SQL Operators

Arithmetic operators

- ☑ **Addition operator (+):** Adds two or more numeric values.

  Eg.:

  ```
  SELECT 5 + 4;
  ```

  The result will be 9.

- ☐ **Subtraction Operator (-):** The – symbol subtracts one number from another.

  Eg.:

  ```
  SELECT 10 - 11;
  ```

  The result will be -1;

- ☐ **Multiplication (*):** The * symbol multiplies two numbers together.

  Eg.:

  ```
  SELECT 10 * 10;
  ```

  The result will be 100;

- ☐ **Division (/):** Divides one numeric value by another.

  Eg.:

  ```
  SELECT 10 / 2;
  ```

  The result will be 5.

- ☐ **Remainder/Modulus (%):** The **%** symbol (sometimes referred to as Modulus) returns the remainder of one number divided by another.

  Eg.:

  ```
  SELECT 10 % 4;
  ```

  The result will be 2.

Bitwise operators

- ☐ **Bitwise AND (&):** Performs a bit-by-bit comparison of two expressions and returns a result that has 1s in the positions where both input expressions have 1s and 0s in all other positions.

Eg.:

SELECT 5 & 3;

The result would be 1.

- **Bitwise OR ()**: Performs a bit-by-bit comparison of two expressions and returns a result that has 1s in the positions where either of the input expressions has 1s and 0s in all other positions.

  Eg.:

  SELECT 5 3;

  The result would be 7.

- **Bitwise XOR (^)**: Performs a bit-by-bit comparison of two expressions and returns a result that has 1s in the positions where either of the input expressions has 1s but not both, and 0s in all other positions.

  Eg.:

  SELECT 5 ^ 3;

  The result would be 6.

- **Bitwise NOT (~)**: Inverts all the bits of an expression, effectively changing all 0s to 1s and all 1s to 0s.

  Eg.:

  SELECT ~3;

  The result would be -4.

- **Left shift (<<)**: Shifts all the bits of an expression to the left by a specified number of positions.

  Eg.:

  SELECT 5 << 1;

  The result would be 10.

- **Right shift (>>)**: Shifts all the bits of an expression to the right by a specified number of positions.

  Eg.:

  SELECT 5 >> 1;

  The result would be 2.

Comparison operators

ID	Name	SALARY
1	John	45000
2	Jane	50000
3	Bob	55000
4	Alice	60000

Table: Employees

- **Equal to (=)**: Returns true if two expressions are equal.

  Eg.:

  SELECT Name FROM employees WHERE salary = 50000;

  This query would return Name from the employees table where the value in the salary column is equal to 50000.

  ~ Output

  Jane

- **Not equal to (<> or !=)**: Returns true if two expressions are not equal.

  Eg.:

  SELECT Name FROM employees WHERE salary != 50000;

  This query would return Name from the employees table where the value in the salary column is not equal to 50000.

  ~ Output

  John
  Bob
  Alice

- **Greater than (>):** Returns true if the first expression is greater than the second expression.

  Eg.:

  SELECT Name FROM employees WHERE salary > 50000;

  This query would return Name from the employees table where the value in the salary column is greater than 50000.

  ~ Output

  Bob
  Alice

☐ **Less than (<):** Returns true if the first expression is less than the second expression.

Eg.:

SELECT Name FROM employees WHERE salary < 50000;

This query would return Name from the employees table where the value in the salary column is less than 50000.

~ Output

John

☐ **Greater than or equal to (>=)**: Returns true if the first expression is greater than or equal to the second expression.

Eg.:

SELECT Name FROM employees WHERE salary >= 55000;

This query would return Name from the employees table where the value in the salary column is greater than or equal to 50000.

~ Output

Bob

Alice

☐ **Less than or equal to (<=)**: Returns true if the first expression is less than or equal to the second expression.

Eg.:

SELECT Name FROM employees WHERE salary <= 50000;

This query would return Name from the employees table where the value in the salary column is less than or equal to 50000.

~ Output

John

Jane

Logical operators

first_name	last_name	age	location
John	Doe	35	New york
Jane	Smith	40	London
Bob	Johnson	45	Paris
Alice	Brown	50	London
Charlie	Wilson	30	Tokyo

Table: users

☐ ALL

The **ALL** operator returns **TRUE** if all of the subquery values meet the specified condition. In the below example, we are filtering all users who have an age that is greater than the highest age of users in London.

```
SELECT first_name, last_name, age, location
FROM users
WHERE age > ALL (SELECT age FROM users WHERE location = 'London');
```

Output:

first_name	last_name	age	location
Bob	Johnson	45	Paris
Alice	Brown	50	London

☐ ANY/SOME

The ANY operator returns TRUE if any of the subquery values meet the specified condition. In the below example, we are filtering all products which have any record in the orders table. The SOME operator achieves the same result.

```
SELECT first_name
FROM users
WHERE age > ANY (SELECT age FROM users);
```

Output:

first_name
John
Jane
Bob
Alice
Charlie

☐ AND

The AND operator returns TRUE if all of the conditions separated by AND are true. In the below example, we are filtering users that have an age of 20 and a location of London.

```
SELECT *
FROM users
WHERE age = 50 AND location = ‘London’;
```

Output:

first_name	last_name	age	location
Alice	Brown	50	London

☐ BETWEEN

The BETWEEN operator filters your query to only return results that fit a specified range.

```
SELECT *
FROM users
WHERE age BETWEEN 40 AND 50;
```

Output:

first_name	last_name	age	location
Jane	Smith	40	London
Bob	Johnson	45	Paris
Alice	Brown	50	London

☐ EXISTS

The EXISTS operator is used to filter data by looking for the presence of any record in a subquery.

```
SELECT *
FROM users
WHERE EXISTS (SELECT 1 FROM users WHERE location = 'London');
```

Output:

first_name	last_name	age	location
Jane	Smith	40	London
Alice	Brown	50	London

☐ IN

The IN operator includes multiple values set into the WHERE clause.

```
SELECT *
FROM users
WHERE first_name IN (‘Bob’, ‘Fred’, ‘Harry’);
```

Output:

first_name	last_name	age	location
Bob	Johnson	45	Paris

☐ NOT

The NOT operator returns results if the condition or conditions are not true.

```
SELECT *
FROM users
WHERE first_name NOT IN ('Bob', 'Fred', 'Harry');
```

Output:

first_name	last_name	age	location
John	Doe	35	New york
Jane	Smith	40	London
Alice	Brown	50	London
Charlie	Wilson	30	Tokyo

☐ OR

The OR operator returns TRUE if any of the conditions separated by OR are true. In the below example, we are filtering users that have an age of 30 or a location of London.

```
SELECT *
FROM users
WHERE age = 30 OR location = 'London';
```

Output:

first_name	last_name	age	location
Jane	Smith	40	London
Alice	Brown	50	London
Charlie	Wilson	30	Tokyo

☐ IS NULL

The IS NULL operator is used to filtering results with a value of NULL.

```
SELECT *
FROM users
WHERE age IS NULL;
```

Output:

Empty result, as no record has age column NULL;

String operators

- [ ] **LIKE**

  The LIKE operator searches for a specified pattern in a column. (For more information on how/why the % is used here, see the section on the wildcard character operator).

  SELECT *
  FROM users
  WHERE first_name LIKE '%Bob%';

  Output:

first_name	last_name	age	location
Bob	Johnson	45	Paris

- [ ] **NOT LIKE operator**: The NOT LIKE operator is used to searching for a specified pattern in a string and return the opposite of the LIKE operator.

  SELECT *
  FROM users
  WHERE first_name NOT LIKE 'J%';

  Output:

first_name	last_name	age	location
John	Doe	35	New york
Jane	Smith	40	London

- [ ] **Concatenation operator (||)**: The concatenation operator is used to combine two or more strings into a single string.

  Example:

  SELECT first_name || ' ' || last_name AS full_name
  FROM users;

Output:

full_name
John Doe
Jane Smith