Lecture Notes – SUM( )

Section 1: Lecture Summary

The lecture introduces the **SUM()** aggregate function as a window function in MySQL, demonstrating its use with the **OVER()** clause on the eCommerceDB **Products** table. It covers basic total sum, repeating the sum across all product rows, cumulative sums ordered by **ProductID**, category-wise cumulative sums using **PARTITION BY CategoryID**, and limited cumulative sums using row framing with **ROWS BETWEEN 1 PRECEDING AND CURRENT ROW**.

Section 2: Key Concepts and Explanations

Window functions combine aggregate functions like **SUM()** with the **OVER()** clause to perform calculations across sets of rows while retaining row details.

- Basic **SUM(Price)** over all products returns a single total value.

- **SUM(Price) OVER()** repeats the total sum on every product row, showing all columns plus the sum.

- **SUM(Price) OVER(ORDER BY ProductID)** computes a running cumulative sum, adding each row's price to the previous total in **ProductID** order (e.g., 45,000; then 135,000; accumulating onward).

- **SUM(Price) OVER(PARTITION BY CategoryID ORDER BY ProductID)** resets the cumulative sum at each **CategoryID** boundary, restarting the running total within each category.

- **Framing** with **ROWS BETWEEN 1 PRECEDING AND CURRENT ROW** limits the sum to only the current row and one prior row (e.g., first row: 45,000; second: 45,000 + 90,000 = 135,000), ordered by **ProductID**.

Section 3: Example Code and Use Cases

```
-- Total sum of all product prices (single value)
SELECT SUM(Price) FROM Products;
```

```
-- All products with total sum repeated on each row
SELECT *, SUM(Price) OVER() AS TotalSumPrice FROM Products;
```

```
-- Cumulative sum of prices ordered by ProductID
SELECT *, SUM(Price) OVER(ORDER BY ProductID) AS CumulativeSum FROM
Products;
```

```
-- Category-wise cumulative sum of prices
SELECT *, SUM(Price) OVER(PARTITION BY CategoryID ORDER BY ProductID) AS
CategoryCumulativeSum FROM Products;
```

```
-- Cumulative sum of current and previous row only
SELECT *, SUM(Price) OVER(ORDER BY ProductID ROWS BETWEEN 1 PRECEDING AND
CURRENT ROW) AS CurrentPrevSum FROM Products;
```

Section 4: Key Takeaways

SUM() as a window function with **OVER()** enables detailed row-level results unlike standard aggregates. Use **ORDER BY** for cumulative sums, **PARTITION BY** to group calculations, and **ROWS** framing for bounded windows. Practice these on the **Products** table to build challenging queries combining these features.