Lecture Notes – Case_When-Having

Section 1: Lecture Summary

This lecture introduces the application of **CASE WHEN** conditional logic within the **HAVING** clause for advanced filtering of grouped data. While **CASE WHEN** is commonly used in SELECT, WHERE, and ORDER BY clauses, this discussion focuses specifically on its integration with HAVING to create complex, multi-condition filtering rules for aggregated groups. The key principle is that instead of writing simple direct conditions in HAVING, you can construct sophisticated conditional statements that evaluate different aggregate functions based on varying column values.

Section 2: Key Concepts and Explanations

CASE WHEN in HAVING Clause

The HAVING clause traditionally filters grouped results using aggregate functions. By embedding **CASE WHEN** within HAVING, you enable category-specific or condition-specific filtering logic. This means different groups can be evaluated against different threshold criteria rather than applying a single uniform condition.

Syntax Structure

The general pattern combines GROUP BY with a HAVING clause containing CASE WHEN logic:

```
SELECT column_name, aggregate_function(column)
FROM table_name
GROUP BY column_name
HAVING CASE
  WHEN condition_1 THEN aggregate_condition_1
  WHEN condition_2 THEN aggregate_condition_2
```

```
  ELSE aggregate_condition_n
END
```

Each WHEN branch must evaluate an aggregate function result, not just return a simple value. The ELSE clause also requires an aggregate condition for consistency.

Why Use This Approach

Complex business requirements often demand different filtering rules for different categories or segments. Rather than writing multiple queries or using complex nested logic, **CASE WHEN** in HAVING provides a clean, readable solution within a single query.

Section 3: Example Code and Use Cases

Example: Filtering Products by Category with Variable Thresholds

Using the eCommerceDB schema, consider this requirement: show the total price of products grouped by category, but apply different filtering rules per category.

```
SELECT
  CategoryID,
  SUM(Price) AS TotalPrice
FROM Products
GROUP BY CategoryID
HAVING CASE
  WHEN CategoryID = 1 THEN SUM(Price) > 100000
  WHEN CategoryID = 2 THEN SUM(Price) > 50000
  ELSE SUM(Price) > 2000
END
```

Query Execution Results

When executed against sample data:

- CategoryID 1 appears because its total price exceeds 100,000

- CategoryID 2 does not appear because its total (45,000 from products priced at 20,000 and 25,000) falls below the 50,000 threshold

- CategoryID 3, 4, 5 appear because they exceed 2,000

- CategoryID 6 does not appear because its total (1,200) fails to meet the 2,000 minimum

Additional Use Case: Order Analytics

You can apply similar logic to Orders or OrderItems for revenue-based filtering:

```
SELECT
  CustomerID,
  COUNT(OrderID) AS OrderCount,
  SUM(TotalAmount) AS CustomerRevenue
FROM Orders
GROUP BY CustomerID
HAVING CASE
  WHEN COUNT(OrderID) >= 5 THEN SUM(TotalAmount) > 50000
  WHEN COUNT(OrderID) >= 2 THEN SUM(TotalAmount) > 10000
  ELSE SUM(TotalAmount) > 1000
END
```

This filters customers based on both order frequency and spending, with stricter revenue requirements for repeat customers.

Section 4: Key Takeaways

- **CASE WHEN extends HAVING functionality** by enabling conditional logic rather than static thresholds

- **Each WHEN branch must contain an aggregate function**, not arbitrary values

- **ELSE clause requires the same structure** as WHEN branches for query consistency

- **Different groups can meet different criteria**, allowing nuanced filtering based on business logic

- **Use this technique when filtering rules depend on column values or group characteristics**, making queries more maintainable than alternative approaches