Window Functions

Window functions in MySQL are powerful tools used to perform calculations across a set of rows related to the current row. Unlike aggregate functions, window functions do not generate the result set into a single row - they retain each individual row while performing calculations.

Syntax

```
<function_name>(<expression>) OVER ([PARTITION BY <columns>] [ORDER BY <columns>])
```

1. **`<function_name>`**:
    a. `ROW_NUMBER`
    b. `RANK`
    c. `SUM`
    d. `AVG`
2. **`OVER`**: Defines the "window" or the subset of rows the function should operate on.
3. **`PARTITION BY`** (optional): Divides the rows into groups (like a `GROUP BY` but without collapsing the data).
4. **`ORDER BY`** (optional): Orders the rows within each partition.

Database

```
-- MySQL Joins Practice Guide

-- Step 1: Creating the Database and Tables
DROP DATABASE IF EXISTS tutorial;
CREATE DATABASE IF NOT EXISTS tutorial;
USE tutorial;

-- Table: Employees
CREATE TABLE Employees (
    EmployeeID INT PRIMARY KEY,
    FirstName VARCHAR(50),
    LastName VARCHAR(50),
    Salary DECIMAL(10,2),
```

```
    DepartmentID INT
);

-- Table: Departments
CREATE TABLE Departments (
    DepartmentID INT PRIMARY KEY,
    DepartmentName VARCHAR(100)
);

-- Table: Projects
CREATE TABLE Projects (
    ProjectID INT PRIMARY KEY,
    ProjectName VARCHAR(100),
    DepartmentID INT
);

-- Table: EmployeeProjects
CREATE TABLE EmployeeProjects (
    EmployeeID INT,
    ProjectID INT,
    HoursWorked INT,
    PRIMARY KEY (EmployeeID, ProjectID)
);

-- Step 2: Inserting Data
-- Employees Table
INSERT INTO Employees VALUES
(1, 'Alice', 'Johnson', 55000, 1),
(2, 'Bob', 'Smith', 65000, 2),
(3, 'Charlie', 'Brown', 60000, NULL),
(4, 'David', 'Wilson', 70000, 1),
(5, 'Eve', 'Davis', 55000, 3);

-- Departments Table
INSERT INTO Departments VALUES
(1, 'Human Resources'),
(2, 'Engineering'),
(3, 'Marketing');

INSERT INTO Departments VALUES
(4, 'Security');

-- Projects Table
INSERT INTO Projects VALUES
```

```
(1, 'Project A', 1),
(2, 'Project B', 2),
(3, 'Project C', 3),
(4, 'Project D', NULL);

-- EmployeeProjects Table
INSERT INTO EmployeeProjects VALUES
(1, 1, 20),
(2, 2, 15),
(4, 1, 25),
(1, 3, 10),
(5, 3, 30);
```

Common Window Functions

1. ROW_NUMBER()

Assigns a unique sequential integer to rows within a partition for each department.

```
SELECT EmployeeID, DepartmentID,
       ROW_NUMBER() OVER (PARTITION BY DepartmentID ORDER BY
EmployeeID) AS RowNum
FROM Employees;
```

2. RANK()

Assigns a rank to rows within a partition based on the `ORDER BY` clause, with gaps for ties.

```
SELECT EmployeeID, DepartmentID, Salary,
       RANK() OVER (PARTITION BY DepartmentID ORDER BY Salary DESC) AS
`Rank`
FROM Employees;
```

3. DENSE_RANK()

Similar to `RANK()`, but without gaps for ties.

```
SELECT EmployeeID, DepartmentID, Salary,
       DENSE_RANK() OVER (PARTITION BY DepartmentID ORDER BY Salary
DESC) AS `DenseRank`
FROM Employees;
```

4. LEAD() and LAG()

Access values from subsequent (`LEAD`) or previous (`LAG`) rows.

```
-- Find the salary difference with the next employee in the same
department
SELECT EmployeeID, DepartmentID, Salary,
       LEAD(Salary) OVER (PARTITION BY DepartmentID ORDER BY Salary
DESC) AS NextSalary,
       Salary - LEAD(Salary) OVER (PARTITION BY DepartmentID ORDER BY
Salary DESC) AS SalaryDiff
FROM Employees;
```

6. Aggregate Functions as Window Functions

These include `SUM`, `AVG`, `MIN`, `MAX`, and `COUNT`, which can be used with the `OVER` clause.

```
-- Calculate cumulative salary within each department
SELECT EmployeeID, DepartmentID, Salary,
       SUM(Salary) OVER (PARTITION BY DepartmentID ORDER BY
EmployeeID) AS CumulativeSalary
FROM Employees;
```

Practical Examples

1. Cumulative Total

```
SELECT EmployeeID, DepartmentID, Salary,
       SUM(Salary) OVER (PARTITION BY DepartmentID ORDER BY Salary
DESC) AS RunningTotal
FROM Employees;
```

2. Percentage of Total

```
SELECT EmployeeID, DepartmentID, Salary,
       Salary * 100.0 / SUM(Salary) OVER (PARTITION BY DepartmentID)
AS Percentage
FROM Employees;
```

3. Ranking Employees by Salary

```
SELECT EmployeeID, DepartmentID, Salary,
       RANK() OVER (PARTITION BY DepartmentID ORDER BY Salary DESC) AS
`Rank`,
       DENSE_RANK() OVER (PARTITION BY DepartmentID ORDER BY Salary
DESC) AS `DenseRank`
FROM Employees;
```

4. Find Top N Records per Partition

```
SELECT EmployeeID, DepartmentID, Salary
FROM (
    SELECT EmployeeID, DepartmentID, Salary,
           ROW_NUMBER() OVER (PARTITION BY DepartmentID ORDER BY
Salary DESC) AS RowNum
    FROM Employees
) AS Ranked
WHERE RowNum <= 3; -- Top 3 employees per department
```

Tips for Using Window Functions

1. **Combination**: Use window functions with common table expressions (CTEs) or subqueries for complex analyses.
2. **Performance**: `PARTITION BY` can be expensive on large datasets. Optimize indexes on partition and order columns.
3. **Clarity**: Use aliases for calculated columns to make queries readable.