Lecture Notes – QueriesOnLAG()_LEAD()

Section 1: Lecture Summary

The lecture demonstrates analytical queries using **LAG** and **LEAD** window functions on eCommerceDB and companyDB. It covers calculating days between customer orders, comparing payment modes, review timelines, and department-wise hiring sequences to analyze customer behavior and patterns.

Section 2: Key Concepts and Explanations

LAG retrieves the previous row's value within a partition, ordered by a specified column, to compare current data with prior records, such as previous order date or payment mode.

LEAD retrieves the next row's value within a partition, used for forward-looking analysis like next review date or next hire.

Both functions require **OVER** clause with **PARTITION BY** for grouping (e.g., by CustomerID or DeptID) and **ORDER BY** for sequencing (e.g., by OrderDate or HireDate) to ensure logical row order.

DATEDIFF computes days between current and previous/next dates when combined with LAG or LEAD.

Joins are used when needed, such as Payments with Orders to access CustomerID.

Section 3: Example Code and Use Cases

Days passed since customer's last order (eCommerceDB):

```
SELECT CustomerID, OrderDate, OrderID,
       DATEDIFF(OrderDate, LAG(OrderDate) OVER (PARTITION BY CustomerID
ORDER BY OrderDate)) AS days_gap
FROM Orders;
```

Shows gaps like 200 days between orders for analysis of return frequency.

Customer's current and previous payment mode (eCommerceDB):

```
SELECT o.CustomerID, p.PaymentID, p.PaymentMode,
       LAG(p.PaymentMode) OVER (PARTITION BY o.CustomerID ORDER BY
```

```
p.PaymentDate) AS prev_mode
FROM Payments p
NATURAL JOIN Orders o;
```

Tracks shifts, e.g., from credit card to UPI.

Customer review timeline with next review date (eCommerceDB):

```
SELECT CustomerID, ProductID, Rating, ReviewDate,
       LEAD(ReviewDate) OVER (PARTITION BY CustomerID ORDER BY ReviewDate)
AS next_review_date
FROM Reviews;
```

Displays next review dates for timeline analysis.

Department-wise next hire analysis (companyDB):

```
SELECT DeptID, FirstName, HireDate,
       LEAD(FirstName) OVER (PARTITION BY DeptID ORDER BY HireDate) AS
next_hire
FROM Employees;
```

Shows sequence like Rahul followed by Sneha in a department.

Section 4: Key Takeaways

Use **LAG** for previous row comparisons and **LEAD** for next row comparisons in window functions.

Always specify **PARTITION BY** for grouping and **ORDER BY** for row sequence to get accurate results.

Combine with **DATEDIFF** for time gaps and joins for related data like CustomerID from Orders.

These enable customer retention analysis, payment trend tracking, and hiring patterns.