# DBMS LAB DEMO-1
# PostgreSQL

## 1. Introduction

PostgreSQL or Postgres is a powerful open-source object-relational database management system that uses and extends the SQL language combined with many features that efficiently store, retrieve, manage and manipulate most of the complicated data.

### 1.1 Database

A database is a collection of data stored in some organized fashion.

### 1.2 DBMS

Database Management System (DBMS) is a software for storing and retrieving users' data while considering appropriate security measures. It consists of a group of programs which manipulate the database. DBMS does all the work of storing, retrieving, managing and manipulating data. Examples are PostgreSQL, MariaDB, MongoDB, etc.

### 1.3 Table:

A structured list of data of a specific type.

- Column: a single field in a table. All tables are made up of one or more columns
- Datatype: a type of allowed data. Every table column has an associated data type that restricts (or allows) specific data in that column. Datatypes restrict the type of data that can be stored in a column.
- Row: a record in a table.
- Primary key: a column (or set of columns) whose values uniquely identify every row in a table.

### SQL (Structured Query Language)

SQL is a language designed specifically for communicating with databases.

## Working with PostgresSQL

**psql** is a terminal-based front-end to PostgreSQL

**Opening PostgresSQL from command line:**

```
psql -U username
```

By default PostgreSQL installed with a super user *'postgres'.* so we can open PostgreSQL as follow,

```
psql -U postgres
```

## Creating new user

For creating a user, we can use CREATE USER command along with username and password.

```
CREATE USER username WITH ENCRYPTED PASSWORD 'password'
```

**List all the users in the current database server:**

```
\du
```

**Setting new password to a user:**

```
ALTER USER username PASSWORD ‘newpassword’;
```

## Creating a database

To create a table, you need to use CREATE DATABASE followed by name of the database you want to create. Syntax is as follow,

```
CREATE DATABASE database_name;
```

For listing all the database in the server, you can use following command

*\l* or *\list*

For connecting to a particular database, you can use the following command,

```
\c database_name or \connect database_name;
```

For directly connect to a database at the starting of psql, you can use following command,

```
psql -U username database_name
```

## Creating a table and Inserting data

To create a table using CREATE TABLE, you must specify the following information:

- The name of the new table specified after the keywords CREATE TABLE.
- The name and definition of the table columns separated by commas.

Example: creating the customer table.

```
CREATE TABLE customer
(
cust_id int NOT NULL AUTO_INCREMENT,
cust_name char(50) NOT NULL ,
cust_address char(50) NULL ,
cust_city char(50) NULL ,
cust_state char(5) NULL ,
cust_zip char(10) NULL ,
cust_country char(50) NULL ,
cust_contact char(50) NULL ,
cust_email char(255) NULL ,
PRIMARY KEY (cust_id)
);
```

**Column constraints**

1. NOT NULL - ensures that values in a column cannot be NULL

2. UNIQUE – ensures the values in a column unique across the rows within the same table.

3. 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.

4. CHECK – a CHECK constraint ensures the data must satisfy a boolean expression.

5. 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

**Inserting data into a table**

INSERT is used to insert (add) rows to a database table.

**Inserting complete rows**

Basic INSERT syntax, which requires that you specify the table name and the values to be inserted into the new row.

```
INSERT INTO Customers
VALUES(NULL, 'Pep E. LaPew','100 Main Street', 'Los Angeles','CA', '90046', 'USA',NULL,NULL);
```

The safer way to write the INSERT statement is as follows:

```
INSERT INTO customers
(cust_name,cust_address, cust_city,cust_state,cust_zip,cust_country,cust_contact,cust_email)
VALUES('Pep E. LaPew','100 Main Street','Los Angeles','CA','90046','USA',NULL,NULL);
```

**Updating Tables**

To update table definitions, the ALTER TABLE statement is used.

Adding a column to a table:

```
ALTER TABLE vendors
ADD vend_phone CHAR(20);
```

This statement adds a column named vend_phone to the vendors table. The datatype must be specified.

**Deleting Tables**

Deleting tables (actually removing the entire table, not just the contents) is easy—arguably too easy. Tables are deleted using the DROP TABLE statement:

```
DROP TABLE customers2;
```

There is no confirmation, nor is there an undo—executing the statement permanently removes the table.

**Renaming Tables**

To rename a table, use the RENAME TABLE statement as follows:

```
RENAME TABLE customers2 TO customers;
```

## Retrieving data

Using the SELECT statement we retrieve one or more columns of data from a table.

**Retrieving Individual Columns:**

```
SELECT prod_name
FROM products;
```

The above statement retrieves a single column called prod_name from the products table.

**Retrieving Multiple Columns:**

```
SELECT prod_id, prod_name, prod_price FROM products;
```

The above statement retrieves mentioned three columns from the products table.

Note: It is important to note that SQL statements are not case sensitive, so SELECT is the same as select, which is the same as Select.

**Retrieving All Columns:**

```
SELECT  *
FROM products;
```

When a wildcard (*) is specified, all the columns in the table are returned. The columns are in the order in which the columns appear in the table definition.

**Retrieving Distinct Rows:**

For retrieving distinct values, we can use DISTINCT keyword followed by SELECT.

```
SELECT DISTINCT vend_id
FROM products;
```

It returns distinct (unique) vend_id. If you apply distinct on multiple columns all specified column values would be retrieved unless both of the specified columns were same.

```
SELECT DISTINCT vend_id, prod_name
FROM products;
```

It returns distinct vend_ids, product_name pairs.

**Limiting Results:**

SELECT statements return all matched rows, possibly every row in the specified table. To return just the first row or rows, use the LIMIT clause

```
SELECT prod_name
FROM products
LIMIT 5;
```

The previous statement uses the SELECT statement to retrieve a single column. LIMIT 5 instructs Postgres to return no more than five rows. LIMIT 3,4 means 3 rows starting from row 4.

**Using Fully Qualified Table Names:**

It is also possible to refer to columns using fully qualified names (using both the table and column names)

```
SELECT products.prod_name
FROM products;
```

## Sorting Retrieved Data

To explicitly sort data retrieved using a SELECT statement, the ORDER BY clause is used. ORDER BY takes the name of one or more columns by which to sort the output

```
SELECT prod_name
FROM products
ORDER BY prod_name;
```

This will sort product_name in alphabetical order (by default ascending order). For sorting it in reverse alphabetical order we can specify the direction of sorting (by default its ascending, ASC). It's shown in following commands,

```
SELECT prod_name
FROM products
ORDER BY prod_name DESC;
```

Some time it is necessary to sort data by more than one column. That we can do as following commands,

```
SELECT prod_id, prod_price, prod_name
FROM products
ORDER BY prod_price, prod_name;
```

This command will first sort the data with prod_price in ascending order and for same price it will order prod_name in alphabetical order.

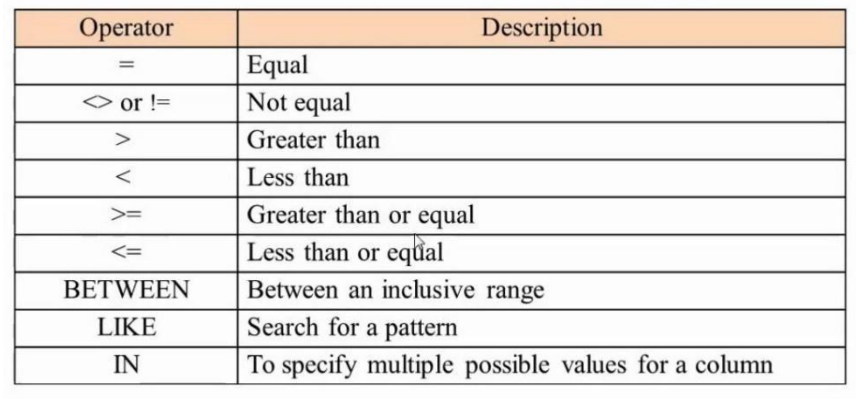
## Filtering Data

Retrieving just the data you want involves specifying search criteria, also known as a filter Condition. Within a SELECT statement, data is filtered by specifying search criteria in the WHERE clause. The WHERE clause is specified right after the table name.

```
SELECT prod_name, prod_price
FROM products
WHERE prod_price = 2.50;
```

This statement retrieves two columns from the products table, but instead of returning all rows, only rows with a prod_price value of 2.50 are returned.

**Where clause operators:**



| Operator | Description |
|---|---|
| = | Equal |
| <> or != | Not equal |
| > | Greater than |
| < | Less than |
| >= | Greater than or equal |
| <= | Less than or equal |
| BETWEEN | Between an inclusive range |
| LIKE | Search for a pattern |
| IN | To specify multiple possible values for a column |

**Checking for a Range of Values**

To check for a range of values, you can use the BETWEEN operator.

```
SELECT prod_name, prod_price
FROM products
WHERE prod_price BETWEEN 5 AND 10;
```

**Combining WHERE Clauses Using the AND Operator:**

To filter by more than one column, you use the AND operator to append conditions to your WHERE clause.

```
SELECT prod_id, prod_price, prod_name
FROM products
WHERE vend_id = 1003 AND prod_price <= 10;
```

**Using the IN Operator:**

The IN operator is used to specify a range of conditions, any of which can be matched. IN takes a comma-delimited list of valid values, all enclosed within parentheses.

```
SELECT prod_name, prod_price
FROM products
WHERE vend_id IN (1002,1003)
```

```
ORDER BY prod_name;
```

This statement will retrieve all products made by vend_ids 1002 and 1003.

## Wildcard Filtering

Wildcards are Special characters used to match parts of a value. Wildcard searches can be performed using the LIKE operator for sophisticated filtering of retrieved data. Using wildcards, you can create search patterns that can be compared against your data.

**The Percent Sign (%) Wildcard**
The most frequently used wildcard is the percent sign (%). Within a search string, % means match any number of occurrences of any character. For example,

```
SELECT prod_id, prod_name
FROM products
WHERE prod_name LIKE 'jet%;
```

This SLECT statement find all products that start with the word jet regardless of how many characters there are.

```
SELECT prod_id, prod_name
FROM products
WHERE prod_name LIKE '%anvil%';
```

The search pattern '%anvil%' means match any value that contains the text anvil anywhere within it, regardless of any characters before or after that text.

**The Underscore (_) Wildcard:**
Another useful wildcard is the underscore (_). The underscore is used just like %, but instead of matching multiple characters, the underscore matches just a single character.

```
SELECT prod_id, prod_name
FROM products
WHERE prod_name LIKE '_ ton anvil';
```

**Reading Resources**

1. https://www.postgresqltutorial.com/

2. https://www.postgresql.org/docs/9.5

3. https://www.javatpoint.com/