shorter than the length of the column, PostgreSQL pads spaces. If you insert a string that is longer than the length of the column, PostgreSQL will issue an error.
- VARCHAR(n) is the variable-length character string. With VARCHAR(n), you can store up to n characters. PostgreSQL does not pad spaces when the stored string is shorter than the length of the column.
- TEXT is the variable-length character string. Theoretically, text data is a character string with unlimited length.

Numeric

PostgreSQL provides two distinct types of numbers:

- integers
- floating-point numbers

Integer

There are three kinds of integers in PostgreSQL:

- Small integer (SMALLINT) is 2-byte signed integer that has a range from -32,768 to 32,767.
- Integer (INT) is a 4-byte integer that has a range from -2,147,483,648 to 2,147,483,647.
- Serial is the same as integer except that PostgreSQL will automatically generate and populate values into the SERIAL column. This is similar to AUTO_INCREMENT column in MySQL or AUTOINCREMENT column in SQLite.

Floating-point number

There three main types of floating-point numbers:

- float(n) is a floating-point number whose precision, at least, n, up to a maximum of 8 bytes.
- realor float8is a 4-byte floating-point number.
- numeric or numeric(p,s) is a real number with p digits with s number after the decimal point. The numeric(p,s) is the exact number.

Temporal data types

The temporal data types allow you to store date and /or time data. PostgreSQL has five main temporal data types:

- DATEstores the dates only.
- TIMEstores the time of day values.
- TIMESTAMPstores both date and time values.
- TIMESTAMPTZ is a timezone-aware timestamp data type. It is the abbreviation for timestamp with the time zone.
- INTERVAL stores periods of time.

The TIMESTAMPTZ is the PostgreSQL's extension to the SQL standard's temporal data types.

Arrays

In PostgreSQL, you can store an array of strings, an array of integers, etc., in array columns. The array comes in handy in some situations e.g., storing days of the week, months of the year.

JSON

PostgreSQL provides two JSON data types: JSON and JSONB for storing JSON data.

The JSON data type stores plain JSON data that requires reparsing for each processing, while JSONB data type stores JSON data in a binary format which is faster to process but slower to insert. In addition, JSONB supports indexing, which can be an advantage.

UUID

The UUID data type allows you to store Universal Unique Identifiers defined by [RFC 4122](). The UUID values guarantee a better uniqueness than [SERIAL]() and can be used to hide sensitive data exposed to the public such as values of id in URL.

Special data types

- Besides the primitive data types, PostgreSQL also provides several special data types related to geometric and network.
- box– a rectangular box.
- line – a set of points.
- point– a geometric pair of numbers.
- lseg– a line segment.
- polygon– a closed geometric.
- inet– an IP4 address.
- macaddr– a MAC address.

Types of SQL Commands

The following sections discuss the basic categories of commands used in SQL to perform various functions. These functions include building database objects, manipulating objects, populating database tables with data, updating existing data in tables, deleting data, performing database queries, controlling database access, and overall database administration. The main categories are:

     a. DDL (Data Definition Language)
     b. DML (Data Manipulation Language)
     c. DQL (Data Query Language)
     d. DCL (Data Control Language)
     e. Data administration commands
     f. Transactional control commands

Data Definition Language

Data Definition Language, DDL, is the part of SQL that allows a database user to create and restructure database objects, such as the creation or the deletion of a table.
Some of the most fundamental DDL commands discussed during following hours include the following:

- CREATE TABLE
- ALTER TABLE
- DROP TABLE
- CREATE INDEX
- ALTER INDEX
- DROP INDEX

- VIEW
- DROP VIEW

Data Manipulation Language

Data Manipulation Language, DML, is the part of SQL used to manipulate data within objects of a relational database.
There are three basic DML commands:

- INSERT
- UPDATE
- DELETE

Data Query Language

Though comprised of only one command, Data Query Language (DQL) is the most concentrated focus of SQL for modern relational database users. The base command is as follows:

- SELECT

This command, accompanied by many options and clauses, is used to compose queries against a relational database. Queries, from simple to complex, from vague to specific, can be easily created. A query is an inquiry to the database for information. A query is usually issued to the database through an application interface or via a command line prompt.

Data Control Language

Data control commands in SQL allow you to control access to data within the database. These DCL commands are normally used to create objects related to user access and also control the distribution of privileges among users. Some data control commands are as follows:

- ALTER PASSWORD
- GRANT
- REVOKE
- CREATE SYNONYM

You will find that these commands are often grouped with other commands.

Data Administration Commands

Data administration commands allow the user to perform audits and perform analyses on operations within the database. They can also be used to help analyse system performance. Two general data administration commands are as follows:

- START AUDIT
- STOP AUDIT

Do not get data administration confused with database administration. Database administration is the overall administration of a database, which envelops the use of all levels of commands. Database administration is much more specific to each SQL implementation than are those core commands of the SQL language.

Transactional Control Commands

In addition to the previously introduced categories of commands, there are commands that allow the user to manage database transactions.

- COMMIT Saves database transactions

- ROLLBACK Undoes database transactions

- SAVEPOINT Creates points within groups of transactions in which to ROLLBACK

SET TRANSACTION Places a name on a transaction

PostgreSQL CREATE TABLE
PostgreSQL CREATE TABLE syntax

A relational database consists of multiple related tables. A table consists of rows and columns. Tables allow you to store structured data like customers, products, employees, etc.

To create a new table, you use the CREATE TABLE statement. The following illustrates the basic syntax of the CREATE TABLE statement:

```
CREATE TABLE [IF NOT EXISTS] table_name (
  column1 datatype(length) column_contraint,
  column2 datatype(length) column_contraint,
  column3 datatype(length) column_contraint,
  table_constraints
);
```

In this syntax:

- First, specify the name of the table after the CREATE TABLE keywords.
- Second, creating a table that already exists will result in a error. The IF NOT EXISTS option allows you to create the new table only if it does not exist. When you use the IF NOT EXISTS option and the table already exists, PostgreSQL issues a notice instead of the error and skips creating the new table.
- Third, specify a comma-separated list of table columns. Each column consists of the column name, the kind of data that column stores, the length of data, and the column constraint. The column constraints specify rules that data stored in the column must follow. For example, the not-null constraint enforces the values in the column cannot be NULL. The column constraints include not null, unique, primary key, check, foreign key constraints.
- Finally, specify the table constraints including primary key, foreign key, and check constraints.

Note that some table constraints can be defined as column constraints like primary key, foreign key, check, unique constraints.

Constraints

PostgreSQL includes the following column constraints:

- NOT NULL – ensures that values in a column cannot be NULL.
- UNIQUE – ensures the values in a column unique across the rows within the same table.
- PRIMARY KEY – a primary key column uniquely identify rows in a table. A table can have one and only one primary key. The primary key constraint allows you to define the primary key of a table.

- **CHECK** – a CHECK constraint ensures the data must satisfy a boolean expression.
- **FOREIGN KEY** – ensures values in a column or a group of columns from a table exists in a column or group of columns in another table. Unlike the primary key, a table can have many foreign keys.

Table constraints are similar to column constraints except that they are applied to more than one column.

PostgreSQL CREATE TABLE examples

We will create a new table called accounts that has the following columns:

- user_id – primary key
- username – unique and not null
- password – not null
- email – unique and not null
- created_on – not null
- last_login – null

The following statement creates the accounts table:

```
CREATE TABLE accounts (
        user_id serial PRIMARY KEY,
        username VARCHAR ( 50 ) UNIQUE NOT NULL,
        password VARCHAR ( 50 ) NOT NULL,
        email VARCHAR ( 255 ) UNIQUE NOT NULL,
        created_on TIMESTAMP NOT NULL,
    last_login TIMESTAMP
);
```

```
accounts
🔑 user_id: int4
   username: varchar(50)
   password: varchar(50)
   email: varchar(255)
   created_on: timestamp(6)
   last_login: timestamp(6)
```
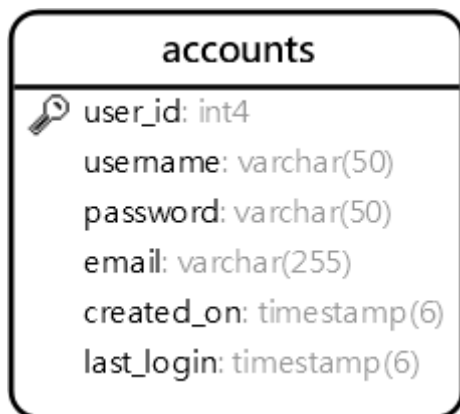
The following statement creates the roles table that consists of two columns: role_id and role_name:

```
CREATE TABLE roles(
   role_id serial PRIMARY KEY,
   role_name VARCHAR (255) UNIQUE NOT NULL
);
```

The following statement creates the account_roles table that has three columns: user_id, role_id and grant_date.

```sql
CREATE TABLE account_roles (
  user_id INT NOT NULL,
  role_id INT NOT NULL,
  grant_date TIMESTAMP,
  PRIMARY KEY (user_id, role_id),
  FOREIGN KEY (role_id)
     REFERENCES roles (role_id),
  FOREIGN KEY (user_id)
     REFERENCES accounts (user_id)
);
```
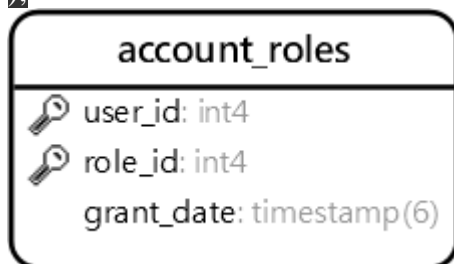

The primary key of the account_roles table consists of two columns: user_id and role_id, therefore, we have to define the primary key constraint as a table constraint.

```sql
PRIMARY KEY (user_id, role_id)
```
Code language: SQL (Structured Query Language) (sql)

Because the user_idcolumn references to the user_idcolumn in the accounts table, we need to define a foreign key constraint for the user_idcolumn:

```sql
FOREIGN KEY (user_id)
REFERENCES accounts (user_id)
```
Code language: SQL (Structured Query Language) (sql)

The role_idcolumn references the role_idcolumn in the roles table, we also need to define a foreign key constraint for the role_idcolumn.

```sql
FOREIGN KEY (role_id)
REFERENCES roles (role_id)
```
Code language: SQL (Structured Query Language) (sql)

The following shows the relationship between the accounts, roles, and account_roles tables:
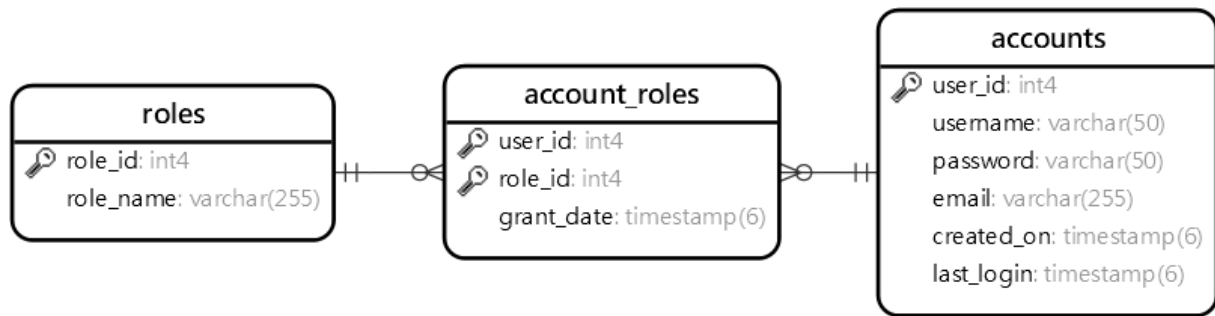
PostgreSQL ALTER TABLE

To change the structure of an existing table, you use PostgreSQL ALTER TABLE statement.

The following illustrates the basic syntax of the ALTER TABLE statement:

`ALTER TABLE table_name action;`

PostgreSQL provides you with many actions:

- Add a column
- Drop a column
- Change the data type of a column
- Rename a column
- Set a default value for the column.
- Add a constraint to a column.
- Rename a table

To add a new column to a table, you use ALTER TABLE ADD COLUMN statement:

```
ALTER TABLE table_name
ADD COLUMN column_name datatype column_constraint;
```
Code language: SQL (Structured Query Language) (sql)

To drop a column from a table, you use ALTER TABLE DROP COLUMN statement:

```
ALTER TABLE table_name
DROP COLUMN column_name;
```
Code language: SQL (Structured Query Language) (sql)

To rename a column, you use the ALTER TABLE RENAME COLUMN TO statement:

```
ALTER TABLE table_name
RENAME COLUMN column_name
TO new_column_name;
```
Code language: SQL (Structured Query Language) (sql)

To change a default value of the column, you use ALTER TABLE ALTER COLUMN SET DEFAULT or DROP DEFAULT:

```
ALTER TABLE table_name
ALTER COLUMN column_name
```

```sql
[SET DEFAULT value | DROP DEFAULT];
```
Code language: SQL (Structured Query Language) (sql)

To change the NOT NULL constraint, you use ALTER TABLE ALTER COLUMN statement:

```sql
ALTER TABLE table_name
ALTER COLUMN column_name
[SET NOT NULL| DROP NOT NULL];
```
Code language: SQL (Structured Query Language) (sql)

To add a CHECK constraint, you use ALTER TABLE ADD CHECK statement:

```sql
ALTER TABLE table_name
ADD CHECK expression;
```
Code language: SQL (Structured Query Language) (sql)

Generailly, to add a constraint to a table, you use ALTER TABLE ADD CONSTRAINT statement:

```sql
ALTER TABLE table_name
ADD CONSTRAINT constraint_name constraint_definition;
```
Code language: SQL (Structured Query Language) (sql)

To rename a table you use ALTER TABLE RENAME TO statement:

```sql
ALTER TABLE table_name
RENAME TO new_table_name;
```
Code language: SQL (Structured Query Language) (sql)

Introduction to PostgreSQL INSERT statement

The PostgreSQL INSERT statement allows you to insert a new row into a table.

The following illustrates the most basic syntax of the INSERT statement:

```sql
INSERT INTO table_name(column1, column2, …)
VALUES (value1, value2, …);
```
Code language: SQL (Structured Query Language) (sql)

In this syntax:

- First, specify the name of the table (table_name) that you want to insert data after the INSERT INTO keywords and a list of comma-separated columns (colum1, column2, ...).

- Second, supply a list of comma-separated values in a parentheses (value1, value2, ...) after the VALUES keyword. The columns and values in the column and value lists must be in the same order.

The INSERT statement returns a command tag with the following form:

```
INSERT oid count
```

OID is an object identifier. PostgreSQL used the OID internally as a [primary key](#) for its system tables. Typically, the INSERT statement returns OID with value 0. The count is the number of rows that the INSERT statement inserted successfully.

RETURNING clause

The INSERT statement also has an optional RETURNING clause that returns the information of the inserted row.

If you want to return the entire inserted row, you use an asterisk (*) after the RETURNING keyword:

```sql
INSERT INTO table_name(column1, column2, …)
VALUES (value1, value2, …)
RETURNING *;
```
Code language: SQL (Structured Query Language) (sql)

If you want to return just some information of the inserted row, you can specify one or more columns after the RETURNING clause.

For example, the following statement returns the id of the inserted row:

```sql
INSERT INTO table_name(column1, column2, …)
VALUES (value1, value2, …)
RETURNING id;
```
Code language: SQL (Structured Query Language) (sql)

To rename the returned value, you use the AS keyword followed by the name of the output. For example:

```sql
INSERT INTO table_name(column1, column2, …)
VALUES (value1, value2, …)
RETURNING output_expression AS output_name;
```
Code language: SQL (Structured Query Language) (sql)

PostgreSQL INSERT statement examples

The following statement [creates a new table](#) calledlinksfor the demonstration:

```sql
DROP TABLE IF EXISTS links;
```

```sql
CREATE TABLE links (
        id SERIAL PRIMARY KEY,
        url VARCHAR(255) NOT NULL,
        name VARCHAR(255) NOT NULL,
        description VARCHAR (255),
    last_update DATE
);
```
Code language: SQL (Structured Query Language) (sql)

Note that you will learn how to create a new table in the subsequent tutorial. In this tutorial, you just need to execute it to create a new table.

1) PostgreSQL INSERT – Inserting a single row into a table

The following statement inserts a new row into the links table:

```sql
INSERT INTO links (url, name)
VALUES('https://www.postgresqltutorial.com','PostgreSQL Tutorial');
```
Code language: SQL (Structured Query Language) (sql)

The statement returns the following output:

```shell
INSERT 0 1
```
Code language: Shell Session (shell)

To insert character data, you enclose it in single quotes (') for example 'PostgreSQL Tutorial'.

If you omit required columns in the INSERT statement, PostgreSQL will issue an error. In case you omit an optional column, PostgreSQL will use the column default value for insert.

In this example, the description is an optional column because it doesn't have a NOT NULL constraint. Therefore, PostgreSQL uses NULL to insert into the description column.

PostgreSQL automatically generates a sequential number for the serial column so you do not have to supply a value for the serial column in the INSERT statement.

The following SELECT statement shows the contents of the links table:

```sql
SELECT * FROM links;
```
Code language: SQL (Structured Query Language) (sql)

| id | url | name | description | last_update |
integer	character varying (255)	character varying (255)	character varying (255)	date
1	1 https://www.postgresqltutorial.com	PostgreSQL Tutorial	[null]	[null]

2) PostgreSQL INSERT – Inserting character string that contains a single quote

If you want to insert a string that contains a single quote (') such as O'Reilly Media, you have to use an additional single quote (') to escape it. For example:

```sql
INSERT INTO links (url, name)
VALUES('http://www.oreilly.com','O''Reilly Media');
```
Code language: SQL (Structured Query Language) (sql)

Output:

```
INSERT 0 1
```

The following statement verifies the insert:

id integer	url character varying (255)	name character varying (255)	description character varying (255)
1	1	https://www.postgresqltutorial.com	PostgreSQL Tutorial	[null]
2	2	http://www.oreilly.com	O'Reilly Media	[null]

3) PostgreSQL INSERT – Inserting a date value

To insert a date value into a column with the DATE type, you use the date in the format 'YYYY-MM-DD'.

The following statement inserts a new row with a specified date into the links table:

```sql
INSERT INTO links (url, name, last_update)
VALUES('https://www.google.com','Google','2013-06-01');
```
Code language: SQL (Structured Query Language) (sql)

Output:

```
INSERT 0 1
```

4) PostgreSQL INSERT- Getting the last insert id

To get the last insert id from inserted row, you use the RETURNING clause of the INSERTstatement.

For example, the following statement inserts a new row into the links table and returns the last insert id:

```sql
INSERT INTO links (url, name)
VALUES('http://www.postgresql.org','PostgreSQL')
RETURNING id;
```
Code language: SQL (Structured Query Language) (sql)

Output:

id integer
1	4

PostgreSQL ALTER TABLE examples

Let's create a new table calledlinks for practicing with the ALTER TABLE statement.

```sql
DROP TABLE IF EXISTS links;
```

```sql
CREATE TABLE links (
    link_id serial PRIMARY KEY,
    title VARCHAR (512) NOT NULL,
    url VARCHAR (1024) NOT NULL
);
```
Code language: SQL (Structured Query Language) (sql)

To add a new column named active, you use the following statement:

```sql
ALTER TABLE links
ADD COLUMN active boolean;
```
Code language: SQL (Structured Query Language) (sql)

The following statement removes the activecolumn from the linkstable:

```sql
ALTER TABLE links
DROP COLUMN active;
```

To change the name of the title column to link_title, you use the following statement:

```sql
ALTER TABLE links
RENAME COLUMN title TO link_title;
```
Code language: SQL (Structured Query Language) (sql)

The following statement adds a new column named target to the links table:

```sql
ALTER TABLE links
ADD COLUMN target VARCHAR(10);
```
Code language: SQL (Structured Query Language) (sql)

To set _blank as the default value for the target column in the links table, you use the following statement:

```sql
ALTER TABLE links
ALTER COLUMN target
SET DEFAULT '_blank';
```
Code language: SQL (Structured Query Language) (sql)

If you insert the new row into the links table without specifying a value for the target column, the target column will take the _blank as the default value. For example:

```sql
INSERT INTO links (link_title, url)
VALUES('PostgreSQL Tutorial','https://www.postgresqltutorial.com/');
```
Code language: SQL (Structured Query Language) (sql)

The following statement selects data from the links table:

```sql
SELECT * FROM links;
```
Code language: SQL (Structured Query Language) (sql)

link_id	link_title	url	target
1	PostgreSQL Tutorial	http://www.postgresqltutorial.com/	_blank

The following statement adds a CHECKcondition to the targetcolumn so that the targetcolumn only accepts the following values: _self, _blank, _parent, and _top:

```sql
ALTER TABLE links
ADD CHECK (target IN ('_self', '_blank', '_parent', '_top'));
```
Code language: SQL (Structured Query Language) (sql)

If you attempt to insert a new row that violates the CHECK constraint set for the targetcolumn, PostgreSQL will issue an error as shown in the following example:

```sql
INSERT INTO links(link_title,url,target)
VALUES('PostgreSQL','http://www.postgresql.org/','whatever');
```
Code language: SQL (Structured Query Language) (sql)

```
ERROR:  new row for relation "links" violates check constraint "links_target_check"
DETAIL:  Failing row contains (2, PostgreSQL, http://www.postgresql.org/,
whatever).DETAIL:  Failing row contains (2, PostgreSQL, http://www.postgresql.org/,
whatever).
```
Code language: Shell Session (shell)

The following statement adds a UNIQUE constraint to the url column of the links table:

```sql
ALTER TABLE links
ADD CONSTRAINT unique_url UNIQUE ( url );
```
Code language: SQL (Structured Query Language) (sql)

The following statement attempts to insert the url that already exists:

```sql
INSERT INTO links(link_title,url)
VALUES('PostgreSQL','https://www.postgresqltutorial.com/');
```
Code language: SQL (Structured Query Language) (sql)

It causes an error due to the unique_url constraint:

```
ERROR:  duplicate key value violates unique constraint "unique_url"
DETAIL:  Key (url)=(https://www.postgresqltutorial.com/) already exists.
```
Code language: Shell Session (shell)

The following statement changes the name of the links table to urls:

```sql
ALTER TABLE links
RENAME TO urls;
```

PostgreSQL DROP TABLE

To drop a table from the database, you use the DROP TABLE statement as follows:

```sql
DROP TABLE [IF EXISTS] table_name
[CASCADE | RESTRICT];
```

In this syntax:

- First, specify the name of the table that you want to drop after the DROP TABLE keywords.
- Second, use the IF EXISTS option to remove the table only if it exists.

If you remove a table that does not exist, PostgreSQL issues an error. To avoid this situation, you can use the IF EXISTS option.

In case the table that you want to remove is used in other objects such as views, triggers, functions, and stored procedures, the DROP TABLE cannot remove the table. In this case, you have two options:

- The CASCADE option allows you to remove the table and its dependent objects.
- The RESTRICT option rejects the removal if there is any object depends on the table. The RESTRICT option is the default if you don't explicitly specify it in the DROP TABLE statement.

To remove multiple tables at once, you can place a comma-separated list of tables after the DROP TABLE keywords:

```
DROP TABLE [IF EXISTS]
   table_name_1,
   table_name_2,
   ...
[CASCADE | RESTRICT];
```
Code language: CSS (css)

Note that you need to have the roles of the superuser, schema owner, or table owner in order to drop tables.

PostgreSQL DROP TABLE examples

Let's take some examples of using the PostgreSQL DROP TABLE statement

1) Drop a table that does not exist

The following statement removes a table named authorin the database:

```
DROP TABLE author;
```
Code language: SQL (Structured Query Language) (sql)

PostgreSQL issues an error because the author table does not exist.

```
[Err] ERROR:  table "author" does not exist
```
Code language: Shell Session (shell)

To avoid the error, you can use the IF EXISTS option like this.

```
DROP TABLE IF EXISTS author;
```
Code language: SQL (Structured Query Language) (sql)
```
NOTICE:  table "author" does not exist, skipping DROP TABLE
```
Code language: Shell Session (shell)

As can be seen clearly from the output, PostgreSQL issued a notice instead of an error.

2) Drop a table that has dependent objects

The following creates new tables called authors and pages:

```sql
CREATE TABLE authors (
        author_id INT PRIMARY KEY,
        firstname VARCHAR (50),
        lastname VARCHAR (50)
);

CREATE TABLE pages (
        page_id serial PRIMARY KEY,
        title VARCHAR (255) NOT NULL,
        contents TEXT,
        author_id INT NOT NULL,
        FOREIGN KEY (author_id)
     REFERENCES authors (author_id)
);
```
Code language: SQL (Structured Query Language) (sql)

The following statement uses the DROP TABLE to drop the authortable:

```sql
DROP TABLE IF EXISTS authors;
```
Code language: SQL (Structured Query Language) (sql)

Because the constraint on the page table depends on the authortable, PostgreSQL issues an error message:

```
ERROR:  cannot drop table authors because other objects depend on it
DETAIL:  constraint pages_author_id_fkey on table pages depends on table authors
HINT:  Use DROP ... CASCADE to drop the dependent objects too.
SQL state: 2BP01
```
Code language: Shell Session (shell)

In this case, you need to remove all dependent objects first before dropping the author table or use CASCADE option as follows:

```sql
DROP TABLE authors CASCADE;
```
Code language: SQL (Structured Query Language) (sql)

PostgreSQL removes the authortable as well as the constraint in the page table.

If the DROP TABLE statement removes the dependent objects of the table that is being dropped, it will issue a notice like this:

```
NOTICE:  drop cascades to constraint pages_author_id_fkey on table pages
```
Code language: Shell Session (shell)

3) Drop multiple tables

The following statements create two tables for the demo purposes:

```php
CREATE TABLE tvshows(
        tvshow_id INT GENERATED ALWAYS AS IDENTITY,
        title VARCHAR,
        release_year SMALLINT,
        PRIMARY KEY(tvshow_id)
);

CREATE TABLE animes(
        anime_id INT GENERATED ALWAYS AS IDENTITY,
        title VARCHAR,
        release_year SMALLINT,
        PRIMARY KEY(anime_id)
);
```
Code language: PHP (php)

The following example uses a single DROP TABLE statement to drop the tvshows and animes tables:

```
DROP TABLE tvshows, animes;
```

PostgreSQL SELECT

One of the most common tasks, when you work with the database, is to query data from tables by using the SELECT statement.

The SELECT statement is one of the most complex statements in PostgreSQL. It has many clauses that you can use to form a flexible query.

Because of its complexity, we will break it down into many shorter and easy-to-understand tutorials so that you can learn about each clause faster.

The SELECTstatement has the following clauses:

- Select distinct rows using DISTINCT operator.
- Sort rows usingORDER BY clause.
- Filter rows using WHERE clause.
- Select a subset of rows from a table using LIMIT or FETCH clause.
- Group rows into groups using GROUP BY clause.
- Filter groups using HAVING clause.
- Join with other tables using joins such as INNER JOIN, LEFT JOIN, FULL OUTER JOIN, CROSS JOIN clauses.
- Perform set operations using UNION, INTERSECT, and EXCEPT.

PostgreSQL SELECT statement syntax

Let's start with the basic form of the SELECT statement that retrieves data from a single table.

The following illustrates the syntax of the SELECT statement:

```sql
SELECT
   select_list
FROM
   table_name;
```
Code language: SQL (Structured Query Language) (sql)

Let's examine the SELECT statement in more detail:

- First, specify a select list that can be a column or a list of columns in a table from which you want to retrieve data. If you specify a list of columns, you need to place a comma (,) between two columns to separate them. If you want to select data from all the columns of the table, you can use an asterisk (∗) shorthand instead of specifying all the column names. The select list may also contain expressions or literal values.
- Second, specify the name of the table from which you want to query data after the FROM keyword.

The FROM clause is optional. If you do not query data from any table, you can omit the FROM clause in the SELECT statement.

PostgreSQL evaluates the FROM clause before the SELECT clause in the SELECT statement:

FROM ➡ SELECT
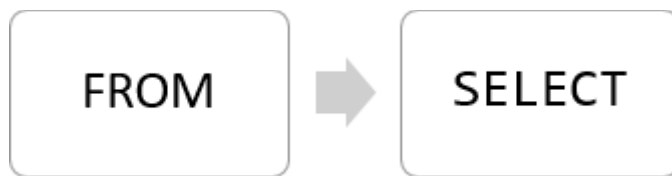
Note that the SQL keywords are case-insensitive. It means that SELECT is equivalent to select or Select. By convention, we will use all the SQL keywords in uppercase to make the queries easier to read.

PostgreSQL SELECT examples

Let's take a look at some examples of using PostgreSQL SELECT statement.

We will use the following customer table in the sample database for the demonstration.

customer

- * customer_id
- store_id
- first_name
- last_name
- email
- address_id
- activebool
- create_date
- last_update
- active

1) Using PostgreSQL SELECT statement to query data from one column example

This example uses the SELECT statement to find the first names of all customers from the customer table:

```sql
SELECT first_name FROM customer;
```

Here is the partial output:

	first_name character varying (45)
1	Jared
2	Mary
3	Patricia
4	Linda
5	Barbara
6	Elizabeth
7	Jennifer
8	Maria
9	Susan
10	Margaret

Notice that we added a semicolon (;) at the end of the SELECT statement. The semicolon is not a part of the SQL statement. It is used to signal PostgreSQL the end of an SQL statement. The semicolon is also used to separate two SQL statements.

2) Using PostgreSQL SELECT statement to query data from multiple columns example

Suppose you just want to know the first name, last name and email of customers, you can specify these column names in the SELECT clause as shown in the following query:

```sql
SELECT
    first_name,
    last_name,
    email
FROM
    customer;
```

Code language: SQL (Structured Query Language) (sql)

	first_name character varying (45)	last_name character varying (45)	email character varying (50)
1	Jared	Ely	jared.ely@sakilacustomer.org
2	Mary	Smith	mary.smith@sakilacustomer.org
3	Patricia	Johnson	patricia.johnson@sakilacustomer.org
4	Linda	Williams	linda.williams@sakilacustomer.org
5	Barbara	Jones	barbara.jones@sakilacustomer.org
6	Elizabeth	Brown	elizabeth.brown@sakilacustomer.org
7	Jennifer	Davis	jennifer.davis@sakilacustomer.org
8	Maria	Miller	maria.miller@sakilacustomer.org
9	Susan	Wilson	susan.wilson@sakilacustomer.org
10	Margaret	Moore	margaret.moore@sakilacustomer.org
11	Dorothy	Taylor	dorothy.taylor@sakilacustomer.org

3) Using PostgreSQL SELECT statement to query data from all columns of a table example

The following query uses the SELECT statement to select data from all columns of the customer table:

```sql
SELECT * FROM customer;
```
Code language: SQL (Structured Query Language) (sql)

	customer_id integer	store_id smallint	first_name character varying (45)	last_name character varying (45)	email character varying (50)	address_id smallint	activebool boolean	create_date date	last_update timestamp without time zone	active integer
1	524	1	Jared	Ely	jared.ely@sakilacustomer.org	530	true	2006-02-14	2013-05-26 14:49:45.738	1
2	1	1	Mary	Smith	mary.smith@sakilacustomer...	5	true	2006-02-14	2013-05-26 14:49:45.738	1
3	2	1	Patricia	Johnson	patricia.johnson@sakilacust...	6	true	2006-02-14	2013-05-26 14:49:45.738	1
4	3	1	Linda	Williams	linda.williams@sakilacusto...	7	true	2006-02-14	2013-05-26 14:49:45.738	1
5	4	2	Barbara	Jones	barbara.jones@sakilacusto...	8	true	2006-02-14	2013-05-26 14:49:45.738	1
6	5	1	Elizabeth	Brown	elizabeth.brown@sakilacust...	9	true	2006-02-14	2013-05-26 14:49:45.738	1
7	6	2	Jennifer	Davis	jennifer.davis@sakilacustom...	10	true	2006-02-14	2013-05-26 14:49:45.738	1
8	7	1	Maria	Miller	maria.miller@sakilacustome...	11	true	2006-02-14	2013-05-26 14:49:45.738	1
9	8	2	Susan	Wilson	susan.wilson@sakilacustom...	12	true	2006-02-14	2013-05-26 14:49:45.738	1
10	9	2	Margaret	Moore	margaret.moore@sakilacust...	13	true	2006-02-14	2013-05-26 14:49:45.738	1
11	10	1	Dorothy	Taylor	dorothy.taylor@sakilacusto...	14	true	2006-02-14	2013-05-26 14:49:45.738	1

In this example, we used an asterisk (∗) in the SELECT clause, which is a shorthand for all columns. Instead of listing all columns in the SELECT clause, we just used the asterisk (∗) to save some typing.

However, it is not a good practice to use the asterisk (∗) in the SELECT statement when you embed SQL statements in the application code like Python, Java, Node.js, or PHP due to the following reasons:

1. Database performance. Suppose you have a table with many columns and a lot of data, the SELECT statement with the asterisk (∗) shorthand will select data from all the columns of the table, which may not be necessary to the application.
2. Application performance. Retrieving unnecessary data from the database increases the traffic between the database server and application server. In consequence, your applications may be slower to respond and less scalable.

Because of these reasons, it is a good practice to explicitly specify the column names in the SELECT clause whenever possible to get only necessary data from the database.

And you should only use the asterisk (*) shorthand for the ad-hoc queries that examine data from the database.

4) Using PostgreSQL SELECT statement with expressions example

The following example uses the SELECT statement to return full names and emails of all customers:

```
SELECT
    first_name || ' ' || last_name,
    email
FROM
    customer;
```

Output:

	?column? text	email character varying (50)
1	Jared Ely	jared.ely@sakilacustomer.org
2	Mary Smith	mary.smith@sakilacustomer.org
3	Patricia Johnson	patricia.johnson@sakilacustomer.org
4	Linda Williams	linda.williams@sakilacustomer.org
5	Barbara Jones	barbara.jones@sakilacustomer.org
6	Elizabeth Brown	elizabeth.brown@sakilacustomer.org
7	Jennifer Davis	jennifer.davis@sakilacustomer.org
8	Maria Miller	maria.miller@sakilacustomer.org
9	Susan Wilson	susan.wilson@sakilacustomer.org
10	Margaret Moore	margaret.moore@sakilacustomer.org
11	Dorothy Taylor	dorothy.taylor@sakilacustomer.org

5) Using PostgreSQL SELECT statement with expressions example

The following example uses the SELECT statement with an expression. It omits the FROM clause:
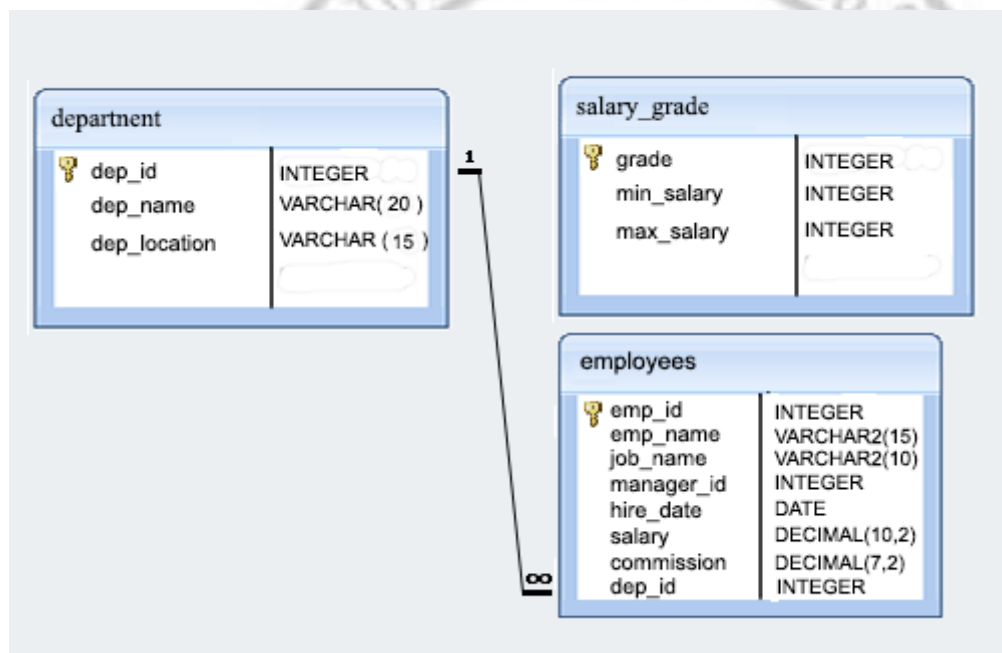
```
SELECT 5 * 3;
```

Here is the output:

	?column? integer
1	15

Exercise:

Structure of employee Database:

1. From the following table return complete information about the employees.

```
emp_id | emp_name  | job_name  | manager_id | hire_date  | salary  | commission | dep_id
--------+----------+----------+-----------+-----------+--------+-----------+--------
 68319 | KAYLING  | PRESIDENT |            | 1991-11-18 | 6000.00 |           |  1001
 66928 | BLAZE    | MANAGER   |     68319  | 1991-05-01 | 2750.00 |           |  3001
 67832 | CLARE    | MANAGER   |     68319  | 1991-06-09 | 2550.00 |           |  1001
 65646 | JONAS    | MANAGER   |     68319  | 1991-04-02 | 2957.00 |           |  2001
 67858 | SCARLET  | ANALYST   |     65646  | 1997-04-19 | 3100.00 |           |  2001
 69062 | FRANK    | ANALYST   |     65646  | 1991-12-03 | 3100.00 |           |  2001
 63679 | SANDRINE | CLERK     |     69062  | 1990-12-18 |  900.00 |           |  2001
 64989 | ADELYN   | SALESMAN  |     66928  | 1991-02-20 | 1700.00 |    400.00 |  3001
 65271 | WADE     | SALESMAN  |     66928  | 1991-02-22 | 1350.00 |    600.00 |  3001
 66564 | MADDEN   | SALESMAN  |     66928  | 1991-09-28 | 1350.00 |   1500.00 |  3001
 68454 | TUCKER   | SALESMAN  |     66928  | 1991-09-08 | 1600.00 |      0.00 |  3001
 68736 | ADNRES   | CLERK     |     67858  | 1997-05-23 | 1200.00 |           |  2001
 69000 | JULIUS   | CLERK     |     66928  | 1991-12-03 | 1050.00 |           |  3001
 69324 | MARKER   | CLERK     |     67832  | 1992-01-23 | 1400.00 |           |  1001
(14 rows)
```

1. Display all the data from the table employees.
2. Write a SQL query to find the salaries of all employees
3. Show the name and job of all employees.
4. Change the column 'emp_id' to 'NO' in the database.
5. Change the database name from employees to emp.