Dt: 26.03.2024

# SQL Workshop Session – 6&7

***Topics Covered:***

1. SQL Operators
2. SQL Aggregate Functions

**SQL Operators:**

Every database administrator and user use SQL queries for manipulating and accessing the data of database tables and views.

The manipulation and retrieving of the data are performed with the help of reserved words and characters, which are used to perform arithmetic operations, logical operations, comparison operations, compound operations, etc.

**What is SQL Operator?**

The SQL reserved words and characters are called operators, which are used with a WHERE clause in a SQL query. In SQL, an operator can either be a unary or binary operator. The unary operator uses only one operand for performing the unary operation, whereas the binary operator uses two operands for performing the binary operation.

**Types of Operators:**

1. SQL Arithmetic Operators
2. SQL Comparison Operators
3. SQL Logical Operators
4. SQL Set Operators
5. SQL Bit-wise Operators
6. **SQL Arithmetic Operators:**

The Arithmetic Operators perform the mathematical operation on the numerical data of the SQL tables. These operators perform addition, subtraction, multiplication, and division operations on the numerical operands.

* **SQL Addition (+) Operator:** The Addition Operator in SQL performs the addition on the numerical data of the database table. In SQL, we can easily add the numerical values of two columns of the same table by specifying both the column names as the first and second operand. We can also add the numbers to the existing numbers of the specific column.

**Syntax:** SELECT operand1 + operand2;

**Ex:** Below is the Employee\_details table to run queries.

|  |  |  |  |
| --- | --- | --- | --- |
| **Emp ID** | **Emp Name** | **Emp Salary** | **Emp Monthly Bonus** |
| 101 | John | 25000 | 4000 |
| 102 | Jhansi | 30000 | 200 |

Suppose, we want to add 20,000 to the salary of each employee specified in the table. Then, we have to write the following query in the SQL:

SELECT Emp\_Salary + 2000 as new\_emp\_sal From Employee\_details;

Suppose, we want to add the Salary and Monthly bonus columnsof the above table, then the query will be:

SELECT Emp\_Salary + Emp\_Monthlybonus as Emp\_Total\_salary from Employee\_details;

* **SQL Subtraction (-) Operator:** The Subtraction Operator in SQL performs the subtraction on the numerical data of the database table. In SQL, we can easily subtract the numerical values of two columns of the same table by specifying both the column names as the first and second operand. We can also subtract the number from the existing number of the specific table column.

**Syntax:** SELECT operand1 - operand2;

**Ex:** Suppose, we want to subtract 5000 from the salary of each employee, then the query will be:

SELECT Emp\_Salary - 5000 as Emp\_New\_Salary FROM Employee\_details;

Suppose, we want to subtract bonus from the salary of each employee, then the query will be:

SELECT Emp\_Salary – Emp\_Monthlybonus as Emp\_Total\_Salary FROM Employee\_details;

* **SQL Multiplication (\*) Operator:** The Multiplication Operator in SQL performs the Multiplication on the numerical data of the database table. In SQL, we can easily multiply the numerical values of two columns of the same table by specifying both the column names as the first and second operand.

**Syntax:** SELECT operand1 \* operand2;

**Ex:** Suppose, we want to double the salary of each employee, then the query will be:

SELECT Emp\_Salary \* 2 as Emp\_New\_Salary FROM Employee\_details;

Suppose, we want to multiply the Emp\_Id column to Emp\_Salary column whose Emp\_Id is 101, then the query will be:

SELECT Emp\_Id \* Emp\_Salary as Emp\_Id \* Emp\_Salary FROM Employee\_details WHERE Emp\_Id = 102;

* **SQL Division (/) Operator:** The Division Operator in SQL divides the operand on the left side by the operand on the right side.

**Syntax:** SELECT operand1 / operand2;

**Ex:** Suppose, we want to half the salary of each employee, then the query will be:

SELECT Emp\_Salary / 2 as Emp\_New\_Salary FROM Employee\_details;

* **SQL Modulus (%) Operator:** The Modulus Operator in SQL provides the remainder when the operand on the left side is divided by the operand on the right side.

**Syntax:** SELECT operand1 % operand2;

**Ex:**

|  |  |  |
| --- | --- | --- |
| **Number** | **First Operand** | **Second Operand** |
| 1 | 56 | 4 |
| 2 | 32 | 8 |
| 3 | 89 | 9 |
| 4 | 18 | 10 |
| 5 | 10 | 5 |

Suppose, we want to get the reminder by dividing the number of First\_Operand column by the number of Second\_Operand column, then the query will be:

SELECT First\_operand % Second\_operand as Remainder FROM Employee\_details

1. **SQL Comparison Operators:**

The Comparison Operators in SQL compare two different data of SQL table and check whether they are the same, greater, and lesser. The SQL comparison operators are used with the WHERE clause in the SQL queries.

* **SQL Equal Operator (=):** This operator is highly used in SQL queries. The Equal Operator in SQL shows only data that matches the specified value in the query.

This operator returns TRUE records from the database table if the value of both operands specified in the query is matched.

**Ex:** Suppose we want to access all the records of those employees, whose salary is 30000, then the query will be:

SELECT \* FROM Employee\_details WHERE Emp\_Salary = 30000;

* **SQL Equal Not (! =) Operator:** The Equal Not Operator in SQL shows only those data that do not match the query's specified value.

This operator returns those records or rows from the database views and tables if the value of both operands specified in the query is not matched with each other.

**Ex:** Suppose we want to access all the records of those employees from the Employee\_details table whose salary is not 45000, then the query will be:

SELECT \* FROM Employee\_details WHERE Emp\_Salary != 45000;

* **SQL Greater Than (>) Operator:** The Greater Than Operator in SQL shows only those data which are greater than the value of the right-hand operand.

**Ex:** Suppose we want to access all the records of those employees from the Employee\_details table whose employee id is greater than 102, then the query will be:

SELECT \* FROM Employee\_details WHERE Emp\_Id > 102;

* **SQL Greater Than Equal To (> =) Operator:** The Greater Than Equals to Operator in SQL shows those data from the table which are greater than and equal to the value of the right-hand operand.

**Ex:** Suppose we want to access all the records of those employees from the Employee\_details table whose employee id is greater than and equals to 202, then the query will be:

SELECT \* FROM Employee\_details WHERE Emp\_Id >= 102;

* **SQL Less Than Operator (<):** The Less Than Operator in SQL shows only those data which are less than the value of the right-side operand.

**Ex:** Suppose we want to access all the records of those employees from the Employee\_details table whose employee id is less than 102, then the query will be:

SELECT \* FROM Employee\_details WHERE Emp\_Id < 102;

* **SQL Less Than Equal To Operator (<=):** The Less Than Equals to Operator in SQL shows those data from the table which are lesser and equal to the value of the right-side operand.

**Ex:** Suppose we want to access all the records of those employees from the Employee\_details table whose employee id is less and equals 102, then the query will be:

SELECT \* FROM Employee\_details WHERE Emp\_Id <= 102;

1. **SQL Logical Operators:**

The Logical Operators in SQL perform the Boolean operations, which give two results True and False. These operators provide True value if both operands match the logical condition.

* **SQL ALL Operator:** The ALL operator in SQL compares the specified value to all the values of a column from the sub-query in the SQL database.

This operator is always used with the statements, **SELECT, HAVING, WHERE**

**Syntax:** SELECT column1, column2 FROM table\_name WHERE column comparison\_operator ALL (SELECT column1 FROM table\_name WHERE column comparison\_operator value);

**Ex:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Emp ID** | **Emp Name** | **Emp Salary** | **Emp City** |
| 201 | Abhay | 25000 | Delhi |
| 202 | Ankit | 45000 | Mumbai |
| 203 | Anjali | 30000 | Pune |
| 204 | Anjana | 29000 | Hyderabad |
| **205** | Andrew | 40000 | Kerala |

Suppose we want to access the employee id and employee names of those employees from the table whose salaries are greater than the salary of employees who lives in Jaipur city, then the query will be:

SELECT Emp\_Id, Emp\_Name FROM Employee\_details WHERE Emp\_Salary > ALL (SELECT Emp\_Salary FROM Employee\_details WHERE Emp\_City = Delhi);

* **SQL AND Operator:** The AND operator in SQL would show the record from the database table if all the conditions separated by the AND operator evaluated to True. It is also known as the conjunctive operator and is used with the WHERE clause.

**Syntax:** SELECT column1, ...., columnN FROM table\_Name WHERE condition1 AND condition2 AND condition3 AND ....... AND conditionN;

**Ex:** Suppose we want to access all the records of those employees from the Employee\_details table whose salary is 25000 and the city is Delhi, then the query will be:

SELECT \* FROM Employee\_details WHERE Emp\_Salary = 25000 OR Emp\_City = 'Delhi';

* **SQL OR Operator:** The OR operator in SQL shows the record from the table if any of the conditions separated by the OR operator evaluates to True. It is also known as the conjunctive operator and is used with the WHERE clause.

**Syntax:** SELECT column1, ...., columnN FROM table\_Name WHERE condition1 OR condition2 OR condition3 OR ....... OR conditionN;

**Ex:** Suppose we want to access all the records of those employees from the Employee\_details table whose salary is 25000 or the city is Delhi, then the query will be:

SELECT \* FROM Employee\_details WHERE Emp\_Salary = 25000 OR Emp\_City = 'Delhi';

* **SQL BETWEEN Operator:** The BETWEEN operator in SQL shows the record within the range mentioned in the SQL query. This operator operates on the numbers, characters, and date/time operands.

**Syntax:** SELECT column\_Name1, column\_Name2 ...., column\_NameN FROM table\_Name WHERE column\_nameBETWEEN value1 and value2;

**Ex:** Suppose we want to access all the information of those employees from the Employee\_details table who is having salaries between 20000 and 40000, then the query will be:

SELECT \* FROM Employee\_details WHERE Emp\_Salary BETWEEN 30000 AND 45000;

* **SQL IN Operator:** The IN operator in SQL allows users to specify two or more values in a WHERE clause. This logical operator minimizes the requirement of multiple OR conditions.

**Syntax:** SELECT column\_Name1, column\_Name2 ...., column\_NameN FROM table\_Name WHERE column\_name IN (list\_of\_values);

**Ex:** Suppose we want to show all the information of those employees from the Employee\_details table whose Employee Id is 202, 204, and 205, then the query will be:

SELECT \* FROM Employee\_details WHERE Emp\_Id IN (202, 204, 205);

Suppose we want to show all the information of those employees from the Employee\_details table whose Employee Id is not equal to 202 and 205, then the query will be:

SELECT \* FROM Employee\_details WHERE Emp\_Id NOT IN (202,205);

* **SQL NOT Operator:** The NOT operator in SQL shows the record from the table if the condition evaluates to false. It is always used with the WHERE clause.

**Syntax:** SELECT column1, column2 ...., columnN FROM table\_Name WHERE NOT condition;

**Ex:** Suppose, we want to show all the information of those employees from the Employee\_details table whose Cityis not Delhi, then the query will be:

SELECT \* FROM Employee\_details WHERE NOT Emp\_City = 'Delhi';

* **SQL ANY Operator:** The ANY operator in SQL shows the records when any of the values returned by the sub-query meet the condition.

**Syntax:** SELECT column1, column2 ...., columnN FROM table\_Name WHERE column\_name comparison\_operator ANY (SELECT column\_name FROM table\_name WHERE condition(s));

* **SQL LIKE Operator:** The LIKE operator in SQL shows those records from the table which match with the given pattern specified in the sub-query.

The percentage (%) sign is a wildcard which is used in conjunction with this logical operator.

This operator is used in the WHERE clause with the statements **SELECT, UPDATE, DELETE.**

**Syntax:** SELECT column\_Name1, column\_Name2 ...., column\_NameN FROM table\_Name WHERE column\_name LIKE pattern;

**Ex:** Suppose, we want to show all the information of those employees from the Employee\_details whose name starts with ''s'', then the query will be:

SELECT \* FROM Employee\_details WHERE Emp\_Name LIKE 's%';

Suppose, we want to show all the information of those employees from the Employee\_detailswhose name ends with ''y'', then the query will be:

SELECT \* FROM Employee\_details WHERE Emp\_Name LIKE '%y';

Suppose, we want to show all the information of those employees from the Employee\_detailswhose name starts with ''S'' and ends with ''y'', then the query will be:

SELECT \* FROM Employee\_details WHERE Emp\_Name LIKE 'S%y';

1. **SQL Set Operators:**

The Set Operators in SQL combine a similar type of data from two or more SQL database tables. It mixes the result, which is extracted from two or more SQL queries, into a single result.

Set operators combine more than one selects statement in a single query and return a specific result set.

* **SQL Union Operator:** The SQL Union Operator combines the result of two or more SELECT statements and provides the single output.

The data type and the number of columns must be the same for each SELECT statement used with the UNION operator. This operator does not show the duplicate records in the output table.

**Syntax:**

SELECT column1, column2 ...., columnN FROM table\_Name1 [WHERE conditions] UNION SELECT column1, column2 ...., columnN FROM table\_Name2 [WHERE conditions];

**Employee\_details1:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Emp ID** | **Emp Name** | **Emp Salary** | **Emp City** |
| 201 | Sanjay | 25000 | Delhi |
| 202 | Ajay | 40000 | Delhi |
| 203 | Saket | 30000 | Alighar |

**Employee\_details2:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Emp ID** | **Emp Name** | **Emp Salary** | **Emp City** |
| 203 | Saket | 30000 | Alighar |
| 204 | Saurabh | 40000 | Delhi |
| 205 | Ram | 30000 | Kerala |
| 201 | Sanjay | 25000 | Delhi |

**Ex:** Suppose, we want to see the employee name and employee id of each employee from both tables in a single output, then the query will be:

SELECT Emp\_ID, Emp\_Name FROM Employee\_details1

UNION

SELECT Emp\_ID, Emp\_Name FROM Employee\_details2;

* **SQL Union All Operator:** The SQL Union Operator is the same as the UNION operator, but the only difference is that it also shows the same record.

**Syntax:**

SELECT column1, column2 ...., columnN FROM table\_Name1 [WHERE conditions] UNION ALL SELECT column1, column2 ...., columnN FROM table\_Name2 [WHERE conditions];

**Ex:** Suppose, we want to see the employee name of each employee of both tables in a single output, then the query will be:

SELECT Emp\_Name FROM Employee\_details1

UNION ALL

SELECT Emp\_Name FROM Employee\_details2;

* **SQL Intersect Operator:** The SQL Intersect Operator shows the common record from two or more SELECT statements. The data type and the number of columns must be the same for each SELECT statement used with the INTERSECT operator.

**Syntax:**

SELECT column1, column2 ...., columnN FROM table\_Name1 [WHERE conditions] INTERSECT SELECT column1, column2 ...., columnN FROM table\_Name2 [WHERE conditions];

**Ex:** Suppose, we want to see a common record of the employee from both the tables in a single output, then the query will be:

SELECT Emp\_Name FROM Employee\_details1

INTERSECT

SELECT Emp\_Name FROM Employee\_details2;

1. **SQL Bitwise Operators:**

The Bitwise Operators in SQL perform the bit operations on the Integer values.

* **Bitwise AND (&):** The Bitwise AND operator performs the logical AND operation on the given Integer values. This operator checks each bit of a value with the corresponding bit of another value.

**Syntax:** SELECT column1 & column2 & .... & columnN FROM table\_Name [WHERE conditions];

This example consists of the following table, which has two columns. Each column holds numerical values.

When we use the Bitwise AND operator in SQL, then SQL converts the values of both columns in binary format, and the AND operation is performed on the converted bits.

After that, SQL converts the resultant bits into user understandable format, i.e., decimal format.

|  |  |
| --- | --- |
| **Column 1** | **Column 2** |
| 1 | 1 |
| 2 | 5 |
| 3 | 4 |
| 4 | 2 |
| 5 | 3 |

**Ex:** Suppose, we want to perform the Bitwise AND operator between both the columns of the above table, then the query will be:

SELECT Column1 & Column2 From TABLE AND;

* **Bitwise OR (|):** The Bitwise OR operator performs the logical OR operation on the given Integer values. This operator checks each bit of a value with the corresponding bit of another value.

**Syntax:** SELECT column1 | column2 | .... | columnN FROM table\_Name [WHERE conditions] ;

**Ex:** Suppose, we want to perform the Bitwise OR operator between both the columns of the above table, then the query will be:

SELECT Column1 | Column2 From TABLE\_OR;

*Ref:* [*https://www.javatpoint.com/sql-operators*](https://www.javatpoint.com/sql-operators)

**SQL Aggregate Functions:**

SQL aggregation function is used to perform the calculations on multiple rows of a single column of a table. It returns a single value.

It is also used to summarize the data.

**Types of Aggregate Functions:**

1. **COUNT Function:** The COUNT () function returns the number of rows that matches a specified criterion.

**Syntax:** SELECT COUNT (column\_name) FROM table\_name WHERE condition;

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **ProductID** | **Company** | **CategoryID** | **Quantity** | **Price** |
| Item1 | Comp1 | 2 | 10 | 20 |
| Item2 | Comp2 | 3 | 25 | 50 |
| Item3 | Comp3 | 2 | 50 | 100 |
| Item4 | Comp4 | 59 | 100 | 200 |
| Item5 | Comp5 | 6 | 150 | 300 |
| Item6 | Comp6 | 7 | 250 | 500 |
| Item7 | Comp7 | 10 | 500 | 1000 |

**Ex:** SELECT COUNT (\*) FROM Products;

Specify Column name: SELECT COUNT (ProductName) FROM Products;

Add a WHERE clause: SELECT COUNT (ProductID) FROM Products WHERE Price > 20;

Ignore Duplicates: SELECT COUNT (DISTINCT Price) FROM Products;

Use an Alias: SELECT COUNT(\*) AS [Number of records] FROM Products;

Use count() with GROUP BY: SELECT COUNT(\*) AS [Number of records], CategoryID FROM Products GROUP BY CategoryID;

Use count() with HAVING: SELECT Company, COUNT(\*) AS [Number of records], CategoryID FROM Products GROUP BY Company HAVING COUNT(\*)>2;

1. **SUM Function:** Itis used to calculate the sum of all selected columns. It works on numeric fields only.

**Syntax:** SELECT SUM (column\_name) FROM table\_name WHERE condition;

**Ex:** SELECT SUM(Price) FROM Products;

SELECT SUM(Price) FROM Products WHERE quantity > 100;

SELECT SUM(Price) FROM Products WHERE quantity > 100 GROUP BY Company;

SELECT Company, SUM(Price) FROM Products GROUP BY Company HAVING SUM(Price) >=100;

1. **AVG Function:** The AVG function is used to calculate the average value of the numeric type.

**Syntax:** SELECT AVG (column\_name) FROM table\_name WHERE condition;

**Ex:** SELECT AVG(Price) FROM Products;

1. **MAX Function:** MAX function is used to find the maximum value of a certain column.

**Syntax:** SELECT MAX(column\_name) FROM table\_name WHERE condition;

**Ex:** SELECT MAX(Price) FROM Products;

1. **MIN Function:** MIN function is used to find the minimum value of a certain column.

**Syntax:** SELECT MIN(column\_name) FROM table\_name WHERE condition;

**Ex:** SELECT MIN(Price) FROM Products;

*Ref:* [*https://www.javatpoint.com/dbms-sql-aggregate-function*](https://www.javatpoint.com/dbms-sql-aggregate-function)