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

```sql
-- Create the database
DROP DATABASE IF EXISTS College;
CREATE DATABASE College;
USE College;

-- Table: Students
CREATE TABLE Students (
    StudentID INT PRIMARY KEY,
    StudentName VARCHAR(50),
    CollegeID INT,
    SubjectID INT,
    MarksObtained INT
);
```

```sql
-- Table: Colleges
CREATE TABLE Colleges (
   CollegeID INT PRIMARY KEY,
   CollegeName VARCHAR(100)
);

-- Table: Subjects
CREATE TABLE Subjects (
   SubjectID INT PRIMARY KEY,
   SubjectName VARCHAR(100)
);

-- Insert data into Colleges
INSERT INTO Colleges VALUES
(1, 'Indian Institute of Technology'),
(2, 'Delhi University'),
(3, 'St. Xavier\'s College');

-- Insert data into Subjects
INSERT INTO Subjects VALUES
(1, 'Mathematics'),
(2, 'Physics'),
(3, 'Chemistry'),
(4, 'History'),
(5, 'English');

-- Insert data into Students
INSERT INTO Students VALUES
(1, 'Aarav', 1, 1, 85),
(2, 'Ananya', 1, 2, 90),
(3, 'Ishaan', 2, 4, 70),
(4, 'Priya', 2, 4, 75),
(5, 'Rohan', 3, 3, 60),
(6, 'Sneha', 3, 3, 65),
(7, 'Vikram', 1, 1, 88),
(8, 'Kavya', 1, 2, 78),
(9, 'Aditya', 2, 5, 80),
(10, 'Meera', 3, 5, 85);
```

```sql
-- Common Window Functions
SELECT
   StudentID AS new_id,
   SubjectID AS new_cat,
   SUM(StudentID) OVER (PARTITION BY SubjectID ORDER BY StudentID) AS
`Total`,
   AVG(StudentID) OVER (PARTITION BY SubjectID ORDER BY StudentID) AS
`Average`,
   COUNT(StudentID) OVER (PARTITION BY SubjectID ORDER BY StudentID) AS
`Count`,
   MIN(StudentID) OVER (PARTITION BY SubjectID ORDER BY StudentID) AS
`Min`,
   MAX(StudentID) OVER (PARTITION BY SubjectID ORDER BY StudentID) AS
`Max`
FROM Students;
```

Explanation of the Query

1. **`SUM()`**: Calculates the cumulative total of `StudentID` for each `SubjectID` group.
2. **`AVG()`**: Computes the running average of `StudentID` for each `SubjectID` group.
3. **`COUNT()`**: Counts the rows processed so far within each `SubjectID` group.
4. **`MIN()`**: Finds the smallest `StudentID` encountered so far within each `SubjectID` group.
5. **`MAX()`**: Finds the largest `StudentID` encountered so far within each `SubjectID` group.

Practice Problems

Problem 1: Cumulative Total

Write a query to calculate the cumulative total of marks scored by students in each subject, ordered by `StudentID`.

Solution:

```
SELECT
    StudentID,
    SubjectID,
    Marks,
    SUM(Marks) OVER (PARTITION BY SubjectID ORDER BY StudentID) AS
CumulativeTotal
FROM Students;
```

Problem 2: Rank Students by Marks

Write a query to rank students within each subject based on their `Marks` in descending order. If two students have the same marks, they should get the same rank.

Solution:

```
SELECT
    StudentID,
    SubjectID,
    Marks,
    RANK() OVER (PARTITION BY SubjectID ORDER BY Marks DESC) AS Rank
FROM Students;
```

Problem 3: Find Top Scorer in Each Subject

Write a query to find the top scorer in each subject.

Solution:

```
SELECT
    StudentID,
    SubjectID,
    Marks,
```

```
    RANK() OVER (PARTITION BY SubjectID ORDER BY Marks DESC) AS Rank
FROM Students
WHERE Rank = 1;
```

Problem 4: Compare Marks with Previous Student

Write a query to compare each student's marks with the previous student's marks within the same subject.

Solution:

```
SELECT
    StudentID,
    SubjectID,
    Marks,
    LAG(Marks) OVER (PARTITION BY SubjectID ORDER BY StudentID) AS
PreviousMarks,
    Marks - LAG(Marks) OVER (PARTITION BY SubjectID ORDER BY
StudentID) AS Difference
FROM Students;
```

Problem 5: Percentage Contribution

Calculate each student's percentage contribution to the total marks in their subject.

Solution:

```
SELECT
    StudentID,
    SubjectID,
    Marks,
    Marks * 100.0 / SUM(Marks) OVER (PARTITION BY SubjectID) AS
PercentageContribution
FROM Students;
```

Problem 6: Find Students Scoring Above Average

Write a query to find students who scored above the average marks in their respective subjects.

Solution:

```
SELECT
    StudentID,
    SubjectID,
    Marks,
    AVG(Marks) OVER (PARTITION BY SubjectID) AS AverageMarks
FROM Students
WHERE Marks > AVG(Marks) OVER (PARTITION BY SubjectID);
```

Problem 7: Cumulative Count of Students

Write a query to calculate the cumulative count of students appearing in each subject.

Solution:

```
SELECT
    StudentID,
    SubjectID,
    COUNT(StudentID) OVER (PARTITION BY SubjectID ORDER BY StudentID)
AS CumulativeCount
FROM Students;
```

Challenge Problem

Problem:

For each subject, find the difference between the highest marks scored so far and the current student's marks.

Solution:

```
SELECT
    StudentID,
    SubjectID,
    Marks,
    MAX(Marks) OVER (PARTITION BY SubjectID ORDER BY StudentID ROWS
BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) AS MaxSoFar,
    Marks - MAX(Marks) OVER (PARTITION BY SubjectID ORDER BY StudentID
ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) AS Difference
FROM Students;
```