Lecture Notes – MultipleCTEs

Section 1: Lecture Summary

The lecture covers advanced usage of **common table expressions (CTEs)**, including syntax review, challenging queries, CTEs in subqueries, CTEs with natural joins, and **multiple CTEs (chaining)**. Examples demonstrate finding orders with specific products, orders by customers from a city, and departments with average salary above company average using multiple CTEs joined in the main query.

Section 2: Key Concepts and Explanations

CTEs are defined with a name and used in the main query, similar to subqueries but reusable. They can be placed in the FROM clause via joins or subqueries with IN. **Natural join** matches CTEs to main tables on common columns like **OrderID** or **CustomerID**. **Multiple CTEs** are chained by comma separation without repeating WITH, enabling complex logic like comparing department averages to company average. CTEs must be referenced in FROM clause or subqueries; conditions use joined CTE columns in WHERE.

Section 3: Example Code and Use Cases

Orders containing Laptop Air 13 inch (ProductID 102) using CTE in subquery:

```
WITH laptop_orders AS (
  SELECT OrderID
  FROM OrderItems
  WHERE ProductID = 102
)
SELECT OrderID, CustomerID
FROM Orders
WHERE OrderID IN (SELECT OrderID FROM laptop_orders);
```

Orders by customers from Delhi using CTE with natural join:

```
WITH delhi_customers AS (
  SELECT CustomerID
  FROM Customers
  WHERE City = 'Delhi'
)
SELECT OrderID, OrderDate
```

```
FROM Orders
NATURAL JOIN delhi_customers;
```

Adapted multiple CTEs example for eCommerceDB (departments replaced by **Categories**, employees by **Reviews** with **Rating** as salary proxy; shows categories with average rating above company average):

```
WITH category_average AS (
  SELECT CategoryID, AVG(Rating) AS category_avg_rating
  FROM Reviews r
  JOIN Products p ON r.ProductID = p.ProductID
  GROUP BY CategoryID
),
company_average AS (
  SELECT AVG(Rating) AS company_avg_rating
  FROM Reviews
)
SELECT c.CategoryID, c.CategoryName, ca.category_avg_rating
FROM Categories c
NATURAL JOIN category_average ca
CROSS JOIN company_average co
WHERE ca.category_avg_rating > co.company_avg_rating;
```

Section 4: Key Takeaways

CTEs simplify complex queries by naming subquery results for reuse in joins or IN subqueries. Multiple CTEs enable comparisons like averages via chaining and natural joins on common keys. Use CTEs in FROM clause for main query integration; apply filters in WHERE on joined CTE columns. Next topic is recursive CTEs.