Tutorial: Group Functions, GROUP BY, HAVING, and CASE in MySQL Workbench

1. Introduction to Group Functions in MySQL Workbench

Group functions are aggregate functions that operate on a set of rows and return a single value. The most common group functions include MAX, MIN, SUM, AVG, COUNT, STDDEV, and VARIANCE. These are useful for summarizing and analyzing data.

Function	Description
MAX	Returns the largest value
MIN	Returns the smallest value
SUM	Returns the sum of values
AVG	Returns the average of values (ignoring NULL values)
COUNT	Returns the count of NOT NULL values
STDDEV	Returns the standard deviation (ignoring NULL values)
VARIANCE	Returns the variation (ignoring NULL values)

2. Group Functions with Examples

2.1 MAX (Maximum)

The **MAX** function returns the maximum value from a set of values.

Syntax:

```
SELECT MAX(column_name) FROM table_name;
```

Example: Find the highest salary from the employees table.

```
SELECT MAX(salary) AS highest_salary FROM employees;
```

2.2 MIN (Minimum)

The **MIN** function returns the minimum value from a set of values.

Syntax:

```
SELECT MIN(column_name) FROM table_name;
```

Example: Find the lowest salary in the employees table.

```
SELECT MIN(salary) AS lowest_salary FROM employees;
```

2.3 SUM (Sum)

The **SUM** function calculates the sum of all numeric values in a column.

Syntax:

```
SELECT SUM(column_name) FROM table_name;
```

Example: Find the total salaries of all employees.

```
SELECT SUM(salary) AS total_salary FROM employees;
```

2.4 AVG (Average)

The **AVG** function calculates the average value of a numeric column.

Syntax:

```
SELECT AVG(column_name) FROM table_name;
```

Example: Find the average salary in the employees table.

```
SELECT AVG(salary) AS average_salary FROM employees;
```

2.5 COUNT

The **COUNT** function returns the number of rows, or non-null values in a column.

Syntax:

```
SELECT COUNT(column_name) FROM table_name;
```

Example: Count the number of employees in the employees table.

```
SELECT COUNT(employee_id) AS employee_count FROM employees;
```

2.6 STDDEV (Standard Deviation)

The **STDDEV** function calculates the standard deviation of numeric values in a column, showing how much the values deviate from the average.

Syntax:

```
SELECT STDDEV(column_name) FROM table_name;
```

Example: Find the standard deviation of salaries in the employees table.

```
SELECT STDDEV(salary) AS salary_stddev FROM employees;
```

2.7 VARIANCE

The **VARIANCE** function calculates the variance of values in a column, showing how spread out the data is.

Syntax:

```
SELECT VARIANCE(column_name) FROM table_name;
```

Example: Find the variance in salaries in the employees table.

```
SELECT VARIANCE(salary) AS salary_variance FROM employees;
```

---

3. GROUP BY Clause

The **GROUP BY** clause is used to group rows that have the same values into summary rows. It is often used with aggregate functions like SUM, AVG, COUNT, etc.

Syntax:

```
SELECT column_name, AGGREGATE_FUNCTION(column_name)
FROM table_name
GROUP BY column_name;
```

Example: List the total salary by department.

```
SELECT department_id, SUM(salary) AS total_salary
FROM employees
GROUP BY department_id;
```

This query groups the data by department_id and calculates the total salary for each department.

---

4. HAVING Clause

The **HAVING** clause is used to filter records after the GROUP BY operation is applied. It works similarly to the WHERE clause but is specifically used with aggregate functions.

Syntax:

```
SELECT column_name, AGGREGATE_FUNCTION(column_name)
FROM table_name
GROUP BY column_name
HAVING condition;
```

Example: List departments where the total salary exceeds 50,000.

```
SELECT department_id, SUM(salary) AS total_salary
FROM employees
GROUP BY department_id
```

```
HAVING total_salary > 50000;
```

In this example, the HAVING clause filters out departments whose total salary is less than or equal to 50,000.

---

5. CASE Statement

The **CASE** statement is used to implement conditional logic in SQL. You can use it to create conditional columns or filtering criteria.

Syntax:

```
SELECT column_name,
    CASE
        WHEN condition THEN result
        ELSE default_result
    END AS new_column
FROM table_name;
```

Example: Classify employees' salaries into categories (Low, Medium, High).

```
SELECT employee_id, salary,
    CASE
        WHEN salary < 30000 THEN 'Low'
        WHEN salary BETWEEN 30000 AND 60000 THEN 'Medium'
        ELSE 'High'
    END AS salary_category
FROM employees;
```

In this example, the CASE statement categorizes salaries based on specified ranges.

---

6. Putting It All Together

Let's combine the **GROUP BY**, **HAVING**, and **CASE** with group functions to perform more advanced analysis.

Example: Get the total salary by department, classify the salary into categories, and filter the results to show only departments with total salaries above 50,000.

```
SELECT department_id, SUM(salary) AS total_salary,
    CASE
        WHEN SUM(salary) < 50000 THEN 'Low'
        WHEN SUM(salary) BETWEEN 50000 AND 100000 THEN 'Medium'
        ELSE 'High'
    END AS salary_category
FROM employees
GROUP BY department_id
HAVING total_salary > 50000;
```

- GROUP BY department_id: Groups the data by department.
- SUM(salary): Calculates the total salary for each department.
- CASE: Categorizes total salaries into Low, Medium, or High.
- HAVING total_salary > 50000: Filters out departments with total salaries below 50,000.