Lab Session 06

6. Introduction to PL/SQL

Aim:

The aim of this experiment is to familiarize students with the PL/SQL, programs, creating reusable procedures & functions and its capabilities for developing database-centric applications.

Description:

The lab experiment involves a procedural description where students gain hands-on experience in writing and executing PL/SQL programs, procedures, and functions. The lab begins with a brief introduction to the PL/SQL language, highlighting its role in developing database-centric applications. Students then proceed to practice implementing PL/SQL programs by writing code blocks that include variable declarations, control structures, and SQL statements. They learn how to create reusable procedures and functions to encapsulate specific logic or calculations. The lab provides students with sample scenarios and real-world problems, enabling them to apply PL/SQL programming concepts to solve database-related tasks. Through this procedural description, students enhance their understanding of PL/SQL programming, develop skills in writing efficient and maintainable code, and gain insights into the benefits of using PL/SQL for database development.

Pre Lab-Task:

1) What Compare SQL & PL/SQL

- **SQL:** Used for querying and manipulating data (SELECT, INSERT, UPDATE).
- **PL/SQL:** Procedural extension of SQL, supports loops, conditions, and functions.

2) How do you declare and initialize variables in a PL/SQL program or procedure in PostgreSQL?

plpgsql Copy Edit

```
DO $$
DECLARE v_name TEXT := 'John';
BEGIN
    RAISE NOTICE '%', v_name;
END $$;
```

3) Can a PL/SQL function return multiple values in PostgreSQL? If so, how can it be achieved?

- **Using OUT parameters**

plpgsql Copy Edit

```
CREATE FUNCTION get_student(IN id INT, OUT name TEXT, OUT age INT) AS $$
BEGIN
    SELECT student_name, student_age INTO name, age FROM students WHERE id = id;
END $$ LANGUAGE plpgsql;
```

- **Returning a Table**

plpgsql Copy Edit

```
CREATE FUNCTION get_student_table(id INT) RETURNS TABLE(name TEXT, age INT) AS $$
BEGIN
    RETURN QUERY SELECT student_name, student_age FROM students WHERE id = id;
END $$ LANGUAGE plpgsql;
```

4) What are the benefits of using packages in PL/SQL programming in PostgreSQL? Provide an example of a situation where packages can be useful.

- **Encapsulation, Reusability, Security, Performance.**
- **Example:** Group student-related functions in a schema (`student_pkg`).

5) How can you handle transactions within PL/SQL programs or procedures in PostgreSQL? Explain the use of the COMMIT and ROLLBACK statements.

- **COMMIT**: Saves changes.
- **ROLLBACK**: Reverts changes.
- **SAVEPOINT**: Creates rollback points.

plpgsql Copy Edit

```
BEGIN;
UPDATE students SET age = age + 1 WHERE id = 1;
SAVEPOINT sp1;
ROLLBACK TO sp1;
COMMIT;
```

6) What is the difference between an anonymous PL/SQL block and a stored procedure in PostgreSQL? When would you choose to use one over the other?

Feature	Anonymous Block	Stored Procedure
Execution	Runs once	Can be reused
Stored?	No	Yes
Use Case	Quick scripts	Business logic

7) How can you debug PL/SQL code in PostgreSQL? Describe some debugging techniques or tools available.

- **RAISE NOTICE** (Print logs).
- **EXCEPTION blocks** (Handle errors).
- **pgAdmin Debugger** (Step through code).

plpgsql Copy Edit

```
RAISE NOTICE 'Debug: %', var_name;
```

8) Can you call a PL/SQL function from a SQL statement in PostgreSQL? If so, provide an example and explain the process

plpgsql Copy Edit

```
CREATE FUNCTION square(n INT) RETURNS INT AS $$
BEGIN RETURN n * n; END $$ LANGUAGE plpgsql;
```

sql Copy Edit

```
SELECT square(5);
```

In Lab Task:

1. Write Use the tables below and implement the below queries

Student:

REGNO	NAME	ADDRESS	PHONE	AGE
101	Sai	Vijayawada	9455123451	18
102	Teja	Guntur	9652431543	18
103	Dinesh	Delhi	9156253131	20
104	Taurn	Chennai	9156768971	18
105	Dhanraj	Mumbai	9156253132	19
106	Rohit	Bangalore	9652431544	18
107	Niraj	Goa	9455123452	18

Course:

COURSE_ID	REG NO
1	101
2	102
2	103
3	104
4	105
5	106
1	107

Execute the following queries:

i) Write a PL/SQL program that retrieves all the records from the "Student" table and displays the student details.

```
DECLARE
   CURSOR student_cursor IS SELECT * FROM Student;
   v_student Student%ROWTYPE;
BEGIN
   OPEN student_cursor;
   LOOP
      FETCH student_cursor INTO v_student;
      EXIT WHEN student_cursor%NOTFOUND;
      DBMS_OUTPUT.PUT_LINE('REGNO: ' || v_student.REGNO || ', NAME: ' ||
v_student.NAME ||
                           ', ADDRESS: ' || v_student.ADDRESS || ', PHONE: ' ||
v_student.PHONE ||
                           ', AGE: ' || v_student.AGE);
   END LOOP;
   CLOSE student_cursor;
END;
/
```

ii) Create a PL/SQL procedure that takes a student's registration number as input and updates their age to 20. Test the procedure by updating the age of a specific student.

```
CREATE OR REPLACE PROCEDURE UpdateStudentAge(p_regno IN NUMBER) IS
BEGIN
   UPDATE Student
   SET AGE = 20
   WHERE REGNO = p_regno;
   COMMIT;
END;
/

-- Test the procedure
BEGIN
   UpdateStudentAge(103);
END;
/
```

iii) Develop a PL/SQL function that calculates the average age of all students in the "Student" table and returns the result.

```
CREATE OR REPLACE FUNCTION GetAverageAge RETURN NUMBER IS
   v_avg_age NUMBER;
BEGIN
   SELECT AVG(AGE) INTO v_avg_age FROM Student;
   RETURN v_avg_age;
END;
/

-- Test the function
DECLARE
   avg_age NUMBER;
BEGIN
   avg_age := GetAverageAge();
   DBMS_OUTPUT.PUT_LINE('Average Age: ' || avg_age);
END;
/
```

iv) Create a PL/SQL procedure that inserts a new row into the "Course" table, enrolling a student with a given registration number into a specified course. Test the procedure by enrolling a student in a course.

```
CREATE OR REPLACE PROCEDURE EnrollStudent(p_course_id IN NUMBER,
p_regno IN NUMBER) IS
BEGIN
   INSERT INTO Course (COURSE_ID, REGNO) VALUES (p_course_id,
p_regno);
   COMMIT;
END;
```

```
/

-- Test the procedure
BEGIN
   EnrollStudent(3, 107);
END;
/
```

v)Write a PL/SQL function that retrieves the count of students in the "Student" table and returns the result.

```
CREATE OR REPLACE FUNCTION CountStudents RETURN NUMBER IS
   v_count NUMBER;
BEGIN
   SELECT COUNT(*) INTO v_count FROM Student;
   RETURN v_count;
END;
/

-- Test the function
DECLARE
   total_students NUMBER;
BEGIN
   total_students := CountStudents();
   DBMS_OUTPUT.PUT_LINE('Total Students: ' || total_students);
END;
/
```

vi) Develop a PL/SQL procedure that deletes a student's record from the "Student" table based on their registration number. Test the procedure by deleting a specific student's record.

```
CREATE OR REPLACE PROCEDURE DeleteStudent(p_regno IN NUMBER)
IS
BEGIN
   DELETE FROM Student WHERE REGNO = p_regno;
   COMMIT;
END;
/

-- Test the procedure
```

```
BEGIN
   DeleteStudent(106);
END;
/
```

vii) Write a PL/SQL program that retrieves all the records from the "Course" table and displays the course details.

```
DECLARE
   CURSOR course_cursor IS SELECT * FROM Course;
   v_course Course%ROWTYPE;
BEGIN
   OPEN course_cursor;
   LOOP
      FETCH course_cursor INTO v_course;
      EXIT WHEN course_cursor%NOTFOUND;
      DBMS_OUTPUT.PUT_LINE('COURSE_ID: ' || v_course.COURSE_ID || ',
REGNO: ' || v_course.REGNO);
   END LOOP;
   CLOSE course_cursor;
END;
/
```

viii) Create a PL/SQL function that counts the number of students enrolled in each course. and returns the result as a cursor.

```
CREATE OR REPLACE FUNCTION CountStudentsPerCourse RETURN
SYS_REFCURSOR IS
   v_cursor SYS_REFCURSOR;
BEGIN
   OPEN v_cursor FOR
   SELECT COURSE_ID, COUNT(REGNO) AS Student_Count
   FROM Course
   GROUP BY COURSE_ID;
   RETURN v_cursor;
END;
/

-- Test the function
DECLARE
   v_cursor SYS_REFCURSOR;
   v_course_id NUMBER;
   v_count NUMBER;
BEGIN
   v_cursor := CountStudentsPerCourse();
   LOOP
      FETCH v_cursor INTO v_course_id, v_count;
      EXIT WHEN v_cursor%NOTFOUND;
      DBMS_OUTPUT.PUT_LINE('COURSE_ID: ' || v_course_id || ', Students
Enrolled: ' || v_count);
   END LOOP;
   CLOSE v_cursor;
```

```
END;
/
```

ix) Develop a PL/SQL procedure that updates the address of a student based on their registration number. Test the procedure by updating a specific student's address.

```
CREATE OR REPLACE PROCEDURE UpdateStudentAddress(p_regno IN
NUMBER, p_new_address IN VARCHAR2) IS
BEGIN
   UPDATE Student
   SET ADDRESS = p_new_address
   WHERE REGNO = p_regno;
   COMMIT;
END;
/

-- Test the procedure
BEGIN
   UpdateStudentAddress(104, 'Hyderabad');
END;
/
```

x)Write a PL/SQL program that accepts an age as input and retrieves all the student details from the "Student" table with that age

```
DECLARE
   v_age NUMBER := 18;
   CURSOR student_cursor IS SELECT * FROM Student WHERE AGE = v_age;
   v_student Student%ROWTYPE;
BEGIN
   OPEN student_cursor;
   LOOP
      FETCH student_cursor INTO v_student;
      EXIT WHEN student_cursor%NOTFOUND;
      DBMS_OUTPUT.PUT_LINE('REGNO: ' || v_student.REGNO || ', NAME: ' ||
v_student.NAME ||
                           ', ADDRESS: ' || v_student.ADDRESS || ', PHONE: ' ||
v_student.PHONE);
   END LOOP;
   CLOSE student_cursor;
END;
/
```

Viva-Voce Questions (In-Lab):

1. How can you pass parameters to PL/SQL procedures and functions, and what are the benefits of doing so?

- **Modes:** `IN` (input), `OUT` (output), `IN OUT` (both).
- **Benefits:** Reusability, flexibility, better maintainability, and reduced redundancy.

2. What are the benefits of encapsulating logic within procedures and functions in PL/SQL?

- Code reuse, modularity, security, performance boost, and easier maintenance.

3. Discuss the usage of cursors in PL/SQL and their significance in handling result sets.

- **Types:** Implicit (auto-handled) & Explicit (manually controlled).
- **Use:** Process multiple rows, optimize memory, and handle complex queries.

4. Explain the concept of packages in PL/SQL and how they aid in organizing and modularizing code.

- **Parts:** Specification (declaration) & Body (implementation).
- **Advantages:** Encapsulation, better performance, reusability, and organized code.

5. How can you use PL/SQL programs, procedures, and functions to enhance the performance and efficiency of database operations?

- Reduces network overhead, enables caching, supports bulk processing, improves security, and manages exceptions.

Post Lab Task:

1. Consider the following Relational Table as shown in below.

Patient:

Pid	Fname	Lname	Email	Phno	Address	Date_Admn
200	David	Samson	aaa@gmail.com	7654328976	Mumbai	13-10-2019
201	Bharath	Kumar	bbb@gmail.com	8765438907	Hyderabad	22-01-2020
202	Eswar	Prasad	ccc@gmail.com	9876548838	Vijayawada	15-03-2020
203	Jaya	Laxmi	ddd@gmail.com	8765489765	Delhi	06-07-2020
204	Laxmi	Devi	eee@gmail.com	7654430692	Mumbai	12-03-2020
205	Pramod	Reddy	fff@gmail.com	6543371619	Hyderabad	05-02-2020
206	Charan	Kumar	ggg@gmail.com	5432312546	Vijayawada	22-04-2020
207	Kalyan	Reddy	hhh@gmail.com	4321253473	Delhi	10-07-2020
208	Rajesh	Yadav	iklmn@gmail.com	3210194400	Chennai	03-03-2020
209	Naveen	Kumar	jghfr@gmail.com	2099135327	Hyderabad	22-04-2020

Bednum	Pid
20	200
21	201
22	202
23	203
24	204
25	205
26	206
27	207
28	208
29	209

Shift			
Doctor Id	Day	Starttime	Endtime
100	12-04-2020	9	5
101	20-06-2020	4	8
102	01-08-2020	6	12
105	05-08-2020	13	18
106	13-10-2019	4	8
107	22-01-2020	6	12
108	15-03-2020	13	18
109	06-07-2020	9	5
110	12-03-2020	4	8
111	05-02-2020	6	12
112	22-04-2020	13	18
113	10-07-2020	4	8
114	03-03-2020	6	12
115	22-04-2020	13	18

Appointment	
Pid	Amt
200	200
201	200
202	300
203	600
204	200
205	300
206	200
207	200
208	200
209	600

Questions:

i) Write a PL/SQL program that retrieves all the records from the "PATIENT" table and displays the patient details.

```
DECLARE
   CURSOR patient_cursor IS SELECT * FROM PATIENT;
   v_patient patient_cursor%ROWTYPE;
BEGIN
   OPEN patient_cursor;
   LOOP
      FETCH patient_cursor INTO v_patient;
      EXIT WHEN patient_cursor%NOTFOUND;
      DBMS_OUTPUT.PUT_LINE('ID: ' || v_patient.Pid || ', Name: ' ||
v_patient.Fname || ' ' || v_patient.Lname || ', Email: ' || v_patient.Email || ',
Phone: ' || v_patient.Phno || ', Address: ' || v_patient.Address || ', Admission
Date: ' || v_patient.Date_Admn);
   END LOOP;
   CLOSE patient_cursor;
END;
/
```

ii) Create a PL/SQL procedure that takes a patient's ID as input and updates their phone number in the "PATIENT" table. Test the procedure by updating the phone number of a specific patient.

```
CREATE OR REPLACE PROCEDURE Update_Phone(p_pid IN NUMBER,
new_phno IN VARCHAR2) AS
BEGIN
   UPDATE PATIENT SET Phno = new_phno WHERE Pid = p_pid;
   COMMIT;
END;
/
```

```
-- Test the procedure
-- Test the procedure
BEGIN
   Update_Phone(200, '9999999999');
END;
/
```

iii) Develop a PL/SQL function that calculates the average appointment amount from the "APPOINTMENT" table and returns the result.

```
CREATE OR REPLACE FUNCTION Avg_Appointment_Amount RETURN NUMBER
AS
   v_avg NUMBER;
BEGIN
   SELECT AVG(Amt) INTO v_avg FROM APPOINTMENT;
   RETURN v_avg;
END;
/
-- Test the procedure
DECLARE
   result NUMBER;
BEGIN
   result := Avg_Appointment_Amount;
   DBMS_OUTPUT.PUT_LINE('Average Appointment Amount: ' || result);
END;
/
```

iv) Create a PL/SQL procedure that inserts a new patient record into the "PATIENT" table with the provided details. Test the procedure by adding a new patient record.

```
CREATE OR REPLACE PROCEDURE Insert_Patient(
   p_pid IN NUMBER,
   p_fname IN VARCHAR2,
```

```
    p_lname IN VARCHAR2,
    p_email IN VARCHAR2,
    p_phno IN VARCHAR2,
    p_address IN VARCHAR2,
    p_date_admn IN DATE
) AS
BEGIN
    INSERT INTO PATIENT (Pid, Fname, Lname, Email, Phno, Address,
Date_Admn)
    VALUES (p_pid, p_fname, p_lname, p_email, p_phno, p_address,
p_date_admn);
    COMMIT;
END;
/

-- Test the procedure
BEGIN
    Insert_Patient(210, 'John', 'Doe', 'john.doe@gmail.com', '9876543210',
'Bangalore', TO_DATE('20-03-2025', 'DD-MM-YYYY'));
END;
/
```

v) Write a PL/SQL function that retrieves the count of patients from the "PATIENT" table and returns the result.

```
CREATE OR REPLACE FUNCTION Count_Patients RETURN NUMBER
AS
    v_count NUMBER;
BEGIN
    SELECT COUNT(*) INTO v_count FROM PATIENT;
    RETURN v_count;
END;
/
```

```
-- Test the procedure
DECLARE
   result NUMBER;
BEGIN
   result := Count_Patients;
   DBMS_OUTPUT.PUT_LINE('Total Number of Patients: ' || result);
END;
/
```

vi) Develop a PL/SQL procedure that deletes a patient's record from the "PATIENT" table based on their ID. Test the procedure by deleting a specific patient's record.

```
CREATE OR REPLACE PROCEDURE Delete_Patient(p_pid IN NUMBER) AS
BEGIN
   DELETE FROM PATIENT WHERE Pid = p_pid;
   COMMIT;
END;
/

-- Test the procedure
BEGIN
   Delete_Patient(210);
END;
/
```

vii) Write a PL/SQL program that retrieves all the records from the "SHIFT" table and displays the doctor shift details.

```
DECLARE
   CURSOR shift_cursor IS SELECT * FROM SHIFT;
   v_shift shift_cursor%ROWTYPE;
BEGIN
```

```
    OPEN shift_cursor;
    LOOP
        FETCH shift_cursor INTO v_shift;
        EXIT WHEN shift_cursor%NOTFOUND;
        DBMS_OUTPUT.PUT_LINE('Doctor ID: ' || v_shift.Doctor_Id || ', Day: ' ||
v_shift.Day || ', Start Time: ' || v_shift.Starttime || ', End Time: ' ||
v_shift.Endtime);
    END LOOP;
    CLOSE shift_cursor;
END;
/
```

viii) Create a PL/SQL function that counts the number of patients per bed from the "PCASE" table and returns the result as a cursor.

```
CREATE OR REPLACE FUNCTION Count_Patients_Per_Bed RETURN
SYS_REFCURSOR AS
    v_cursor SYS_REFCURSOR;
BEGIN
    OPEN v_cursor FOR
        SELECT Bednum, COUNT(Pid) AS Patient_Count
        FROM BEDNUM
        GROUP BY Bednum;
    RETURN v_cursor;
END;
/

-- Test the procedure
DECLARE
    v_cursor SYS_REFCURSOR;
    v_bednum NUMBER;
    v_count NUMBER;
BEGIN
```

```
    v_cursor := Count_Patients_Per_Bed;
    LOOP
        FETCH v_cursor INTO v_bednum, v_count;
        EXIT WHEN v_cursor%NOTFOUND;
        DBMS_OUTPUT.PUT_LINE('Bed Number: ' || v_bednum || ', Patient Count: ' ||
v_count);
    END LOOP;
    CLOSE v_cursor;
END;
/
```

ix) Develop a PL/SQL procedure that updates the end time of a doctor's shift based on their ID and the date. Test the procedure by updating a specific doctor's shift end time.

```
CREATE OR REPLACE PROCEDURE Update_Shift_Endtime(p_doctor_id IN
NUMBER, p_day IN DATE, new_endtime IN NUMBER) AS
BEGIN
    UPDATE SHIFT SET Endtime = new_endtime WHERE Doctor_Id =
p_doctor_id AND Day = p_day;
    COMMIT;
END;
/

-- Test the procedure
BEGIN
    Update_Shift_Endtime(100, TO_DATE('12-04-2020', 'DD-MM-YYYY'), 6);
END;
/
```

x) Write a PL/SQL program that accepts an address as input and retrieves all the patient details from the "PATIENT" table with that address.

```
DECLARE
   CURSOR address_cursor(p_address VARCHAR2) IS
      SELECT * FROM PATIENT WHERE Address = p_address;
   v_patient address_cursor%ROWTYPE;
   v_input_address VARCHAR2(50);
BEGIN
   v_input_address := 'Hyderabad';  -- Change this value as needed
   OPEN address_cursor(v_input_address);
   LOOP
      FETCH address_cursor INTO v_patient;
      EXIT WHEN address_cursor%NOTFOUND;
      DBMS_OUTPUT.PUT_LINE('ID: ' || v_patient.Pid || ', Name: ' || v_patient.Fname
|| ' ' || v_patient.Lname || ', Email: ' || v_patient.Email || ', Phone: ' || v_patient.Phno
|| ', Address: ' || v_patient.Address || ', Admission Date: ' || v_patient.Date_Admn);
   END LOOP;
   CLOSE address_cursor;
END;
/
```

Students Signature

(For Evaluator's use only)

Comment of the Evaluator (if Any)	Evaluator's Observation Marks Secured:_____ out of _____ Full Name of the Evaluator: Signature of the Evaluator Date of Evaluation: