SQL operators are essential tools for working with databases. They help perform calculations, filter data, combine datasets, and manipulate text efficiently.This guide provides an overview of SQL operators, their types, and practical examples to help you understand and use them effectively. Whether you're new to SQL or looking to improve your skills, this resource has something for everyone.

Comprehensive Guide to SQL Operators

Introduction to SQL Operators

SQL operators are symbols or keywords used to perform operations on database data. They help retrieve, manipulate, and compare data effectively, forming the backbone of SQL queries. Operators are used in conditions, expressions, or calculations to extract meaningful information from the database.

Categories of SQL Operators

- Arithmetic Operators
- Comparison Operators
- Logical Operators
- Bitwise Operators
- Set Operators
- String Operators

Below is a comprehensive list with real-world examples and detailed explanations for each category.

Arithmetic Operators

Arithmetic operators are used to perform mathematical calculations like addition, subtraction, multiplication, division, etc. They operate on numerical data types and help to perform basic math operations in programming.

Operator	Description	Example	Explanation
+	Addition	SELECT price + tax AS total_cost FROM products; **Explanation**: If the price is 100 and tax is 20, the result is 120.	Adds the price and tax columns to calculate the total cost of a product.
-	Subtraction	SELECT salary - deductions AS net_salary FROM employees; **Explanation**: If the salary is 50000 and deductions are 5000, the result is 45000.	Subtracts deductions from salary to calculate net pay for employees.
*	Multiplication	SELECT quantity * unit_price AS total_amount FROM orders; **Explanation**: For 10 items at $15 each, the result is $150.	Multiplies quantity and unit_price to calculate total sales amount.
/	Division	SELECT total_sales / number_of_days AS avg_sales_per_day FROM sales; **Explanation**: If total_sales is 50000 over 10 days, the result is 5000.	Divides total_sales by number_of_days to calculate daily average sales.
%	Modulus (Remainder)	SELECT employee_id % 2 AS is_even FROM employees; **Explanation**: If employee_id is 5, the result is 1 (odd). For 4, it's 0 (even).	Checks if employee_id is even or odd.

Comparison Operators

Comparison operators are used to compare two values. They return a boolean result (True or False) based on whether the comparison holds true or not. These operators help in making decisions or controlling the flow of programs.

Operator	Description	Example	Explanation
=	Equal to	SELECT * FROM customers WHERE city = 'New York'; **Explanation**: Matches exact text or value. E.g., city = 'New York' finds rows where the city is exactly New York.	Retrieves all customers who live in New York.
!= or <>	Not equal to	SELECT * FROM products WHERE category != 'Electronics'; **Explanation**: Finds rows where the category is anything except "Electronics".	Retrieves all products that do not belong to the Electronics category.
>	Greater than	SELECT * FROM employees WHERE age > 40; **Explanation**: Filters rows where age is greater than 40.	Retrieves all employees older than 40 years.
<	Less than	SELECT * FROM employees WHERE experience < 5; **Explanation**: Filters rows where experience is less than 5.	Retrieves employees with less than 5 years of experience.
>=	Greater than or equal to	SELECT * FROM sales WHERE total_amount >= 1000; **Explanation**: Matches rows where total_amount is greater than or equal to 1000.	Retrieves all sales transactions where the total is at least $1000.
<=	Less than or equal to	SELECT * FROM sales WHERE discount <= 20; **Explanation**: Matches rows where discount is less than or equal to 20.	Retrieves all transactions with discounts up to 20%.
<=>	NULL-safe equal to (MySQL only)	SELECT * FROM orders WHERE shipping_date <=> NULL; **Explanation**: Unlike = NULL, this handles NULL safely without causing errors.	Retrieves orders where shipping_date is NULL.

Logical Operators

Logical operators are used to combine multiple conditions or expressions. They are often used in conditional statements to evaluate logical relationships. These operators help in decision-making where multiple conditions need to be checked simultaneously.

Operator	Description	Example	Explanation
AND	Combines conditions, all must be true	SELECT * FROM employees WHERE department = 'Sales' AND age > 30; **Explanation**: Both conditions (department = 'Sales' AND age > 30) must be true for a row to match.	Retrieves all employees in the Sales department who are older than 30.
OR	Combines conditions, at least one is true	SELECT * FROM products WHERE category = 'Books' OR category = 'Stationery'; **Explanation**: Matches rows where at least one condition (category = 'Books' or category = 'Stationery') is true.	Retrieves all products in either the Books or Stationery category.
NOT	Reverses the condition	SELECT * FROM customers WHERE NOT (city = 'London'); **Explanation**: Reverses the logic of the condition (city = 'London'), excluding rows where this is true.	Retrieves all customers who do not live in London.

Set Operators

Set operators are used to perform operations on sets, which are collections of unique elements. These operators help in combining, comparing, and manipulating sets (e.g., finding common elements, differences, or combining multiple sets).

Operator	Description	Example	Explanation
UNION	Combines results, removes duplicates	SELECT customer_id FROM orders_2022 **UNION** SELECT customer_id FROM orders_2023; **Explanation**: Ensures unique customer IDs across both tables in the result.	Combines customer IDs from 2022 and 2023 orders, removing duplicates.
UNION ALL	Combines results, includes duplicates	SELECT product_id FROM old_stock **UNION ALL** SELECT product_id FROM new_stock;	Combines product IDs from old and new stock, including duplicates.

		Explanation: Keeps all occurrences of product_id from both queries.	
INTERSECT	Returns common rows	SELECT employee_id FROM project_a INTERSECT SELECT employee_id FROM project_b; **Explanation**: Only rows present in both queries are returned.	Finds employees who worked on both projects A and B.
EXCEPT	Returns rows in first query only	SELECT product_id FROM catalog_a EXCEPT SELECT product_id FROM catalog_b; **Explanation**: Excludes rows from catalog B that are also present in catalog A.	Finds products in catalog A but not in catalog B.

Bitwise Operators

Bitwise operators work directly on binary representations of integers. They are used to perform bit-level operations. These are less commonly used in everyday SQL queries but are powerful for tasks like permission handling, bitmask comparisons, or low-level data manipulation.

Operator	Description	Example	Explanation
&	Bitwise AND	SELECT 5 & 3 AS result; **Explanation**: Useful in scenarios like checking specific flags in bitmask data.	Performs a bitwise AND operation on two numbers. Example: 5 (0101) & 3 (0011) results in 1 (0001) because only the last bit matches in both numbers.
|	|	Bitwise OR	**Explanation**: Can be used for combining permissions or features represented as bits.
^	Bitwise XOR	SELECT 5 ^ 3 AS result; **Explanation**: Useful for detecting bit changes or toggling specific bits.	Performs a bitwise XOR (exclusive OR) operation. Example: 5 (0101) ^ 3 (0011) results in 6 (0110) because bits differ in these positions.

String Operators

String operators are used to manipulate text data. These operators allow you to perform operations such as concatenating strings, repeating strings, and checking for the presence of substrings within a string. They are essential when working with textual information in programming.

Operator	Description	Example	Explanation
LIKE Pattern Matching	Searches for patterns in a string using wildcards (% for any characters, _ for a single character).	SELECT * FROM employees WHERE name LIKE 'A%';	Finds employees whose names start with "A".
ILIKE Case-Insensitive	Similar to LIKE, but performs a case-insensitive search (PostgreSQL only).	SELECT * FROM employees WHERE name ILIKE 'a%';	Finds employees whose names start with "a" or "A".
NOT LIKE Negated Pattern	Excludes rows matching the pattern.	SELECT * FROM employees WHERE name NOT LIKE 'A%';	Excludes employees whose names start with "A".

Note: There are many other string operators and functions available. For a detailed list, refer to SQL resources or documentation online.