SIMPLE PLSQL PROCEDURES

1. **Get All Students in a Specific Major**

```
CREATE PROCEDURE GetStudentsByMajor(IN majorName VARCHAR(100))
BEGIN
   SELECT StudentID, Name, Email
   FROM Student
   WHERE Major = majorName;
END
```

Usage:

```
CALL GetStudentsByMajor('Computer Science');
```

2. Get Enrollments for a Student

```
CREATE PROCEDURE GetStudentEnrollments(IN stuID INT)
BEGIN
   SELECT C.Title, E.Semester, E.Grade
   FROM Enrollment E
   JOIN Course C ON E.CourseID = C.CourseID
   WHERE E.StudentID = stuID;
END
```

Usage:

```
CALL GetStudentEnrollments(1001);
```

3. Insert a New Enrollment

```
CREATE PROCEDURE EnrollStudent(
   IN stuID INT,
   IN courseID INT,
   IN semester VARCHAR(50)
)
BEGIN
   INSERT INTO Enrollment (StudentID, CourseID, Semester)
   VALUES (stuID, courseID, semester);
```

```
END
```

Usage:

```
CALL EnrollStudent(1002, 201, 'Fall2025');
```

4. Update a Student's Grade

```
CREATE PROCEDURE UpdateGrade(
    IN stuID INT,
    IN courseID INT,
    IN newGrade CHAR(2)
)
BEGIN
    UPDATE Enrollment
    SET Grade = newGrade
    WHERE StudentID = stuID AND CourseID = courseID;
END
```

Usage:

```
CALL UpdateGrade(1002, 201, 'A');
```

PLSQL FUNCTIONS

1. **Function: Get Full Student Name by ID**

```
CREATE FUNCTION GetStudentName(stuID INT)
RETURNS VARCHAR(100)
DETERMINISTIC
BEGIN
   DECLARE stuName VARCHAR(100);
   SELECT Name INTO stuName FROM Student WHERE StudentID = stuID;
   RETURN stuName;
END
```

Usage:

```
SELECT GetStudentName(1001);
```

2. Function: Get Student Email by ID

```
CREATE FUNCTION GetStudentEmail(stuID INT)
RETURNS VARCHAR(100)
DETERMINISTIC
BEGIN
   DECLARE email VARCHAR(100);
   SELECT Email INTO email FROM Student WHERE StudentID = stuID;
   RETURN email;
END
```

Usage:

```
SELECT GetStudentEmail(1002);
```

3. Function: Get Total Enrollments in a Semester

```
CREATE FUNCTION TotalEnrollments(sem VARCHAR(50))
RETURNS INT
DETERMINISTIC
BEGIN
   DECLARE total INT;
   SELECT COUNT(*) INTO total FROM Enrollment WHERE Semester = sem;
   RETURN total;
```

```
END
```

Usage:

```
SELECT TotalEnrollments('Fall2024');
```

4. Function: Convert Grade to Grade Point

```
CREATE FUNCTION get_grade_point(p_grade CHAR(1))
RETURNS INT
DETERMINISTIC
BEGIN
  CASE p_grade
    WHEN 'A' THEN RETURN 4;
    WHEN 'B' THEN RETURN 3;
    WHEN 'C' THEN RETURN 2;
    WHEN 'D' THEN RETURN 1;
    WHEN 'F' THEN RETURN 0;
    ELSE RETURN NULL;
  END CASE;
END;
```

USAGE:

```
SELECT get_grade_point('A') AS GradePoint;
```

5. Function: Check If Instructor Is in a Department

```
CREATE FUNCTION is_instructor_in_dept(p_instructor_id INT, p_dept_id INT)
RETURNS VARCHAR(3)
DETERMINISTIC
BEGIN
  DECLARE v_exists INT;

  SELECT COUNT(*) INTO v_exists
  FROM Instructor
  WHERE InstructorID = p_instructor_id AND DeptID = p_dept_id;
```

```
  IF v_exists > 0 THEN
    RETURN 'YES';
  ELSE
    RETURN 'NO';
  END IF;
END;
```

USAGE:

```
SELECT is_instructor_in_dept(101, 1) AS Result;
```

6. Function: Get Department Name for Course

```
CREATE FUNCTION get_dept_name(p_course_id INT)
RETURNS VARCHAR(100)
DETERMINISTIC
BEGIN
  DECLARE v_dept_name VARCHAR(100);
  SELECT D.DeptName
  INTO v_dept_name
  FROM Course C
  JOIN Department D ON C.DeptID = D.DeptID
  WHERE C.CourseID = p_course_id;
  RETURN v_dept_name;
END;
```

USAGE:

```
SELECT get_dept_name(201) AS DepartmentName;
```

PLSQL TRIGGERS

1. **Trigger to Set Default Grade to 'N/A' if Not Provided**

```
DELIMITER //
CREATE TRIGGER before_enrollment_insert
BEFORE INSERT ON Enrollment
FOR EACH ROW
BEGIN
   IF NEW.Grade IS NULL THEN
      SET NEW.Grade = 'N/A';
   END IF;
END;
//
DELIMITER ;
```

Explanation: If the `Grade` is not provided when inserting into `Enrollment`, it sets it to `'N/A'`.

2. **Trigger to Uppercase Student Name Before Insertion**

```
DELIMITER //
CREATE TRIGGER before_student_insert
BEFORE INSERT ON Student
FOR EACH ROW
BEGIN
   SET NEW.Name = UPPER(NEW.Name);
END;
//
DELIMITER ;
```

Explanation: Automatically converts the student name to uppercase before storing it.

3. **Trigger to Track When a New Instructor is Added**

```
CREATE TABLE Instructor_Log (
   LogID INT AUTO_INCREMENT PRIMARY KEY,
   InstructorID INT,
   Name VARCHAR(100),
   LoggedAt DATETIME
```

```
);


DELIMITER //

CREATE TRIGGER after_instructor_insert

AFTER INSERT ON Instructor

FOR EACH ROW

BEGIN

   INSERT INTO Instructor_Log (InstructorID, Name, LoggedAt)

   VALUES (NEW.InstructorID, NEW.Name, NOW());

END;

//

DELIMITER ;
```

Explanation: Adds an entry to the `Instructor_Log` table every time a new instructor is added.

4. **Trigger to Prevent Enrolling in More Than 5 Courses**

```
DELIMITER //

CREATE TRIGGER before_enrollment_limit

BEFORE INSERT ON Enrollment

FOR EACH ROW

BEGIN

   DECLARE course_count INT;


   SELECT COUNT(*) INTO course_count

   FROM Enrollment

   WHERE StudentID = NEW.StudentID;


   IF course_count >= 5 THEN

      SIGNAL SQLSTATE '45000'

      SET MESSAGE_TEXT = 'Student cannot enroll in more than 5 courses';

   END IF;

END;

//

DELIMITER ;
```

Explanation: Prevents a student from enrolling in more than 5 courses.

5. **Trigger to Automatically Fill Semester if Not Provided**

```
DELIMITER //
CREATE TRIGGER before_enrollment_semester
BEFORE INSERT ON Enrollment
FOR EACH ROW
BEGIN
   IF NEW.Semester IS NULL OR NEW.Semester = '' THEN
      SET NEW.Semester = 'Fall2025';
   END IF;
END;
//
DELIMITER ;
```

Explanation: Assigns `'Fall2025'` as default semester if none is specified.

SOME ADVANCED QUERIES:

1. Trigger to Log Course Enrollment

```
CREATE TABLE Enrollment_Log (
   LogID INT AUTO_INCREMENT PRIMARY KEY,
   StudentID INT,
   CourseID INT,
   ActionType VARCHAR(10),
   ActionTime DATETIME
);

DELIMITER //
CREATE TRIGGER after_enrollment_insert
AFTER INSERT ON Enrollment
FOR EACH ROW
BEGIN
   INSERT INTO Enrollment_Log (StudentID, CourseID, ActionType, ActionTime)
```

```
    VALUES (NEW.StudentID, NEW.CourseID, 'INSERT', NOW());
END;
//
DELIMITER ;
```

Purpose: Logs whenever a student enrolls in a course.

5. Trigger to Block Duplicate Email in Instructor (Beyond Constraint)

```
DELIMITER //
CREATE TRIGGER before_instructor_insert
BEFORE INSERT ON Instructor
FOR EACH ROW
BEGIN
    DECLARE count_email INT;
    SELECT COUNT(*) INTO count_email FROM Instructor WHERE Email = NEW.Email;
    IF count_email > 0 THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Duplicate email not allowed in Instructor';
    END IF;
END;
//
DELIMITER ;
```

Purpose: Programmatically blocks duplicate emails with a custom message.