Lecture Notes – QueriesOnROW_NUMBER()

Section 1: Lecture Summary

This lecture focuses on the **ROW_NUMBER()** window function in MySQL and its practical applications for assigning sequential row numbers to records within specific partitions. The lecture demonstrates how to use ROW_NUMBER() combined with the OVER clause to organize and analyze data by categories and time periods, enabling easy identification of highest and lowest value items based on specified ordering criteria.

Section 2: Key Concepts and Explanations

Window Functions and ROW_NUMBER()

A **window function** operates on a set of rows and returns a value for each row within that set. The ROW_NUMBER() function assigns a unique sequential integer to each row within a partition, starting from 1. This differs from aggregate functions because it returns a result for every row rather than combining rows into a single result.

PARTITION BY Clause

The PARTITION BY clause divides the result set into partitions or groups. ROW_NUMBER() resets to 1 for each new partition. This allows you to assign row numbers independently within categories, departments, or other logical groupings without mixing data across partitions.

ORDER BY Clause

The ORDER BY clause within the OVER statement determines the sequence in which rows receive their numbers. Combined with PARTITION BY, it enables you to rank or number rows based on specific criteria like price, amount, or date within each partition.

Basic Syntax

The fundamental structure combines SELECT with ROW_NUMBER():

```
SELECT column1, column2,
       ROW_NUMBER() OVER (PARTITION BY partition_column ORDER BY
sort_column DESC) AS row_num
FROM table_name;
```

Section 3: Example Code and Use Cases

Example 1: Assign Row Numbers to Products Within Each Category by Price

This query assigns sequential numbers to products within each category, ordered by price in descending order:

```
SELECT ProductID, ProductName, CategoryID, Price,
       ROW_NUMBER() OVER (PARTITION BY CategoryID ORDER BY Price DESC) AS
CategorySequence
FROM Products;
```

This produces results where products are grouped by CategoryID, arranged by highest to lowest price within each category, and assigned row numbers 1, 2, 3, and so on for each category. This structure allows analysts to easily identify premium products (row 1) and budget options (highest row number) within each category.

Example 2: Assign Row Numbers to 2025 Orders Based on Total Amount

This query filters orders placed in 2025 and assigns row numbers based on order amount in descending order:

```
SELECT OrderID, OrderDate, TotalAmount,
       ROW_NUMBER() OVER (ORDER BY TotalAmount DESC) AS amount_rank
FROM Orders
WHERE YEAR(OrderDate) = 2025;
```

The WHERE clause filters records to include only orders from the year 2025 using the YEAR() function. The ROW_NUMBER() window function then assigns sequential numbers with ORDER BY TotalAmount DESC, placing the highest-value order at rank 1. This enables quick identification of top-performing orders and positioning of orders by their monetary value.

Section 4: Key Takeaways

ROW_NUMBER() assigns unique sequential integers to each row within partitions, resetting to 1 for each new partition group.

PARTITION BY creates independent groups, allowing row numbering to occur separately within each category or logical division.

ORDER BY determines the sequence in which row numbers are assigned, whether ascending or descending.

Practical applications include ranking products by price within categories and ordering transactions by amount, enabling straightforward analysis and positioning of records.

The combination of PARTITION BY and ORDER BY provides flexibility to solve complex analytical queries without requiring subqueries or complex joins.