Project 34–37 SQL Solutions

Schema

```
Students(student_id, name, dob, department_id)
Departments(department_id, department_name)
Courses(course_id, course_name, department_id)
Enrollments(enrollment_id, student_id, course_id, grade)
```

Part A – Advanced SQL (10 Marks)

11. Create a view named TopStudents

```
CREATE VIEW TopStudents AS
SELECT
    s.student_id,
    s.name,
    AVG(e.grade) AS average_grade
FROM Students s
JOIN Enrollments e ON s.student_id = e.student_id
GROUP BY s.student_id, s.name
HAVING AVG(e.grade) > 3.5;
```

12. Stored procedure to increase the grade of all students in a given course by 0.5 (max 4.0)

```
DELIMITER $$
CREATE PROCEDURE IncreaseCourseGrades(IN p_course_id INT)
BEGIN
    UPDATE Enrollments
    SET grade = CASE
                    WHEN grade + 0.5 > 4.0 THEN 4.0
                    ELSE grade + 0.5
                END
    WHERE course_id = p_course_id;
END $$
DELIMITER ;
```

13. Correlated subquery – list students whose grade in any course is the highest for that course

```
SELECT DISTINCT
    s.student_id,
    s.name
FROM Students s
JOIN Enrollments e ON s.student_id = e.student_id
WHERE e.grade = (
    SELECT MAX(e2.grade)
    FROM Enrollments e2
    WHERE e2.course_id = e.course_id
);
```

Part B – Applied Scenario (15 Marks)

Report Query

```
SELECT
    d.department_name,
    COUNT(DISTINCT s.student_id) AS total_students,
    AVG(e.grade) AS average_grade,
    COUNT(DISTINCT c.course_id) AS number_of_courses
FROM Departments d
JOIN Courses c ON d.department_id = c.department_id
JOIN Students s ON d.department_id = s.department_id
JOIN Enrollments e ON s.student_id = e.student_id
                   AND e.course_id = c.course_id
GROUP BY d.department_name
HAVING COUNT(DISTINCT c.course_id) >= 2
ORDER BY average_grade DESC;
```