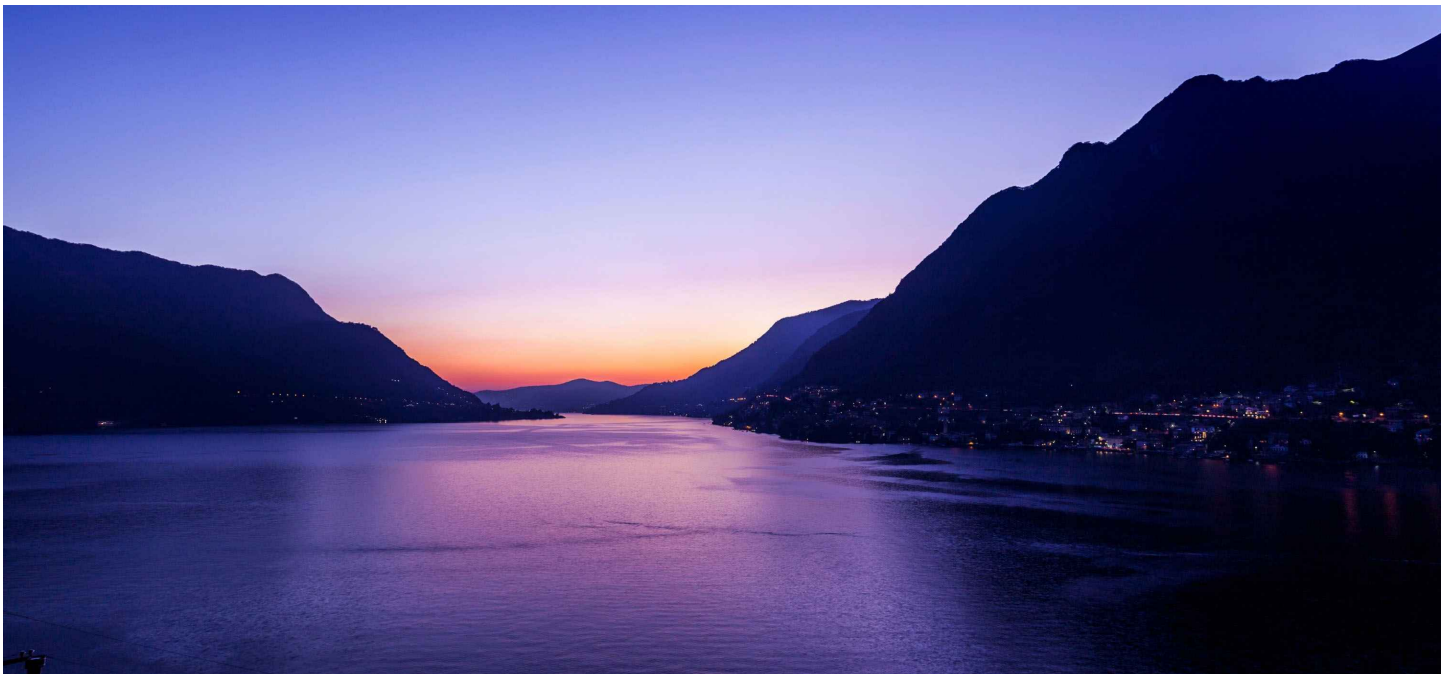

4-SQL Operators

Author: Vincent Lau

Learning Objectives

- Understand the meaning of various SQL operators.
- Understand conditional functions in SQL to achieve results based on criteria.

Introduction

SQL operators are used to perform various operations on data within SQL queries. These operators are categorized into different types, including arithmetic, comparison, compound, and logical operators. Here's an introduction to each type:

Operators

Arithmetic Operators

Arithmetic operators are used to perform mathematical operations on numeric data types.

Common arithmetic operators include:

- `+` (Addition): Adds two values.
- `-` (Subtraction): Subtracts the right operand from the left operand.
- `*` (Multiplication): Multiplies two values.
- `/` (Division): Divides the left operand by the right operand.
- `%` (Modulus): Returns the remainder of the division of the left operand by the right operand.

Example (MySQL and PostgreSQL):

```sql
SELECT 5 + 3 AS addition_result,
       10 - 4 AS subtraction_result,
       6 * 2 AS multiplication_result,
       20 / 4 AS division_result,
       17 % 5 AS modulus_result;
```

Comparison Operators

Comparison operators are used to compare values in SQL queries. They return a Boolean value (true or false) based on the comparison result.

Common comparison operators include:

- `=` (Equal to): Checks if two values are equal.
- `!=` or `<>` (Not equal to): Checks if two values are not equal.
- `<` (Less than): Checks if the left operand is less than the right operand.
- `>` (Greater than): Checks if the left operand is greater than the right operand.
- `<=` (Less than or equal to): Checks if the left operand is less than or equal to the right operand.
- `>=` (Greater than or equal to): Checks if the left operand is greater than or equal to the right operand.

Example (MySQL and PostgreSQL):

```sql
SELECT 5 = 5 AS equal,
       5 <> 3 AS not_equal, -- !=
       5 < 10 AS less_than,
       7 > 4 AS greater_than,
       6 <= 6 AS less_than_or_equal,
       9 >= 9 AS greater_than_or_equal;
```

Compound Operators (Not Valid in DBMS)

Compound operators combine an arithmetic operation with an assignment. They allow you to update the value of a column with the result of an arithmetic operation. Common compound operators include `+=`, `-=`, `*=`, `/=`, and `%=`.

In most DBMS, including popular ones like MySQL, PostgreSQL, Oracle, SQL Server, etc., **you don't use `+=` or similar operators for updating or modifying values in database tables directly within SQL queries**. Instead, you use SQL `UPDATE` statements with the `SET` clause to specify how values should be updated.

DBMS - MySQL, PostgreSQL, Oracle & SQL Server:

```
1 UPDATE your_table
2 SET your_column = your_column + 1
3 WHERE some_condition;
```

Logical Operators

Logical operators are used to combine conditions in SQL queries to form more complex conditions.

1. `ALL` Operator

The `ALL` operator is used to compare a value to all values in a result set or a subquery. It returns true if the comparison is true for all values; otherwise, it returns false.

Example (MySQL and PostgreSQL):

```
1 SELECT * FROM products
2 WHERE price > ALL (SELECT price FROM competitors);
```

This query selects products with prices greater than all prices in the subquery (prices from competitors).

2. `AND` Operator

The `AND` operator combines two or more conditions and returns true only if all conditions are true.

Example (MySQL and PostgreSQL):

```sql
SELECT * FROM employees
WHERE age > 30
AND department = 'Sales';
```

This query selects employees who are older than 30 years and work in the Sales department.

3. `OR` Operator

The `OR` operator combines two or more conditions and returns true if at least one condition is true.

Example (MySQL and PostgreSQL):

```sql
SELECT * FROM employees
WHERE department = 'HR' OR department = 'Finance';
```

This query selects employees in either the HR or Finance department.

4. `ANY` Operator

The `ANY` operator is used to compare a value to any value in a result set or a subquery. It returns true if the comparison is true for at least one value.

Example (MySQL and PostgreSQL):

```sql
SELECT * FROM products
WHERE price < ANY (SELECT price FROM competitors);
```

This query selects products with prices lower than at least one price in the subquery (prices from competitors).

5. `BETWEEN` Operator

The `BETWEEN` operator is used to specify a range for a value. It returns true if the value falls within the specified range (inclusive).

Example (MySQL and PostgreSQL):

```sql
SELECT * FROM orders
WHERE order_date BETWEEN '2023-01-01' AND '2023-12-31';
```

This query selects orders with order dates between January 1, 2023, and December 31, 2023.

6. `EXISTS` Operator

The `EXISTS` operator is used to check if a subquery returns any rows. It returns true if the subquery returns at least one row; otherwise, it returns false.

Example (MySQL and PostgreSQL):

```
SELECT *
FROM customers
WHERE EXISTS (SELECT 1 FROM orders WHERE orders.customer_id = customers.id);
```

This query selects customers who have placed at least one order.

7. `IN` Operator

The `IN` operator is used to specify a list of values to compare against. It returns true if the value matches any value in the list.

Example (MySQL and PostgreSQL):

```
SELECT * FROM products
WHERE category_id IN (1, 3, 5);
```

This query selects products in categories 1, 3, or 5.

8. `LIKE` Operator

The `LIKE` operator is used to perform pattern matching on text data. It returns true if a string matches a specified pattern.

Example (MySQL and PostgreSQL):

```
SELECT * FROM employees
WHERE last_name LIKE 'Lau%';
```

This query selects employees with last names starting with "Lau"

9. `NOT` Operator

The `NOT` operator negates a condition. It returns true if the condition is false and vice versa.

Example (MySQL and PostgreSQL):

```sql
SELECT * FROM products
WHERE NOT(price > 100);
```

This query selects products with price <= 100. Be aware of null case if any.

10. `SOME` Operator

The `SOME` operator is similar to the `ANY` operator. It is used to compare a value to any value in a result set or a subquery and returns true if the comparison is true for at least one value.

Example (MySQL and PostgreSQL):

```sql
SELECT * FROM products
WHERE price < SOME (SELECT price FROM competitors);
```

This query selects products with prices lower than at least one price in the subquery (prices from competitors).

Manipulation

In database management systems (DBMS), the concepts of NULL and the CASE function are fundamental for handling and manipulating data.

CASE

The **CASE** function in DBMS is used to perform conditional logic within SQL queries. It allows you to define conditions and specify the value to return based on those conditions.

The basic syntax of the CASE function looks like this:

```sql
CASE
    WHEN condition1 THEN result1
    WHEN condition2 THEN result2
    ...
    ELSE default_result
END
```

- `condition1`, `condition2`, etc., are the conditions to evaluate.

- `result1`, `result2`, etc., are the values to return if the corresponding condition is true.
- `default_result` is the value to return if none of the conditions are true (optional).

Example (MySQL and PostgreSQL):

```sql
SELECT first_name,
       CASE
           WHEN salary > 50000 THEN 'High'
           WHEN salary > 30000 THEN 'Medium'
           ELSE 'Low'
       END AS salary_category
FROM employees;
```

In this example, the CASE function is used to categorize employees into salary categories based on their salary values.

The CASE function is particularly useful when you need to perform conditional operations in your SQL queries, allowing you to make decisions and transform data based on specified conditions. It's a powerful tool for data manipulation and reporting in DBMS.