Lecture 4 Activity: Advanced SQL & Query Optimization

(Slow Learner)

Case Study: Student Performance Analysis System

You are helping the university analyze student performance data to generate reports efficiently. The database includes:

Tables Overview:

Students

StudentID	Name	Department
101	Alice	CSE
102	Bob	ECE
103	Clara	CSE

Courses

CourseID	CourseName	Department
C1	DBMS	CSE
C2	Networks	ECE

Results

StudentID	CourseID	Marks
101	C1	85
102	C2	78
103	C1	92

Activity Tasks (With Solutions)

1. Nested Query

Task: Find students who scored above the **average marks**

```
SELECT Name
FROM Students
WHERE StudentID IN (
    SELECT StudentID
    FROM Results
                WHERE Marks > (SELECT AVG(Marks) FROM Results) );
```

2. Correlated Subquery

Task: List students who scored the **highest marks in each department**.

```
SELECT s.Name, s.Department, r.Marks
FROM Students s
JOIN Results r ON s.StudentID = r.StudentID
JOIN Courses c ON r.CourseID = c.CourseID
WHERE r.Marks = (
    SELECT MAX(r2.Marks)
    FROM Results r2
    JOIN Courses c2 ON r2.CourseID = c2.CourseID
    WHERE c2.Department = c.Department
);
```

3. Set Operation - UNION

Task: Get a list of all students in either CSE or those who have taken the DBMS course.

```
-- Students in CSE
SELECT Name FROM Students WHERE Department = 'CSE'
UNION
-- Students who took DBMS
SELECT s.Name
FROM Students s
JOIN Results r ON s.StudentID = r.StudentID
JOIN Courses c ON r.CourseID = c.CourseID
WHERE c.CourseName = 'DBMS';
```

4. Set Operation - INTERSECT (Emulated in MySQL using INNER JOIN)

Task: Find students who are in CSE **and** have taken DBMS.

```
SELECT s.Name
FROM Students s
JOIN Results r ON s.StudentID = r.StudentID
JOIN Courses c ON r.CourseID = c.CourseID
WHERE s.Department = 'CSE' AND c.CourseName = 'DBMS';
```

5. Basic Query Optimization

Task: Optimize the following query:

```
SELECT * FROM Students, Results
WHERE Students.StudentID = Results.StudentID
AND Results.Marks > 80;
```

Optimized:

```
SELECT s.Name, r.Marks
FROM Students s
JOIN Results r ON s.StudentID = r.StudentID
WHERE r.Marks > 80;
```

6. Nested Query – Second Highest Marks

Task: Find the student(s) who got the **second highest marks**.

```
SELECT Name
FROM Students
WHERE StudentID IN (
    SELECT StudentID
    FROM Results
    WHERE Marks = (
        SELECT MAX(Marks)
        FROM Results
        WHERE Marks < (SELECT MAX(Marks) FROM Results)
    )
);
```

7. Correlated Subquery – Students Who Scored Above Average in Each Course

Task: Find students who scored **above average in their respective courses**.

```
SELECT s.Name, c.CourseName, r.Marks
FROM Students s
JOIN Results r ON s.StudentID = r.StudentID
JOIN Courses c ON r.CourseID = c.CourseID
WHERE r.Marks > (
    SELECT AVG(r2.Marks)
    FROM Results r2
    WHERE r2.CourseID = r.CourseID
);
```

8. Aggregation with Optimization – Average Marks per Student

Task: Display average marks for each student and show only those with average > 80.

```
SELECT s.Name, AVG(r.Marks) AS AvgMarks
FROM Students s
JOIN Results r ON s.StudentID = r.StudentID
GROUP BY s.Name
HAVING AVG(r.Marks) > 80;
```

9. Set Operation – MINUS (Students who didn't take DBMS)

Task: Get names of students who did **not** take the DBMS course.

```
SELECT Name FROM Students
MINUS
SELECT DISTINCT s.Name
FROM Students s
JOIN Results r ON s.StudentID = r.StudentID
JOIN Courses c ON r.CourseID = c.CourseID
WHERE c.CourseName = 'DBMS';
```