DataBase2 Documentation:-

1.Connection to SQL Plus…

create (sys as sysdba) //Password: ADMIN

create manager_user with password: manager123.

```
SQL*Plus: Release 21.0.0.0.0 - Production on Thu Dec 18 16:59:37 2025
Version 21.3.0.0.0

Copyright (c) 1982, 2021, Oracle.  All rights reserved.

Enter user-name: sys as sysdba
Enter password:

Connected to:
Oracle Database 21c Express Edition Release 21.0.0.0.0 - Production
Version 21.3.0.0.0

SQL> SHOW PDBS;

    CON_ID CON_NAME                       OPEN MODE  RESTRICTED
---------- ------------------------------ ---------- ----------
         2 PDB$SEED                       READ ONLY  NO
         3 XEPDB1                         READ WRITE NO
SQL> ALTER SESSION SET CONTAINER =XEPDB1;

Session altered.

SQL> CREATE USER   manager_user IDENTIFIED BY manager123;

User created.
```

Create tables such as: Students, Professors, Courses, Registers, Exams, Warnings, Audit trial.

```
SQL> CREATE TABLE Students (
  2      id NUMBER PRIMARY KEY,
  3      name VARCHAR2(100) NOT NULL,
  4      academic_status VARCHAR2(20)
  5          CHECK (academic_status IN ('Active', 'Suspended')),
  6      total_credits NUMBER DEFAULT 0
  7  );

Table created.
```

```
SQL> CREATE TABLE Courses (
  2      id NUMBER PRIMARY KEY,
  3      name VARCHAR2(100) NOT NULL,
  4      professor_id NUMBER,
  5      credit_hours NUMBER
  6  );

Table created.
```

```
SQL> CREATE TABLE Register (
  2      id NUMBER PRIMARY KEY,
  3      student_id NUMBER NOT NULL,
  4      course_id NUMBER NOT NULL,
  5      CONSTRAINT fk_reg_student FOREIGN KEY (student_id) REFERENCES Students(id),
  6      CONSTRAINT fk_reg_course FOREIGN KEY (course_id) REFERENCES Courses(id),
  7      CONSTRAINT uq_student_course UNIQUE (student_id, course_id)
  8  );

Table created.
```

```
SQL> CREATE TABLE Exams (
  2      id NUMBER PRIMARY KEY,
  3      course_id NUMBER NOT NULL,
  4      exam_date DATE NOT NULL,
  5      exam_type VARCHAR2(20),
  6      CONSTRAINT fk_exam_course FOREIGN KEY (course_id) REFERENCES Courses(id)
  7  );

Table created.
```

```
SQL> CREATE TABLE Professors (
  2      id NUMBER PRIMARY KEY,
  3      name VARCHAR2(100) NOT NULL,
  4      department VARCHAR2(100)
  5  );

Table created.
```

```
SQL> CREATE TABLE ExamResults (
  2      id NUMBER PRIMARY KEY,
  3      registration_id NUMBER NOT NULL,
  4      grade NUMBER,
  5      status VARCHAR2(20),
  6      CONSTRAINT fk_examresults_register
  7          FOREIGN KEY (registration_id)
  8          REFERENCES Register(id)
  9  );

Table created.
```

```
SQL> CREATE TABLE AuditTrail (
  2      id NUMBER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
  3      table_name VARCHAR2(50),
  4      operation VARCHAR2(10),
  5      old_data CLOB,
  6      new_data CLOB,
  7      timestamp TIMESTAMP DEFAULT CURRENT_TIMESTAMP
  8  );

Table created.
```

```
SQL> CREATE TABLE Warnings (
  2      id NUMBER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
  3      student_id NUMBER NOT NULL,
  4      warning_reason VARCHAR2(255) NOT NULL,
  5      warning_date DATE DEFAULT SYSDATE,
  6      CONSTRAINT fk_student
  7          FOREIGN KEY (student_id)
  8          REFERENCES Students(id)
  9  );

Table created.
```

```
SQL> GRANT CREATE SESSION TO usertwo;

Grant succeeded.
```

User2 insert in Students table and Register table.

```
SQL> INSERT INTO Students(id, name, academic_status) VALUES (1, 'Salma', 'Active');

1 row created.

SQL> INSERT INTO Students(id, name, academic_status) VALUES (2, 'Fatma', 'Active');

1 row created.

SQL> INSERT INTO Students(id, name, academic_status) VALUES (3, 'Marina', 'Active');

1 row created.

SQL> INSERT INTO Students(id, name, academic_status) VALUES (4, 'Mariam', 'Active');

1 row created.

SQL> INSERT INTO Students(id, name, academic_status) VALUES (5, 'Shahd', 'Active');

1 row created.

SQL> INSERT INTO Students(id, name, academic_status) VALUES (6, 'Nada', 'Active');

1 row created.
```

```
SQL> INSERT INTO Professors (id, name, department)
  2  VALUES (101, 'Dr. Ahmed', 'Computer Science');

1 row created.
```

```
SQL> INSERT INTO Professors (id, name, department)
  2  VALUES (101, 'Dr. Ahmed', 'Computer Science');

1 row created.

SQL> INSERT INTO Professors (id, name, department)
  2  VALUES (102, 'Dr. Mona', 'Mathematics');

1 row created.
```

```
SQL>   INSERT INTO Courses (id, name, professor_id, credit_hours)
  2  VALUES
  3      (2, 'DB2', 102, 3);

1 row created.
```

```
SQL> INSERT INTO Register (id, student_id, course_id)
  2  VALUES
  3      (1, 1, 2);

1 row created.
```

```
SQL> INSERT INTO Register (id,student_id,course_id) VALUES(2,2,1);

1 row created.
```

```
SQL> INSERT INTO Register (id,student_id,course_id)VALUES(3,4,2);

1 row created.
```

```
SQL> INSERT INTO Register (id,student_id,course_id)VALUES(4,4,1);

1 row created.
```

```
SQL> INSERT INTO Register (id,student_id,course_id)VALUES(5,5,1);

1 row created.
```

```
SQL> INSERT INTO Register (id,student_id,course_id)VALUES(6,3,1);

1 row created.
```

Select from Students table

```
SQL> SELECT* FROM Students;

        ID
----------
NAME
--------------------------------------------------------------------------------
ACADEMIC_STATUS      TOTAL_CREDITS STUDENT_STATUS
-------------------- ------------- --------------------
         1
Salma
Active                           0

         2
Fatma
Active                           0

        ID
----------
NAME
--------------------------------------------------------------------------------
ACADEMIC_STATUS      TOTAL_CREDITS STUDENT_STATUS
-------------------- ------------- --------------------

         3
Marina
Active                           0
```

Write a PL/SQL Procedure to automatically log the creation of new database users into the DBUserCreationLog table, capturing the username, the user who created them, and the timestamp:

```
SQL> CREATE TABLE DBUserCreationLog (
  2      log_id NUMBER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
  3      new_username VARCHAR2(50),
  4      created_by VARCHAR2(50),
  5      created_at TIMESTAMP DEFAULT SYSTIMESTAMP
  6  );

Table created.
```

```
SQL> CREATE OR REPLACE PROCEDURE LogNewUser(
  2      p_new_username IN VARCHAR2
  3  )
  4  IS
  5      v_created_by VARCHAR2(50);
  6  BEGIN
  7
  8      v_created_by := SYS_CONTEXT('USERENV', 'SESSION_USER');
  9
 10
 11      INSERT INTO DBUserCreationLog(new_username, created_by)
 12      VALUES (p_new_username, v_created_by);
 13
 14      COMMIT;
 15  END;
 16  /

Procedure created.
```

To test this Procedure:

```
SQL> CREATE USER test_user IDENTIFIED BY password123;

User created.

SQL> EXEC LogNewUser('TEST_USER');

PL/SQL procedure successfully completed.
```

Task 2: Exam Eligibility Validation:

```
CREATE OR REPLACE TRIGGER CheckPrerequisite
BEFORE INSERT ON Register
FOR EACH ROW
DECLARE
    v_prereq NUMBER;
    v_count NUMBER;
    v_status VARCHAR2(20);
BEGIN
    SELECT academic_status INTO v_status FROM Students WHERE id = :NEW.student_id;

    IF v_status = 'Suspended' THEN
        RAISE_APPLICATION_ERROR(-20003, 'Registration denied: student is suspended');
    END IF;

    SELECT prerequisite INTO v_prereq FROM Courses WHERE id = :NEW.course_id;

    IF v_prereq IS NOT NULL THEN
        SELECT COUNT(*) INTO v_count FROM CompletedCourses
        WHERE StudentID = :NEW.student_id AND CourseID = v_prereq;

        IF v_count = 0 THEN
            RAISE_APPLICATION_ERROR(-20001, 'Registration denied: prerequisite course not compl
        END IF;
```

Task 3:

Grade Calculation Function - Create a PL/SQL function that calculates the grade for a student based on their exam performance. The function should take the ExamResults ID as input and compute the grade based on predefined ranges (e.g., 90-100 = A, 80-89 = B). The function updates the grade column in the ExamResults table and returns the grade as output.

```
SQL> CREATE OR REPLACE FUNCTION CalculateGrade(p_registration_id IN NUMBER, p_grade NUMBER)
  2  RETURN VARCHAR2
  3  IS
  4      v_grade VARCHAR2(2);
  5      v_status VARCHAR2(10);
  6  BEGIN
  7
  8      IF p_grade >= 90 THEN
  9          v_grade := 'A';
 10          v_status := 'Pass';
 11      ELSIF p_grade >= 80 THEN
 12          v_grade := 'B';
 13          v_status := 'Pass';
 14      ELSIF p_grade >= 70 THEN
 15          v_grade := 'C';
 16          v_status := 'Pass';
 17      ELSIF p_grade >= 60 THEN
 18          v_grade := 'D';
 19          v_status := 'Pass';
 20      ELSE
 21          v_grade := 'F';
 22          v_status := 'Fail';
 23      END IF;
 24
 25
 26      UPDATE ExamResults
 27      SET grade = v_grade,
 28          status = v_status
 29      WHERE registration_id = p_registration_id;
 30
```

```
25
26      UPDATE ExamResults
27      SET grade = v_grade,
28          status = v_status
29      WHERE registration_id = p_registration_id;
30
31      COMMIT;
32
33      RETURN v_grade;
34  END;
35  /

Function created.

SQL>
```

```
SQL> DECLARE
  2      v_result VARCHAR2(2);
  3  BEGIN
  4      v_result := CalculateGrade(1, 85);
  5      DBMS_OUTPUT.PUT_LINE('Assigned grade: ' || v_result);
  6  END;
  7  /

PL/SQL procedure successfully completed.
```

Task 4:

Automated Warning Issuance - Develop a PL/SQL procedure to issue warnings automatically for students who have received a failing grade (status = 'Fail') in two or more courses. The procedure should check the Exam Results table and log warnings in the Warnings table with details.

```
SQL> CREATE SEQUENCE warnings_seq
  2  START WITH 1
  3  INCREMENT BY 1
  4  NOCACHE
  5  NOCYCLE;
```

```
SQL> CREATE OR REPLACE PROCEDURE issue_automatic_warnings
  2  IS
  3      v_warnings_issued NUMBER := 0;
  4  BEGIN
  5
  6      DBMS_OUTPUT.PUT_LINE('Automated Warning Issuance Started');
  7
  8      INSERT INTO Warnings(id, student_id, warning_reason, warning_date)
  9      SELECT warnings_seq.NEXTVAL, f.student_id, 'Multiple Course Failures', SYSDATE
 10      FROM (
 11
 12          SELECT r.student_id, COUNT(*) AS fail_count
 13          FROM ExamResults er
 14          JOIN Register r ON er.registration_id = r.id
 15          WHERE er.status = 'Fail'
 16          GROUP BY r.student_id
 17          HAVING COUNT(*) >= 2
 18      ) f
 19      WHERE NOT EXISTS (
 20
 21          SELECT 1
 22          FROM Warnings w
 23          WHERE w.student_id = f.student_id
 24            AND w.warning_reason = 'Multiple Course Failures'
 25            AND w.warning_date >= SYSDATE - 30
 26      );
 27
 28
 29      v_warnings_issued := SQL%ROWCOUNT;
```

```
21          SELECT 1
22          FROM Warnings w
23          WHERE w.student_id = f.student_id
24            AND w.warning_reason = 'Multiple Course Failures'
25            AND w.warning_date >= SYSDATE - 30
26      );
27
28
29      v_warnings_issued := SQL%ROWCOUNT;
30
31      COMMIT;
32
33      DBMS_OUTPUT.PUT_LINE('Total Warnings Issued: ' || v_warnings_issued);
34      DBMS_OUTPUT.PUT_LINE('Procedure Completed Successfully');
35
36  EXCEPTION
37      WHEN OTHERS THEN
38          ROLLBACK;
39          DBMS_OUTPUT.PUT_LINE('ERROR: ' || SQLERRM);
40          RAISE_APPLICATION_ERROR(-20010,
41              'Error issuing warnings: ' || SQLERRM);
42  END issue_automatic_warnings;
43  /

Procedure created.
```

```
SQL> SELECT object_name, status
  2  FROM user_objects
  3  WHERE object_type = 'PROCEDURE'
  4  AND object_name = 'ISSUE_AUTOMATIC_WARNINGS';

OBJECT_NAME
--------------------------------------------------------------------------------
STATUS
-------
ISSUE_AUTOMATIC_WARNINGS
VALID
```

```
SQL> BEGIN
  2      issue_automatic_warnings;
  3  END;
  4  /

PL/SQL procedure successfully completed.
```

```
SQL> SELECT object_name, status
  2  FROM user_objects
  3  WHERE object_type = 'PROCEDURE'
  4  AND object_name = 'ISSUE_AUTOMATIC_WARNINGS';

OBJECT_NAME
--------------------------------------------------------------------------------
STATUS
-------
ISSUE_AUTOMATIC_WARNINGS
VALID
```

Task 5:

```
CREATE OR REPLACE TRIGGER trg_register_audit_insert
BEFORE INSERT ON Register
FOR EACH ROW
DECLARE
    v_new_data CLOB;
BEGIN
    v_new_data := '{"student_id":' || :NEW.student_id || ',"course_id":' || :NEW.course_id || '
    INSERT INTO AuditTrail (table_name, operation, old_data, new_data, timestamp)
    VALUES ('Register', 'INSERT', NULL, v_new_data, SYSTIMESTAMP);
END;
/

CREATE OR REPLACE TRIGGER trg_register_audit_delete
BEFORE DELETE ON Register
FOR EACH ROW
DECLARE
    v_old_data CLOB;
BEGIN
    v_old_data := '{"student_id":' || :OLD.student_id || ',"course_id":' || :OLD.course_id || '
    INSERT INTO AuditTrail (table_name, operation, old_data, new_data, timestamp)
    VALUES ('Register', 'DELETE', v_old_data, NULL, SYSTIMESTAMP);
END;
/
```

Task 6: -

6. Course Performance Report - Write a PL/SQL cursor that generates a performance report for a specific course. The cursor should retrieve information from the ExamResults and Register tables, including student IDs, grades, and overall pass/fail statistics for the course. The report should summarize the number of students who passed and failed the course.

```
SQL> SET SERVEROUTPUT ON;
SQL> DECLARE
  2      v_course_id NUMBER := 101;
  3      v_pass_count NUMBER := 0;
  4      v_fail_count NUMBER := 0;
  5
  6      CURSOR c_course_perf IS
  7          SELECT r.student_id, er.grade, er.status
  8          FROM ExamResults er
  9          JOIN Register r ON er.registration_id = r.id
 10          WHERE r.course_id = v_course_id;
 11
 12  BEGIN
 13      DBMS_OUTPUT.PUT_LINE('Course Performance Report for Course ID: ' || v_course_id);
 14      DBMS_OUTPUT.PUT_LINE('-----------------------------------------');
 15
 16      FOR rec IN c_course_perf LOOP
 17          DBMS_OUTPUT.PUT_LINE('Student ID: ' || rec.student_id ||
 18                               ' | Grade: ' || rec.grade ||
 19                               ' | Status: ' || rec.status);
 20
 21          IF rec.status = 'Pass' THEN
 22              v_pass_count := v_pass_count + 1;
 23          ELSE
 24              v_fail_count := v_fail_count + 1;
 25          END IF;
 26      END LOOP;
 27
```

```
 16      FOR rec IN c_course_perf LOOP
 17          DBMS_OUTPUT.PUT_LINE('Student ID: ' || rec.student_id ||
 18                               ' | Grade: ' || rec.grade ||
 19                               ' | Status: ' || rec.status);
 20
 21          IF rec.status = 'Pass' THEN
 22              v_pass_count := v_pass_count + 1;
 23          ELSE
 24              v_fail_count := v_fail_count + 1;
 25          END IF;
 26      END LOOP;
 27
 28      DBMS_OUTPUT.PUT_LINE('-----------------------------------------');
 29      DBMS_OUTPUT.PUT_LINE('Total Passed: ' || v_pass_count);
 30      DBMS_OUTPUT.PUT_LINE('Total Failed: ' || v_fail_count);
 31
 32  END;
 33  /
Course Performance Report for Course ID: 101
-----------------------------------------
-----------------------------------------
Total Passed: 0
Total Failed: 0
```

```
Course Performance Report for Course ID: 101
-----------------------------------------
-----------------------------------------
Total Passed: 0
Total Failed: 0

PL/SQL procedure successfully completed.
```

---

7. Exam Schedule Management - Create a PL/SQL block to retrieve and display the schedule of exams for a specific course. The block should query the Exams

table and display the course name, exam date, and type (e.g., midterm or final). If no exams are scheduled, the block should display an appropriate message.

```
SQL> SET SERVEROUTPUT ON;
SQL> DECLARE
  2      v_course_id NUMBER := 101;
  3      v_count NUMBER := 0;
  4  BEGIN
  5      FOR rec IN (
  6          SELECT c.name AS course_name, e.exam_date, e.exam_type
  7          FROM Exams e
  8          JOIN Courses c ON e.course_id = c.id
  9          WHERE c.id = v_course_id
 10      ) LOOP
 11          DBMS_OUTPUT.PUT_LINE('Course: ' || rec.course_name ||
 12                               ' | Exam Date: ' || TO_CHAR(rec.exam_date, 'DD-MON-YYYY') ||
 13                               ' | Type: ' || rec.exam_type);
 14          v_count := v_count + 1;
 15      END LOOP;
 16
 17      IF v_count = 0 THEN
 18          DBMS_OUTPUT.PUT_LINE('No exams scheduled for Course ID: ' || v_course_id);
 19      END IF;
 20  END;
 21  /
No exams scheduled for Course ID: 101

PL/SQL procedure successfully completed.
```

```
SQL> SET SERVEROUTPUT ON;
SQL> DECLARE
  2      TYPE t_reg_ids IS TABLE OF NUMBER;
  3      TYPE t_grades IS TABLE OF VARCHAR2(2);
  4
  5      v_reg_ids t_reg_ids := t_reg_ids(1, 2, 3);
  6      v_new_grades t_grades := t_grades('A', 'B', 'C');
  7
  8  BEGIN
  9      FOR i IN 1 .. v_reg_ids.COUNT LOOP
 10          UPDATE ExamResults
 11          SET grade = v_new_grades(i)
 12          WHERE registration_id = v_reg_ids(i);
 13
 14          DBMS_OUTPUT.PUT_LINE('Updated registration_id: ' || v_reg_ids(i) ||
 15                               ' to grade: ' || v_new_grades(i));
 16      END LOOP;
 17
 18      COMMIT;
 19      DBMS_OUTPUT.PUT_LINE('All grade updates committed successfully.');
 20
 21  EXCEPTION
 22      WHEN OTHERS THEN
 23          ROLLBACK;
 24          DBMS_OUTPUT.PUT_LINE('Error occurred: ' || SQLERRM);
 25          DBMS_OUTPUT.PUT_LINE('All changes rolled back.');
 26  END;
 27  /
Updated registration_id: 1 to grade: A
Updated registration_id: 2 to grade: B
```

```
 26  END;
 27  /
Updated registration_id: 1 to grade: A
Updated registration_id: 2 to grade: B
Updated registration_id: 3 to grade: C
All grade updates committed successfully.

PL/SQL procedure successfully completed.

SQL>
```

Task8:

8. Multi-Exam Grade Update with Transactions - Develop a PL/SQL block that processes grade updates for multiple exams in a single transaction. The block should update the ExamResults table for a list of registration_ids with new grades. If any error occurs, the block should roll back all changes to ensure data consistency.

```
SQL> SET SERVEROUTPUT ON;
SQL> DECLARE
  2      TYPE t_reg_ids IS TABLE OF NUMBER;
  3      TYPE t_grades IS TABLE OF VARCHAR2(2);
  4
  5      v_reg_ids t_reg_ids := t_reg_ids(1, 2, 3);
  6      v_new_grades t_grades := t_grades('A', 'B', 'C');
  7
  8  BEGIN
  9      FOR i IN 1 .. v_reg_ids.COUNT LOOP
 10          UPDATE ExamResults
 11          SET grade = v_new_grades(i)
 12          WHERE registration_id = v_reg_ids(i);
 13
 14          DBMS_OUTPUT.PUT_LINE('Updated registration_id: ' || v_reg_ids(i) ||
 15                               ' to grade: ' || v_new_grades(i));
 16      END LOOP;
 17
 18      COMMIT;
 19      DBMS_OUTPUT.PUT_LINE('All grade updates committed successfully.');
 20
 21  EXCEPTION
 22      WHEN OTHERS THEN
 23          ROLLBACK;
 24          DBMS_OUTPUT.PUT_LINE('Error occurred: ' || SQLERRM);
 25          DBMS_OUTPUT.PUT_LINE('All changes rolled back.');
 26  END;
 27  /
Updated registration_id: 1 to grade: A
Updated registration_id: 2 to grade: B
```

```
 26  END;
 27  /
Updated registration_id: 1 to grade: A
Updated registration_id: 2 to grade: B
Updated registration_id: 3 to grade: C
All grade updates committed successfully.

PL/SQL procedure successfully completed.

SQL>
```

Task 9:

9. Student Suspension Based on Warnings - Create a PL/SQL procedure to suspend students who have received three or more warnings. The procedure should query the Warnings table to identify affected students and update their

academic_status in the Students table to "Suspended." Ensure the procedure logs these updates into the AuditTrail table.

```
SQL> CREATE SEQUENCE audit_seq
  2  START WITH 1
  3  INCREMENT BY 1
  4  NOCACHE
  5  NOCYCLE;
```

```
SQL> CREATE OR REPLACE PROCEDURE suspend_students_based_on_warnings
  2  IS
  3      CURSOR c_students_to_suspend IS
  4          SELECT s.id AS student_id, s.academic_status AS old_status
  5          FROM Students s
  6          JOIN (
  7              SELECT student_id
  8              FROM Warnings
  9              GROUP BY student_id
 10              HAVING COUNT(*) >= 3
 11          ) w ON s.id = w.student_id
 12          WHERE s.academic_status != 'Suspended';
 13
 14  BEGIN
 15      FOR rec IN c_students_to_suspend LOOP
 16          UPDATE Students
 17          SET academic_status = 'Suspended'
 18          WHERE id = rec.student_id;
 19
 20          INSERT INTO AuditTrail(id, table_name, operation, old_data, new_data, timestamp)
 21          VALUES (
 22              audit_seq.NEXTVAL,
 23              'Students',
 24              'UPDATE',
 25              'academic_status=' || rec.old_status,
 26              'academic_status=Suspended',
 27              SYSDATE
 28          );
```

```
17          SET academic_status = 'Suspended'
18          WHERE id = rec.student_id;
19
20          INSERT INTO AuditTrail(id, table_name, operation, old_data, new_data, timestamp)
21          VALUES (
22              audit_seq.NEXTVAL,
23              'Students',
24              'UPDATE',
25              'academic_status=' || rec.old_status,
26              'academic_status=Suspended',
27              SYSDATE
28          );
29
30          DBMS_OUTPUT.PUT_LINE('Student ID ' || rec.student_id || ' suspended.');
31      END LOOP;
32
33      COMMIT;
34      DBMS_OUTPUT.PUT_LINE('All eligible students have been suspended.');
35
36  EXCEPTION
37      WHEN OTHERS THEN
38          ROLLBACK;
39          DBMS_OUTPUT.PUT_LINE('Error: ' || SQLERRM);
40          RAISE_APPLICATION_ERROR(-20020, 'Error suspending students: ' || SQLERRM);
41  END;
42  /

rocedure created.
```

Task 10:

10. Advanced Grade Management and Data Integrity: - Write a PL/SQL Function that accepts a student's ID and calculates their current Grade Point Average (GPA) based on the grades in the ExamResults table and the credit hours from the Courses table. The function should return the calculated GPA.

```
SQL> CREATE OR REPLACE FUNCTION calculate_gpa(p_student_id NUMBER)
  2  RETURN NUMBER
  3  IS
  4      v_total_points NUMBER := 0;
  5      v_total_credits NUMBER := 0;
  6      v_gpa NUMBER := 0;
  7
  8      CURSOR c_grades IS
  9          SELECT er.grade, c.credit_hours
 10          FROM ExamResults er
 11          JOIN Register r ON er.registration_id = r.id
 12          JOIN Courses c ON r.course_id = c.id
 13          WHERE r.student_id = p_student_id;
 14
 15      FUNCTION grade_to_points(p_grade VARCHAR2) RETURN NUMBER IS
 16      BEGIN
 17          IF p_grade = 'A' THEN
 18              RETURN 4;
 19          ELSIF p_grade = 'B' THEN
 20              RETURN 3;
 21          ELSIF p_grade = 'C' THEN
 22              RETURN 2;
 23          ELSIF p_grade = 'D' THEN
 24              RETURN 1;
 25          ELSE
 26              RETURN 0;
 27          END IF;
 28      END;
 29
 30  BEGIN
```

```
 19          ELSIF p_grade = 'B' THEN
 20              RETURN 3;
 21          ELSIF p_grade = 'C' THEN
 22              RETURN 2;
 23          ELSIF p_grade = 'D' THEN
 24              RETURN 1;
 25          ELSE
 26              RETURN 0;
 27          END IF;
 28      END;
 29
 30  BEGIN
 31      FOR rec IN c_grades LOOP
 32          v_total_points := v_total_points + grade_to_points(rec.grade) * rec.credit_hours;
 33          v_total_credits := v_total_credits + rec.credit_hours;
 34      END LOOP;
 35
 36      IF v_total_credits > 0 THEN
 37          v_gpa := v_total_points / v_total_credits;
 38      ELSE
 39          v_gpa := 0;
 40      END IF;
 41
 42      RETURN ROUND(v_gpa, 2);
 43  END;
 44  /

Function created.
```

```
SQL> SET SERVEROUTPUT ON;
SQL>
SQL> DECLARE
  2      v_gpa NUMBER;
  3  BEGIN
  4      v_gpa := calculate_gpa(1);
  5      DBMS_OUTPUT.PUT_LINE('GPA for Student ID 1: ' || v_gpa);
  6  END;
  7  /
GPA for Student ID 1: 0

PL/SQL procedure successfully completed.
```

Task 10 part B

```sql
CREATE OR REPLACE TRIGGER trg_authorize_grade_update
BEFORE UPDATE OF grade ON ExamResults
FOR EACH ROW
DECLARE
    v_user VARCHAR2(50);
    v_calling_proc VARCHAR2(100);
BEGIN
    v_user := SYS_CONTEXT('USERENV', 'SESSION_USER');

    BEGIN
        SELECT object_name INTO v_calling_proc
        FROM user_procedures
        WHERE object_name = 'CALCULATEGRADE';
    EXCEPTION
        WHEN NO_DATA_FOUND THEN
            v_calling_proc := NULL;
    END;

    IF v_user NOT IN ('ADMIN', 'SYS', 'SYSTEM', 'MANGER', 'MANAGER')
       AND v_user NOT LIKE 'PROFESSOR%' THEN
        RAISE_APPLICATION_ERROR(-20040, 'Unauthorized user cannot modify exam grades');
    END IF;
END;
/
```

Task 11:

```
PROMPT BLOCKER-WAITING DEMONSTRATION
PROMPT ================================================
PROMPT;

PROMPT Creating blocker_session.sql...
SPOOL blocker_session.sql
SELECT 'SET SERVEROUTPUT ON;' FROM DUAL;
SELECT 'PROMPT ===== BLOCKER SESSION STARTED =====' FROM DUAL;
SELECT 'UPDATE Students SET academic_status = ''OnHold'' WHERE id = 2;' FROM DUAL;
SELECT 'PROMPT Row LOCKED for Student ID 2' FROM DUAL;
SELECT 'PROMPT;' FROM DUAL;
SELECT 'SELECT ''Blocker Session Info:'' as Info, sid, serial#, username, status FROM v$session
SELECT 'PROMPT;' FROM DUAL;
SELECT 'PROMPT Keeping lock for 30 seconds...' FROM DUAL;
SELECT 'EXEC DBMS_LOCK.SLEEP(30);' FROM DUAL;
SELECT 'COMMIT;' FROM DUAL;
SELECT 'PROMPT BLOCKER COMMITTED - Lock Released' FROM DUAL;
SPOOL OFF
```

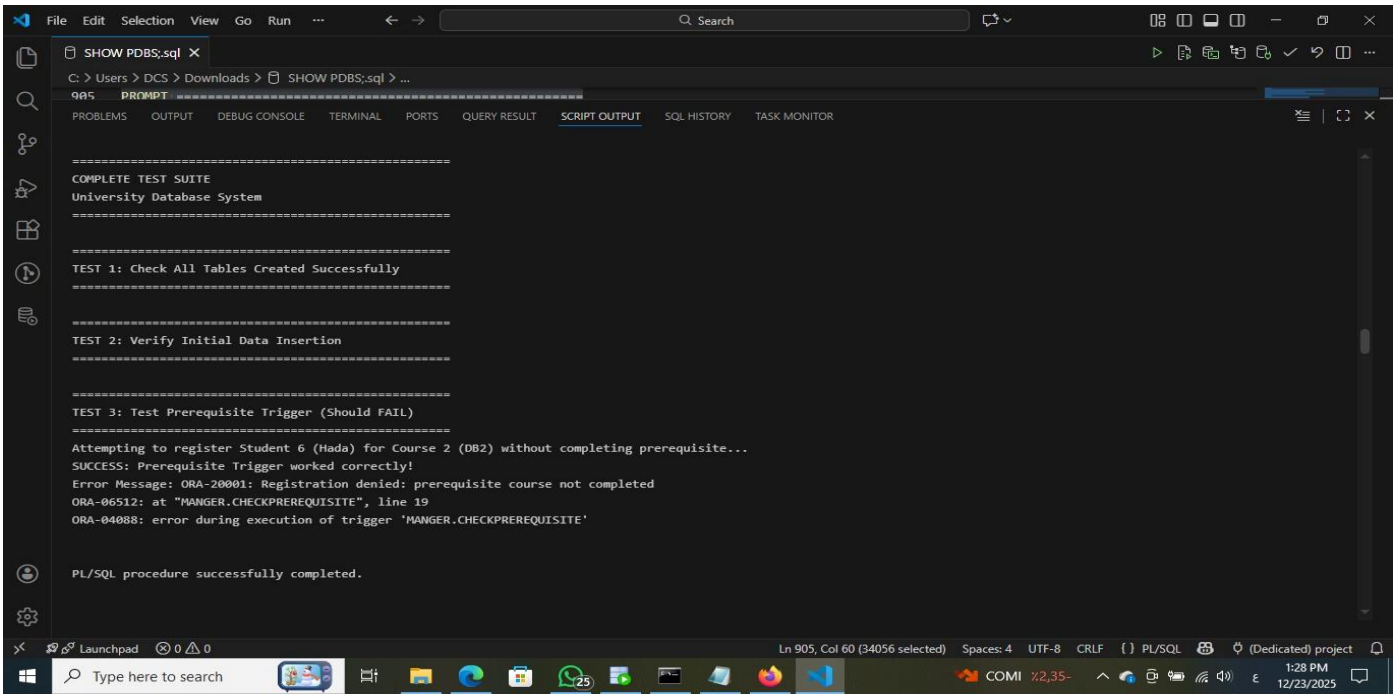
```
PROMPT Creating waiting_session.sql...
SPOOL waiting_session.sql
SELECT 'SET SERVEROUTPUT ON;' FROM DUAL;
SELECT 'PROMPT ===== WAITING SESSION STARTED =====' FROM DUAL;
SELECT 'PROMPT Attempting to update Student ID 2...' FROM DUAL;
SELECT 'PROMPT This will WAIT for blocker session to release lock' FROM DUAL;
SELECT 'UPDATE Students SET name = ''Updated_Name'' WHERE id = 2;' FROM DUAL;
SELECT 'PROMPT Update completed after lock released' FROM DUAL;
SELECT 'COMMIT;' FROM DUAL;
SELECT 'PROMPT WAITING SESSION COMPLETED' FROM DUAL;
SPOOL OFF

PROMPT Creating monitor_blocking.sql...
SPOOL monitor_blocking.sql
SELECT 'SET SERVEROUTPUT ON;' FROM DUAL;
SELECT 'SET LINESIZE 200;' FROM DUAL;
SELECT 'SET PAGESIZE 100;' FROM DUAL;
SELECT 'PROMPT ===== BLOCKING SESSIONS MONITOR =====' FROM DUAL;
SELECT 'PROMPT;' FROM DUAL;
SELECT 'PROMPT Current Locked Objects:' FROM DUAL;
SELECT 'SELECT s.sid, s.serial#, s.username, s.status, o.object_name, DECODE(l.locked_mode, 0,
SELECT 'PROMPT;' FROM DUAL;
SELECT 'PROMPT Sessions with Blocking:' FROM DUAL;
SELECT 'SELECT s.sid AS session_id, s.serial#, s.username, s.status, s.blocking_session, s.even
```

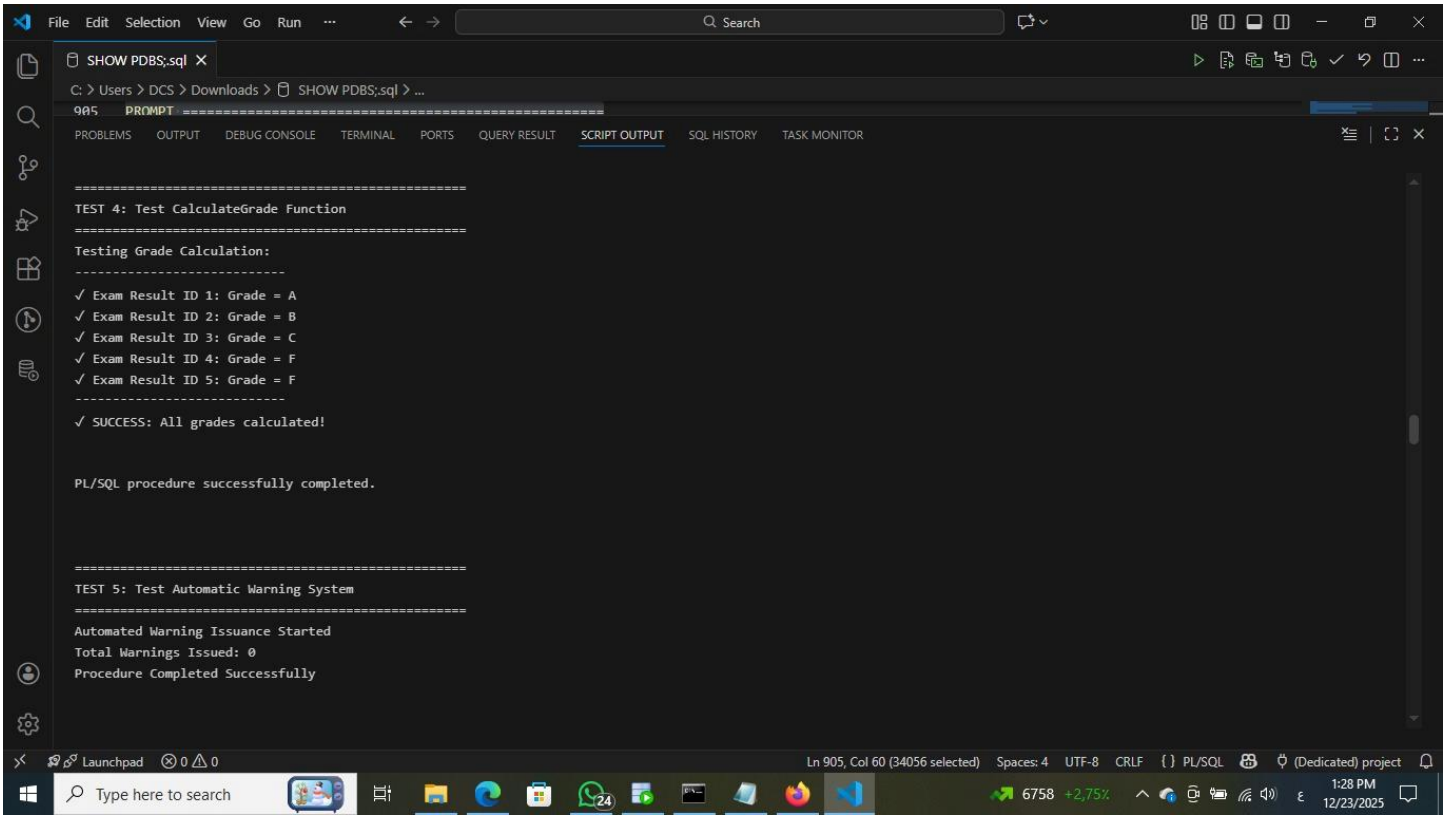
Task 12:

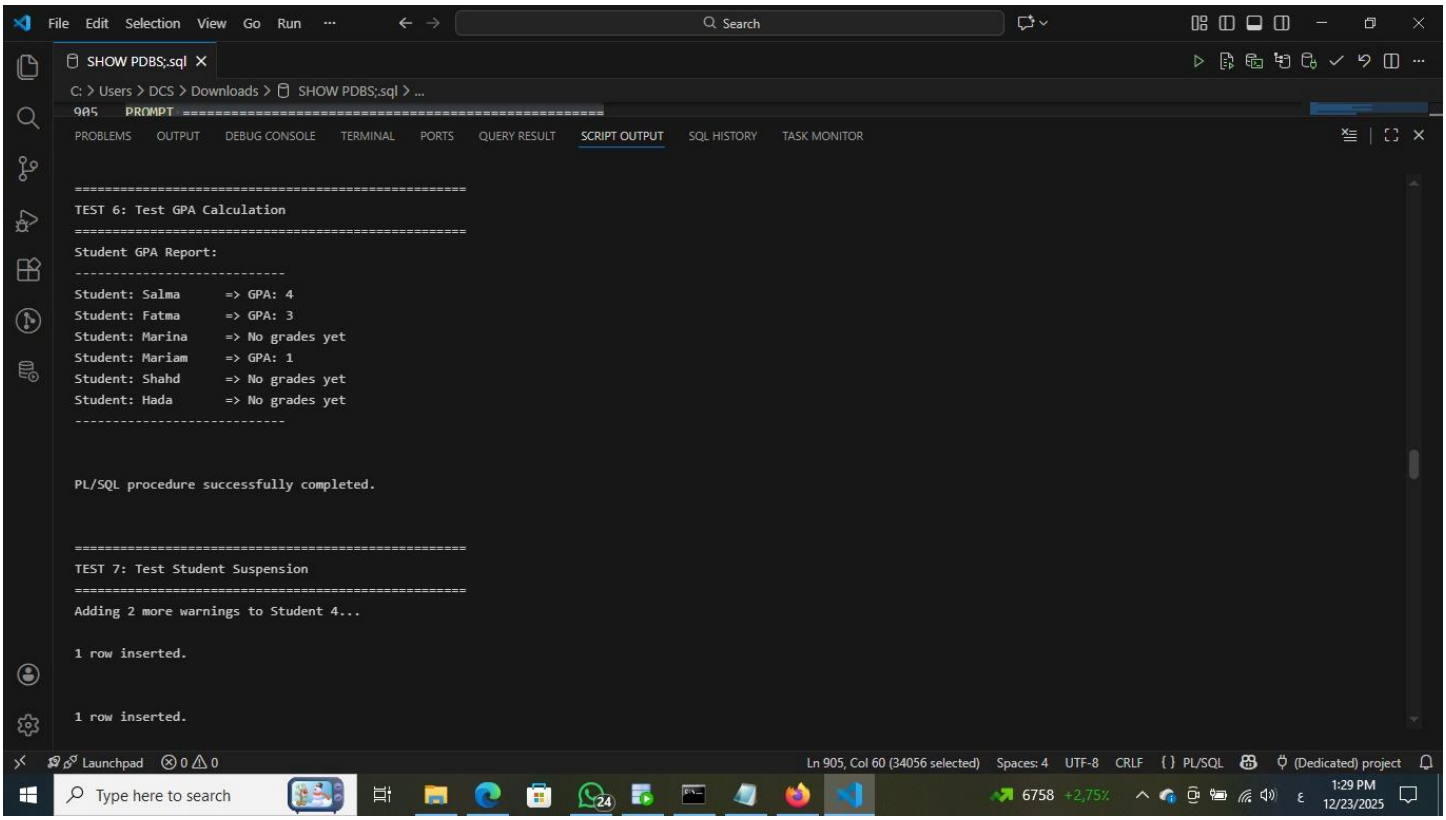
```
SELECT 'SET SERVEROUTPUT ON;' FROM DUAL;
SELECT 'SET LINESIZE 200;' FROM DUAL;
SELECT 'SET PAGESIZE 100;' FROM DUAL;
SELECT 'PROMPT ===== BLOCKING SESSIONS MONITOR =====' FROM DUAL;
SELECT 'PROMPT;' FROM DUAL;
SELECT 'PROMPT Current Locked Objects:' FROM DUAL;
SELECT 'SELECT s.sid, s.serial#, s.username, s.status, o.object_name, DECODE(l.locked_mode, 0,
SELECT 'PROMPT;' FROM DUAL;
SELECT 'PROMPT Sessions with Blocking:' FROM DUAL;
SELECT 'SELECT s.sid AS session_id, s.serial#, s.username, s.status, s.blocking_session, s.even
SELECT 'PROMPT;' FROM DUAL;
SELECT 'PROMPT Detailed Blocker-Waiter Relationship:' FROM DUAL;
SELECT 'SELECT DISTINCT s1.sid || '',' || s1.serial# AS blocker_session, s1.username AS blocke
SPOOL OFF

PROMPT;
PROMPT ======================================================
PROMPT EXECUTION INSTRUCTIONS:
PROMPT ======================================================
PROMPT;
PROMPT 1. Open THREE separate SQL*Plus windows
PROMPT 2. In Window 1: Run @blocker_session.sql
PROMPT 3. In Window 2: Wait 2 seconds then run @waiting_session.sql
PROMPT 4. In Window 3: Run @monitor_blocking.sql multiple times
PROMPT;
```

```
PROMPT To view all current locks in database:
SELECT
    s.sid,
    s.serial#,
    s.username,
    o.object_name,
    DECODE(l.locked_mode,
        0, 'None',
        1, 'Null',
        2, 'Row-S',
        3, 'Row-X',
        4, 'Share',
        5, 'S/Row-X',
        6, 'Exclusive',
        'Unknown') AS lock_mode
FROM v$locked_object l
JOIN v$session s ON l.session_id = s.sid
JOIN dba_objects o ON l.object_id = o.object_id;

PROMPT;
PROMPT ======================================================
PROMPT ALL TESTS COMPLETED SUCCESSFULLY!
PROMPT ======================================================
PROMPT;
PROMPT Database is ready for production use!
```

Some test cases:-

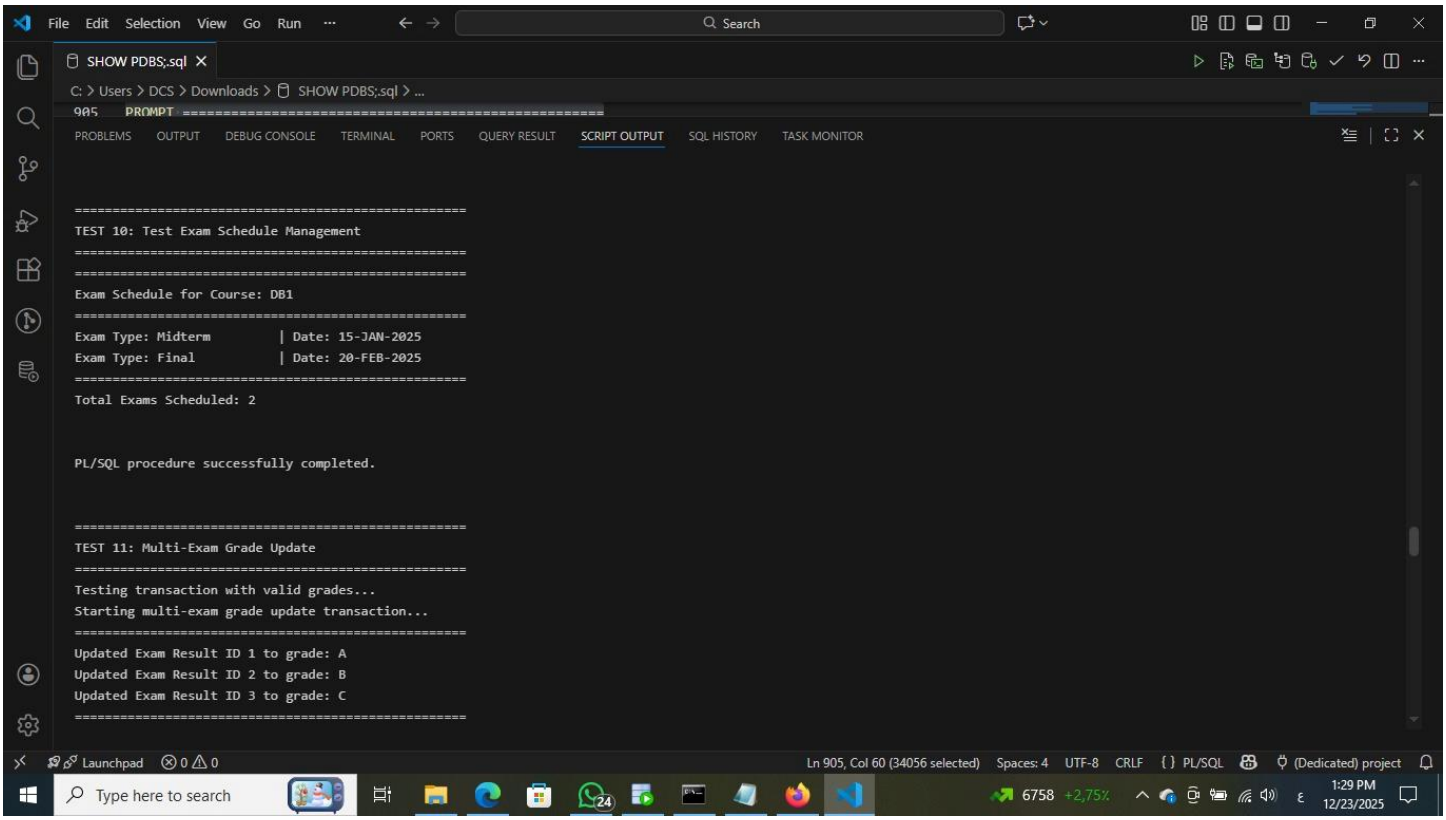


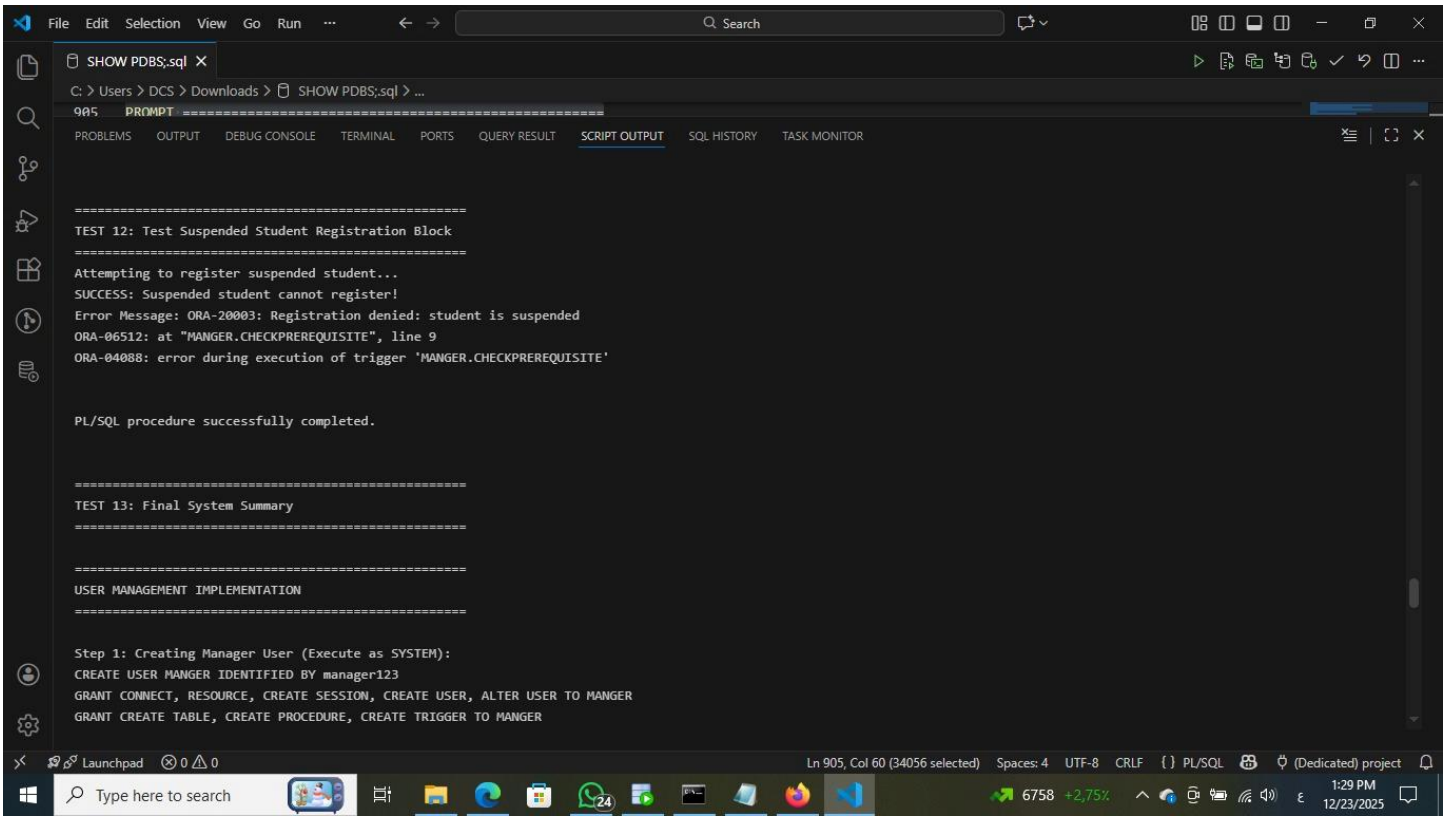


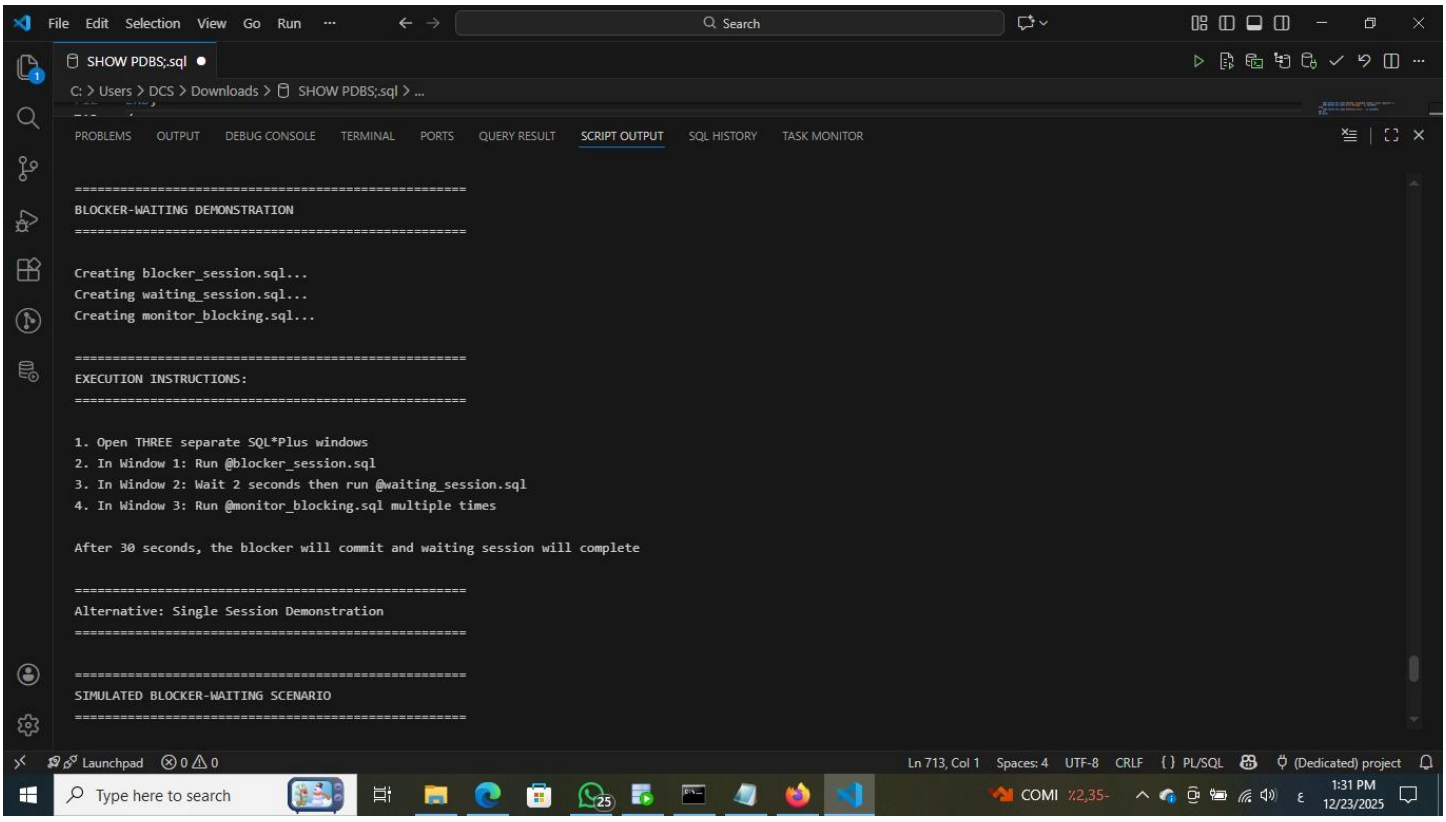

File Edit Selection View Go Run ···
Search
SHOW PDBS;.sql
C: > Users > DCS > Downloads > SHOW PDBS;.sql > ...
905 PROMPT ==========
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS QUERY RESULT SCRIPT OUTPUT SQL HISTORY TASK MONITOR
==========
TEST 6: Test GPA Calculation
==========
Student GPA Report:

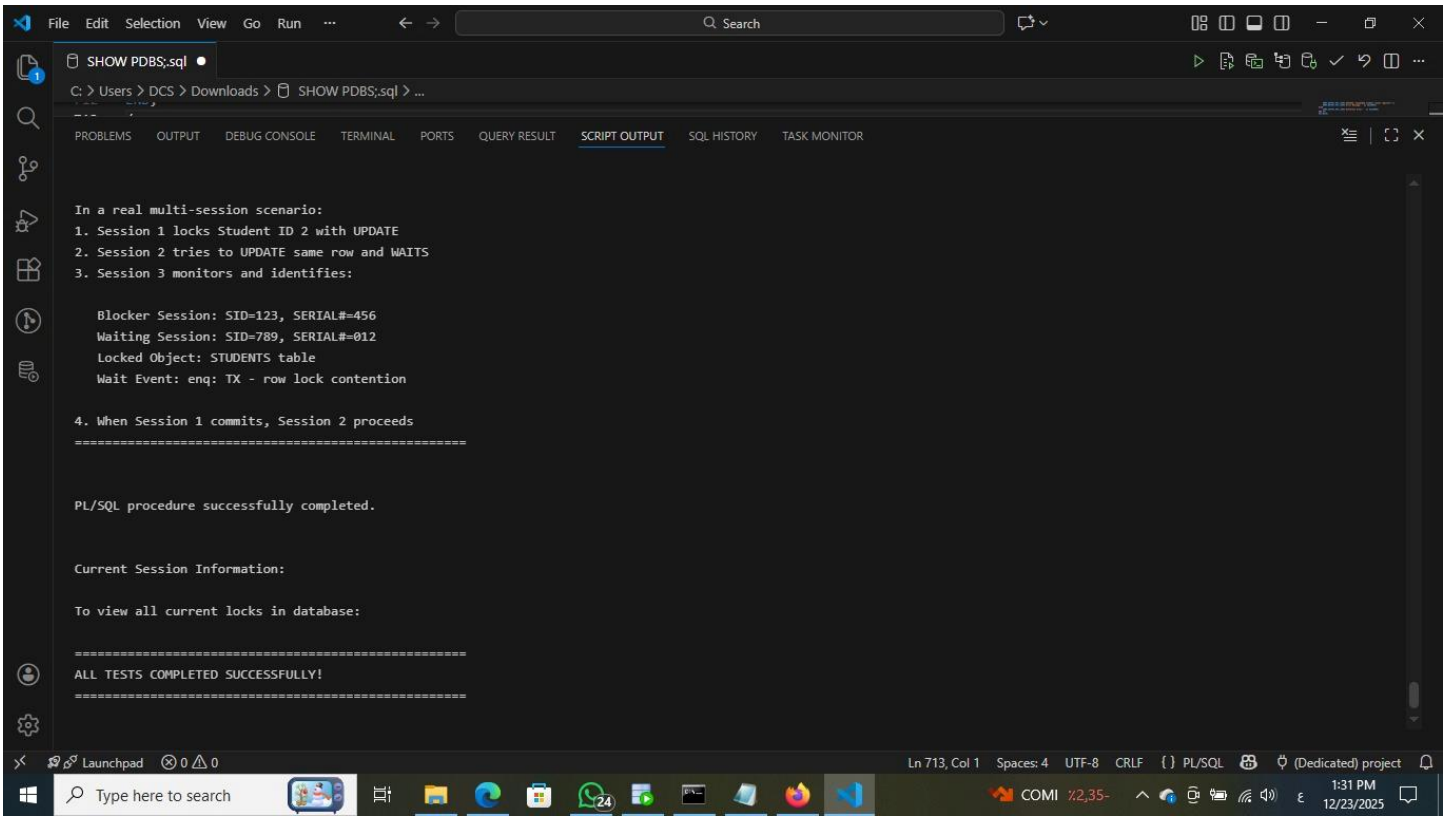
Student: Salma => GPA: 4
Student: Fatma => GPA: 3
Student: Marina => No grades yet
Student: Mariam => GPA: 1
Student: Shahd => No grades yet
Student: Hada => No grades yet

PL/SQL procedure successfully completed.
==========
TEST 7: Test Student Suspension
==========
Adding 2 more warnings to Student 4...
1 row inserted.
1 row inserted.
Launchpad
Ln 905, Col 60 (34056 selected) Spaces: 4 UTF-8 CRLF { } PL/SQL (Dedicated) project
Type here to search
6758 +2,75%
1:29 PM
12/23/2025


File Edit Selection View Go Run ···
Search
SHOW PDBS;.sql
C: > Users > DCS > Downloads > SHOW PDBS;.sql > ...
905 PROMPT ==========
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS QUERY RESULT SCRIPT OUTPUT SQL HISTORY TASK MONITOR
==========
TEST 10: Test Exam Schedule Management
==========
==========
Exam Schedule for Course: DB1
==========
Exam Type: Midterm | Date: 15-JAN-2025
Exam Type: Final | Date: 20-FEB-2025
==========
Total Exams Scheduled: 2
PL/SQL procedure successfully completed.
==========
TEST 11: Multi-Exam Grade Update
==========
Testing transaction with valid grades...
Starting multi-exam grade update transaction...
==========
Updated Exam Result ID 1 to grade: A
Updated Exam Result ID 2 to grade: B
Updated Exam Result ID 3 to grade: C
==========
Launchpad
Ln 905, Col 60 (34056 selected) Spaces: 4 UTF-8 CRLF { } PL/SQL (Dedicated) project
Type here to search
6758 +2,75%
1:29 PM
12/23/2025


File Edit Selection View Go Run
Search
SHOW PDBS;.sql
C: > Users > DCS > Downloads > SHOW PDBS;.sql > ...
905 PROMPT ==========
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS QUERY RESULT SCRIPT OUTPUT SQL HISTORY TASK MONITOR
==========
TEST 12: Test Suspended Student Registration Block
==========
Attempting to register suspended student...
SUCCESS: Suspended student cannot register!
Error Message: ORA-20003: Registration denied: student is suspended
ORA-06512: at "MANGER.CHECKPREREQUISITE", line 9
ORA-04088: error during execution of trigger 'MANGER.CHECKPREREQUISITE'
PL/SQL procedure successfully completed.
==========
TEST 13: Final System Summary
==========
==========
USER MANAGEMENT IMPLEMENTATION
==========
Step 1: Creating Manager User (Execute as SYSTEM):
CREATE USER MANGER IDENTIFIED BY manager123
GRANT CONNECT, RESOURCE, CREATE SESSION, CREATE USER, ALTER USER TO MANGER
GRANT CREATE TABLE, CREATE PROCEDURE, CREATE TRIGGER TO MANGER
Launchpad
Ln 905, Col 60 (34056 selected) Spaces: 4 UTF-8 CRLF PL/SQL (Dedicated) project
Type here to search
6758 +2,75%
1:29 PM
12/23/2025


File Edit Selection View Go Run
Search
SHOW PDBS;.sql
C: > Users > DCS > Downloads > SHOW PDBS;.sql > ...
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS QUERY RESULT SCRIPT OUTPUT SQL HISTORY TASK MONITOR
==========
BLOCKER-WAITING DEMONSTRATION
==========
Creating blocker_session.sql...
Creating waiting_session.sql...
Creating monitor_blocking.sql...
==========
EXECUTION INSTRUCTIONS:
==========
1. Open THREE separate SQL*Plus windows
2. In Window 1: Run @blocker_session.sql
3. In Window 2: Wait 2 seconds then run @waiting_session.sql
4. In Window 3: Run @monitor_blocking.sql multiple times
After 30 seconds, the blocker will commit and waiting session will complete
==========
Alternative: Single Session Demonstration
==========
==========
SIMULATED BLOCKER-WAITING SCENARIO
==========
Launchpad
Ln 713, Col 1 Spaces: 4 UTF-8 CRLF PL/SQL (Dedicated) project
Type here to search
COMI %2,35-
1:31 PM
12/23/2025


File Edit Selection View Go Run
Search
SHOW PDBS;.sql
C: > Users > DCS > Downloads > SHOW PDBS;.sql > ...
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS QUERY RESULT SCRIPT OUTPUT SQL HISTORY TASK MONITOR
In a real multi-session scenario:
1. Session 1 locks Student ID 2 with UPDATE
2. Session 2 tries to UPDATE same row and WAITS
3. Session 3 monitors and identifies:
Blocker Session: SID=123, SERIAL#=456
Waiting Session: SID=789, SERIAL#=012
Locked Object: STUDENTS table
Wait Event: enq: TX - row lock contention
4. When Session 1 commits, Session 2 proceeds
PL/SQL procedure successfully completed.
Current Session Information:
To view all current locks in database:
ALL TESTS COMPLETED SUCCESSFULLY!
Launchpad
Ln 713, Col 1 Spaces: 4 UTF-8 CRLF { } PL/SQL (Dedicated) project
Type here to search
COMI
1:31 PM
12/23/2025