

debabratapalit98@gmail.com



DATA ANALYSIS

WITH

SQL

SQL OPERATORS



[in](#) /debabrata-palit03

INTRODUCTION

SQL operators are important in database management systems (DBMS) as they allow us to manipulate and retrieve data efficiently. Operators in SQL perform arithmetic, logical, comparison, bitwise, and other operations to work with database values. Understanding SQL operators is crucial for performing complex data manipulations, calculations, and filtering operations in queries.



TYPES OF SQL OPERATORS

SQL operators can be categorized based on the type of operation they perform. Here are the primary types of SQL operators:

- **Arithmetic Operators**
- **Comparison Operators**
- **Logical Operators**
- **Bitwise Operators**
- **Compound Operators**
- **Special Operators**



ARITHMETIC OPERATORS

Used to perform mathematical operations on numeric data types

Operator	Description	Example	Output
+	Addition	<code>SELECT 3 + 5;</code>	8
-	Subtraction	<code>SELECT 3 - 5;</code>	-2
*	Multiplication	<code>SELECT 3 * 5;</code>	15
/	Division	<code>SELECT 5 / 4;</code>	1.25
DIV	Integer Division	<code>SELECT 14 DIV 4;</code>	3
% or MOD	Modulo (Remainder of a Division)	<code>SELECT 15 % 4;</code> <code>SELECT 14 MOD 4;</code>	2 2

Example:

```
-- calculate the total revenue from sales
SELECT SUM(quantity * price) AS total_revenue
FROM sales;
```



COMPARISON OPERATORS

Used to compare one expression's value to other expressions.

Operator	Description
=	The SQL Equal Operator checks if the values of two operands are equal.
!= or <>	The SQL Not Equal Operator checks if the values of two operands are not equal.
>=	The SQL Greater Than Equals Operator checks if the left operand is \geq right one.
<	The SQL Less Than Operator checks if the left operand is $<$ right one.
>	The SQL Greater Than Operator checks if the left operand is $>$ right one.
<=	The SQL Less Than Equals Operator checks if the left operand is \leq right one.
IS	Tests a value against a Boolean ('TRUE', 'FALSE', or 'NULL').

Example:

```
-- find the results where marks are 60 or more
SELECT * FROM results WHERE marks>=60;

-- example of IS operator
SELECT * FROM employees WHERE is_active IS TRUE;
SELECT * FROM employees WHERE email IS NULL;
```



LOGICAL OPERATORS

Used to combine or manipulate conditions in SQL queries to retrieve or manipulate data based on specified criteria.

Operator	Description
AND	Logical AND compares two Booleans as expressions and returns true when both expressions are true.
OR	Logical OR compares two Booleans as expressions and returns true when one of the expressions is true.
NOT	NOT takes a single Boolean as an argument and changes its value from false to true or from true to false.

Example:

```
-- find the employees who belong to Pune and earn 25k per month  
SELECT * FROM employees WHERE city = 'Pune' AND salary >= 25000;
```

```
-- find the employees who are either system analyst or from Mumbai  
SELECT * FROM employees WHERE role = 'System Analyst' OR city = 'Mumbai';
```

```
-- find the employees who are not from India  
SELECT * FROM employee WHERE NOT country = 'India';
```



BITWISE OPERATORS

Used to perform bitwise operations on binary values in SQL queries, manipulating individual bits to perform logical operations at the bit level.

Operator	Description	Example	Output
&	Bitwise AND operator	<code>SELECT 5 & 3;</code>	1
	Bitwise OR operator	<code>SELECT 5 3;</code>	7
^	Bitwise XOR (exclusive OR)	<code>SELECT 5 ^ 3;</code>	6
~	Bitwise NOT (complement)	<code>SELECT ~5;</code>	-6
<<	Left shift operator	<code>SELECT 5 << 1;</code>	10
>>	Right shift operator	<code>SELECT 5 >> 1;</code>	2

To understand this, let's take the example of '**SELECT 5 & 3**'

5 → 0101 (binary)

3 → 0011 (binary)

0101

& 0011

0001 → (Decimal: 1)

Hence, 5 & 3 = 1



COMPOUND OPERATORS

Combination of an operation with assignment.

These operators modify the value of a column and store the result in the same column in a single step.

Operator	Description
+=	Add and assign
-=	Subtract and assign
*=	Multiply and assign
/=	Divide and assign
%=	Modulo and assign
&=	Bitwise AND and assign
^=	Bitwise XOR and assign
=	Bitwise OR and assign

Note:

- MySQL does NOT support +=, -=, *= operators. You must use **SET x = x + value** instead.
- SQL Server supports these operators. Syntax: **x += value** e.g., **x+=5**
- PostgreSQL also does NOT support these operators.



SPECIAL OPERATORS

provides several special operators that serve specific functions such as filtering data based on a range, checking for existence, and comparing sets of values.

Operator	Description
ALL	Selects all records; compares value to every value in a subquery.
ANY/SOME	Returns true if at least one row in a subquery matches.
BETWEEN	Checks if a value is within a specified range (inclusive).
IN	Checks if a value exists in a given set of values.
EXISTS	Returns TRUE if a subquery returns at least one row.
LIKE	Matches a pattern using wildcards (`%` for multiple chars, `_` for single char).



Let's take an example table to understand the application of the ***SPECIAL OPERATORS***. Here we have two tables.

Employees

id	name	age	salary	department	hire_date
1	Alice	30	60000	HR	2020-05-15
2	Bob	25	55000	IT	2021-06-20
3	Charlie	35	75000	Finance	2019-08-10
4	David	28	48000	IT	2022-01-05
5	Eve	40	70000	HR	2018-07-12

Managers

id	name	department
1	Alice	HR
3	Charlie	Finance
5	Eve	HR



ALL()

-- Find employees whose salary is greater than the salary of all HR employees

```
SELECT * FROM Employees WHERE salary > ALL (SELECT salary FROM Employees WHERE department = 'HR');
```

-- Output:

id	name	age	salary	department	hire_date
3	Charlie	35	75000	Finance	2019-08-10

ANY() / SOME()

-- Find employees whose age is greater than the age of any Finance employees

```
SELECT * FROM Employees WHERE age > ANY (SELECT age FROM Employees WHERE department = 'Finance');
```

-- Output:

id	name	age	salary	department	hire_date
5	Eve	40	70000	HR	2018-07-12



BETWEEN()

```
-- Find employees who were hired between 2019 and 2021.
SELECT * FROM Employees
WHERE hire_date BETWEEN '2019-01-01' AND '2021-12-31';
```

```
-- Output:
```

id	name	age	salary	department	hire_date
1	Alice	30	60000	HR	2020-05-15
2	Bob	25	55000	IT	2021-06-20
3	Charlie	35	75000	Finance	2019-08-10

IN()

```
-- Find employees who work in IT or Finance departments
SELECT * FROM Employees
WHERE department IN ('IT', 'Finance');
```

```
-- Output:
```

id	name	age	salary	department	hire_date
2	Bob	25	55000	IT	2021-06-20
3	Charlie	35	75000	Finance	2019-08-10
4	David	28	48000	IT	2022-01-05



EXISTS()

```
-- Find employees who exist in the Managers table
SELECT * FROM Employees WHERE EXISTS (SELECT * FROM Managers WHERE
Employees.id = Managers.id);
```

```
-- Output:
```

id	name	age	salary	department	hire_date
1	Alice	30	60000	HR	2020-05-15
3	Charlie	35	75000	Finance	2019-08-10
5	Eve	40	70000	HR	2018-07-12

LIKE()

```
-- Find employees whose name starts with 'A'
SELECT * FROM Employees WHERE name LIKE 'A%';
```

```
-- Output:
```

id	name	age	salary	department	hire_date
1	Alice	30	60000	HR	2020-05-15



WILDCARDS

SQL wildcard characters are special characters used in SQL queries to search for patterns within string data. Wildcard characters are used with the LIKE operator.

% (Ampersand)	Represents zero or more characters
_ (Underscore)	Represents a single character

Some Examples:

```
-----  
a%      => Finds any value starts with 'a'  
%a      => Finds any value ends with 'a'  
%a%    => Finds any value that have 'a' in any position  
_a%    => Finds any value that have 'a' in the 2nd position  
__a%   => Finds any value that have 'a' in the 3rd position  
%a_    => Finds any value that have 'a' in the 2nd last position  
a_%    => Finds any value starts with 'a' and has at least two  
         characters in length  
a%b    => Finds any value starts with 'a' and ends with 'b'  
a_b    => Finds any value that has three characters, starts with 'a'  
         and ends with 'b'
```



debabratapalit98@gmail.com



THANK YOU!
SHARE YOUR
INSIGHTS 
AND STAY
CONNECTED!



 /debabrata-palit03