Lecture Notes – AVG( )

Section 1: Lecture Summary

The lecture introduces the **AVG() window function** and demonstrates how to apply it in various scenarios. Starting with a basic aggregate query that returns a single average value, the lecture progresses to show cumulative averages ordered by price, then extends to category-wise cumulative averages using window function partitioning. The instructor illustrates how the same aggregate function produces different results depending on whether it is used as a simple aggregate or combined with window function clauses like ORDER BY and PARTITION BY.

Section 2: Key Concepts and Explanations

Basic Aggregate AVG(): When using AVG() without window function syntax, it returns a single scalar value representing the average across all rows in the result set.

Cumulative Average with ORDER BY: When AVG() is combined with an ORDER BY clause in a window function context, it calculates a running cumulative average. Each row shows the average of all rows from the first row up to the current row, sorted by the specified column. Rows with identical values in the ORDER BY column receive the same cumulative average value.

Partitioned Average with PARTITION BY: The PARTITION BY clause divides the dataset into logical groups (partitions). The AVG() function then calculates a cumulative average within each partition independently. When a new partition begins, the cumulative calculation resets and starts fresh for that partition.

Window Frame Behavior: The window frame implicitly extends from the first row of the partition (or result set) to the current row, creating the cumulative effect.

Section 3: Example Code and Use Cases

Example 1: Basic Average Price Across All Products

```
SELECT AVG(Price) AS AveragePrice
FROM Products;
```

This returns a single value representing the average price of all products in the Products table.

Example 2: Product Details with Cumulative Average Ordered by Price

```
SELECT
    ProductID,
    ProductName,
    Price,
    AVG(Price) OVER (ORDER BY Price) AS CumulativeAvgPrice
FROM Products
ORDER BY Price;
```

This query displays each product with its cumulative average price. Products with the same price receive the same cumulative average. For example, if two products both cost 1500, and they are the second and third products when sorted by price, both rows will show the cumulative average of the first three products combined.

Example 3: Category-Wise Cumulative Average Ordered by Price

```
SELECT
    ProductID,
    ProductName,
    CategoryID,
    Price,
    AVG(Price) OVER (PARTITION BY CategoryID ORDER BY Price) AS
CategoryAvgPrice
```

```
FROM Products
ORDER BY CategoryID, Price;
```

This query partitions products by CategoryID, then calculates a cumulative average price within each category. When the query transitions to a new category, the cumulative average calculation resets. For instance, if Category 1 products have a final cumulative average of 36300 and Category 2 products have a final cumulative average of 22500, each category maintains its own independent cumulative average progression.

Example 4: Using eCommerceDB - Category-Wise Average Order Amount

```
SELECT
    o.OrderID,
    c.CategoryID,
    o.TotalAmount,
    AVG(o.TotalAmount) OVER (PARTITION BY c.CategoryID ORDER BY
o.OrderDate) AS CategoryCumulativeAvg
FROM Orders o
JOIN OrderItems oi ON o.OrderID = oi.OrderID
JOIN Products p ON oi.ProductID = p.ProductID
JOIN Categories c ON p.CategoryID = c.CategoryID
ORDER BY c.CategoryID, o.OrderDate;
```

This applies the category-wise cumulative average concept to the eCommerceDB schema, showing how order amounts accumulate by category over time.

Section 4: Key Takeaways

AVG() serves dual purposes: As a basic aggregate function returning a single value, or as a window function providing row-by-row cumulative or partitioned averages.

ORDER BY controls the window frame: Adding ORDER BY to AVG() creates a cumulative window that grows from the first row to the current row.

PARTITION BY isolates calculations: Using PARTITION BY ensures that average calculations restart for each group, preventing data from different categories or segments from influencing each other's results.

Identical values share results: When multiple rows have the same value in the ORDER BY column, they all receive the same cumulative average, since they occupy the same logical position in the ordered window.

Practical applications include tracking trends: These techniques enable meaningful analysis such as monitoring how average prices or amounts evolve, with the ability to segment by category, customer, or other dimensions.