Experiment 1.2

Student Name: Shivansh
UID: 23BCS13205
Branch: B.E CSE
Section: 23BCS_KRG-2_A
Semester: 5th
Date of Performance:24/07/25
Subject Name: ADBMS
Subject Code: 23CSH-333

1. **Aim:** To design and implement advanced SQL Server features such as Views, Triggers, Stored Procedures, Functions, and Cursors to automate data validation, enable modular query logic, and support iterative data processing in relational databases.

Easy-Level Problem

Title: Filtered Student View with Aggregated Data

Procedure (Step-by-Step):

1. Create tables for students and their subject-wise marks.
2. Insert at least 5 students and 10 marks entries.
3. Create a view that filters students who scored more than 80 marks.
4. Query the view to get performance grouped by department.

Medium-Level Problem

Title: Logging Grade Updates with Triggers & Functions

Procedure (Step-by-Step):

1. Add two new tables: one for grades and one for logging updates.
2. Write a function to allow only valid grades (A, B, C, D, F).
3. Create a trigger that logs every grade update into the log table with old value, new value, and time.
4. Update a grade to check if the log is recorded properly.

Hard-Level Problem

Title: GPA Summary Using Procedures and Cursors

Procedure (Step-by-Step):

1. Use student, course, and enrollment data.
2. Write a stored procedure that takes a department as input.
3. Use a cursor to go through each student in that department.
4. Calculate GPA by converting grades to numeric values (A=10, B=8, etc.).
5. Display each student's name and GPA.
6. Call the procedure with a specific department.

2. **Tools Used:** Oracle Live SQL

3. Code & Output:

Easy-Level Problem Code:

```
CREATE TABLE Students2 (
    student_id INT PRIMARY KEY,
    student_name VARCHAR(50),
    department VARCHAR(50)
);

CREATE TABLE Marks (
    mark_id INT PRIMARY KEY,
    student_id INT,
    subject_name VARCHAR(50),
    marks INT,
    FOREIGN KEY (student_id) REFERENCES Students(student_id)
);

INSERT INTO Students2 (student_id, student_name, department) VALUES
(1, 'Amit Sharma', 'CSE'),
(2, 'Neha Singh', 'ECE'),
(3, 'Ravi Kumar', 'ME'),
(4, 'Priya Verma', 'CSE'),
(5, 'Vikas Gupta', 'EEE');

INSERT INTO Marks (mark_id, student_id, subject_name, marks) VALUES
(101, 1, 'DBMS', 85),
(102, 1, 'CN', 78),
(103, 2, 'Digital Electronics', 88),
(104, 2, 'Signals', 91),
(105, 3, 'Thermodynamics', 75),
(106, 3, 'Fluid Mechanics', 82),
(107, 4, 'Data Structures', 92),
(108, 4, 'OS', 89),
(109, 5, 'Machines', 84),
(110, 5, 'Power Systems', 65);

CREATE VIEW HighScorers AS
SELECT
    Students2.student_id,
    Students2.student_name,
    Students2.department,
    Marks.subject_name,
    Marks.marks
FROM Students2
INNER JOIN Marks ON Students2.student_id = Marks.student_id
```

```
WHERE Marks.marks > 80;

SELECT
   department,
   COUNT(*) AS high_score_count
FROM HighScorers
GROUP BY department;
```

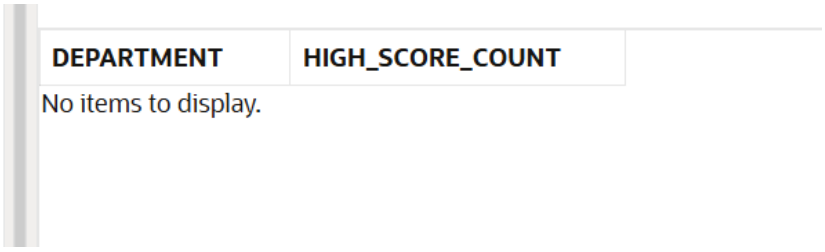

Figure 1: Output of Easy-Level Problem

Medium-Level Problem:

```
CREATE TABLE StudentGrades (
   sg_id NUMBER PRIMARY KEY,
   full_name VARCHAR2(50),
   letter_grade CHAR(1)
);

CREATE TABLE GradeChangeLog (
   log_entry_id NUMBER GENERATED BY DEFAULT ON NULL AS IDENTITY
     PRIMARY KEY,
   sg_id NUMBER,
   prev_grade CHAR(1),
   new_grade CHAR(1),
   change_time TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE OR REPLACE TRIGGER tr_log_grade_update
BEFORE UPDATE ON StudentGrades
FOR EACH ROW
BEGIN
   IF :NEW.letter_grade NOT IN ('A', 'B', 'C', 'D', 'F') THEN
      RAISE_APPLICATION_ERROR(-20001, 'Invalid letter grade!');
   END IF;

   INSERT INTO GradeChangeLog (sg_id, prev_grade, new_grade)
   VALUES (:OLD.sg_id, :OLD.letter_grade, :NEW.letter_grade);
END;
/
```

```
INSERT INTO StudentGrades (sg_id, full_name, letter_grade) VALUES (1, 'Amit Sharma', 'B');
INSERT INTO StudentGrades (sg_id, full_name, letter_grade) VALUES (2, 'Neha Singh', 'A');

UPDATE StudentGrades SET letter_grade = 'C' WHERE sg_id = 1;

SELECT * FROM GradeChangeLog;
```

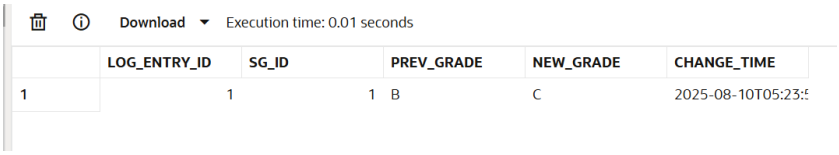

Figure 2: Output of Middle-Level Problem

Hard-Level Problem:

```
CREATE TABLE Students3 (
    student_id NUMBER PRIMARY KEY,
    full_name VARCHAR2(50),
    department VARCHAR2(50)
);

CREATE TABLE Courses (
    course_id NUMBER PRIMARY KEY,
    course_name VARCHAR2(50),
    department VARCHAR2(50)
);

CREATE TABLE Enrollments3 (
    enrollment_id NUMBER PRIMARY KEY,
    student_id NUMBER REFERENCES Students(student_id),
    course_id NUMBER REFERENCES Courses(course_id),
    grade CHAR(1)
);

INSERT INTO Students3 VALUES (1, 'Amit Sharma', 'Computer Science');
INSERT INTO Students3 VALUES (2, 'Neha Singh', 'Computer Science');
INSERT INTO Students3 VALUES (3, 'Raj Mehta', 'Mathematics');

INSERT INTO Courses VALUES (101, 'DBMS', 'Computer Science');
INSERT INTO Courses VALUES (102, 'Algorithms', 'Computer Science');
INSERT INTO Courses VALUES (201, 'Linear Algebra', 'Mathematics');

INSERT INTO Enrollments3 VALUES (1, 1, 101, 'A');
INSERT INTO Enrollments3 VALUES (2, 1, 102, 'B');
INSERT INTO Enrollments3 VALUES (3, 2, 101, 'A');
INSERT INTO Enrollments3 VALUES (4, 2, 102, 'A');
```

```
INSERT INTO Enrollments3 VALUES (5, 3, 201, 'B');

CREATE OR REPLACE PROCEDURE Dept_GPA_Summary(p_department IN VARCHAR2)
     IS
   CURSOR student_cur IS
      SELECT student_id, full_name
      FROM Students3
      WHERE department = p_department;

   v_student_id Students3.student_id%TYPE;
   v_full_name Students3.full_name%TYPE;
   v_gpa NUMBER;
BEGIN
   FOR stu_rec IN student_cur LOOP
      v_student_id := stu_rec.student_id;
      v_full_name := stu_rec.full_name;

      SELECT AVG(CASE grade
                 WHEN 'A' THEN 10
                 WHEN 'B' THEN 8
                 WHEN 'C' THEN 6
                 WHEN 'D' THEN 4
                 WHEN 'F' THEN 0
                END)
      INTO v_gpa
      FROM Enrollments3 e
      JOIN Courses c ON e.course_id = c.course_id
      WHERE e.student_id = v_student_id
        AND c.department = p_department;

      DBMS_OUTPUT.PUT_LINE('Student: ' || v_full_name || ', GPA: ' ||
     NVL(TO_CHAR(v_gpa), 'N/A'));
   END LOOP;
END;
/

SET SERVEROUTPUT ON
BEGIN
   Dept_GPA_Summary('Computer Science');
END;
/
```

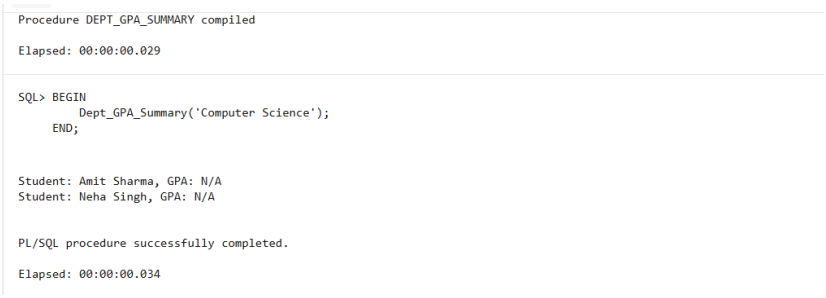

Figure 3: Output of Hard-Level Problem

5. Learning Outcomes (What I have learnt):

- Understand how to create and query views to filter and aggregate data for specific analytical needs.
- Implement triggers and user-defined functions to enforce business rules and automatically log database changes.
- Use stored procedures and cursors to process data iteratively and perform complex calculations like GPA computation.
- Apply grade-to-numeric mapping techniques to translate qualitative data into quantitative metrics for reporting.
- Integrate multiple SQL features (Views, Triggers, Functions, Procedures, Cursors) to build modular, maintainable, and automated database solutions.