# PostgreSQL

## What is PostgreSQL

**An SQL engine** which **stores data** and **reads queries** and **returns information**

## What is PgAdmin

A graphical user interface to connect with PostgreSQL

## Credentials

Local laptop superadmin password – password

Master password - password

## Points to note in sql query

* Sql keywords will be typed in uppercase, even though if you don’t type them in upper case, still SQL engine executes them, but typing them in upper case is what will separate your dynamic database information (such as table name, column name) from constant information (ex: SELECT, UPDATE, JOIN, GROUP BY)
  + Ex: SELECT employee\_id FROM employee

# Clauses that can be used as a part of SQL query

SQL queries typically consist of various clauses that provide instructions and specify conditions for retrieving, manipulating, or organizing data. The commonly used clauses in an SQL query are mentioned below with a basic explanation:

* SELECT: Specifies the columns or expressions to retrieve from the database.
* FROM: Specifies the table or tables from which to retrieve data.
* WHERE: Filters rows based on specified conditions.
* GROUP BY: Groups the result set by one or more columns.
* HAVING: Filters groups based on specified conditions after the GROUP BY clause.
* JOIN: Combines rows from multiple tables based on a related column between them.
* ORDER BY: Sorts the result set based on specified columns in ascending or descending order.
* LIMIT (or TOP, FETCH FIRST): Limits the number of rows returned by the query.
* INSERT INTO: Inserts new rows into a table.
* UPDATE: Modifies existing rows in a table.
* DELETE: Removes rows from a table.
* SET: Specifies the column values to be updated in an UPDATE statement.
* DISTINCT: Retrieves only unique/distinct values from a column or a combination of columns.
* UNION: Combines the result sets of multiple SELECT statements.
* EXISTS: Checks for the existence of rows in a subquery.
* IN: Tests for a value's presence in a list or a subquery.
* LIKE: Searches for patterns in a column using wildcard characters.
* BETWEEN: Filters rows based on a range of values.
* CASE: Performs conditional logic and returns different values based on specified conditions.

These clauses can be combined and used in different combinations to create SQL queries that retrieve, manipulate, or organize data in various ways.

# SELECT statement

It is used to fetch (select) certain amount of information from a table.

## How is select query executed by sql engine?

First engine checks whether the table that is being queried exists or not, then it will check whether the columns that are being requested are existing or not, then the data is returned by the SQL engine

## Syntax

SELECT col1,col2,.. FROM table\_name;

## note

* It is not a good practice to use SELECT \* FROM table\_name; as this increases the traffic between the database server and application, which can increase the latency of the results, If you don’t need all the columns, you can only mention the columns that you need
* **All other clauses (such as are used as a part of select**

## SELECT DISTINCT

Distinct is a particular keyword that can be passed inside a SELECT statement.

The functionality of DISTINCT is, if you want to fetch unique results for a particular set of columns.

**Similarly, there are other few functionalities that you can add to SELECT query, such as WHERE, JOIN, HAVING (will be discussed later)**

Ex: if you want to know unique combinations of age and their sex for a particular classroom called B1, you can use DISTINCT keyword

Sample query: SELECT DISTINCT age, sex from school where class\_name = ‘B1’

## SELECT COUNT

COUNT can be used as a parameter to SELECT query to achieve the count of particular result of a SELECT query

COUNT returns the count of the result that SELECT query returns

Example situations:

* Fetch the count of students in a school
  + SELECT COUNT(student\_id) FROM students;
* Fetch the count of students per class in a school
  + SELECT class\_room, COUNT(class\_room) FROM students GROUP BY class\_room;
    - we can also use GROUP BY as part of which we will discuss later
    - note that we need to pass class\_room as an argument because only then we will be able to know which classroom has how many numbers of students
* if you want to fetch the count of distinct names in the school
  + SELECT COUNT(DISTINCT names) from student;
    - Notice that DISTINCT comes after COUNT, similar to d comes after c in alphabets.

### will COUNT include null in its count?

NO, if you pass a particular column, it won’t include null entries of that particular column in to the count

If you want to fetch the count of that column or the count of the records in the database, you need to pass COUNT(\*)

Ex: SELECT COUNT(\*) FROM students;

## SELECT query having all the keywords used in SQL

In the below query you can see major keywords being used, the order is SELECT, COUNT, DISTINCT, FROM, JOIN, WHERE, GROUP BY, HAVING, ORDER BY, LIMIT

SELECT DISTINCT

c.customer\_id,

c.customer\_name,

COUNT(o.order\_id) AS order\_count,

SUM(o.order\_amount) AS total\_amount

FROM

customers c

JOIN

orders o ON c.customer\_id = o.customer\_id

JOIN

products p ON o.product\_id = p.product\_id

WHERE

c.city = 'New York'

AND p.category = 'Electronics'

AND o.order\_date BETWEEN '2022-01-01' AND '2022-12-31'

GROUP BY

c.customer\_id,

c.customer\_name

HAVING

COUNT(o.order\_id) > 2

ORDER BY

total\_amount DESC;

# WHERE clause

Let’s think of use case, if you want to fetch students named Sharadha’s address who is from class 5, you can’t use select alone and get the result

You need a filtering condition, and that filtering is achieved in SQL using WHERE clause, the SQL query for above use case is as follows

SELECT student\_address FROM students WHERE student\_name = “Sharadha” and class=5;

## operators used in SQL WHERE clause

* Equal to (=): Matches rows where the column value is equal to the specified value.
* Not equal to (<> or !=): Matches rows where the column value is not equal to the specified value.
* Greater than (>): Matches rows where the column value is greater than the specified value.
* Less than (<): Matches rows where the column value is less than the specified value.
* Greater than or equal to (>=): Matches rows where the column value is greater than or equal to the specified value.
* Less than or equal to (<=): Matches rows where the column value is less than or equal to the specified value.
* BETWEEN: Matches rows where the column value is within a specified range.
* LIKE: Matches rows where the column value matches a specified pattern (often used with wildcard characters).
* IN: Matches rows where the column value is contained in a specified list of values.
* IS NULL: Matches rows where the column value is null (i.e., has no value).
* IS NOT NULL: Matches rows where the column value is not null.

### Important point to note

A subset of above operators are also used in HAVING, JOIN and DESC as well, the operators that can be used on these clauses are in the descending order of HAVING, JOIN and DESC.

## Logical operators used in where clause

logical operators are used to combine or modify conditions in a query's WHERE clause. The logical operators in SQL are:

* AND: Combines two or more conditions, and the result is true only if all the conditions are true.
  + Example: SELECT \* FROM table WHERE condition1 AND condition2
* OR: Combines two or more conditions, and the result is true if at least one of the conditions is true.
  + Example: SELECT \* FROM table WHERE condition1 OR condition2
* NOT: Negates a condition, and the result is true if the condition is false and vice versa.
  + Example: SELECT \* FROM table WHERE NOT condition

These logical operators allow you to create complex conditions by combining multiple individual conditions. They help you express more advanced logic and provide flexibility in querying and filtering data based on various criteria.

Additionally, **parentheses can be used to group conditions and define the order of evaluation. This is particularly useful when combining different logical operators to create complex and nested conditions.**

# JOIN clause

## Operators that you can use as a part of join clause

In the JOIN clause of an SQL query, you typically use the equality operator (=) to specify the join condition. However, in addition to the equality operator, you can also use other operators in certain situations. The possible operators used in the JOIN clause are:

* Equality (=): The most commonly used operator in joins. It matches rows where the values in the joined columns are equal.
  + SELECT \* FROM table1 JOIN table2 ON table1.column = table2.column
* Non-equality (<> or !=): This operator is used to match rows where the values in the joined columns are not equal.
  + SELECT \* FROM table1 JOIN table2 ON table1.column <> table2.column
* Greater than (>), Less than (<), Greater than or equal to (>=), Less than or equal to (<=): These comparison operators can be used when you want to join rows based on a range of values.
  + SELECT \* FROM table1 JOIN table2 ON table1.column >= table2.column
* IN: This operator allows you to specify a list of values and matches rows where the value in the joined column is present in that list.
  + SELECT \* FROM table1 JOIN table2 ON table1.column IN (value1, value2, value3)
* LIKE: The LIKE operator, often used with wildcard characters (% or \_), matches rows where the value in the joined column matches a specified pattern.
  + SELECT \* FROM table1 JOIN table2 ON table1.column LIKE 'ABC%'
* BETWEEN: The BETWEEN operator is used to match rows where the value in the joined column falls within a specified range.
  + SELECT \* FROM table1 JOIN table2 ON table1.column BETWEEN value1 AND value2

It's important to note that not all database systems support all of these operators in the JOIN clause. The available operators may vary depending on the specific database management system (e.g., MySQL, PostgreSQL, Oracle) you are using.

## CROSS JOIN

Also known as Cartesian join. In a cross join, each row from one table is combined with every row from another table, resulting in a combination of all possible pairs of rows.

Here's an example of a cross join:

SELECT \*

FROM table1

CROSS JOIN table2;

it's important to note that cross joins can potentially produce a large number of rows, especially if the tables involved have many rows. Therefore, cross joins should be used with caution and typically require a specific use case where all possible combinations are desired.

# HAVING clause

In the HAVING clause of an SQL query, you can use various operators to specify conditions for filtering groups based on aggregate values.

## The possible operators used in the HAVING clause are:

* Equality (=): Matches groups where the aggregate value is equal to the specified value.
* Non-equality (<> or !=): Matches groups where the aggregate value is not equal to the specified value.
* Greater than (>): Matches groups where the aggregate value is greater than the specified value.
* Less than (<): Matches groups where the aggregate value is less than the specified value.
* Greater than or equal to (>=): Matches groups where the aggregate value is greater than or equal to the specified value.
* Less than or equal to (<=): Matches groups where the aggregate value is less than or equal to the specified value.
* IN: Matches groups where the aggregate value is contained in a specified list of values.
* LIKE: Matches groups where the aggregate value matches a specified pattern (often used with wildcard characters).
* BETWEEN: Matches groups where the aggregate value falls within a specified range.
* IS NULL: Matches groups where the aggregate value is null (i.e., has no value).
* IS NOT NULL: Matches groups where the aggregate value is not null.

These operators allow you to define specific conditions in the HAVING clause for filtering groups based on the results of aggregate functions such as COUNT, SUM, AVG, MAX, MIN, etc. The choice of operator depends on the desired filtering logic and the type of aggregate value being considered.

## Logical operators used in having clause

logical operators are used to combine or modify conditions in a query's WHERE clause. The logical operators in SQL are:

* AND: Combines two or more conditions, and the result is true only if all the conditions are true.
  + Example: SELECT \* FROM table WHERE condition1 AND condition2
* OR: Combines two or more conditions, and the result is true if at least one of the conditions is true.
  + Example: SELECT \* FROM table WHERE condition1 OR condition2
* NOT: Negates a condition, and the result is true if the condition is false and vice versa.
  + Example: SELECT \* FROM table WHERE NOT condition

These logical operators allow you to create complex conditions by combining multiple individual conditions. They help you express more advanced logic and provide flexibility in querying and filtering data based on various criteria.

Additionally, **parentheses can be used to group conditions and define the order of evaluation. This is particularly useful when combining different logical operators to create complex and nested conditions.**

# Code format style

SELECT \*

FROM table1

CROSS JOIN table2;