**Lab Exercises on Windows Functions in MySQL**

**Dataset Creation**

Before starting the exercises, create the sample dataset:

CREATE DATABASE IF NOT EXISTS lab\_exercises;

USE lab\_exercises;

CREATE TABLE employees (

emp\_id INT PRIMARY KEY,

emp\_name VARCHAR(50),

department VARCHAR(50),

salary DECIMAL(10,2),

join\_date DATE

);

INSERT INTO employees (emp\_id, emp\_name, department, salary, join\_date) VALUES

(1, 'Alice', 'HR', 5000, '2020-01-15'),

(2, 'Bob', 'IT', 7000, '2019-06-23'),

(3, 'Charlie', 'Finance', 6000, '2018-09-17'),

(4, 'David', 'IT', 7500, '2021-04-11'),

(5, 'Emma', 'HR', 5500, '2020-08-05'),

(6, 'Frank', 'Finance', 6500, '2017-12-01'),

(7, 'Grace', 'IT', 7200, '2022-03-20'),

(8, 'Hannah', 'HR', 4800, '2021-07-19'),

(9, 'Isaac', 'Finance', 6200, '2020-02-10'),

(10, 'Jack', 'IT', 7100, '2018-07-25');

**Exercise 1: Rank Employees by Salary**

**Problem:** Display all employees along with their rank based on salary in descending order.

SELECT emp\_id, emp\_name, department, salary,

RANK() OVER (ORDER BY salary DESC) AS salary\_rank

FROM employees;

**Exercise 2: Dense Rank of Employees by Salary**

**Problem:** Display employees with their salaries and dense ranking.

SELECT emp\_id, emp\_name, salary,

DENSE\_RANK() OVER (ORDER BY salary DESC) AS salary\_dense\_rank

FROM employees;

**Exercise 3: Row Number for Employees**

**Problem:** Assign a row number to each employee ordered by their joining date.

SELECT emp\_id, emp\_name, join\_date,

ROW\_NUMBER() OVER (ORDER BY join\_date) AS row\_num

FROM employees;

**Exercise 4: Running Total of Salaries**

**Problem:** Calculate a running total of salaries.

SELECT emp\_id, emp\_name, salary,

SUM(salary) OVER (ORDER BY emp\_id) AS running\_total

FROM employees;

**Exercise 5: Average Salary Per Department**

**Problem:** Compute the average salary per department using a window function.

SELECT emp\_id, emp\_name, department, salary,

AVG(salary) OVER (PARTITION BY department) AS avg\_salary

FROM employees;

**Exercise 6: First Salary in Each Department**

**Problem:** Retrieve the first salary recorded in each department.

SELECT emp\_id, emp\_name, department, salary,

FIRST\_VALUE(salary) OVER (PARTITION BY department ORDER BY join\_date) AS first\_salary

FROM employees;

**Exercise 7: Last Salary in Each Department**

**Problem:** Retrieve the last salary recorded in each department.

SELECT emp\_id, emp\_name, department, salary,

LAST\_VALUE(salary) OVER (PARTITION BY department ORDER BY join\_date ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) AS last\_salary

FROM employees;

**Exercise 8: Employees' Salary Difference from Department Average**

**Problem:** Find how much an employee’s salary deviates from the department’s average salary.

SELECT emp\_id, emp\_name, department, salary,

salary - AVG(salary) OVER (PARTITION BY department) AS salary\_deviation

FROM employees;

**Exercise 9: Percentage Contribution to Total Salary**

**Problem:** Calculate the percentage of total salary contributed by each employee.

SELECT emp\_id, emp\_name, salary,

(salary / SUM(salary) OVER ()) \* 100 AS salary\_percentage

FROM employees;

**Exercise 10: Find the Lead Employee**

**Problem:** Show the next employee’s salary in the ordered list.

SELECT emp\_id, emp\_name, salary,

LEAD(salary, 1) OVER (ORDER BY salary DESC) AS next\_salary

FROM employees;

**Exercise 11: Find the Lag Employee**

**Problem:** Show the previous employee’s salary in the ordered list.

SELECT emp\_id, emp\_name, salary,

LAG(salary, 1) OVER (ORDER BY salary DESC) AS previous\_salary

FROM employees;

**Exercise 12: Number of Employees Before Each One**

**Problem:** Count the number of employees who joined before each employee.

SELECT emp\_id, emp\_name, join\_date,

COUNT(\*) OVER (ORDER BY join\_date ROWS BETWEEN UNBOUNDED PRECEDING AND 1 PRECEDING) AS count\_before

FROM employees;

**Exercise 13: Cumulative Salary Per Department**

**Problem:** Calculate the running total salary per department.

SELECT emp\_id, emp\_name, department, salary,

SUM(salary) OVER (PARTITION BY department ORDER BY emp\_id) AS department\_running\_total

FROM employees;

**Exercise 14: Identify Employees with Highest Salary Per Department**

**Problem:** Find the employees with the highest salary in each department.

SELECT emp\_id, emp\_name, department, salary

FROM (

SELECT emp\_id, emp\_name, department, salary,

RANK() OVER (PARTITION BY department ORDER BY salary DESC) AS rnk

FROM employees

) ranked

WHERE rnk = 1;

**Exercise 15: Moving Average of Salaries**

**Problem:** Compute the moving average of salaries.

SELECT emp\_id, emp\_name, salary,

AVG(salary) OVER (ORDER BY emp\_id ROWS BETWEEN 2 PRECEDING AND CURRENT ROW) AS moving\_avg

FROM employees;

**Exercise 16: Employees Who Earn More Than the Previous One**

**Problem:** Identify employees whose salary is greater than the previous employee in the salary order.

SELECT emp\_id, emp\_name, salary

FROM (

SELECT emp\_id, emp\_name, salary,

LAG(salary) OVER (ORDER BY salary) AS prev\_salary

FROM employees

) subquery

WHERE salary > prev\_salary;

**Exercise 17: Top 3 Earners per Department**

**Problem:** List the top 3 highest-paid employees in each department.

SELECT emp\_id, emp\_name, department, salary

FROM (

SELECT emp\_id, emp\_name, department, salary,

DENSE\_RANK() OVER (PARTITION BY department ORDER BY salary DESC) AS rnk

FROM employees

) ranked

WHERE rnk <= 3;

**Exercise 18: Employees' Salary Growth Over Time**

**Problem:** Find the salary difference compared to the previous employee by join date.

SELECT emp\_id, emp\_name, salary, join\_date,

salary - LAG(salary) OVER (ORDER BY join\_date) AS salary\_growth

FROM employees;

**Exercise 19: Median Salary Per Department**

**Problem:** Find the median salary per department.

SELECT emp\_id, emp\_name, department, salary

FROM (

SELECT emp\_id, emp\_name, department, salary,

NTILE(2) OVER (PARTITION BY department ORDER BY salary) AS median\_group

FROM employees

) subquery

WHERE median\_group = 1;

**Exercise 20: Identify Salary Gaps**

**Problem:** Find employees with a significant salary gap compared to the next one.

SELECT emp\_id, emp\_name, salary,

LEAD(salary) OVER (ORDER BY salary DESC) AS next\_salary,

LEAD(salary) OVER (ORDER BY salary DESC) - salary AS salary\_gap

FROM employees;

**Explanation of Window Functions in MySQL**

Window functions are special types of SQL functions that **perform calculations across a set of table rows related to the current row**. They allow you to calculate **running totals, ranks, moving averages**, and more without grouping the result into a single output row.

Unlike aggregate functions (SUM(), AVG(), etc.), window functions **do not collapse rows**; instead, they maintain the original row count while computing results over partitions or the entire dataset.

**1. RANK() - Ranking Employees by Salary**

SELECT emp\_id, emp\_name, department, salary,

RANK() OVER (ORDER BY salary DESC) AS salary\_rank

FROM employees;

* **RANK()** assigns a ranking to each row based on salary (highest to lowest).
* **If two employees have the same salary, they get the same rank, but the next rank is skipped**.

**2. DENSE\_RANK() - Dense Ranking of Employees**

SELECT emp\_id, emp\_name, salary,

DENSE\_RANK() OVER (ORDER BY salary DESC) AS salary\_dense\_rank

FROM employees;

* **Similar to RANK() but does not skip ranks**.
* If two employees have the same salary, they get the same rank, but the next rank follows sequentially.

**3. ROW\_NUMBER() - Assigning Row Numbers**

SELECT emp\_id, emp\_name, join\_date,

ROW\_NUMBER() OVER (ORDER BY join\_date) AS row\_num

FROM employees;

* **Assigns a unique row number to each row**, even if values are repeated.

**4. SUM() - Running Total of Salaries**

SELECT emp\_id, emp\_name, salary,

SUM(salary) OVER (ORDER BY emp\_id) AS running\_total

FROM employees;

* **Calculates cumulative salary** by summing up salaries **row by row**.

**5. AVG() - Average Salary Per Department**

SELECT emp\_id, emp\_name, department, salary,

AVG(salary) OVER (PARTITION BY department) AS avg\_salary

FROM employees;

* **Partitions data by department and calculates average salary per department**.
* Each employee sees the department’s average salary.

**6. FIRST\_VALUE() - First Salary in Each Department**

SELECT emp\_id, emp\_name, department, salary,

FIRST\_VALUE(salary) OVER (PARTITION BY department ORDER BY join\_date) AS first\_salary

FROM employees;

* **Finds the earliest salary in each department based on join date**.

**7. LAST\_VALUE() - Last Salary in Each Department**

SELECT emp\_id, emp\_name, department, salary,

LAST\_VALUE(salary) OVER (PARTITION BY department ORDER BY join\_date ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) AS last\_salary

FROM employees;

* **Finds the last salary recorded in each department**.

**8. Salary Deviation from Department Average**

SELECT emp\_id, emp\_name, department, salary,

salary - AVG(salary) OVER (PARTITION BY department) AS salary\_deviation

FROM employees;

* **Calculates how much an employee’s salary deviates from the department's average salary**.

**9. Percentage Contribution of Salary**

SELECT emp\_id, emp\_name, salary,

(salary / SUM(salary) OVER ()) \* 100 AS salary\_percentage

FROM employees;

* **Finds the percentage of total salary that each employee earns**.

**10. LEAD() - Next Employee’s Salary**

SELECT emp\_id, emp\_name, salary,

LEAD(salary, 1) OVER (ORDER BY salary DESC) AS next\_salary

FROM employees;

* **Retrieves the next employee’s salary in descending order**.

**11. LAG() - Previous Employee’s Salary**

SELECT emp\_id, emp\_name, salary,

LAG(salary, 1) OVER (ORDER BY salary DESC) AS previous\_salary

FROM employees;

* **Retrieves the previous employee’s salary**.

**12. COUNT() - Number of Employees Before Each One**

SELECT emp\_id, emp\_name, join\_date,

COUNT(\*) OVER (ORDER BY join\_date ROWS BETWEEN UNBOUNDED PRECEDING AND 1 PRECEDING) AS count\_before

FROM employees;

* **Counts how many employees joined before the current employee**.

**13. SUM() - Running Total Salary Per Department**

SELECT emp\_id, emp\_name, department, salary,

SUM(salary) OVER (PARTITION BY department ORDER BY emp\_id) AS department\_running\_total

FROM employees;

* **Calculates running total of salaries per department**.

**14. Finding Employees with the Highest Salary in Each Department**

SELECT emp\_id, emp\_name, department, salary

FROM (

SELECT emp\_id, emp\_name, department, salary,

RANK() OVER (PARTITION BY department ORDER BY salary DESC) AS rnk

FROM employees

) ranked

WHERE rnk = 1;

* **Ranks employees in each department by salary and selects the top earners**.

**15. Moving Average of Salaries**

SELECT emp\_id, emp\_name, salary,

AVG(salary) OVER (ORDER BY emp\_id ROWS BETWEEN 2 PRECEDING AND CURRENT ROW) AS moving\_avg

FROM employees;

* **Computes moving average over the last 3 employees**.

**16. Finding Employees Who Earn More Than the Previous One**

SELECT emp\_id, emp\_name, salary

FROM (

SELECT emp\_id, emp\_name, salary,

LAG(salary) OVER (ORDER BY salary) AS prev\_salary

FROM employees

) subquery

WHERE salary > prev\_salary;

* **Identifies employees who earn more than the previous employee**.

**17. Top 3 Earners Per Department**

SELECT emp\_id, emp\_name, department, salary

FROM (

SELECT emp\_id, emp\_name, department, salary,

DENSE\_RANK() OVER (PARTITION BY department ORDER BY salary DESC) AS rnk

FROM employees

) ranked

WHERE rnk <= 3;

* **Finds the top 3 highest-paid employees in each department**.

**18. Salary Growth Over Time**

SELECT emp\_id, emp\_name, salary, join\_date,

salary - LAG(salary) OVER (ORDER BY join\_date) AS salary\_growth

FROM employees;

* **Calculates salary growth compared to the previous employee in joining order**.

**19. Median Salary Per Department**

SELECT emp\_id, emp\_name, department, salary

FROM (

SELECT emp\_id, emp\_name, department, salary,

NTILE(2) OVER (PARTITION BY department ORDER BY salary) AS median\_group

FROM employees

) subquery

WHERE median\_group = 1;

* **Finds the median salary by splitting employees into two equal groups**.

**20. Finding Salary Gaps**

SELECT emp\_id, emp\_name, salary,

LEAD(salary) OVER (ORDER BY salary DESC) AS next\_salary,

LEAD(salary) OVER (ORDER BY salary DESC) - salary AS salary\_gap

FROM employees;

* **Finds employees with large salary gaps between them and the next highest salary**.

**Conclusion**

These window functions allow **ranking, aggregation, and analytics** without losing row-level details. They are useful for **reporting, trend analysis, and performance insights** in MySQL.