**SQL Operators / Keywords**

SQL (Structured Query Language) supports various types of operators, which can be used to perform various operations on data stored in a database. Here are some commonly used SQL operators:

1. **Arithmetic Operators**: +, -, \*, /, % (modulo)
2. **Comparison Operators**: =, <>, >, <, >=, <=
3. **Logical Operators**: AND, OR, NOT
4. **LIKE operator**: used to match a string pattern, e.g., 'LIKE '%John%'' matches all strings containing 'John'.
5. **IN operator**: used to match a value against a set of values, e.g., 'IN (1, 2, 3)' matches any of the values 1, 2, or 3.
6. **BETWEEN operator**: used to match a value against a range of values, e.g., 'BETWEEN 10 AND 20' matches any value that is greater than or equal to 10 and less than or equal to 20.
7. **LIMIT operator:** is Use to Limit Records
8. **IS NULL operator**: used to test for NULL values, e.g., 'IS NULL' matches only NULL values.
9. **Exist , Not Exist** : Very Importent

**Arithmetic Operators**: +, -, \*, /, % (modulo)

Yes, that's correct! The arithmetic operators in SQL include addition (+), subtraction (-), multiplication (*), division (/), and modulo division (%). These operators can be used in SQL expressions to perform mathematical operations on values stored in the database. For example, you can use the "+" operator to add two numbers, the "*" operator to multiply two numbers, and so on. The modulo operator (%) returns the remainder of the division of one number by another.

Here's an example of how you could use these operators in a SQL query:

vbnetCopy code

SELECT ( column1 + column2 ) AS result1, ( column3 \* column4 ) AS result2, ( column5 % column6 ) AS result3 FROM table\_name;

**Comparison Operators:** =, <>, >, <, >=, <=

The comparison operators in SQL allow you to compare values stored in a database. The comparison operators include equality (=), inequality (<> or !=), greater than (>), less than (<), greater than or equal to (>=), and less than or equal to (<=).

Here's an example of how you could use these operators in a SQL query:

SELECT column1, column2 FROM table\_name WHERE column1 >= 100 AND column2 < 50;

In this example, the SQL query selects two columns from the "table\_name" table and filters the resulting data based on the conditions specified in the "WHERE" clause. The "WHERE" clause specifies that the values in "column1" must be greater than or equal to 100, and the values in "column2" must be less than 50. The "AND" operator combines the two conditions, ensuring that both conditions must be true in order for a row to be included in the result set.

**Logical Operators:** AND, OR, NOT

Yes, that's correct! The logical operators in SQL allow you to combine conditions in a "WHERE" clause to create more complex expressions. The logical operators include AND, OR, and NOT.

* The AND operator requires that both conditions be true in order for a row to be included in the result set.
* The OR operator requires that at least one condition be true in order for a row to be included in the result set.
* The NOT operator negates the result of a condition, allowing you to exclude rows from the result set that would otherwise be included.

Here's an example of how you could use these operators in a SQL query:

sqlCopy code

SELECT column1, column2 FROM table\_name WHERE (column1 >= 100 AND column2 < 50) OR column3 = 'value';

In this example, the SQL query selects two columns from the "table\_name" table and filters the resulting data based on the conditions specified in the "WHERE" clause. The "WHERE" clause specifies that either the values in "column1" must be greater than or equal to 100 and the values in "column2" must be less than 50, or the values in "column3" must be equal to the string 'value'. The "OR" operator combines the two conditions, ensuring that at least one of the conditions must be true in order for a row to be included in the result set.

**LIKE operator:** used to match a string pattern, e.g., 'LIKE '%John%'' matches all strings containing 'John'

Yes, that's correct! The LIKE operator in SQL is used to match a string pattern against values stored in a database. The syntax for the LIKE operator is as follows:

sqlCopy code

column\_name LIKE ‘pattern’

The "pattern" argument can include wildcard characters, such as the percent sign (%), which matches any number of characters, and the underscore (\_), which matches exactly one character.

For example, the following SQL query uses the LIKE operator to match all strings containing the word "John":

sqlCopy code

SELECT column1, column2 FROM table\_name WHERE column1 LIKE '%John%';

SELECT \* FROM customer where name like '%\_\_a%'

**IN operator:** used to match a value against a set of values, e.g., 'IN (1, 2, 3)' matches any of the values 1, 2, or 3.

Yes, that's correct! The IN operator in SQL is used to match a value against a set of values. The syntax for the IN operator is as follows:

column\_name IN (value1, value2, ..., value\_n)

The "value1", "value2", ..., "value\_n" argument is a list of values that you want to match against the values stored in "column\_name".

For example, the following SQL query uses the IN operator to match against the values 1, 2, and 3:

sqlCopy code

SELECT column1, column2 FROM table\_name WHERE column1 IN (1, 2, 3);

In this example, the SQL query selects two columns from the "table\_name" table and filters the resulting data based on the condition specified in the "WHERE" clause. The "WHERE" clause specifies that the values in "column1" must be equal to one of the values in the set (1, 2, 3). Any rows in which the value in "column1" is not equal to one of these values will be excluded from the result set.

**BETWEEN operator:** used to match a value against a range of values, e.g., 'BETWEEN 10 AND 20' matches any value that is greater than or equal to 10 and less than or equal to 20.

The BETWEEN operator in SQL is used to match a value against a range of values. The syntax for the BETWEEN operator is as follows:

Its work on 2 Operands

column\_name BETWEEN value1 AND value2

Example, the following SQL query uses the BETWEEN operator to match against the range of values from 10 to 20:

SELECT column1, column2 FROM table\_name WHERE column1 BETWEEN 10 AND 20;

**(MySQL) LIMIT operator:** It is often used at the end of a SELECT statement to limit the number of rows returned, particularly when working with large datasets.

Example

1. To get 1st 5 Rows
2. To get 5 rows after skipping 1st 5 rows

**(MsSQL) Top Keyword:** Its Used to Select Top Rows.

1. SELECT TOP 5 \* FROM employees; (select 5 rows top)
2. SELECT TOP 50 PERCENT \* FROM customer; (50% record from 100%)

**IS NULL operator:** used to test for NULL values, e.g., 'IS NULL' matches only NULL values.

IS NULL operator in SQL is used to get for NULL values.

column\_name IS NULL

**Example**,

select \* from customer where genderId is null

insert into customer values (1, 'sajidfgd',null,22)

insert into customer values (2, 'maleer',1,24234)

insert into customer values (3, 'rashid',2,24234)

insert into customer values (4, 'dsdsdff',1,24234)

insert into customer values (5, 'ajsdfdsfid',null,24234)

**Exist , Not Exist:** (Exist check Record Exist) , (Not Exist check Records not Exist)

-- Create the customers table

CREATE TABLE customers (

id INT PRIMARY KEY,

name VARCHAR(50),

email VARCHAR(50)

);

-- Insert 10 records into the customers table

INSERT INTO customers (id, name, email)

VALUES (1, 'Alice', 'alice@example.com'),

(2, 'Bob', 'bob@example.com'),

(3, 'Charlie', 'charlie@example.com'),

(4, 'David', 'david@example.com'),

(5, 'Emily', 'emily@example.com'),

(6, 'Frank', 'frank@example.com'),

(7, 'Grace', 'grace@example.com'),

(8, 'Henry', 'henry@example.com'),

(9, 'Isabel', 'isabel@example.com'),

(10, 'John', 'john@example.com');

-- Create the orders table

CREATE TABLE orders (

id INT PRIMARY KEY,

customer\_id INT REFERENCES customers(id),

order\_date DATE,

total\_amount DECIMAL(10,2)

);

-- Insert 10 records into the orders table

INSERT INTO orders (id, customer\_id, order\_date, total\_amount)

VALUES (1, 1, '2022-01-01', 100.00),

(2, 2, '2022-01-02', 200.00),

(3, 3, '2022-01-03', 300.00),

(4, 4, '2022-01-04', 400.00),

(5, 5, '2022-01-05', 500.00),

(6, 6, '2022-01-06', 600.00),

(7, 7, '2022-01-07', 700.00),

(8, 8, '2022-01-08', 800.00),

(9, 9, '2022-01-09', 900.00),

(10, 10, '2022-01-10', 1000.00);

1. **Exist :** Give or Select Those Records on The Base of Condition , Any Particular Values

**Example 1 : -** (Exist)

if exists(select name from customers where id = 5)

begin

print 'pakistna'

end

else

begin

print 'Error '

end

select \* from customers

**Example 2 : -** (Exist)

Find all customers who have placed at least one order, we can use the EXISTS keyword as follows:

SELECT \*

FROM customers

WHERE EXISTS (

SELECT \*

FROM orders

WHERE orders.customer\_id = customers.id

);

1. **Not Exist :** *Return Those Record in Which Condition value Not Exists.*

**Example 1 : -** (not Exist)

if not exists(select name from customers where id = 5)

begin

print 'pakistna'

end

else

begin

print 'Error '

end

select \* from customers

**Example 2 : -**

To find all customers who have not placed any orders, we can use the NOT EXISTS keyword as follows:

SELECT \* FROM customers

WHERE NOT EXISTS (

SELECT \*

FROM orders

WHERE orders.customer\_id = customers.id

);