Lecture Notes – Sub Queries-HAVING Clause

Section 1: Lecture Summary

Subqueries can be used in the **HAVING** clause to filter grouped results based on aggregate comparisons. The **HAVING** clause requires a **GROUP BY** and works with aggregate functions like **AVG** or **SUM**, where the subquery must return a single scalar value. Examples include finding departments with average salary higher than the finance department and products with total order quantity above the average across all products, emphasizing correct subquery logic to match the intended grouping.

Section 2: Key Concepts and Explanations

Subqueries in **HAVING** follow the structure: SELECT columns FROM tables GROUP BY columns HAVING aggregate > (SELECT scalar FROM tables WHERE condition). The subquery must return one value, not a list. **HAVING** filters groups after **GROUP BY** and aggregates, unlike **WHERE** which filters rows before grouping. Common aggregates include **AVG(salary)** or **SUM(quantity)**. A key pitfall is using **AVG(quantity)** on order items, which averages per row instead of per product; correct it with **SUM(quantity) / COUNT(DISTINCT productID)** for per-product average.

Section 3: Example Code and Use Cases

Using **companyDB**:

Find departments whose average salary is higher than finance department (DeptID 3):

```
SELECT DeptID, AVG(Salary) AS avg_salary
FROM Employees
GROUP BY DeptID
HAVING AVG(Salary) > (
    SELECT AVG(Salary)
    FROM Employees
    WHERE DeptID = 3
);
```

This groups employees by **DeptID**, computes average **Salary** per department, and filters using the scalar subquery average for finance.

Using **companyDB** (adapted for projects, assuming total employee count per project as quantity proxy):

Products (projects) whose total employee assignments exceed average across projects:

```
SELECT ProjectID, COUNT(EmpID) AS total_assignments
FROM EmployeeProjects
GROUP BY ProjectID
HAVING COUNT(EmpID) > (
    SELECT SUM(quantity) / COUNT(DISTINCT p.ProjectID)
    FROM Projects p
);
```

Incorrect version (averages per assignment row):

```
SELECT ProjectID, COUNT(EmpID) AS total_assignments
FROM EmployeeProjects
GROUP BY ProjectID
HAVING COUNT(EmpID) > (
    SELECT AVG(quantity)
    FROM some_quantity_table
);
```

Corrected to average per project using total sum divided by distinct count.

Section 4: Key Takeaways

Subqueries in **HAVING** enable group-level comparisons with scalar results from aggregates. Always verify subquery returns the expected scalar (e.g., per-group average, not per-row). Use **GROUP BY** in main query for **HAVING**; test results to ensure logic matches intent, like dividing totals by distinct groups rather than row count. Read queries to predict outcomes without execution.