**PROMT** : EXPLAIN SELECT COMMAND IN POSTGRESQL IN DETALIED WITH 2 EXAMPLES AS A BEGGINER.EXPLAIN ALL THE OPERATORS WHICH ARE USED WITH SELECT STATEMENT.

**What is SELECT in PostgreSQL?**

The **SELECT** command is used to **retrieve data from a table** (or even from multiple tables).  
It lets you choose **which columns**, **which rows**, and **in what format** you want the data to appear.

The SELECT command in PostgreSQL is used to retrieve data from a table. It's the most common command in SQL and allows you to specify which columns you want to see and which rows you want to filter.

**Syntax**

The basic syntax for the SELECT command is as follows:

SQL

SELECT column\_1, column\_2, ...

FROM table\_name

WHERE condition

GROUP BY column\_name

HAVING condition

ORDER BY column\_name ASC|DESC

LIMIT number;

* **SELECT column\_1, column\_2, ...**: This is where you specify the columns you want to retrieve. You can list specific column names, or use \* to select all columns.
* **FROM table\_name**: This specifies the table from which you want to retrieve data.
* **WHERE condition**: This optional clause filters rows based on a specific condition. It's used to retrieve a subset of rows from the table.
* **GROUP BY column\_name**: This clause groups rows that have the same values in specified columns into summary rows, like "find the number of customers in each city."
* **HAVING condition**: This clause is used with GROUP BY to filter groups based on an aggregate function's result.
* **ORDER BY column\_name ASC|DESC**: This clause sorts the result set in either ascending (ASC) or descending (DESC) order.
* **LIMIT number**: This clause restricts the number of rows returned by the query.

**Operators Used with SELECT**

Operators are like tools that make your queries more powerful.  
Here’s a breakdown:

**A) Comparison Operators**

Used in WHERE to compare values.

| **Operator** | **Meaning** | **Example** |
| --- | --- | --- |
| = | Equal to | age = 25 |
| <> or != | Not equal to | age <> 30 |
| > | Greater than | salary > 50000 |
| < | Less than | age < 18 |
| >= | Greater than or equal | marks >= 50 |
| <= | Less than or equal | price <= 1000 |

**B) Logical Operators**

Used to combine multiple conditions.

| **Operator** | **Meaning** | **Example** |
| --- | --- | --- |
| AND | All conditions must be true | age > 18 AND city = 'Delhi' |
| OR | At least one condition must be true | city = 'Delhi' OR city = 'Mumbai' |
| NOT | Negates a condition | NOT city = 'Delhi' |

**C) Pattern Matching Operators**

Used for searching text patterns.

| **Operator** | **Meaning** | **Example** |
| --- | --- | --- |
| LIKE | Matches a pattern (case-sensitive) | name LIKE 'A%' (starts with A) |
| ILIKE | Matches a pattern (case-insensitive) | name ILIKE 'a%' |
| % | Wildcard: any number of characters | name LIKE '%a' (ends with a) |
| \_ | Wildcard: exactly one character | name LIKE 'A\_' (A + 1 letter) |

**D) Range & Set Operators**

Work with lists and ranges.

| **Operator** | **Meaning** | **Example** |
| --- | --- | --- |
| BETWEEN | Between two values (inclusive) | age BETWEEN 18 AND 25 |
| IN | Matches any value in a list | city IN ('Delhi', 'Mumbai') |
| NOT IN | Not in the list | city NOT IN ('Delhi', 'Mumbai') |

**E) NULL Operators**

Used to check for NULL values.

| **Operator** | **Meaning** | **Example** |
| --- | --- | --- |
| IS NULL | Value is NULL | email IS NULL |
| IS NOT NULL | Value is not NULL | email IS NOT NULL |

**F) Arithmetic Operators**

Used for calculations.

| **Operator** | **Meaning** | **Example** |
| --- | --- | --- |
| + | Addition | salary + 5000 |
| - | Subtraction | marks - 10 |
| \* | Multiplication | price \* quantity |
| / | Division | total / count |
| % | Modulus (remainder) | age % 2 |

**G) Other Useful Clauses with SELECT**

* **DISTINCT** → Removes duplicate rows  
  SELECT DISTINCT city FROM customers;
* **ORDER BY** → Sort results  
  SELECT \* FROM products ORDER BY price DESC;
* **LIMIT** → Restrict number of rows  
  SELECT \* FROM students LIMIT 5;

**Examples**

**Example 1: Basic SELECT with WHERE Clause**

Let's say you have a table named products and you want to find the names and prices of all products that cost more than $50.

SQL

-- Selects the 'product\_name' and 'price' columns

-- from the 'products' table,

-- but only for rows where the 'price' is greater than 50.

SELECT product\_name, price

FROM products

WHERE price > 50;

**Explanation:** This query selects two specific columns, product\_name and price. The WHERE clause then filters the results, ensuring that only rows where the price column has a value greater than 50 are included in the final output. The other rows are ignored.

**Example 2: SELECT with Multiple Operators and Sorting**

Imagine you have an employees table and you want to find the names of employees in the 'Sales' or 'Marketing' departments who were hired after January 1, 2020. You also want the results to be sorted alphabetically by the employee's last name.

SQL

-- Selects the 'first\_name' and 'last\_name' columns.

-- Filters for employees in 'Sales' or 'Marketing' and hired after '2020-01-01'.

-- Sorts the result set by 'last\_name' in ascending order.

SELECT first\_name, last\_name

FROM employees

WHERE (department = 'Sales' OR department = 'Marketing')

AND hire\_date > '2020-01-01'

ORDER BY last\_name ASC;

**Explanation:** This query demonstrates the use of multiple operators. The WHERE clause combines two conditions:

1. The employee's department is either 'Sales' or 'Marketing' (using OR).
2. The employee's hire date is after '2020-01-01' (using >).

The parentheses around the OR condition are important to ensure it's evaluated correctly before the AND condition. Finally, the ORDER BY last\_name ASC clause sorts the filtered results alphabetically by last name, making the output easy to read.

**Key Notes for Beginners**

* Always use WHERE when deleting or updating to avoid affecting all rows by mistake.
* Use **LIMIT** when testing queries to avoid huge outputs.
* SELECT works with not only tables but also **views, subqueries, and joins**.