**Data** **Base :**

* Data is nothing but value.
* Data base is a repository and is used to store data
* It is an example for secondary memory.
* There are many database available like MySQL, Sybase, Oracle, Mango DB, Informix, Postgre, SQL Server etc.

**DBMS** :

* It is a software management system to maintain database.
* the software required to perform the task of database management is called as DBMS.

**RDBMS :**

* A relational database management system is an implementation of database management system or software that is used to maintain database.
* In RDABMS data will be stored in many tables and all the tables will be related wirth each other with the help of Primary key and Foreign Key.
* Rdbms supports normalization.

**ORDBMS :**

* An object relational database management system is also a database management system.
* ORDBMS database is used to bridge the gap between relational world and object Oriented Programming language like java,c++ etc. we can map directly any language object with the tabls of relational database.

**SQL Commands**

These are the some important SQL command:

CREATE DATABASE : it creates a new database.

ALTER DATABASE : It is used to modify a database.

CREATE TABLE : it creates a new table.

ALTER TABLE : it is used to modify the table.

DROP TABLE : it deletes a table.

SELECT : it extracts data from a database.

UPDATE : it updates data in database.

DELETE : it deletes data from database.

INSERT INTO : it inserts new data into a database.

CREATE INDEX : it is used to create an index (search key).

DROP INDEX : it deletes an index.

**Postgresql :**

* PostgreSQL is an open-source object-relational database management system (ORDBMS).
* It is used to store data securely.
* PostgreSQL (also pronounced as Post-gress-Q-L) is developed by the PostgreSQL Global Development Group
* PostgreSQL is cross platform and runs on many operating systems such as Linux,and Microsoft Windows etc.

**PostgreSQL Tools :**

psql :

* It is a command line tool and the primary tool to manage PostgreSQL. pgAdmin
* It is a free and open source graphical user interface administration tool for PostgreSQL.

PgAdmin :

* It is a web-based administration tool for PostgreSQL written in PHP. It is based on phpMyAdmin tool to manage MySQL.OpenOffice.org Base
* It can be used as a front end tool to PostgreSQL.

Steps to create postgresql DataBase :

#\l - show the databases.

#create database test - to create Database.

#\c - connect to database;

#/d - Describe table.

#\q - Disconnect all databases.

#drop database test - drop the database.

#\! - to clear screen

PostgreSQL Data Types :

**String Datatypes :**

**char(size)** : Where size is the number of characters to store. Fixed-length strings.

**varchar(size)** : Where size is the number of characters to store. Variable-length string.

**text :** Variable-length string.

**Numeric Datatypes :**

**integer** : Equivalent to int4. 4-byte signed integer.

**money** : Currency value.

**Date/Time Datatypes :**

**date** : Displayed as 'YYYY-MM-DD'.

**timestamp**  : Displayed as 'YYYY-MM-DD HH:MM:SS'.

**timestamp without time zone**: Displayed as 'YYYY-MM-DD HH:MM:SS'.

# PostgreSQL Create Table :

In PostgreSQL, CREATE TABLE statement is used to create a new table in any of the given database.

**Syntax:**

CREATE TABLE table\_name(

   column1 datatype,

    column2 datatype,

    .....

    columnN datatype,

    PRIMARY KEY( one or more columns )

**Example :**

CREATE TABLE employees(

ID INT PRIMARY KEY NOT NULL,

NAME TEXT NOT NULL,

AGE INT NOT NULL,

ADDRESS CHAR(50),

SALARY REAL

);

# PostgreSQL INSERT Statement

In PostgreSQL, the INSERT query is used to insert new rows into a table. You can insert a single row or multiple rows at a time into the table.

**Syntax:**

INSERT INTO TABLE\_NAME (column1, column2, column3,...columnN)

VALUES (value1, value2, value3,...valueN);

**Note: column1, column2,...columnN are the names of the columns in the table into which you want to insert data.**

**Example :**

INSERT INTO "EMPLOYEES"(

"ID", "NAME", "AGE", "ADDRESS", "SALARY")

VALUES (1, 'Ajeet', 25, 'Mau ', 65000.00 )

# PostgreSQL SELECT Query :

In PostgreSQL, the SELECT statement is used to retrieve data from a database table. Data is returned in the form of result table. These result tables are called result-sets.

**Syntax:**

SELECT "column1", "column2".."column" FROM "table\_name";

Here, column1, column2,?..columnN specifies the columns from where you retrieve the data. If you want to retrieve all the fields from the table then you have to use the following

**syntax:**

SELECT \* FROM  "table\_name";

**Example :**

SELECT "ID", "NAME", "AGE", "ADDRESS", "SALARY"

FROM "EMPLOYEES";

or

SELECT\* FROM "EMPLOYEES";

PostgreSQL UPDATE statement :

In PostgreSQL, UPDATE statement is used to modify the existing records in a table. To update selected rows, you have to use WHERE clause otherwise all rows would be updated.

**Syntax:**

Following is the basic syntax of update statement:

UPDATE table\_name

SET column1 = value1, column2 = value2...., columnN = valueN

WHERE [condition];

# PostgreSQL DELETE statement

The DETETE statement is used to delete the existing records from a table. The "WHERE" clause is used to delete the selected records, otherwise all the records would be deleted.

**Syntax:**

Following is the basic syntax for DELETE statement:

DELETE FROM table\_name

WHERE [condition];

**Example |:**

DELETE FROM "EMPLOYEES"

WHERE "ID" = 1;

# PostgreSQL WHERE Clause

The WHERE clause specifies a condition while you fetch data from a table or a join of multiple tables. It is generally used with SELECT, UPDATE and DELETE statements to filter the results. It returns the specific result only when the condition is satisfied.

## WHERE Clause with SELECT statement

**Syntax:**

**SELECT** column1, column2, ..... columnN

**FROM** table\_name

**WHERE** [search\_condition]

**Note: WHERE condition can be used with logical operators such as >, <, =, LIKE, NOT, OR, AND etc**

**Execute the following query to return a single record:**

**SELECT** \*

**FROM** "EMPLOYEES"

**WHERE** "NAME" = 'Ajeet';

WHERE clause with multiple conditions

**Execute the following query:**

**SELECT** \*

**FROM** "EMPLOYEES"

**WHERE** "SALARY" = '65000'

AND "ID" <= 4;

# PostgreSQL ORDER BY clause

The PostgreSQL ORDER BY clause is used for sorting data in ascending or descending order. The data is sorted on the base on one or more columns.

**Syntax:**

**SELECT** **column**-list

**FROM** table\_name

[**WHERE** condition]

[**ORDER** **BY** column1, column2, .. columnN] [**ASC** | **DESC**];

Parameters explanation:

**column\_list:** It specifies the columns or calculations that you want to retrieve.

**table\_name:** It specifies the tables that you want to retrieve records from. There must be at least one table listed in the FROM clause.

**WHERE conditions:** It is optional. It specifies that the condition must be fulfilled to retrieve records.

**ASC:** It is also optional. It sorts the result set in ascending order by expression (default, if no modifier is provider).

**DESC:** It is also optional. It sorts the result set in descending order by expression.

ORDER BY: ASC

Execute the following query to retrieve the records ORDER BY AGE in ascending order:

**SELECT** \*

**FROM** "EMPLOYEES"

**ORDER** **BY** "AGE" **ASC**;

ORDER BY: DESC

Execute the following query to retrieve the records ORDER BY NAME in descending order:

**SELECT** \*

**FROM** "EMPLOYEES"

**ORDER** **BY** "NAME" **DESC**;

# PostgreSQL GROUP BY Clause

The PostgreSQL GROUP BY clause is used to group together those rows in a table that have identical data. It is used with SELECT statement.

The GROUP BY clause collects data across multiple records and groups the result by one or more columns. It is also used to reduce the redundancy in the output.

**Syntax:**

SELECT column-list

FROM  table\_name

WHERE [conditions ]

GROUP BY column1, column2....columnN

ORDER BY column1, column2....columnN

**Note:** In case of GROUP BY multiple columns you make sure whatever column you are using to group, that column should be available in column-list.

**Example :**

SELECT "NAME", SUM("SALARY")

FROM "EMPLOYEES"

GROUP BY "NAME";

# PostgreSQL Having Clause

In PostgreSQL, HAVING clause is used in combination with GROUP BY clause to pick out particular rows where the function's result meets some condition.

**Syntax:**

SELECT column1, column2

FROM table1, table2

WHERE [ conditions ]

GROUP BY column1, column2

HAVING [ conditions ]

ORDER BY column1, column2

**Example :**

SELECT "NAME"

FROM "EMPLOYEES"

GROUP BY "NAME" HAVING COUNT ("NAME") **<** **2**;

# PostgreSQL Conditions

PostgreSQL conditions are used to get more specific results form a database. They are generally used with WHERE clause. Conditions with a clause act like a double layer filter.

Following is a list of PostgreSQL conditions:

* AND Condition
* OR Condition
* AND & OR Condition
* NOT Condition
* LIKE Condition
* IN Condition
* NOT IN Condition
* BETWEEN Condition

# PostgreSQL AND condition

The PostgreSQL AND condition is used with WHERE clause to select unique data from more than one column in a table.

**Syntax:**

 SELECT column1, column2, ..... columnN

FROM table\_name

WHERE [search\_condition]

AND [search\_condition];

**Example :**

SELECT \*

FROM "EMPLOYEES"

WHERE "SALARY" = '65000'  AND "ID" **<**= 4;

# PostgreSQL OR condition

The PostgreSQL OR condition is used with WHERE clause to select unique data either from one column or more than one column in a table.

**Syntax:**

SELECT column1, column2, ..... columnN

FROM table\_name

WHERE [search\_condition]

OR [search\_condition];

Example :

SELECT \*

FROM "EMPLOYEES"

WHERE "NAME" = 'Ajeet'

OR "ADDRESS" = 'Shimla';

# PostgreSQL AND & OR condition

The PostgreSQL AND & OR condition provides the advantages of AND and OR condition both in just a single query.

**Syntax:**

SELECT column1, column2, ..... columnN

FROM table\_name

WHERE [search\_condition]  AND [search\_condition]

OR [search\_condition];

**Example :**

SELECT \*

FROM "EMPLOYEES"

WHERE ("NAME" = ''Ajeet' AND "ADDRESS" = 'Mau')

OR ("ID"**>**= '1');

# PostgreSQL NOT condition

The PostgreSQL NOT condition is used with WHERE clause to negate a condition in a query.

**Syntax:**

SELECT column1, column2, ..... columnN

FROM table\_name

WHERE [search\_condition] NOT [condition];

**Example** :

SELECT \*

FROM "EMPLOYEES"

WHERE "AGE" IS NOT NULL;

# PostgreSQL LIKE condition

The PostgreSQL LIKE condition is used with WHERE clause to fetch data from a table where specified condition satisfies the LIKE condition.

**Syntax:**

SELECT column1, column2, ..... columnN

FROM table\_name

WHERE [search\_condition] LIKE [condition];

Example :

SELECT \*

FROM "EMPLOYEES"

WHERE "NAME" LIKE 'Aj%';

# PostgreSQL IN condition

The PostgreSQL IN condition is used with WHERE clause to fetch data from a table where specified condition satisfies the IN condition.

**Syntax:**

SELECT column1, column2, ..... columnN

FROM table\_name

WHERE [search\_condition] IN [condition];

**Example :**

SELECT \*

FROM "EMPLOYEES"

WHERE "AGE" IN (25, 27);

# PostgreSQL NOT IN condition

The PostgreSQL NOT IN condition is used with WHERE clause to fetch data from a table where specified condition negates the IN condition.

**Syntax:**

SELECT column1, column2, ..... columnN

FROM table\_name

WHERE [search\_condition] NOT IN [condition];

**Example** :

 SELECT \*

FROM "EMPLOYEES"

WHERE "AGE" NOT IN (25, 27);

# PostgreSQL BETWEEN condition

The PostgreSQL BETWEEN condition is used with WHERE clause to fetch data from a table between two specified conditions.

**Syntax:**

SELECT column1, column2, ..... columnN

FROM table\_name

WHERE [search\_condition] BETWEEN [condition];

**Example** :

SELECT \*

FROM "EMPLOYEES"

WHERE "AGE" BETWEEN 25 AND 27;

Primary key :

A column or columns is called primary key (PK) that uniquely identifies each row in the table.

If you want to create a primary key, you should define a PRIMARY KEY constraint when you create or modify a table.

|  |  |
| --- | --- |
|  | **Syntax :** |

ALTER TABLE products

ADD PRIMARY KEY (product\_no)

Remove primary key :

ALTER TABLE products

DROP CONSTRAINT products\_pkey;

FOREIGN KEY :

A FOREIGN KEY is a key used to link two tables together.

A FOREIGN KEY is a field (or collection of fields) in one table that refers to the PRIMARY KEY in another table

ALTER TABLE child\_table

ADD CONSTRAINT constraint\_fk

FOREIGN KEY (c1)

REFERENCES parent\_table(p1)

ON DELETE CASCADE;

Sequences :

Use the CREATE SEQUENCE statement to create a sequence, which is a database object from which multiple users may generate unique integers. You can use sequences to automatically generate primary key values

CREATE SEQUENCE DEPT\_SEQ

START WITH 10

INCREMENT BY 10;

INSERT INTO DEPARTMENT(DEPARTMENT\_ID,DEPARTMENT\_NAME) VALUES(NEXTVAL('DEPT\_SEQ'),'TRANSPORT');

# PostgreSQL JOIN

In PostgreSQL, there are following types of joins:

* The INNER JOIN
* The LEFT OUTER JOIN
* The RIGHT OUTER JOIN
* The FULL OUTER JOIN
* The CROSS JOIN

## PostgreSQL INNER JOIN

The PostgreSQL Inner Join is also known as join or simple join. This is the most common type of join. This join returns all rows from multiple tables where the join conditions are satisfied.

**Syntax:**

**SELECT** table1.columns, table2.columns

**FROM** table1

**INNER** JOIN table2

**ON** table1.common\_filed = table2.common\_field;

PostgreSQL INNER JOIN Example

**SELECT** "EMPLOYEES"."ID", "EMPLOYEES"."NAME", "DEPARTMENT"."DEPT"

**FROM** "EMPLOYEES"

**INNER** JOIN "DEPARTMENT"

**ON** "EMPLOYEES"."ID" = "DEPARTMENT"."ID";

# PostgreSQL Outer Join

The Outer Join is an extension of Inner Join. There are three types of Outer Joins.

* Left Outer Join
* Right Outer Join
* Full Outer Join

## Left Outer Join:

The left outer join returns all rows from the left hand table specified in the "ON" condition and only rows from the other table where the condition is fulfilled.

**Syntax:**

SELECT table1.columns, table2.columns

FROM table1

LEFT OUTER JOIN table2

ON table1.common\_filed = table2.common\_field;

Example :

SELECT "EMPLOYEES"."ID", "EMPLOYEES"."NAME", "DEPARTMENT"."DEPT"

FROM "EMPLOYEES"

LEFT OUTER JOIN "DEPARTMENT"

ON "EMPLOYEES"."ID" = "DEPARTMENT"."ID";

# Right Outer Join

The right outer join returns all rows from the right hand table specified in the ON condition and only those rows from the other table where the inner join condition is true.

The right outer join is absolute opposite of left outer join.

**Syntax:**

SELECT table1.columns, table2.columns

FROM table1

RIGHT OUTER JOIN table2

ON table1.common\_filed = table2.common\_field;

**Example** :

 SELECT "EMPLOYEES"."ID", "EMPLOYEES"."NAME", "DEPARTMENT"."DEPT"

FROM "EMPLOYEES"

RIGHT OUTER JOIN "DEPARTMENT"

ON "EMPLOYEES"."ID" = "DEPARTMENT"."ID";

# Full Outer Join

The FULL outer join returns all rows from the LEFT-hand table and RIGHT-hand table both. It puts NULL in place where the join condition is not satisfied.

**Syntax:**

SELECT table1.columns, table2.columns

FROM table1

FULL OUTER JOIN table2

ON table1.common\_filed = table2.common\_field;

**Example :**

SELECT "EMPLOYEES"."ID", "EMPLOYEES"."NAME", "DEPARTMENT"."DEPT"

FROM "EMPLOYEES"

FULL OUTER JOIN "DEPARTMENT"

ON "EMPLOYEES"."ID" = "DEPARTMENT"."ID";

# PostgreSQL Cross Join

The CROSS JOIN matches every row of the first table with every row of the second table. It is also known as Cartesian product join. If the table1 has "x" columns and table2 has "y" columns then the resultant table will has the (x+y) columns.

**Syntax:**

SELECT coloums

FROM table1

CROSS JOIN table2

Example :

SELECT "NAME", "DEPT"

FROM "EMPLOYEES"

CROSS JOIN "DEPARTMENT";

# PostgreSQL Alias

PostgreSQL Aliases are used to provide temporary names for columns or tables. You can create a temporary name for a column or a table by using PostgreSQL Alias.

Generally a temporary table is created when you perform self join.

## PostgreSQL Alias Column

**Syntax:**

**SELECT** column\_name **AS** alias\_name

**FROM** table\_name

conditions...  ;

**Parameter explanation:**

**column\_name:** It specifies the original name of the column that you want to alias.

**alias\_name:** It specifies the temporary name that is assigned to the column.

**table\_name:** It specifies the name of the table.

**AS:** It is optional. Most programmers will specify the AS keyword when aliasing a column name, but not when aliasing a table name.

**Note:**

* If the alias\_name contains spaces, you must enclose the alias\_name in quotes.
* It is acceptable to use spaces when you are aliasing a column name. However, it is not generally a good practice to use spaces when you are aliasing a table name.
* The alias\_name is only valid within the scope of the SQL statement.

**Example :**

**SELECT** "NAME", **MAX**("SALARY") **AS** PACKAGE

**FROM** "EMPLOYEES"

**GROUP** **BY** "NAME";

# PostgreSQL Indexes

## What are Indexes

Indexes are the special lookup tables that are used to speed up the retrieval of data from the databases.

A database index is similar like the index of a book. An index creates an entry for each value that appears in the indexed columns.

## Important features of database indexes

* An index speeds up data output with SELECT query and WHERE clause but it slows down data input with INSERT and UPDATE statement.
* You can create or drop indexes without affecting the data.
* You can create an index by using CREATE INDEX statement by specifying the index name and table or column name on which index is created.
* You can also create a unique index, similar to the UNIQUE constraint, in that the index prevents duplicate entries in the column or combination of columns on which there's an index.

## PostgreSQL Create Index

The CREATE INDEX statement is used to create a PostgreSQL index.

**Syntax:**

**CREATE** **INDEX** index\_name **ON** table\_name;

## Index Types

There are several index types in PostgreSQL like B-tree, Hash, GiST, SP-GiST and GIN etc. Each index types use a different algorithm according to different queries. By default, the CREATE INDEX command uses B-tree indexes.

## Single Column Indexes

An index is called a single column index if it is created based on only one table column.

**Syntax:**

**CREATE** **INDEX** index\_name

**ON** table\_name (column\_name);

Example :

**CREATE** **INDEX** employees\_index

**ON** EMPLOYEES (**name**);

PostgreSQL Drop Indexes

The DROP INDEX method is used to drop an index in PostgreSQL. If you drop an index then it can slow or improve the performance.

**Syntax:**

**DROP** **INDEX** index\_name;

# PostgreSQL View

In PostgreSQL, a VIEW is a pseudo table. It is not a physical table but appear as ordinary table to select.

A view can also represent joined tables. It can contain all rows of a table or selected rows from one or more tables.

A View facilitates users to do the following:

* It structures data in a natural and intuitive way and make it easy to find.
* It restricts access to the data such that a user can only see limited data instead of complete data.
* It summarizes data from various tables to generate reports.

## PostgreSQL Create View

PostgreSQL view can be created by using CREATE VIEW statement. You can create it from a single table, multiple tables and another view.

**Syntax:**

CREATE [TEMP | TEMPORARY] VIEW view\_name AS

SELECT column1, column2.....

FROM table\_name

WHERE [condition];

**Example** :

CREATE VIEW current\_employees AS

SELECT "NAME", "ID", "SALARY"

FROM "EMPLOYEES";

**Java JDBC connection :**

Class.forName("org.postgresql.Driver");

Connection connection=DriverManager.getConnection ( "jdbc:postgresql://hostname:port/dbname","username", "password");

**Ex :**

Class.forName("org.postgresql.Driver");

connection = DriverManager.getConnection("jdbc:postgresql://127.0.0.1:5432/testdb", "sachinpostresql", "12345");