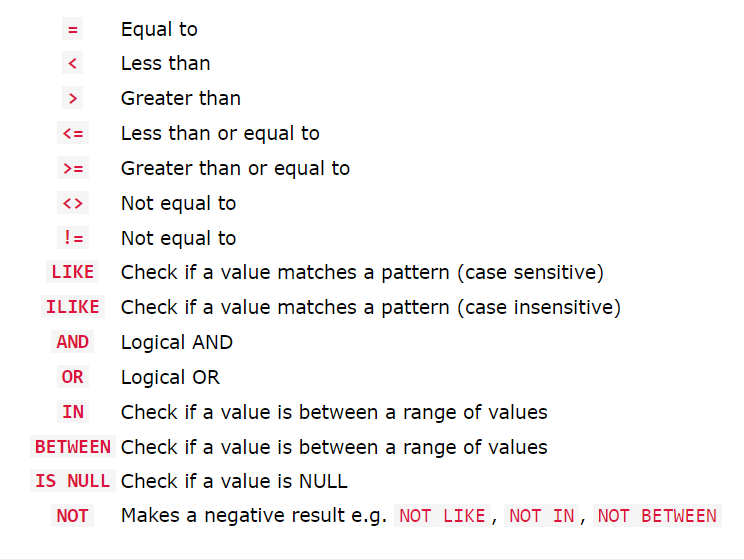
**SQL Operator**

-- Following are the operators we can operate in WHERE clause:



Comparison Operators

1. Equal To

-- The = operator is used when you want to return all records where a column is equal to a specified value.

-- For example, return all the employees whose salary is equal to 300000:

> *SELECT \* FROM dev\_schema.employee WHERE salary = 300000;*

2. Greater than and Less than

-- The < operator is used when you want to return all records where a column is less than a specified value.

-- For example, return all the employee names whose salary is less than 30000:

*> SELECT \* FROM dev\_schema.employee WHERE salary < 30000;*

-- The > operator is used when you want to return all records where a column is greater than a specified value.

-- For example, return all the employee names whose salary is greater than 30000:

*> SELECT \* FROM dev\_schema.employee WHERE salary > 30000;*

3. Greater than or Equal to and Less than or Equal to

-- The >= operator is used when you want to return all records where a column is greater than, or equal to, a specified value.

-- For example, return all the employee names whose salary is greater than or equal to 30000:

*> SELECT \* FROM dev\_schema.employee WHERE salary >= 30000;*

-- The >= operator is used when you want to return all records where a column is less than, or equal to, a specified value.

-- For example, return all the employee names whose salary is less than or equal to 30000:

*> SELECT \* FROM dev\_schema.employee WHERE salary <= 30000;*

4. Not Equal To

-- The <> operator is used when you want to return all records where a column is NOT equal to a specified value.

-- For example, return all the employee names whose salary is not equal to 300000:

*> SELECT \* FROM dev\_schema.employee WHERE salary <> 300000;*

*> SELECT \* FROM dev\_schema.employee WHERE salary != 300000;*

Logical Operators

1. And Operator

-- The logical AND operator is used when you want to check more than one condition.

-- For example, fetch an employee where last\_name = ‘Kulkarni’ and salary = 300000;

> SELECT \* FROM dev\_schema.employee WHERE salary >= 30000;

2. OR Operator

-- The logical OR operator is used when you can accept that only one of many conditions is true.

-- For example, return the employees whose salary is either 3 lacs or 2 lacs.

*> SELECT \* FROM dev\_schema.employee WHERE salary = 300000 OR salary = 200000;*

3. LIKE Operator

-- The LIKE operator is used when you want to return all records where a column is equal to a specified pattern.

-- The PostgreSQL LIKE operator is used query data using pattern matching techniques.

-- Its result includes strings that are **case-sensitive** and follow the mentioned pattern.

-- It is important to know that PostgreSQL provides with 2 special wildcard characters for the purpose of patterns matching as below:

* Percent ( %) for matching any sequence of characters.
* Underscore ( \_) for matching any single character.

-- % wildcard character:

1. Finds any values that start with K

> *SELECT \* FROM dev\_schema.employee WHERE last\_name LIKE 'K%';*

2. Finds any values that have p in any position

> *SELECT \* FROM dev\_schema.employee WHERE last\_name LIKE '%p%';*

3. Finds any value that ends with 'a'

> *SELECT \* FROM dev\_schema.employee WHERE last\_name LIKE '%a';*

-- \_ wildcard character:

1 Finds any values that have 00 in the second and third positions

> *SELECT \* FROM dev\_schema.employee WHERE salary::text LIKE '\_00%';*

2. Finds any values that have 8 in the second position and end with 0

> *SELECT \* FROM dev\_schema.employee WHERE salary::text LIKE '\_8%0';*

4. ILIKE Operator

-- The PostgreSQL ILIKE operator is used query data using pattern matching techniques. Its result includes strings that are **case-insensitive** and follow the mentioned pattern.

-- It is important to know that PostgreSQL provides with 2 special wildcard characters for the purpose of patterns matching as below:

* Percent ( %) for matching any sequence of characters.
* Underscore ( \_) for matching any single character.

-- % wildcard character:

1. Finds any values that start with K

> *SELECT \* FROM dev\_schema.employee WHERE last\_name ILIKE 'K%';*

2. Finds any values that have p in any position

> *SELECT \* FROM dev\_schema.employee WHERE last\_name ILIKE '%p%';*

3. Finds any value that ends with 'a'

> *SELECT \* FROM dev\_schema.employee WHERE last\_name ILIKE '%a';*

-- \_ wildcard character:

1 Finds any values that have 00 in the second and third positions

> *SELECT \* FROM dev\_schema.employee WHERE salary::text ILIKE '\_00%';*

2. Finds any values that have 8 in the second position and end with 0

> *SELECT \* FROM dev\_schema.employee WHERE salary::text ILIKE '\_8%0';*

5. IN Operator

-- The PostgreSQL IN operator is used with the WHERE clause to check against a list of values.

-- Using IN operator with the WHERE clause to check against a list of values which returns a boolean value depending upon the match.

1. Select the employees with list of emp id:

> SELECT \* FROM dev\_schema.employee WHERE employee\_id IN ('TechM11', 'TechM13', 'TechM16');

6. NOT IN Operator

-- The NOT IN operator is used to exclude the rows that match values in the list.

-- It returns all the rows except the excluded rows.

> SELECT \* FROM dev\_schema.employee WHERE employee\_id NOT IN ('TechM11', 'TechM13', 'TechM16');

7. BETWEEN Operator

-- In SQL, the BETWEEN operator is used with the WHERE clause to match values in a range.

-- The syntax of the SQL BETWEEN operator is:

SELECT column1, column2, ...

FROM table

WHERE column BETWEEN value1 AND value2;

> SELECT first\_name, last\_name FROM dev\_schema.employee WHERE salary BETWEEN 200000 AND 300000;

-- Here, the SQL command selects the first\_name and last\_name columns from the employee table where the salary is between 200000 and 300000 (including 200000 and 300000).

8. ANY and ALL

*-- After subquery is done.*

9. EXISTS

*-- After subquery is done.*

10. IS NULL

-- In SQL, IS NULL and IS NOT NULL are used to check if a column in a table contains a NULL value or not.

> SELECT first\_name, last\_name FROM dev\_schema.employee WHERE last\_name IS NULL;

Arithmatic Operator

1. Addition

> SELECT first\_name, last\_name, salary, salary+10000 as appraised\_salary FROM dev\_schema.employee;

2. Subtraction

> SELECT first\_name, last\_name, salary, salary-10000 as depraised\_salary FROM dev\_schema.employee;

3. Multiplication

> SELECT first\_name, last\_name, salary, salary\*2 as appraised\_salary FROM dev\_schema.employee;

4. Division

> SELECT first\_name, last\_name, salary, salary/2 as appraised\_salary FROM dev\_schema.employee;