**PostgreS-Q-L**

**DATA:** Data is nothing but a value.

**DATA BASE** **:** Data base is a repository where we store the data.

**DBMS:**

* Data Base Management System is a software used to maintain the database.
* The software required to perform the task of database management is called DBMS.
* Management of data means 1) Providing mechanism for data manipulation.

2) Providing data security against unauthorized access.

**RDBMS:**

* RDBMS is a implementation of DBMS.
* In RDBMS data will be stored in tabular format.
* In RDBMS data will be stored in many tables and all the tables are related with Primary key and Foreign key.

**ORDBMS:**

* Object relational database management system has some features of RDBMS.
* ORDBMS is used to bridge the gap between Relational world and Object oriented programming language.
* In java application we rarely use ORDBMS, we generally use RDBMS and support of ORM(object relational mapping).

**POSTGRES**: “post – greS - Q –L” is a open source relational database management system and it is not controlled by any corporation or other private entity and the source code is available free of cost.

**DATA TYPES IN POSTGRES:**

**1)Numeric types:** It consist of 2-byte, 4-byte and 8-byte integers, 4-byte and 8-byte floating point numbers and selectable precision decimals.

**Different types of numeric data types are:**

* Small int (2-bytes)
* Integer (4-bytes)
* Big int (8-bytes)
* Decimal (variable)
* Numeric (variable)
* Real (4-bytes)
* Double precision (8-bytes)
* Small serial (2-bytes)
* Serial (4-bytes)
* Big serial(8-bytes)

**2)Monetory types:** The money type stores a currency amount with a fixed fractional precision. Values of the numeric, int and big int data types can be cost to money.

**3)Character types :** general purpose characters are available in postgres are

* Character varying(n), varchar(n) : variable length with limit.
* Character(n),char(n) : fixed length, blank padded.
* Text : variable unlimited length.

**4)Binary data types**: The bytea data type allows storage of binary strings.

**5)Date/Time types:** Postgres support full set of date and time type.

**6)Boolean types:** postgres provides the standard SQL type Boolean. The Boolean data type can have the state true, false and third state unknown which is represented by SQL null values.

**7)Enumerated types:** Enumerated(enum) types are data types that comprise a static,ordered set of values. Unlike other types, enumerated types need to be created using CREATE TYPE command. This type is used to store a static, ordered set of values.

Ex: CREATE TYPE week AS ENUM(‘mon’, ‘tue’, ‘wed’, ‘thu’, ‘fri’, ‘sat’,’sun’);

**8)Geometric type:** It represents 2 dimensional special objects.

**9)Network Address Type:** Postgres offers data type to store IPV4, IPV6 and MAC address.

**10)Range types:** Range types represent data types that uses a range of data. Range type can be discrete ranges(eg:all integer values 1 to n10) or Continuous ranges(eg: any point in time b/w 10:00 am to 11:00 am).

**DDL queries in postgres**

## CREATE DATABASE:

This command will create a database from PostgreSQL shell prompt, but you should have appropriate privilege to create a database

### Syntax–

CREATE DATABASE dbname

where *dbname* is the name of a database to create.

**DROP TABLE** :

statement is used to remove a table definition and all associated data, indexes, rules, triggers, and constraints for that table.

You have to be careful while using this command because once a table is deleted then all the information available in the table would also be lost forever.

**Syntax−**

DROP TABLE table name;

**ALTER TABLE** :

command is used to add, delete or modify columns in an existing table.

You would also use ALTER TABLE command to add and drop various constraints on an existing table.

## Syntax

## ALTER TABLE table name ADD column name datatype;

**TRUNCATE TABLE**

 command is used to delete complete data from an existing table. You can also use DROP TABLE command to delete complete table but it would remove complete table structure from the database and you would need to re-create this table once again if you wish to store some data.It has the same effect as DELETE on each table, but since it does not actually scan the tables, it is faster. Furthermore, it reclaims disk space immediately.

**Syntax**

TRUNCATE TABLE table name

## DML queries in postgres

**INSERT INTO** :

The statement allows one to insert new rows into a table. One can insert a single row at a time or several rows as a result of a query.

## Syntax

## INSERT INTO TABLE\_NAME(column1, column2, column3…… column) VALUES (value1,value2,value3…valueN);

**SELECT:**

  The statement is used to fetch the data from a database table, which returns data in the form of result table. These result tables are called result-sets.

## Syntax

## SELECT column1,column2,column FROM table name;

**UPDATE:**

The statemnt is used to modify the existing records in a table. You can use WHERE clause with UPDATE query to update the selected rows. Otherwise, all the rows would be updated.

## Syntax

## SET column1=vakue1, column2=value2…., column=valueN

## WHERE[condition];

**DELETE:**

The  statement is used to delete the existing records from a table. You can use WHERE clause with DELETE query to delete the selected rows. Otherwise, all the records would be deleted.

## Syntax

## DELETE FROM table name

## WHERE[condition]

**CONSTRTAINTS IN POSTGRES:**

Constraints are the rules enforced on data columns on table. These are used to prevent invalid data from being entered into the database. This ensures the accuracy and reliability of the data in the database.

Constraints could be column level or table level. Column level constraints are applied only to one column whereas table level constraints are applied to the whole table.

CONSTRAINTS in postgres:

**1)NOT NULL Constraint**

**2)UNIQUE Constraint**

**3)PRIMARY Key**

**4)FOREIGN Key**

**5)CHECK Constraint**

**6)EXCLUSION Constraint**

**NOT NULL Constraint:**

By default, a column can hold NULL values. If you do not want a column to have a NULL value, then you need to define NOT NULL constraint on this column.

## UNIQUE Constraint:

The UNIQUE Constraint prevents two records from having identical values in a particular column.

## PRIMARY KEY Constraint

The PRIMARY KEY constraint uniquely identifies each record in a database table. There can be more UNIQUE columns, but only one primary key in a table. Primary keys are important when designing the database tables. Primary keys are unique ids.

## FOREIGN KEY Constraint

A foreign key constraint specifies that the values in a column (or a group of columns) must match the values appearing in some row of another table.

## CHECK Constraint

The CHECK Constraint enables a condition to check the value being entered into a record. If the condition evaluates to false, the record violates the constraint and is not entered into the table.

## EXCLUSION Constraint

Exclusion constraints ensure that if any two rows are compared on the specified columns or expressions using the specified operators, at least one of these operator comparisons will return false or null.

**SEQUENCE IN POSTGRES:**

Sequence is used for creating number sequence. Sequence creation is useful for generating unique numbers.

**Create sequence syntax:**

CREATE SEQUENCE seq\_name

START WITH 10

INCREMENT BY 1;

**Drop sequence syntax:**

DROP SEQUENCE seq\_name;

**Inserting values using sequence:**

INSERT INTO table\_name(column1, column2) VALUES(NEXTVAL(“seq\_name”), value2);

**CLUASES IN POSTGRES:**

**WHERE clause**: It is used to specify a condition while fetching the data from single table or joining with multiple tables.

If the given condition is satisfied, only then it returns specific value from the table. You can filter out rows that you do not want included in the result-set by using the WHERE clause.

The WHERE clause not only is used in SELECT statement, but it is also used in UPDATE, DELETE statement, etc., which we would examine in subsequent chapters.

## Syntax

## SELECT column1, column2, column

## FROM table name

## WHERE[search\_condition]

**LIMIT clause:**

It is used to limit the data amount returned by the SELECT statement.

## Syntax

The basic syntax of SELECT statement with LIMIT clause is as follows –

SELECT column1, column2, column

FROM table\_name

LIMIT [no of rows]

**ORDER BY clause:**

It is used to sort the data in ascending or descending order, based on one or more columns.

## Syntax

The basic syntax of ORDER BY clause is as follows –

SELECT column-list

FROM table\_name

[WHERE condition]

[ORDER BY column1,column2,…..columnN][asc|desc]

**GROUP BY clause**:

It is used in collaboration with the SELECT statement to group together those rows in a table that have identical data. This is done to eliminate redundancy in the output and/or compute aggregates that apply to these groups

The GROUP BY clause follows the WHERE clause in a SELECT statement and precedes the ORDER BY clause.

## Syntax

The basic syntax of GROUP BY clause is given below. The GROUP BY clause must follow the conditions in the WHERE clause and must precede the ORDER BY clause if one is used.

SELECT column-list

FROM table\_name

WHERE [ condition]

GROUP BY column1,column2,…..columnN

ORDER BY column1,column2,…..columnN

**HAVING clause**

It allows us to pick out particular rows where the function's result meets some condition.

The WHERE clause places conditions on the selected columns, whereas the HAVING clause places conditions on groups created by the GROUP BY clause.

## Syntax

The following is the position of the HAVING clause in a SELECT query –

SELECT

FROM

WHERE

GROUP BY

HAVING

ORDER BY

**JOINS IN POSTGRES**

The PostgreSQL **Joins** clause is used to combine records from two or more tables in a database. A JOIN is a means for combining fields from two tables by using values common to each.

Join Types in PostgreSQL are −

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

## The CROSS JOIN

A CROSS JOIN matches every row of the first table with every row of the second table. If the input tables have x and y columns, respectively, the resulting table will have x+y columns. Because CROSS JOINs have the potential to generate extremely large tables, care must be taken to use them only when appropriate.

**syntax**

The following is the syntax of CROSS JOIN –

SELECT FROM table1 CROSS JOIN table2

## The INNER JOIN :

A INNER JOIN creates a new result table by combining column values of two tables based upon the join-predicate. The query compares each row of table1 with each row of table2 to find all pairs of rows, which satisfy the join-predicate. An INNER JOIN is the most common type of join and is the default type of join. You can use INNER keyword optionally.

**syntax**

The following is the syntax of INNER JOIN –

SELECT table1.column1, table2.column2…

FROM table INNER JOIN table2 ON table1.common\_field=table2.common\_field;

## The LEFT OUTER JOIN:

SQL standard defines three types of OUTER JOINs: LEFT, RIGHT, and FULL and PostgreSQL supports all of these.

In case of LEFT OUTER JOIN, an inner join is performed first. Then, for each row in table T1 that does not satisfy the join condition with any row in table T2, a joined row is added with null values in columns of T2. Thus, the joined table always has at least one row for each row in T1.

**syntax**

The following is the syntax of LEFT OUTER JOIN –

SELECT … FROM table1 LEFT OUTER JOIN table2 ON conditional expression

## The RIGHT OUTER JOIN

First, an inner join is performed. Then, for each row in table T2 that does not satisfy the join condition with any row in table T1, a joined row is added with null values in columns of T1. This is the converse of a left join; the result table will always have a row for each row in T2.

**syntax**

The following is the syntax of RIGHT OUTER JOIN –

SELECT… FROM TABLE1 right outer join table2 ON conditional expression..

## The FULL OUTER JOIN

First, an inner join is performed. Then, for each row in table T1 that does not satisfy the join condition with any row in table T2, a joined row is added with null values in columns of T2. In addition, for each row of T2 that does not satisfy the join condition with any row in T1, a joined row with null values in the columns of T1 is added.

**syntax**

The following is the syntax of FULL OUTER JOIN –

SELECT… FROM table1 FULL OUTER JOIN table2 ON conditionale expression

**OPERATORS:**

**AND and OR operators** are used to combine multiple conditions to narrow down selected data in a PostgreSQL statement. These two operators are called conjunctive operators.

These operators provide a means to make multiple comparisons with different operators in the same PostgreSQL statement.

## The AND Operator

The **AND** operator allows the existence of multiple conditions in a PostgreSQL statement's WHERE clause. While using AND operator, complete condition will be assumed true when all the conditions are true. For example [condition1] AND [condition2] will be true only when both condition1 and condition2 are true.

### Syntax

SELECT column1,column2, column

FROM table\_name

WHWRE [condition1] AND [condition];

## The OR Operator

## The OR operator is also used to combine multiple conditions in a PostgreSQL statement's WHERE clause. While using OR operator, complete condition will be assumed true when at least any of the conditions is true. For example [condition1] OR [condition2] will be true if either condition1 or condition2 is true.

### Syntax

SELECT column1,column2,column

FROM table\_name

WHERE [condition1] OR [condition2]…. OR[condition]

**AGGREGATE FUNCTIONS:**

PostgreSQL built-in functions, also called as Aggregate functions, are used for performing processing on string or numeric data.

The following is the list of all general-purpose PostgreSQL built-in functions −

* [PostgreSQL COUNT Function](https://www.tutorialspoint.com/postgresql/postgresql_count_function.htm) − The PostgreSQL COUNT aggregate function is used to count the number of rows in a database table.
* [PostgreSQL MAX Function](https://www.tutorialspoint.com/postgresql/postgresql_max_function.htm) − The PostgreSQL MAX aggregate function allows us to select the highest (maximum) value for a certain column.
* [PostgreSQL MIN Function](https://www.tutorialspoint.com/postgresql/postgresql_min_function.htm) − The PostgreSQL MIN aggregate function allows us to select the lowest (minimum) value for a certain column.
* [PostgreSQL AVG Function](https://www.tutorialspoint.com/postgresql/postgresql_avg_function.htm) − The PostgreSQL AVG aggregate function selects the average value for certain table column.
* [PostgreSQL SUM Function](https://www.tutorialspoint.com/postgresql/postgresql_sum_function.htm) − The PostgreSQL SUM aggregate function allows selecting the total for a numeric column.
* [PostgreSQL ARRAY Functions](https://www.tutorialspoint.com/postgresql/postgresql_array_functions.htm) − The PostgreSQL ARRAY aggregate function puts input values, including nulls, concatenated into an array.
* [PostgreSQL Numeric Functions](https://www.tutorialspoint.com/postgresql/postgresql_numeric_functions.htm) − Complete list of PostgreSQL functions required to manipulate numbers in SQL.
* [PostgreSQL String Functions](https://www.tutorialspoint.com/postgresql/postgresql_string_functions.htm) − Complete list of PostgreSQL functions required to manipulate strings in PostgreSQL.

**VIEWS:**

Views are pseudo-tables. That is, they are not real tables; nevertheless appear as ordinary tables to SELECT. A view can represent a subset of a real table, selecting certain columns or certain rows from an ordinary table. A view can even represent joined tables. Because views are assigned separate permissions, you can use them to restrict table access so that the users see only specific rows or columns of a table.

A view can contain all rows of a table or selected rows from one or more tables. A view can be created from one or many tables, which depends on the written PostgreSQL query to create a view.

Views, which are kind of virtual tables, allow users to do the following −

* Structure data in a way that users or classes of users find natural or intuitive.
* Restrict access to the data such that a user can only see limited data instead of complete table.
* Summarize data from various tables, which can be used to generate reports.

Since views are not ordinary tables, you may not be able to execute a DELETE, INSERT, or UPDATE statement on a view. However, you can create a RULE to correct this problem of using DELETE, INSERT or UPDATE on a view.

**Creating Views**

The PostgreSQL views are created using the **CREATE VIEW**statement. The PostgreSQL views can be created from a single table, multiple tables, or another view.

The basic CREATE VIEW syntax is as follows –

CREATE [TEMP | TEMPORARY] VIEW view name AS

SELECT column1, column2…

FROM table name

WHERE [condition];

**Dropping Views:**

To drop a view, simply use the DROP VIEW statement with the **view\_name**. The basic DROP VIEW syntax is as follows – DROP VIEW view name;

**INDEXES:**

Indexes are special lookup tables that the database search engine can use to speed up data retrieval. Simply put, an index is a pointer to data in a table

For example, if you want to reference all pages in a book that discusses a certain topic, you have to first refer to the index, which lists all topics alphabetically and then refer to one or more specific page numbers.

An index helps to speed up SELECT queries and WHERE clauses; however, it slows down data input, with UPDATE and INSERT statements. Indexes can be created or dropped with no effect on the data.

Creating an index involves the CREATE INDEX statement, which allows you to name the index, to specify the table and which column or columns to index, and to indicate whether the index is in ascending or descending order.

Indexes can also be unique, 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.

### The CREATE INDEX Command

The basic syntax of **CREATE INDEX** is as follows −

CREATE INDEX index\_name ON table\_name;

## Index Types:

PostgreSQL provides several index types: B-tree, Hash, GiST, SP-GiST and GIN. Each Index type uses a different algorithm that is best suited to different types of queries. By default, the CREATE INDEX command creates B-tree indexes, which fit the most common situations.

### Single-Column Indexes

A single-column index is one that is created based on only one table column. The basic syntax is as follows –

CREATE INDEX index name

ON table name (column name);

**Multicolumn Indexes:**

A multicolumn index is defined on more than one column of a table. The basic syntax is as follows –

CREATE INDEX index name

ON table name (column1\_name, column2\_name);

### Unique Indexes:

Unique indexes are used not only for performance, but also for data integrity. A unique index does not allow any duplicate values to be inserted into the table. The basic syntax is as follows –

CREATE UNIQUE INDEX index\_name

ON table\_name (column\_name);

## Partial Indexes:

A partial index is an index built over a subset of a table; the subset is defined by a conditional expression (called the predicate of the partial index). The index contains entries only for those table rows that satisfy the predicate. The basic syntax is as follows –

CREATE INDEX index\_name

ON table\_name (conditional\_expression);

## Implicit Indexes:

Implicit indexes are indexes that are automatically created by the database server when an object is created. Indexes are automatically created for primary key constraints and unique constraints.

**TRIGGERS:**

PostgreSQL **Triggers** are database callback functions, which are automatically performed/invoked when a specified database event occurs.

The following are important points about PostgreSQL triggers −

* PostgreSQL trigger can be specified to fire
  + Before the operation is attempted on a row (before constraints are checked and the INSERT, UPDATE or DELETE is attempted)
  + After the operation has completed (after constraints are checked and the INSERT, UPDATE, or DELETE has completed)
  + Instead of the operation (in the case of inserts, updates or deletes on a view)
* The BEFORE, AFTER or INSTEAD OF keyword determines when the trigger actions will be executed relative to the insertion, modification or removal of the associated row.
* Triggers are automatically dropped when the table that they are associated with is dropped.
* The table to be modified must exist in the same database as the table or view to which the trigger is attached and one must use just **tablename**, not **database.tablename**.

Syntax

The basic syntax of creating a **trigger** is as follows –

CREATE TRIGGER trigger\_name[BEFORE|AFTER|INSTEAD OF] event\_name

ON table\_name

[

--Trigger logic goes here….

];