Lecture Notes – QueriesOnRANK()_DENSERANK()

Section 1: Lecture Summary

The lecture demonstrates writing queries using **RANK()** and **DENSE_RANK()** window functions on the eCommerceDB. It covers ranking products by total units sold from the **OrderItems** table and ranking categories by total revenue using joins between **Products** and **OrderItems** tables with grouping.

Section 2: Key Concepts and Explanations

RANK() assigns rankings based on ordered values but skips numbers after ties (e.g., 1,1,3). **DENSE_RANK()** assigns rankings without skipping numbers after ties (e.g., 1,1,2). Both require an **OVER** clause with **ORDER BY** in descending order for highest values first. Use **GROUP BY** and aggregate functions like **SUM(Quantity)** or **SUM(Subtotal)** for per-product or per-category summaries before applying the window function. Joins connect **Products.CategoryID** to **OrderItems.ProductID** for category revenue calculations.

Section 3: Example Code and Use Cases

Query to rank products by total units sold:

```
SELECT
    ProductID,
    SUM(Quantity) AS units_sold,
    RANK() OVER (ORDER BY SUM(Quantity) DESC) AS rank
FROM OrderItems
GROUP BY ProductID;
```

Results show ties at rank 1 for products with 8 units, skipping to rank 3 for 7 units.

Using **DENSE_RANK()** instead:

```
SELECT
    ProductID,
```

```
    SUM(Quantity) AS units_sold,
    DENSE_RANK() OVER (ORDER BY SUM(Quantity) DESC) AS dense_rank
FROM OrderItems
GROUP BY ProductID;
```

Results show ties at rank 1, then 2 without skips.

Query to rank categories by total revenue:

```
SELECT
    p.CategoryID,
    SUM(oi.Subtotal) AS category_revenue,
    RANK() OVER (ORDER BY SUM(oi.Subtotal) DESC) AS revenue_rank
FROM Products p
NATURAL JOIN OrderItems oi
GROUP BY p.CategoryID;
```

Results rank categories 1 through 6 by descending revenue, with ties assigned appropriately.

Section 4: Key Takeaways

Group data by the ranking key (e.g., **ProductID**, **CategoryID**) and sum relevant fields before applying **RANK()** or **DENSE_RANK()** over ordered aggregates. Use descending order for top performers first. **RANK()** skips ranks on ties; **DENSE_RANK()** does not, making it suitable when consecutive ranks matter.