Assignment 1: Stored Procedure

Set A

a. Write a procedure to display addition, subtraction and multiplication of three numbers.

Answer

```
CREATE OR REPLACE PROCEDURE CalculateOperations(
   IN num1 INT,
   IN num2 INT,
   IN num3 INT
)
LANGUAGE plpgsql
AS $$
BEGIN
   RAISE NOTICE 'Addition: %', num1 + num2 + num3;
   RAISE NOTICE 'Subtraction: %', num1 - num2 - num3;
   RAISE NOTICE 'Multiplication: %', num1 * num2 * num3;
END;
$$;
```

//To Call It://

```
CALL CalculateOperations(10, 5, 2);
```

b. Write a procedure to display division of two numbers use raise to display error messages.

Answer

```
CREATE OR REPLACE PROCEDURE DivideNumbers(
   IN num1 NUMERIC,
   IN num2 NUMERIC
)
LANGUAGE plpgsql
AS $$
DECLARE
   result NUMERIC;
BEGIN
   -- Check for division by zero
   IF num2 = 0 THEN
      RAISE EXCEPTION 'Error: Division by zero is not allowed.';
   END IF;

   result := num1 / num2;

   RAISE NOTICE 'Division Result: %', result;
END;
$$;
```

//To Call It://

CALL DivideNumbers(10, 2);

c.Create table Department (dno, dname, empname, city).

Answer

//Create Database//

```
CREATE DATABASE company_db;
```

//Create table //

```
CREATE TABLE Department (
   dno INT PRIMARY KEY,
   dname VARCHAR(50),
   empname VARCHAR(50),
   city VARCHAR(50)
);
```

i) Write a procedure to insert values in Department table.

Answer

```
CREATE OR REPLACE PROCEDURE insert_department(
   p_dno INT,
   p_dname VARCHAR,
   p_empname VARCHAR,
   p_city VARCHAR
)
LANGUAGE plpgsql
AS $$
BEGIN
   INSERT INTO Department(dno, dname, empname, city)
   VALUES (p_dno, p_dname, p_empname, p_city);

   RAISE NOTICE 'Record inserted successfully';
END;
$$;
```

//To Call It://

```
CALL insert_department(1, 'HR', 'Amit', 'Pune');
CALL insert_department(2, 'IT', 'Sneha', 'Mumbai');
CALL insert_department(3, 'Finance', 'Rahul', 'Pune');
```

ii) Write a procedure to display all employees working in ‘Pune’ city.

Answer

```
CREATE OR REPLACE PROCEDURE display_employees_pune()
LANGUAGE plpgsql
```

```
AS $$
DECLARE
   rec RECORD;
BEGIN
   FOR rec IN
      SELECT empname, dname
      FROM Department
      WHERE city = 'Pune'
   LOOP
      RAISE NOTICE 'Employee: %, Department: %',
         rec.empname, rec.dname;
   END LOOP;
END;
$$;
```

//To Call It://

```
CALL display_employees_pune();
```

SET B
A) Consider the following relationship
Route(route_no,source, destination, no_of_station) Bus
(bus_no, capacity, depot_name)
Relationship between Route and Bus is one-to-many

Answer

//Create Database//

```
CREATE DATABASE transport_db;
```

//Create table //

```
CREATE TABLE Route (
   route_no INT PRIMARY KEY,
   source VARCHAR(50),
   destination VARCHAR(50),
   no_of_station INT
);

CREATE TABLE Bus (
   bus_no INT PRIMARY KEY,
   capacity INT,
   depot_name VARCHAR(50),
   route_no INT,
   FOREIGN KEY (route_no) REFERENCES Route(route_no)
);

INSERT INTO Route VALUES (101, 'Pune', 'Mumbai', 12);
INSERT INTO Route VALUES (102, 'Delhi', 'Agra', 8);
INSERT INTO Bus VALUES (1, 50, 'Central Depot', 101);
INSERT INTO Bus VALUES (2, 45, 'West Depot', 101);
INSERT INTO Bus VALUES (3, 40, 'North Depot', 102);
```

a. Write a procedure which display all bus details for a given route.

Answer

```
CREATE OR REPLACE PROCEDURE display_bus_details(p_route_no INT)
LANGUAGE plpgsql
AS $$
DECLARE
   rec RECORD;
BEGIN
   FOR rec IN
      SELECT bus_no, capacity, depot_name
      FROM Bus
      WHERE route_no = p_route_no
   LOOP
      RAISE NOTICE 'Bus No: %, Capacity: %, Depot: %',
         rec.bus_no, rec.capacity, rec.depot_name;
   END LOOP;
END;
$$;
```

//To Call It://

```
CALL display_bus_details(101);
```

b. Write a procedure to update source of route no 101

Answer

```
CREATE OR REPLACE PROCEDURE update_source_101(p_new_source VARCHAR)
LANGUAGE plpgsql
AS $$
BEGIN
   UPDATE Route
   SET source = p_new_source
   WHERE route_no = 101;

   RAISE NOTICE 'Source updated successfully for Route 101';
END;
$$;
```

//To Call It://

```
CALL update_source_101('Nagpur');
```

B) Consider the following relationship
Patient (p_no, p_name, p_addr) Doctor
(d_no, d_name, d_addr, city)
Relationship between Patient and Doctor is many-to-many with descriptive attribute disease and no_of_visits.

Answer

```
CREATE TABLE Patient (
   p_no INT PRIMARY KEY,
   p_name VARCHAR(50),
   p_addr VARCHAR(100)
);

CREATE TABLE Doctor (
   d_no INT PRIMARY KEY,
   d_name VARCHAR(50),
   d_addr VARCHAR(100),
   city VARCHAR(50)
);

CREATE TABLE Patient_Doctor (
   p_no INT REFERENCES Patient(p_no),
   d_no INT REFERENCES Doctor(d_no),
   disease VARCHAR(50),
   no_of_visits INT,
   PRIMARY KEY (p_no, d_no, disease)
);
INSERT INTO Patient VALUES
(1, 'Rahul', 'Delhi'),
(2, 'Anita', 'Mumbai'),
(3, 'Suresh', 'Chennai'),
(4, 'Priya', 'Pune');

INSERT INTO Doctor VALUES
(101, 'Kumar', 'MG Road', 'Delhi'),
(102, 'Sharma', 'Brigade Road', 'Bangalore');

INSERT INTO Patient_Doctor VALUES
(1, 101, 'Diabetes', 5),
(2, 101, 'Diabetes', 2),
(3, 101, 'Diabetes', 4),
(4, 101, 'Fever', 6),
(2, 102, 'Diabetes', 7);
```

a. Write a procedure which display patient detail who has visited more than 3 times to the given doctor for ‘Diabetes’.

Answer

```
CREATE OR REPLACE FUNCTION get_diabetes_patients(doc_id INT)
RETURNS TABLE (
   p_no INT,
   p_name VARCHAR,
   p_addr VARCHAR
)
LANGUAGE plpgsql
AS $$
BEGIN
   RETURN QUERY
   SELECT p.p_no, p.p_name, p.p_addr
   FROM Patient p
   JOIN Patient_Doctor pd ON p.p_no = pd.p_no
   WHERE pd.d_no = doc_id
     AND pd.disease = 'Diabetes'
     AND pd.no_of_visits > 3;
END;
$$;
```

//To Call It://

```
SELECT * FROM get_diabetes_patients(101);
```

b. Write a procedure which displays total number of visits of Dr.Kumar.

Answer

```
CREATE OR REPLACE FUNCTION total_visits_kumar()
RETURNS INT
LANGUAGE plpgsql
AS $$
DECLARE
   total INT;
BEGIN
   SELECT SUM(pd.no_of_visits)
   INTO total
   FROM Patient_Doctor pd
   JOIN Doctor d ON pd.d_no = d.d_no
   WHERE d.d_name = 'Kumar';

   RETURN total;
END;
$$;
```

//To Call It://

```
SELECT total_visits_kumar();
```

Assignment 2 - Stored Functions

SET A

Using If.-Then-else,case,for,while and unconditional loops

1.Find minimum and maximum from two numbers

Answer

```
CREATE OR REPLACE FUNCTION MinMaxNumbers(
   num1 INT,
   num2 INT
)
RETURNS TABLE (
   Minimum INT,
   Maximum INT
)
LANGUAGE plpgsql
AS $$
BEGIN
   RETURN QUERY
   SELECT
      CASE WHEN num1 < num2 THEN num1 ELSE num2 END AS Minimum,
      CASE WHEN num1 > num2 THEN num1 ELSE num2 END AS Maximum;
END;
$$;
```

//To Call It://

```
SELECT * FROM MinMaxNumbers(10, 25);
```

2. Check the number is positive, negative or zero.

Answer

```
CREATE OR REPLACE PROCEDURE CheckNumber(
   IN num INT
)
LANGUAGE plpgsql
AS $$
BEGIN
   IF num > 0 THEN
      RAISE NOTICE 'The number % is Positive.', num;
   ELSIF num < 0 THEN
      RAISE NOTICE 'The number % is Negative.', num;
   ELSE
      RAISE NOTICE 'The number is Zero.';
   END IF;
END;
$$;
```

//To Call It://

```
CALL CheckNumber(10);
CALL CheckNumber(-5);
CALL CheckNumber(0);
```

3. Find maximum and minimum from three numbers

Answer

```
CREATE OR REPLACE FUNCTION MinMaxThreeNumbers(
   num1 INT,
   num2 INT,
   num3 INT
)
RETURNS TABLE (
   Minimum INT,
   Maximum INT
)
LANGUAGE plpgsql
AS $$
BEGIN
   RETURN QUERY
   SELECT
      LEAST(num1, num2, num3) AS Minimum,
      GREATEST(num1, num2, num3) AS Maximum;
END;
$$;
```

//To Call It://

```
SELECT * FROM MinMaxThreeNumbers(10, 25, 5);
```

4. Find number is even or odd

Answer

```
CREATE OR REPLACE PROCEDURE CheckEvenOdd(
   IN num INT
)
LANGUAGE plpgsql
AS $$
BEGIN
   IF num % 2 = 0 THEN
      RAISE NOTICE 'The number % is Even.', num;
   ELSE
      RAISE NOTICE 'The number % is Odd.', num;
   END IF;
END;
$$;
```

//To Call It://

```
CALL CheckEvenOdd(10);
CALL CheckEvenOdd(7);
```

5. Find sum of first 20 numbers (using unconditional loop)

Answer

```
CREATE OR REPLACE PROCEDURE SumFirst20Numbers()
LANGUAGE plpgsql
AS $$
DECLARE
   i INT := 1;
   sum INT := 0;
BEGIN
   LOOP
      sum := sum + i;
      i := i + 1;

      EXIT WHEN i > 20;  -- Exit loop after 20 numbers
   END LOOP;

   RAISE NOTICE 'Sum of first 20 numbers is %', sum;
END;
$$;
```

//To Call It://

```
CALL SumFirst20Numbers();
```

6. Display all even numbers from 1 to 50.

Answer

```
CREATE OR REPLACE PROCEDURE DisplayEvenNumbers()
LANGUAGE plpgsql
AS $$
DECLARE
   i INT := 1;
BEGIN
   LOOP
      IF i % 2 = 0 THEN
         RAISE NOTICE '%', i;
      END IF;

      i := i + 1;

      EXIT WHEN i > 50;  -- Stop loop after 50
   END LOOP;
END;
$$;

//To Call It://
CALL DisplayEvenNumbers();
```

7. Find sum and average of first n numbers using conditional loop(while)

Answer

```
CREATE OR REPLACE PROCEDURE SumAndAverage(n INT)
LANGUAGE plpgsql
AS $$
DECLARE
   i INT := 1;
   sum INT := 0;
   avg NUMERIC;
BEGIN
   -- Conditional loop using WHILE
   WHILE i <= n LOOP
      sum := sum + i;
      i := i + 1;
   END LOOP;

   avg := sum::NUMERIC / n;  -- Calculate average

   RAISE NOTICE 'Sum of first % numbers = %', n, sum;
   RAISE NOTICE 'Average of first % numbers = %', n, avg;
END;
$$;

//To Call It://

CALL CountOddNumbers(1, 10);
```

8. Count odd numbers from given range (m to n) (for)

Answer

```
CREATE OR REPLACE PROCEDURE CountOddNumbers(
   IN m INT,
   IN n INT
)
LANGUAGE plpgsql
AS $$
DECLARE
   i INT;
   count_odd INT := 0;
BEGIN
   FOR i IN m..n LOOP
      IF i % 2 <> 0 THEN
         count_odd := count_odd + 1;
      END IF;
   END LOOP;

   RAISE NOTICE 'Count of odd numbers between % and % = %', m, n, count_odd;
END;
$$;
```

//To Call It://

```
CALL CountOddNumbers(1, 10);
```

9. Search the given number is in given range.

Answer

```
CREATE OR REPLACE PROCEDURE SearchNumberInRange(
   IN num INT,
   IN start_range INT,
   IN end_range INT
)
LANGUAGE plpgsql
AS $$
BEGIN
   IF num BETWEEN start_range AND end_range THEN
      RAISE NOTICE 'The number % is in the range [% , %].', num, start_range, end_range;
   ELSE
      RAISE NOTICE 'The number % is NOT in the range [% , %].', num, start_range,
end_range;
   END IF;
END;
$$;
```

//To Call It://

```
CALL SearchNumberInRange(15, 10, 20);
CALL SearchNumberInRange(25, 10, 20);
```

10. Display a number in word (Using Case) and loop.

Answer

```
CREATE OR REPLACE PROCEDURE NumberInWords()
LANGUAGE plpgsql
AS $$
DECLARE
   i INT := 0;
BEGIN
   LOOP
      CASE i
         WHEN 0 THEN RAISE NOTICE '0 = Zero';
         WHEN 1 THEN RAISE NOTICE '1 = One';
         WHEN 2 THEN RAISE NOTICE '2 = Two';
         WHEN 3 THEN RAISE NOTICE '3 = Three';
         WHEN 4 THEN RAISE NOTICE '4 = Four';
         WHEN 5 THEN RAISE NOTICE '5 = Five';
         WHEN 6 THEN RAISE NOTICE '6 = Six';
         WHEN 7 THEN RAISE NOTICE '7 = Seven';
         WHEN 8 THEN RAISE NOTICE '8 = Eight';
         WHEN 9 THEN RAISE NOTICE '9 = Nine';
         ELSE
            EXIT;  -- Exit the loop after 9
      END CASE;

      i := i + 1;
   END LOOP;
END;
$$;
```

//To Call It://

```
CALL NumberInWords();
```

SET B

Bank database
Consider the following database maintained by a Bank. The Bank maintains information about its branches, customers and their loan applications.
Following are the tables:
branch (bid integer, brname char (30), brcity char (10))
customer (cno integer, cname char (20), caddr char (35), city char(20))
loan_application (lno integer, lamtrequired money, lamtapproved money, l_date date)
The relationship is as follows: branch, customer, loan_application are related with ternary relationship. TERNARY (bid integer, cno integer, lno integer).

Answer

```
-- Branch Table
CREATE TABLE branch (
   bid INTEGER PRIMARY KEY,
   brname CHAR(30) NOT NULL,
   brcity CHAR(10)
);

-- Customer Table
CREATE TABLE customer (
   cno INTEGER PRIMARY KEY,
   cname CHAR(20),
   caddr CHAR(35),
   city CHAR(20)
);

-- Loan Application Table
CREATE TABLE loan_application (
   lno INTEGER PRIMARY KEY,
   lamtrequired MONEY,
   lamtapproved MONEY,
   l_date DATE
);

-- Ternary Relationship Table
CREATE TABLE ternary (
   bid INTEGER,
   cno INTEGER,
   lno INTEGER,
   PRIMARY KEY (bid, cno, lno),
   FOREIGN KEY (bid) REFERENCES branch(bid),
   FOREIGN KEY (cno) REFERENCES customer(cno),
   FOREIGN KEY (lno) REFERENCES loan_application(lno)
);


-- Insert Branch
INSERT INTO branch VALUES
(1, 'Shivaji Nagar', 'Pune'),
(2, 'Andheri', 'Mumbai'),
(3, 'MG Road', 'Delhi');

-- Insert Customers
INSERT INTO customer VALUES
(101, 'Amit', 'Aundh Pune', 'Pune'),
(102, 'Neha', 'Baner Pune', 'Pune'),
(103, 'Raj', 'Andheri West', 'Mumbai'),
(104, 'Sneha', 'Karol Bagh', 'Delhi');

-- Insert Loan Applications
INSERT INTO loan_application VALUES
```

```
(1001, 300000, 250000, '2024-01-10'),
(1002, 150000, 150000, '2024-02-15'),
(1003, 500000, 450000, '2024-03-01'),
(1004, 200000, 180000, '2024-04-05');

-- Insert into Ternary (AFTER all above inserts)
INSERT INTO ternary VALUES
(1, 101, 1001),
(1, 102, 1002),
(2, 103, 1003),
(3, 104, 1004);
```

a) Write a function that returns the total number of customers of a particular branch.(Accept branch name as input parameter.)

Answer

```
CREATE OR REPLACE FUNCTION total_customers_by_branch(p_branch_name
VARCHAR)
RETURNS INTEGER AS
$$
DECLARE
   total INTEGER;
BEGIN
   SELECT COUNT(DISTINCT t.cno)
   INTO total
   FROM ternary t
   JOIN branch b ON t.bid = b.bid
   WHERE TRIM(b.brname) = p_branch_name;

   RETURN total;
END;
$$ LANGUAGE plpgsql;

//To Call It://

SELECT total_customers_by_branch('Shivaji Nagar');
```

b) Write a function to find the minimum loan amount approved.

Answer

```
CREATE OR REPLACE FUNCTION minimum_loan_approved()
RETURNS MONEY AS
$$
DECLARE
   min_amount MONEY;
BEGIN
   SELECT MIN(lamtapproved)
   INTO min_amount
   FROM loan_application;
```

```
    RETURN min_amount;
END;
$$ LANGUAGE plpgsql;
```

//To Call It://

```
SELECT minimum_loan_approved();
```

c) Create a function which returns the total number of customers who have applied for a loan more than Rs.200000. (Accept loan amount as input parameter)

Answer

```
CREATE OR REPLACE FUNCTION customers_above_amount(p_amount NUMERIC)
RETURNS INTEGER AS
$$
DECLARE
    total INTEGER;
BEGIN
    SELECT COUNT(DISTINCT t.cno)
    INTO total
    FROM ternary t
    JOIN loan_application l ON t.lno = l.lno
    WHERE l.lamtrequired::NUMERIC > p_amount;

    RETURN total;
END;
$$ LANGUAGE plpgsql;
```

//To Call It://

```
SELECT customers_above_amount(200000);
```

SET C

Student- Teacher database
Consider the following database maintained by a school. The school maintains information about students and the teachers. It also gives information of the subject taught by the teacher.
Following are the tables:
student (sno integer, s_name char(30), s_class char(10), s_addr char(50))
teacher (tno integer, t_name char (20), qualification char (15), experience integer)
The relationship is as follows: STUDENT-TEACHER: Many to Many with descriptive attribute SUBJECT.

Answer

```
-- Student Table
CREATE TABLE student (
    sno INTEGER PRIMARY KEY,
    s_name CHAR(30),
    s_class CHAR(10),
    s_addr CHAR(50)
);

-- Teacher Table
CREATE TABLE teacher (
   tno INTEGER PRIMARY KEY,
   t_name CHAR(20),
   qualification CHAR(15),
   experience INTEGER
);

-- Junction Table (Many-to-Many with descriptive attribute SUBJECT)
CREATE TABLE student_teacher (
   sno INTEGER,
   tno INTEGER,
   subject CHAR(20),
   PRIMARY KEY (sno, tno, subject),
   FOREIGN KEY (sno) REFERENCES student(sno),
   FOREIGN KEY (tno) REFERENCES teacher(tno)
);

-- Insert Students
INSERT INTO student VALUES
(1, 'Amit', '10A', 'Pune'),
(2, 'Neha', '10A', 'Pune'),
(3, 'Raj', '9B', 'Mumbai'),
(4, 'Sneha', '8C', 'Delhi');

-- Insert Teachers
INSERT INTO teacher VALUES
(101, 'Mr. Sharma', 'NET', 15),
(102, 'Mrs. Patil', 'PhD', 20),
(103, 'Mr. Khan', 'NET', 10),
(104, 'Ms. Roy', 'MSc', 8);

-- Insert Relationship (Subjects)
INSERT INTO student_teacher VALUES
(1,101,'Computer'),
(2,101,'Computer'),
(3,102,'Computer'),
(4,103,'Maths'),
(1,102,'Physics'),
(2,103,'Maths'),
(3,101,'Computer');
```

a) Write a function to find name of the most experienced teacher for “Computer”.
36

Answer

```
CREATE OR REPLACE FUNCTION most_experienced_computer_teacher()
RETURNS VARCHAR AS
$$
```

```
DECLARE
   teacher_name VARCHAR;
BEGIN
   SELECT t.t_name
   INTO teacher_name
   FROM teacher t
   JOIN student_teacher st ON t.tno = st.tno
   WHERE TRIM(st.subject) = 'Computer'
   ORDER BY t.experience DESC
   LIMIT 1;

   RETURN teacher_name;
END;
$$ LANGUAGE plpgsql;

//To Call It://

SELECT most_experienced_computer_teacher();
```

b) Write a function to find the teacher teaching maximum number of subjects.

Answer

```
CREATE OR REPLACE FUNCTION teacher_max_subjects()
RETURNS VARCHAR AS
$$
DECLARE
   teacher_name VARCHAR;
BEGIN
   SELECT t.t_name
   INTO teacher_name
   FROM teacher t
   JOIN student_teacher st ON t.tno = st.tno
   GROUP BY t.tno, t.t_name
   ORDER BY COUNT(DISTINCT st.subject) DESC
   LIMIT 1;

   RETURN teacher_name;
END;
$$ LANGUAGE plpgsql;

//To Call It://

SELECT teacher_max_subjects();
```

c) Write a function to find the number of teachers having qualification "NET".

Answer

```
CREATE OR REPLACE FUNCTION count_net_teachers()
RETURNS INTEGER AS
$$
```

```
DECLARE
   total INTEGER;
BEGIN
   SELECT COUNT(*)
   INTO total
   FROM teacher
   WHERE TRIM(qualification) = 'NET';

   RETURN total;
END;
$$ LANGUAGE plpgsql;
```

//To Call It://

```
SELECT count_net_teachers();
```

SET D

Q.Project – Employee Database
Consider the following database
Project (pno int, pname char (30), ptype char (20), duration int)
Employee (empno int, ename char (20), joining_date date)
The relationship between Project and Employee is many to many with descriptive attribute start_date.
Create the above database in PostGreSQL and insert sufficient records.
Execute any two of the following using PL/pgSQL

Answer

```
-- Drop tables if they already exist
DROP TABLE IF EXISTS project_employee;
DROP TABLE IF EXISTS project;
DROP TABLE IF EXISTS employee;

-- Project Table
CREATE TABLE project (
   pno INTEGER PRIMARY KEY,
   pname VARCHAR(30),
   ptype VARCHAR(20),
   duration INTEGER
);

-- Employee Table
CREATE TABLE employee (
   empno INTEGER PRIMARY KEY,
   ename VARCHAR(20),
   joining_date DATE
);

-- Junction Table for Many-to-Many Relationship with descriptive attribute start_date
```

```
CREATE TABLE project_employee (
   pno INTEGER,
   empno INTEGER,
   start_date DATE,
   PRIMARY KEY (pno, empno),
   FOREIGN KEY (pno) REFERENCES project(pno),
   FOREIGN KEY (empno) REFERENCES employee(empno)
);

-- Insert Projects
INSERT INTO project VALUES
(1, 'Banking System', 'Software', 12),
(2, 'Bridge Construction', 'Civil', 24),
(3, 'School App', 'Software', 8),
(4, 'Metro Rail', 'Infrastructure', 36);

-- Insert Employees
INSERT INTO employee VALUES
(101, 'Amit', '2020-01-10'),
(102, 'Neha', '2019-03-15'),
(103, 'Raj', '2021-06-20'),
(104, 'Sneha', '2018-11-05');

-- Insert Many-to-Many relationship with start_date
INSERT INTO project_employee VALUES
(1,101,'2023-01-01'),
(1,102,'2023-02-01'),
(2,103,'2022-05-10'),
(3,101,'2023-03-01'),
(3,104,'2023-04-15'),
(4,102,'2021-07-01');
```

a. Write a stored function to accept project type as an input and display all project names of that type.

Answer

```
CREATE OR REPLACE FUNCTION projects_by_type(p_project_type VARCHAR)
RETURNS TABLE(project_name VARCHAR) AS
$$
BEGIN
   RETURN QUERY
   SELECT pname
   FROM project
   WHERE ptype = p_project_type;
END;
$$ LANGUAGE plpgsql;
```

//To Call It://

```
SELECT * FROM projects_by_type('Software');
```

b. Write a function which accepts employee name and prints the details of the project which the employee works on.

Answer

```
CREATE OR REPLACE FUNCTION projects_of_employee(p_employee_name VARCHAR)
RETURNS TABLE(
   project_name VARCHAR,
   project_type VARCHAR,
   duration INTEGER,
   start_date DATE
) AS
$$
BEGIN
   RETURN QUERY
   SELECT
      p.pname,
      p.ptype,
      p.duration,
      pe.start_date
   FROM project p
   JOIN project_employee pe ON p.pno = pe.pno
   JOIN employee e ON e.empno = pe.empno
   WHERE e.ename = p_employee_name;
END;
$$ LANGUAGE plpgsql;
```

//To Call It://

```
SELECT * FROM projects_of_employee('Amit');
```

c. Write a function to accept project name as input and returns the number of employees working on the project.

Answer

```
CREATE OR REPLACE FUNCTION count_employees_in_project(p_project_name
VARCHAR)
RETURNS INTEGER AS
$$
DECLARE
   total INTEGER;
BEGIN
   SELECT COUNT(pe.empno)
   INTO total
   FROM project_employee pe
   JOIN project p ON p.pno = pe.pno
   WHERE p.pname = p_project_name;

   RETURN total;
END;
$$ LANGUAGE plpgsql;
```

//To Call It://

```
SELECT count_employees_in_project('Banking System');
```

SET E

Student - Subject Database
Consider the following database
Student (roll_no integer, name varchar(30), address varchar(50), class varchar(10)) Subject (scode varchar(10), subject_name varchar(20))
Student-Subject are related with Many to Many relationship with attributes marks_scored. Create the above database in PostGreSQL and insert sufficient records.

//1 Create Database (Optional)//

```
CREATE DATABASE student_subject_db;
```

//Create Tables//

```
-- Student Table
CREATE TABLE Student (
   roll_no INTEGER PRIMARY KEY,
   name VARCHAR(30),
   address VARCHAR(50),
   class VARCHAR(10)
);

-- Subject Table
CREATE TABLE Subject (
   scode VARCHAR(10) PRIMARY KEY,
   subject_name VARCHAR(20)
);

-- Junction Table (Many-to-Many Relationship)
CREATE TABLE Student_Subject (
   roll_no INTEGER REFERENCES Student(roll_no) ON DELETE CASCADE,
   scode VARCHAR(10) REFERENCES Subject(scode) ON DELETE CASCADE,
   marks_scored INTEGER,
   PRIMARY KEY (roll_no, scode)
);
```

Insert Sample Records

```
-- Insert Students
INSERT INTO Student VALUES
(1, 'Amit Sharma', 'Pune', 'FYBSc'),
(2, 'Neha Patil', 'Mumbai', 'SYBSc'),
(3, 'Rahul Mehta', 'Delhi', 'FYBSc'),
(4, 'Sneha Iyer', 'Chennai', 'TYBSc');
```

```
-- Insert Subjects
INSERT INTO Subject VALUES
('S101', 'Mathematics'),
('S102', 'Physics'),
('S103', 'Chemistry'),
('S104', 'Computer');

-- Insert Student-Subject Records
INSERT INTO Student_Subject VALUES
(1, 'S101', 85),
(1, 'S102', 78),
(2, 'S101', 88),
(2, 'S104', 91),
(3, 'S103', 72),
(3, 'S104', 80),
(4, 'S101', 90),
(4, 'S102', 87);
```

a. Write a function which will accept the name and print all the details of that student.

Answer

```
CREATE OR REPLACE FUNCTION get_student_by_name(p_name VARCHAR)
RETURNS TABLE(
   roll_no INTEGER,
   name VARCHAR,
   address VARCHAR,
   class VARCHAR,
   subject_name VARCHAR,
   marks_scored INTEGER
) AS
$$
BEGIN
   RETURN QUERY
   SELECT s.roll_no, s.name, s.address, s.class,
          sub.subject_name, ss.marks_scored
   FROM Student s
   JOIN Student_Subject ss ON s.roll_no = ss.roll_no
   JOIN Subject sub ON ss.scode = sub.scode
   WHERE s.name = p_name;
END;
$$ LANGUAGE plpgsql;

//To Call It://

SELECT * FROM get_student_by_name('Amit Sharma');
```

b. Write a function to accept student roll_no as input and displays details of that student.

Answer

```
CREATE OR REPLACE FUNCTION get_student_by_roll(p_roll INTEGER)
RETURNS TABLE(
   roll_no INTEGER,
   name VARCHAR,
   addrAnsweress VARCHAR,
   class VARCHAR,
   subject_name VARCHAR,
   marks_scored INTEGER
) AS
$$
BEGIN
   RETURN QUERY
   SELECT s.roll_no, s.name, s.address, s.class,
        sub.subject_name, ss.marks_scored
   FROM Student s
   JOIN Student_Subject ss ON s.roll_no = ss.roll_no
   JOIN Subject sub ON ss.scode = sub.scode
   WHERE s.roll_no = p_roll;
END;
$$ LANGUAGE plpgsql;

//To Call It://

SELECT * FROM get_student_by_roll(1);
```

c. Write a stored function using cursors, to accept **c**lass from the user and display the details of the students of that class.

Answer

```
CREATE OR REPLACE FUNCTION get_students_by_class(p_class VARCHAR)
RETURNS VOID AS
$$
DECLARE
   rec RECORD;
   cur CURSOR FOR
      SELECT roll_no, name, address, class
      FROM Student
      WHERE class = p_class;
BEGIN
   OPEN cur;

   LOOP
      FETCH cur INTO rec;
      EXIT WHEN NOT FOUND;

      RAISE NOTICE 'Roll No: %, Name: %, Address: %, Class: %',
                rec.roll_no, rec.name, rec.address, rec.class;
   END LOOP;

   CLOSE cur;
END;
```

$$ LANGUAGE plpgsql;

//To Call It://

SELECT get_students_by_class('FYBSc');

Assignment 3: Cursors

SET A

Bus – Route Database
Consider the following database
Bus (bus_no int, capacity int , depot_name varchar(20))
Route (route_no int, source varchar(20), destination varchar(20), No_of_stations int) Bus and Route are related with many to many relationship.
Create the above database in PostGreSQL and insert sufficient records.

Answer

//Create Database//

CREATE DATABASE bus_route_db;

//Create Tables//

```
CREATE TABLE Bus (
   bus_no INT PRIMARY KEY,
   capacity INT,
   depot_name VARCHAR(20)
);

CREATE TABLE Route (
   route_no INT PRIMARY KEY,
   source VARCHAR(20),
   destination VARCHAR(20),
   no_of_stations INT
);

CREATE TABLE Bus_Route (
   bus_no INT REFERENCES Bus(bus_no) ON DELETE CASCADE,
   route_no INT REFERENCES Route(route_no) ON DELETE CASCADE,
   PRIMARY KEY (bus_no, route_no)
);

//Insert Sample Records//

INSERT INTO Bus VALUES
(101, 50, 'Central Depot'),
(102, 45, 'North Depot'),
(108, 60, 'City Depot'),
(110, 55, 'East Depot');

INSERT INTO Route VALUES
(1, 'Station', 'Airport', 10),
(2, 'Market', 'University', 8),
(3, 'Station', 'Mall', 6),
(4, 'Airport', 'City Center', 7);

INSERT INTO Bus_Route VALUES
(108, 1),
(108, 3),
(101, 2),
(102, 1),
(110, 4);
```

a. Write a stored function using cursor, which will give details of all routes on which bus no 108 is running.

Answer

```
CREATE OR REPLACE FUNCTION get_routes_of_bus_108()
RETURNS VOID AS
$$
DECLARE
   rec RECORD;
   cur CURSOR FOR
      SELECT r.route_no, r.source, r.destination, r.no_of_stations
      FROM Route r
      JOIN Bus_Route br ON r.route_no = br.route_no
      WHERE br.bus_no = 108;
BEGIN
   OPEN cur;

   LOOP
      FETCH cur INTO rec;
      EXIT WHEN NOT FOUND;
```

```
        RAISE NOTICE 'Route No: %, Source: %, Destination: %, Stations: %',
            rec.route_no, rec.source, rec.destination, rec.no_of_stations;
    END LOOP;

    CLOSE cur;
END;
$$ LANGUAGE plpgsql;
```

//To Call It://

```
SELECT get_routes_of_bus_108();
```

b. Write a stored function using cursor, which will give details of all buses on route from “Station” to “Airport”.

Answer

```
CREATE OR REPLACE FUNCTION get_buses_station_to_airport()
RETURNS VOID AS
$$
DECLARE
    rec RECORD;
    cur CURSOR FOR
        SELECT b.bus_no, b.capacity, b.depot_name
        FROM Bus b
        JOIN Bus_Route br ON b.bus_no = br.bus_no
        JOIN Route r ON br.route_no = r.route_no
        WHERE r.source = 'Station' AND r.destination = 'Airport';
BEGIN
    OPEN cur;

    LOOP
        FETCH cur INTO rec;
        EXIT WHEN NOT FOUND;

        RAISE NOTICE 'Bus No: %, Capacity: %, Depot: %',
            rec.bus_no, rec.capacity, rec.depot_name;
    END LOOP;

    CLOSE cur;
END;
$$ LANGUAGE plpgsql;
```

//To Call It://

```
SELECT get_buses_station_to_airport();
```

SET B

Student –Teacher database

Consider the following database
Teacher(t_no int, t_name varchar(20), age int, yr_experience int)
Subject (s_no int, s_namevarchar(15))
Teacher and Subject are related with many to many relationship
Create the above database in PostGreSQL and insert sufficient records.

Answer

//Create Database//

```
CREATE DATABASE student_teacher_db;
```

//Create Tables//

```
CREATE TABLE Teacher (
   t_no INT PRIMARY KEY,
   t_name VARCHAR(20),
   age INT,
   yr_experience INT
);

CREATE TABLE Subject (
   s_no INT PRIMARY KEY,
   s_name VARCHAR(15)
);

CREATE TABLE Teacher_Subject (
   t_no INT REFERENCES Teacher(t_no) ON DELETE CASCADE,
   s_no INT REFERENCES Subject(s_no) ON DELETE CASCADE,
   PRIMARY KEY (t_no, s_no)
);

INSERT INTO Teacher VALUES
(1, 'Mr. Sharma', 40, 15),
(2, 'Ms. Patil', 35, 10),
(3, 'Dr. Mehta', 50, 25),
(4, 'Mrs. Iyer', 38, 12);

INSERT INTO Subject VALUES
(101, 'Mathematics'),
(102, 'Physics'),
(103, 'Chemistry'),
(104, 'Computer');

INSERT INTO Teacher_Subject VALUES
(1, 101),
(1, 102),
(2, 103),
(3, 101),
(3, 104),
(4, 102);
```

a. Write a stored function using cursor which will accept the subject name and print the names of all teachers teaching that subject.

Answer

```
CREATE OR REPLACE FUNCTION get_teachers_by_subject(p_subject_name VARCHAR)
RETURNS VOID AS
$$
DECLARE
   rec RECORD;
   cur CURSOR FOR
      SELECT t.t_name
      FROM Teacher t
      JOIN Teacher_Subject ts ON t.t_no = ts.t_no
      JOIN Subject s ON ts.s_no = s.s_no
      WHERE s.s_name = p_subject_name;
BEGIN
   OPEN cur;

   LOOP
      FETCH cur INTO rec;
      EXIT WHEN NOT FOUND;

      RAISE NOTICE 'Teacher Name: %', rec.t_name;
   END LOOP;

   CLOSE cur;
END;
$$ LANGUAGE plpgsql;

//To Call It://

SELECT get_teachers_by_subject('Mathematics');
```

b. Write a cursor to accept the subject's name from the user as an input and display names of all teachers teaching that student.

Answer

```
CREATE OR REPLACE FUNCTION display_teachers_by_subject(p_subject_name
VARCHAR)
RETURNS VOID AS
$$
DECLARE
   rec RECORD;
   cur CURSOR FOR
      SELECT t.t_name
      FROM Teacher t
      JOIN Teacher_Subject ts ON t.t_no = ts.t_no
      JOIN Subject s ON ts.s_no = s.s_no
      WHERE s.s_name = p_subject_name;
```

```
BEGIN
   OPEN cur;

   LOOP
      FETCH cur INTO rec;
      EXIT WHEN NOT FOUND;

      RAISE NOTICE 'Teacher Name: %', rec.t_name;
   END LOOP;

   CLOSE cur;
END;
$$ LANGUAGE plpgsql;
```

//To Call It://

```
SELECT display_teachers_by_subject('Physics');
```

SET C

Person - Area Database
Person (pno int, name varchar (20), birthdate date, income money) Area
(aid int, aname varchar (20), area_type varchar (5))
The person and area related to many to one relationship. The attribute ‘area_type’ can have values either ‘urban’ or ‘rural’.
Create the above database in PostGreSQL and insert sufficient records.

Answer

//Create Database//

```
CREATE DATABASE person_area_db;
```

//Create Tables//

```
CREATE TABLE Area (
   aid INT PRIMARY KEY,
   aname VARCHAR(20),
   area_type VARCHAR(5) CHECK (area_type IN ('urban','rural'))
);

CREATE TABLE Person (
   pno INT PRIMARY KEY,
   name VARCHAR(20),
   birthdate DATE,
   income MONEY,
   aid INT REFERENCES Area(aid) ON DELETE SET NULL
);
```

```
INSERT INTO Area VALUES
(1, 'Shivaji Nagar', 'urban'),
(2, 'Green Village', 'rural'),
(3, 'City Center', 'urban'),
(4, 'Hill Side', 'rural');

INSERT INTO Person VALUES
(101, 'Amit', '1998-01-15', 75000, 1),
(102, 'Neha', '1995-02-20', 55000, 3),
(103, 'Rahul', '1990-01-10', 120000, 2),
(104, 'Sneha', '1997-03-05', 65000, 1),
(105, 'Karan', '1996-02-25', 48000, 4),
(106, 'Pooja', '1994-01-30', 90000, 3);
```

a. Write a cursor to accept a month as an input parameter from the user and display the names of persons whose birthday falls in that particular month.

Answer

```
CREATE OR REPLACE FUNCTION get_persons_by_birth_month(p_month INT)
RETURNS VOID AS
$$
DECLARE
   rec RECORD;
   cur CURSOR FOR
      SELECT name
      FROM Person
      WHERE EXTRACT(MONTH FROM birthdate) = p_month;
BEGIN
   OPEN cur;

   LOOP
      FETCH cur INTO rec;
      EXIT WHEN NOT FOUND;

      RAISE NOTICE 'Name: %', rec.name;
   END LOOP;

   CLOSE cur;
END;
$$ LANGUAGE plpgsql;
```

//To Call It://

```
SELECT get_persons_by_birth_month(1);
```

b. Write a cursor to display the names of persons living in urban area.

Answer

```
CREATE OR REPLACE FUNCTION get_urban_persons()
RETURNS VOID AS
```

```
$$
DECLARE
   rec RECORD;
   cur CURSOR FOR
      SELECT p.name
      FROM Person p
      JOIN Area a ON p.aid = a.aid
      WHERE a.area_type = 'urban';
BEGIN
   OPEN cur;

   LOOP
      FETCH cur INTO rec;
      EXIT WHEN NOT FOUND;

      RAISE NOTICE 'Name: %', rec.name;
   END LOOP;

   CLOSE cur;
END;
$$ LANGUAGE plpgsql;
```

//To Call It://

```
SELECT get_urban_persons();
```

c. Write a cursor to print names of all persons having income between50,000 and 1,00,000.

Answer

```
CREATE OR REPLACE FUNCTION get_persons_by_income_range()
RETURNS VOID AS
$$
DECLARE
   rec RECORD;
   cur CURSOR FOR
      SELECT name
      FROM Person
      WHERE income BETWEEN 50000::money AND 100000::money;
BEGIN
   OPEN cur;

   LOOP
      FETCH cur INTO rec;
      EXIT WHEN NOT FOUND;

      RAISE NOTICE 'Name: %', rec.name;
   END LOOP;

   CLOSE cur;
END;
$$ LANGUAGE plpgsql;
```

//To Call It://

```
SELECT get_persons_by_income_range();
```

SET D

Student – Competition Database
 Consider the following entities and their relationship.
 Student (s_reg_no int, s_name varchar(20), s_class varchar(20))
 Competition (comp_no int, comp_name varchar(20), comp_type varchar(20))
Relationship between Student and Competition is many-to-many with descriptive attribute rank and year.

Answer

//Create Database//

```
CREATE DATABASE student_competition_db;
```

//Create Tables//

```
CREATE TABLE student (
   s_reg_no INT PRIMARY KEY,
   s_name VARCHAR(20),
   s_class VARCHAR(20)
);

CREATE TABLE competition (
   comp_no INT PRIMARY KEY,
   comp_name VARCHAR(20),
   comp_type VARCHAR(20)
);

CREATE TABLE participation (
   s_reg_no INT,
   comp_no INT,
   rank INT,
   year INT,
   PRIMARY KEY (s_reg_no, comp_no, year),
   FOREIGN KEY (s_reg_no) REFERENCES student(s_reg_no),
   FOREIGN KEY (comp_no) REFERENCES competition(comp_no)
);

INSERT INTO student VALUES
(1, 'Amit', 'TYBCA'),
(2, 'Neha', 'SYBCA'),
(3, 'Rahul', 'FYBCA');

INSERT INTO competition VALUES
```

```
(101, 'Coding', 'Technical'),
(102, 'Debate', 'Cultural'),
(103, 'Quiz', 'Academic');

INSERT INTO participation VALUES
(1, 101, 1, 2023),
(1, 102, 2, 2023),
(2, 101, 3, 2023),
(2, 103, 1, 2024),
(3, 102, 2, 2024);
```

a) Write a cursor which will display year wise details of competitions held. (Use parameterized cursor)

Answer

```
CREATE OR REPLACE FUNCTION year_wise_competitions(p_year INT)
RETURNS REFCURSOR AS $$
DECLARE
   comp_ref REFCURSOR;
BEGIN
   OPEN comp_ref FOR
      SELECT c.comp_no, c.comp_name, c.comp_type, p.rank
      FROM competition c
      JOIN participation p
      ON c.comp_no = p.comp_no
      WHERE p.year = p_year;
   RETURN comp_ref;
END;
$$ LANGUAGE plpgsql;
```

b) Write a cursor which will display student wise total count of competition participated.

Answer

```
CREATE OR REPLACE FUNCTION student_competition_count()
RETURNS REFCURSOR AS $$
DECLARE
   student_ref REFCURSOR;
BEGIN
   OPEN student_ref FOR
      SELECT s.s_reg_no, s.s_name, COUNT(p.comp_no) AS total_count
      FROM student s
      JOIN participation p
      ON s.s_reg_no = p.s_reg_no
      GROUP BY s.s_reg_no, s.s_name;
   RETURN student_ref;
END;
$$ LANGUAGE plpgsql;



//To Call It://
```

```
BEGIN;
SELECT student_competition_count();  -- returns cursor name
FETCH ALL FROM "<cursor name>";      -- fetch results
COMMIT;
```

SET E

Car – Driver Database
Consider the following database:
/To Call It://
Car (c_no int, owner varchar(20), model varchar(10), color varchar(10)
Driver (driver_no int, driver_namevarchar(20), license_no int, d_age int, salary float) Car and Driver are related with many to many relationship
Create the above database in PostGreSQL and insert sufficient records.

Answer

```
-- Drop tables if already exist
DROP TABLE IF EXISTS Car_Driver;
DROP TABLE IF EXISTS Car;
DROP TABLE IF EXISTS Driver;

-- Car Table
CREATE TABLE Car (
   c_no INT PRIMARY KEY,
   owner VARCHAR(20),
   model VARCHAR(10),
   color VARCHAR(10)
);

-- Driver Table
CREATE TABLE Driver (
   driver_no INT PRIMARY KEY,
   driver_name VARCHAR(20),
   license_no INT,
   d_age INT,
   salary FLOAT
);
/To Call It://

-- Many-to-Many Relationship Table
CREATE TABLE Car_Driver (
   c_no INT REFERENCES Car(c_no),
   driver_no INT REFERENCES Driver(driver_no),
   PRIMARY KEY (c_no, driver_no)
);

-- Insert Cars
```

```
INSERT INTO Car VALUES
(1, 'Ramesh', 'Honda', 'Red'),
(2, 'Suresh', 'Hyundai', 'Blue'),
(3, 'Mahesh', 'Toyota', 'Red'),
(4, 'Anita', 'Ford', 'Black'),
(5, 'Kiran', 'BMW', 'White');

-- Insert Drivers
INSERT INTO Driver VALUES
(101, 'Arun', 5001, 30, 25000),
(102, 'Bala', 5002, 35, 30000),
(103, 'Charan', 5003, 28, 22000),
(104, 'David', 5004, 40, 35000);

-- Insert Car-Driver Relations
INSERT INTO Car_Driver VALUES
(1,101),
(2,101),
(3,102),
(4,103),
(5,104),
(1,104);
```

a. Write a stored function with cursor which accepts the color and prints the names of all owners who own a car of that color.
Answer

```
CREATE OR REPLACE FUNCTION get_owner_by_color(p_color VARCHAR)
RETURNS VOID AS
$$
DECLARE
   rec RECORD;
   car_cursor CURSOR FOR
      SELECT DISTINCT owner
      FROM Car
      WHERE color = p_color;
BEGIN
   OPEN car_cursor;

   LOOP
      FETCH car_cursor INTO rec;
      EXIT WHEN NOT FOUND;
      RAISE NOTICE 'Owner: %', rec.owner;
   END LOOP;

   CLOSE car_cursor;

   IF NOT FOUND THEN
      RAISE NOTICE 'No cars found with color %', p_color;
   END IF;
```

```
END;
$$
LANGUAGE plpgsql;

//To Call It://

SELECT get_owner_by_color('Red');
```

b. Write a cursor which accepts the driver name and prints the details of all cars that this driver has driven, if the driver name is invalid, print an appropriate message

Answer

```
CREATE OR REPLACE FUNCTION get_cars_by_driver(p_driver_name VARCHAR)
RETURNS VOID AS
$$
DECLARE
   rec RECORD;
   v_driver_id INT;
   car_cursor CURSOR FOR
      SELECT c.c_no, c.owner, c.model, c.color
      FROM Car c
      JOIN Car_Driver cd ON c.c_no = cd.c_no
      WHERE cd.driver_no = v_driver_id;
BEGIN
   -- Check if driver exists
   SELECT driver_no INTO v_driver_id
   FROM Driver
   WHERE driver_name = p_driver_name;

   IF v_driver_id IS NULL THEN
      RAISE NOTICE 'Invalid Driver Name!';
      RETURN;
   END IF;

   OPEN car_cursor;

   LOOP
      FETCH car_cursor INTO rec;
      EXIT WHEN NOT FOUND;
      RAISE NOTICE 'Car No: %, Owner: %, Model: %, Color: %',
                 rec.c_no, rec.owner, rec.model, rec.color;
   END LOOP;

   CLOSE car_cursor;
END;
$$
LANGUAGE plpgsql;

//To Call It://

SELECT get_cars_by_driver('Arun');
```

Assignment 4: Handling errors and Exceptions

SET A

Project-Employee database
Consider the following database:
Project (pno int, pname char (30), ptype char (20), duration int)
Employee (empno int, ename char (20), joining_date date)
The relationship between Project and Employee is many to many with descriptive attribute start_date.
Create the above database in PostGreSQL and insert sufficient records.

Answer

//-- Create a new database//

CREATE DATABASE project_employee_db;

-- Drop tables if they exist
DROP TABLE IF EXISTS Project_Employee;
DROP TABLE IF EXISTS Project;
DROP TABLE IF EXISTS Employee;

-- Project Table
CREATE TABLE Project (

```
    pno INT PRIMARY KEY,
    pname CHAR(30) UNIQUE,
    ptype CHAR(20),
    duration INT
);

-- Employee Table
CREATE TABLE Employee (
    empno INT PRIMARY KEY,
    ename CHAR(20),
    joining_date DATE
);

-- Relationship Table (Many-to-Many)
CREATE TABLE Project_Employee (
    pno INT REFERENCES Project(pno),
    empno INT REFERENCES Employee(empno),
    start_date DATE,
    PRIMARY KEY (pno, empno)
);

-- Insert Projects
INSERT INTO Project VALUES
(1, 'Banking System', 'Software', 12),
(2, 'Ecommerce App', 'Web', 8),
(3, 'AI Research', 'Research', 18);

-- Insert Employees
INSERT INTO Employee VALUES
(101, 'Amit', '2022-01-15'),
(102, 'Sneha', '2021-03-10'),
(103, 'Rahul', '2023-06-01'),
(104, 'Priya', '2020-09-20');

-- Insert Relationship Data
INSERT INTO Project_Employee VALUES
(1,101,'2023-01-01'),
(1,102,'2023-02-01'),
(2,103,'2023-03-15'),
(2,101,'2023-04-01'),
(3,104,'2023-05-10');
```

a. Write a stored function to accept project name as input and print the names of employees working on the project. Also print the total number of employees working on that project. Raise an exception for an invalid project name.

Answer

```
CREATE OR REPLACE FUNCTION get_employees_by_project(p_project_name
VARCHAR)
RETURNS VOID AS
$$
```

```
DECLARE
   rec RECORD;
   v_pno INT;
   v_count INT := 0;
BEGIN
   -- Check if project exists
   SELECT pno INTO v_pno
   FROM Project
   WHERE TRIM(pname) = TRIM(p_project_name);

   IF v_pno IS NULL THEN
      RAISE EXCEPTION 'Invalid Project Name!';
   END IF;

   -- Loop through employees
   FOR rec IN
      SELECT e.ename
      FROM Employee e
      JOIN Project_Employee pe ON e.empno = pe.empno
      WHERE pe.pno = v_pno
   LOOP
      RAISE NOTICE 'Employee Name: %', TRIM(rec.ename);
      v_count := v_count + 1;
   END LOOP;

   RAISE NOTICE 'Total Employees Working on Project: %', v_count;
END;
$$
LANGUAGE plpgsql;
//To Call It://

SELECT get_employees_by_project('Banking System');
```

b. Write a stored function to accept empno as an input parameter from the user and count the number of projects of a given employee. Raise an exception if the employee number is invalid.

Answer

```
CREATE OR REPLACE FUNCTION count_projects_by_employee(p_empno INT)
RETURNS VOID AS
$$
DECLARE
   v_count INT;
   v_exists INT;
BEGIN
   -- Check if employee exists
   SELECT COUNT(*) INTO v_exists
   FROM Employee
   WHERE empno = p_empno;

   IF v_exists = 0 THEN
```

```
        RAISE EXCEPTION 'Invalid Employee Number!';
    END IF;

    -- Count projects
    SELECT COUNT(*) INTO v_count
    FROM Project_Employee
    WHERE empno = p_empno;

    RAISE NOTICE 'Total Projects of Employee % : %', p_empno, v_count;
END;
$$
LANGUAGE plpgsql;

//To Call It://

SELECT count_projects_by_employee(101);
```

SET B

Person – Area database
Person (pno int, name varchar (20), birthdate date, income money) Area
(aid int, aname varchar (20), area_type varchar (5))
The person and area related to many to one relationship. The attribute 'area_type' can have values either 'urban' or 'rural'.Create the above database in PostGreSQL and insert sufficient records.

Answer

```
//-- Create a new database//

CREATE DATABASE person_area_db;

-- Drop tables if already exist
DROP TABLE IF EXISTS Person;
DROP TABLE IF EXISTS Area;

-- Area Table
CREATE TABLE Area (
    aid INT PRIMARY KEY,
    aname VARCHAR(20) UNIQUE,
    area_type VARCHAR(5) CHECK (area_type IN ('urban','rural'))
);

-- Person Table
CREATE TABLE Person (
    pno INT PRIMARY KEY,
    name VARCHAR(20),
    birthdate DATE,
    income MONEY,
    aid INT REFERENCES Area(aid)
);

-- Insert Areas
INSERT INTO Area VALUES
```

```
(1, 'Pune', 'urban'),
(2, 'Shirur', 'rural'),
(3, 'Mumbai', 'urban'),
(4, 'Satara', 'rural');

-- Insert Persons
INSERT INTO Person VALUES
(101, 'Amit', '1995-05-10', 50000, 1),
(102, 'Sneha', '1998-07-15', 45000, 1),
(103, 'Rohan', '1992-03-20', 30000, 2),
(104, 'Priya', '1990-11-25', 75000, 3),
(105, 'Kiran', '1997-01-30', 28000, 4);
```

a. Write a stored function that accepts the area name as an input parameter from the user and displays the details of persons living in that area. Raise an exception if area name is invalid.

Answer

```
CREATE OR REPLACE FUNCTION get_persons_by_area(p_area_name VARCHAR)
RETURNS VOID AS
$$
DECLARE
   rec RECORD;
   v_aid INT;
BEGIN
   -- Check if area exists
   SELECT aid INTO v_aid
   FROM Area
   WHERE TRIM(aname) = TRIM(p_area_name);

   IF v_aid IS NULL THEN
      RAISE EXCEPTION 'Invalid Area Name!';
   END IF;

   -- Display persons living in that area
   FOR rec IN
      SELECT pno, name, birthdate, income
      FROM Person
      WHERE aid = v_aid
   LOOP
      RAISE NOTICE 'PNo: %, Name: %, Birthdate: %, Income: %',
                 rec.pno, rec.name, rec.birthdate, rec.income;
   END LOOP;
END;
$$
LANGUAGE plpgsql;

//To Call It://

SELECT get_persons_by_area('Pune');
```

SET C

Wholesaler – Product database
Consider the following entities and their relationships.
Wholesaler (w_no, w_name, address, city)
Product (product_no, product_name, rate)
Relation between Wholesaler and Product is Many to Many with quantity as descriptive attribute.
Create the above database in PostGreSQL and insert sufficient records.

Answer

```
-- Table: Wholesaler
CREATE TABLE Wholesaler (
    w_no SERIAL PRIMARY KEY,
    w_name VARCHAR(100) NOT NULL,
    address TEXT,
    city VARCHAR(50)
);

-- Table: Product
CREATE TABLE Product (
    product_no SERIAL PRIMARY KEY,
    product_name VARCHAR(100) NOT NULL,
    rate NUMERIC(10,2) NOT NULL CHECK (rate > 0)
);

-- Junction Table: Wholesaler_Product
CREATE TABLE Wholesaler_Product (
    w_no INT REFERENCES Wholesaler(w_no),
    product_no INT REFERENCES Product(product_no),
    quantity INT NOT NULL,
    PRIMARY KEY (w_no, product_no)
);

-- Sample Wholesalers
INSERT INTO Wholesaler (w_name, address, city) VALUES
('Alpha Traders', '123 Main St', 'New York'),
('Beta Distributors', '456 Market Rd', 'Chicago'),
('Gamma Wholesalers', '789 Central Ave', 'Los Angeles');

-- Sample Products
INSERT INTO Product (product_name, rate) VALUES
('Laptop', 1200.50),
('Mobile Phone', 650.00),
('Printer', 300.00),
('Router', 80.00);

-- Sample Many-to-Many Records
INSERT INTO Wholesaler_Product (w_no, product_no, quantity) VALUES
(1, 1, 100),
(1, 2, 150),
(2, 3, 80),
```

(2, 4, 60),
(3, 1, 70),
(3, 4, 120);

a. Write a function to accept quantity from user. Quantity must be within range 50-200. If user enters
the quantity out of range, then raise a user defined exception “quantity_out_of _range” otherwise enter the record in table.

Answer

```
-- Step 1: Create user-defined exception for quantity
DO $$
BEGIN
   -- Drop type if already exists
   BEGIN
      DROP TYPE quantity_out_of_range;
   EXCEPTION WHEN undefined_object THEN
      NULL;
   END;

   CREATE TYPE quantity_out_of_range AS ENUM ('Quantity must be between 50 and
200');
END$$;

-- Step 2: Function to insert quantity
CREATE OR REPLACE FUNCTION insert_quantity(p_w_no INT, p_product_no INT,
p_quantity INT)
RETURNS TEXT AS $$
BEGIN
   IF p_quantity < 50 OR p_quantity > 200 THEN
      RAISE EXCEPTION 'quantity_out_of_range: %', p_quantity;
   ELSE
      INSERT INTO Wholesaler_Product (w_no, product_no, quantity)
      VALUES (p_w_no, p_product_no, p_quantity);
      RETURN 'Record inserted successfully';
   END IF;
END;
$$ LANGUAGE plpgsql;

//To Call It://

SELECT insert_quantity(1, 3, 120);  -- valid
SELECT insert_quantity(1, 3, 250);  -- raises quantity_out_of_range
```

b. Write a function which accept rate from user. If user enters rate less than or equal to zero then raise an user defined exception “Invalid_Rate_Value” otherwise display message “Correct Input”.

Answer

```
CREATE OR REPLACE FUNCTION check_rate(p_rate NUMERIC)
RETURNS TEXT AS $$
BEGIN
```

```
   IF p_rate <= 0 THEN
      RAISE EXCEPTION 'Invalid_Rate_Value: %', p_rate;
   ELSE
      RETURN 'Correct Input';
   END IF;
END;
$$ LANGUAGE plpgsql;
```

//To Call It://

```
SELECT check_rate(100);  -- Correct Input
SELECT check_rate(-5);   -- raises Invalid_Rate_Value
```

c. Write a function to accept product name as parameter. If entered product name is not valid then raise an user defined exception"Invalid_Product_Name" otherwise display product details of Specified product

Answer

```
CREATE FUNCTION get_product_details(v_product_name VARCHAR)
RETURNS TABLE(product_no INT, product_name VARCHAR, rate NUMERIC) AS $$
BEGIN
   -- Check if the product exists
   IF NOT EXISTS (SELECT 1 FROM Product p WHERE p.product_name =
v_product_name) THEN
      RAISE EXCEPTION 'Invalid_Product_Name: %', v_product_name;
   END IF;

   -- Return product details with fully qualified column names
   RETURN QUERY
   SELECT p.product_no, p.product_name, p.rate
   FROM Product p
   WHERE p.product_name = v_product_name;
END;
$$ LANGUAGE plpgsql;
```

//To Call It://

```
-- Valid product
SELECT * FROM get_product_details('Laptop');

-- Invalid product
SELECT * FROM get_product_details('Tablet');  -- raises Invalid_Product_Name
```

SET D

Student Teacher Database
Student (sno integer, s_name char(30), s_class char(10), s_addr Char(50))
Teacher (tno integer, t_name char (20), qualification char (15), experience integer)

The relationship is as follows:
Student-Teacher: Many to Many with descriptive attribute Subject.
a. Write a stored function to count the number of the teachers teaching to a student named “ ”. (Accept student name as input parameter). Raise an exception if student name does not exist
b. Write a stored function to count the number of the students who are studying subject named “ ” (Accept subject name as input parameter). Display error message if subject name is not valid.
c. Write a stored function to display teacher details who have qualification as “ ” (Accept teacher’s qualification as input parameter). Raise an exception for invalid qualification
Answer

```
-- ================================
-- 1□ Drop existing tables safely
-- ================================
DROP TABLE IF EXISTS Student_Teacher CASCADE;
DROP TABLE IF EXISTS Student CASCADE;
DROP TABLE IF EXISTS Teacher CASCADE;

-- ================================
-- 2□ Create Tables
-- ================================

-- Teacher Table
CREATE TABLE Teacher (
   tno INT PRIMARY KEY,
   t_name CHAR(20),
   qualification CHAR(15),
   experience INT
);

-- Student Table
CREATE TABLE Student (
   sno INT PRIMARY KEY,
   s_name CHAR(30),
   s_class CHAR(10),
   s_addr CHAR(50)
);

-- Junction Table: Student_Teacher (Many-to-Many with Subject)
CREATE TABLE Student_Teacher (
   sno INT REFERENCES Student(sno),
   tno INT REFERENCES Teacher(tno),
   subject CHAR(20),
   PRIMARY KEY (sno, tno, subject)
);

-- ================================
-- 3□ Insert Sample Data
-- ================================

-- Students
```

```
INSERT INTO Student VALUES
(1, 'Amit', '10A', 'Pune'),
(2, 'Sneha', '10B', 'Mumbai'),
(3, 'Rohan', '10A', 'Nashik'),
(4, 'Priya', '10C', 'Pune');

-- Teachers
INSERT INTO Teacher VALUES
(101, 'Mr. Sharma', 'MSc', 10),
(102, 'Ms. Gupta', 'BEd', 5),
(103, 'Mr. Iyer', 'MSc', 12),
(104, 'Ms. Roy', 'PhD', 8);

-- Student-Teacher Relationships
INSERT INTO Student_Teacher VALUES
(1,101,'Math'),
(1,102,'English'),
(2,102,'English'),
(3,103,'Science'),
(4,104,'Math'),
(2,101,'Math'),
(3,101,'Math');

-- ================================
-- 4□ Function (a) Count Teachers for a Student
-- ================================
CREATE OR REPLACE FUNCTION count_teachers_by_student(p_student_name
CHAR(30))
RETURNS VOID AS
$$
DECLARE
   v_sno INT;
   v_count INT;
BEGIN
   -- Check if student exists
   SELECT sno INTO v_sno
   FROM Student
   WHERE TRIM(s_name) = TRIM(p_student_name);

   IF v_sno IS NULL THEN
      RAISE EXCEPTION 'Invalid Student Name: %', p_student_name;
   END IF;

   -- Count teachers teaching this student
   SELECT COUNT(DISTINCT tno) INTO v_count
   FROM Student_Teacher
   WHERE sno = v_sno;

   RAISE NOTICE 'Total Teachers teaching %: %', p_student_name, v_count;
END;
$$
LANGUAGE plpgsql;
```

```
-- ================================
-- 5□ Function (b) Count Students for a Subject
-- ================================
CREATE OR REPLACE FUNCTION count_students_by_subject(p_subject CHAR(20))
RETURNS VOID AS
$$
DECLARE
   v_count INT;
BEGIN
   -- Count students for the subject
   SELECT COUNT(DISTINCT sno) INTO v_count
   FROM Student_Teacher
   WHERE TRIM(subject) = TRIM(p_subject);

   IF v_count = 0 THEN
      RAISE EXCEPTION 'Invalid Subject Name: %', p_subject;
   END IF;

   RAISE NOTICE 'Total Students studying %: %', p_subject, v_count;
END;
$$
LANGUAGE plpgsql;

-- ================================
-- 6□ Function (c) Display Teacher Details by Qualification
-- ================================
CREATE OR REPLACE FUNCTION get_teachers_by_qualification(p_qualification
CHAR(15))
RETURNS VOID AS
$$
DECLARE
   rec RECORD;
   v_found BOOLEAN := FALSE;
BEGIN
   -- Loop through teachers with the qualification
   FOR rec IN
      SELECT tno, t_name, qualification, experience
      FROM Teacher
      WHERE TRIM(qualification) = TRIM(p_qualification)
   LOOP
      RAISE NOTICE 'TNo: %, Name: %, Qualification: %, Experience: %',
               rec.tno, rec.t_name, rec.qualification, rec.experience;
      v_found := TRUE;
   END LOOP;

   -- If no teacher found, raise exception
   IF NOT v_found THEN
      RAISE EXCEPTION 'Invalid Qualification: %', p_qualification;
   END IF;
END;
$$
```

```
LANGUAGE plpgsql;

-- ==================================
-- 7□ Example Calls
-- ==================================

-- a) Count teachers for a student
-- SELECT count_teachers_by_student('Amit');
-- SELECT count_teachers_by_student('Unknown'); -- Will raise exception

-- b) Count students for a subject
-- SELECT count_students_by_subject('Math');
-- SELECT count_students_by_subject('History'); -- Will raise exception

-- c) Display teachers by qualification
-- SELECT get_teachers_by_qualification('MSc');
-- SELECT get_teachers_by_qualification('MBA'); -- Will raise exception
```

SET E

Railway Reservation System Database
TRAIN: (train_no int, train_name varchar(20), depart_time time , arrival_time time, source_stn varchar (20),dest_stn varchar (20), no_of_res_bogies int ,bogie_capacity int)
PASSENGER : (passenger_id int, passenger_name varchar(20), address varchar(30), age int ,gender char)
Relationships:
Train _Passenger: Many to Many relationship named ticket with descriptive attributes as follows
TICKET: (train_no int, passenger_id int, ticket_no int ,bogie_no int, no_of_berths int ,tdate date , ticket_amt decimal(7,2),status char)
Constraints: The status of a berth can be 'W' (waiting) or 'C' (confirmed).
a. Write a stored function to print the details of train wise confirmed bookings on date
“ ” (Accept date as input parameter).Raise an error in case of invalid date.
b. Write a stored function to accept date and passenger name and display no of berths reserved and ticket amount paid by him. Raise exception if passenger name is invalid.
c. Write a stored function to display the ticket details of a train. (Accept train name as input parameter).Raise an exception in case of invalid train name.

Answer

```
-- ==========================================
-- 1□ Drop Existing Tables (Clean Setup)
-- ==========================================
DROP TABLE IF EXISTS Ticket CASCADE;
DROP TABLE IF EXISTS Passenger CASCADE;
DROP TABLE IF EXISTS Train CASCADE;

-- ==========================================
-- 2□ Create Tables
-- ========================================
-- Train Table
CREATE TABLE Train (
   train_no INT PRIMARY KEY,
   train_name VARCHAR(20) UNIQUE,
   depart_time TIME,
   arrival_time TIME,
   source_stn VARCHAR(20),
   dest_stn VARCHAR(20),
   no_of_res_bogies INT,
   bogie_capacity INT
);

-- Passenger Table
CREATE TABLE Passenger (
   passenger_id INT PRIMARY KEY,
   passenger_name VARCHAR(20),
   address VARCHAR(30),
   age INT,
   gender CHAR(1)
);

-- Ticket Table (Many-to-Many)
CREATE TABLE Ticket (
   train_no INT REFERENCES Train(train_no),
   passenger_id INT REFERENCES Passenger(passenger_id),
   ticket_no INT PRIMARY KEY,
   bogie_no INT,
   no_of_berths INT,
   tdate DATE,
   ticket_amt DECIMAL(7,2),
   status CHAR(1) CHECK (status IN ('W','C'))
);

-- ========================================
-- 3□ Insert Sample Data
-- ========================================
-- Trains
INSERT INTO Train VALUES
(101,'Express1','08:00','14:00','Pune','Mumbai',5,72),
(102,'Express2','09:30','16:00','Mumbai','Delhi',8,80),
(103,'Express3','07:00','12:00','Delhi','Chennai',6,60);

-- Passengers
INSERT INTO Passenger VALUES
(1,'Amit','Pune',30,'M'),
(2,'Sneha','Mumbai',25,'F'),
(3,'Rohan','Delhi',28,'M'),
(4,'Priya','Chennai',22,'F');

-- Tickets
INSERT INTO Ticket VALUES
```

```
(101,1,1001,1,2,'2026-02-16',1500.00,'C'),
(101,2,1002,2,1,'2026-02-16',750.00,'W'),
(102,3,1003,1,3,'2026-02-16',2250.00,'C'),
(102,4,1004,2,2,'2026-02-16',1500.00,'C'),
(103,1,1005,1,1,'2026-02-16',800.00,'W');

-- ========================================
-- 4□ Function (a) – Confirmed Bookings by Date
-- ========================================
CREATE OR REPLACE FUNCTION confirmed_bookings_by_date(p_date DATE)
RETURNS VOID AS
$$
DECLARE
   rec RECORD;
BEGIN
   IF NOT EXISTS (SELECT 1 FROM Ticket WHERE tdate = p_date AND status='C')
THEN
       RAISE EXCEPTION 'No confirmed bookings found on date: %', p_date;
   END IF;

   FOR rec IN
       SELECT t.train_no, t.train_name, p.passenger_name, tk.ticket_no, tk.bogie_no,
tk.no_of_berths, tk.ticket_amt
       FROM Ticket tk
       JOIN Train t ON tk.train_no = t.train_no
       JOIN Passenger p ON tk.passenger_id = p.passenger_id
       WHERE tk.tdate = p_date AND tk.status='C'
   LOOP
       RAISE NOTICE 'Train No: %, Train Name: %, Passenger: %, Ticket No: %, Bogie: %,
Berths: %, Amount: %',
                    rec.train_no, rec.train_name, rec.passenger_name, rec.ticket_no, rec.bogie_no,
rec.no_of_berths, rec.ticket_amt;
   END LOOP;
END;
$$
LANGUAGE plpgsql;

-- ========================================
-- 5□ Function (b) – Passenger Booking Info
-- ========================================
CREATE OR REPLACE FUNCTION passenger_booking_info(p_date DATE,
p_passenger_name VARCHAR)
RETURNS VOID AS
$$
DECLARE
   v_passenger_id INT;
   rec RECORD;
BEGIN
   SELECT passenger_id INTO v_passenger_id
   FROM Passenger
   WHERE TRIM(passenger_name) = TRIM(p_passenger_name);
   IF v_passenger_id IS NULL THEN
      RAISE EXCEPTION 'Invalid Passenger Name: %', p_passenger_name;
   END IF;

   SELECT SUM(no_of_berths) AS total_berths, SUM(ticket_amt) AS total_amount
   INTO rec
   FROM Ticket
   WHERE passenger_id = v_passenger_id AND tdate = p_date;

   IF rec.total_berths IS NULL THEN
      RAISE NOTICE 'No bookings found for % on %', p_passenger_name, p_date;
   ELSE
      RAISE NOTICE 'Passenger: %, Total Berths: %, Total Amount Paid: %',
                p_passenger_name, rec.total_berths, rec.total_amount;
   END IF;
END;
$$
LANGUAGE plpgsql;

-- ==========================================
-- 6 Function (c) – Ticket Details by Train Name
-- ==========================================
CREATE OR REPLACE FUNCTION ticket_details_by_train(p_train_name VARCHAR)
RETURNS VOID AS
$$
DECLARE
   rec RECORD;
BEGIN
   IF NOT EXISTS (SELECT 1 FROM Train WHERE TRIM(train_name) =
TRIM(p_train_name)) THEN
      RAISE EXCEPTION 'Invalid Train Name: %', p_train_name;
   END IF;

   FOR rec IN
      SELECT t.train_no, t.train_name, p.passenger_name, tk.ticket_no, tk.bogie_no,
tk.no_of_berths, tk.tdate, tk.ticket_amt, tk.status
      FROM Ticket tk
      JOIN Train t ON tk.train_no = t.train_no
      JOIN Passenger p ON tk.passenger_id = p.passenger_id
      WHERE TRIM(t.train_name) = TRIM(p_train_name)
   LOOP
      RAISE NOTICE 'Train No: %, Train Name: %, Passenger: %, Ticket No: %, Bogie: %,
Berths: %, Date: %, Amount: %, Status: %',
                rec.train_no, rec.train_name, rec.passenger_name, rec.ticket_no, rec.bogie_no,
rec.no_of_berths, rec.tdate, rec.ticket_amt, rec.status;
   END LOOP;
END;
$$
LANGUAGE plpgsql;

-- ==========================================
-- 7 Example Calls
-- ==========================================
-- a) Confirmed bookings on a date
SELECT confirmed_bookings_by_date('2026-02-16');
-- SELECT confirmed_bookings_by_date('2026-02-15'); -- Invalid date test

-- b) Passenger booking info
SELECT passenger_booking_info('2026-02-16','Amit');
-- SELECT passenger_booking_info('2026-02-16','Unknown'); -- Invalid passenger test

-- c) Ticket details by train
SELECT ticket_details_by_train('Express2');
-- SELECT ticket_details_by_train('UnknownTrain'); -- Invalid train test
```

Assignment 5: Triggers.

SET A

Movie – Actor Database
Consider the following database
Movie (m_name varchar (25), release_year integer, budget money)
Actor (a_name varchar(30), role varchar(30), charges money, a_address varchar(30)) Movie and Actor are related with many to many relationship.
Create the above database in PostGreSQL and insert sufficient records.

Answer

```
//For create database//

CREATE DATABASE movie_actor_db;

-- Drop the junction table first because it references others
DROP TABLE IF EXISTS Movie_Actor;

//-- Drop Movie and Actor tables//

DROP TABLE IF EXISTS Movie;
DROP TABLE IF EXISTS Actor;

//-- Create Movie table//

CREATE TABLE Movie (
   m_name VARCHAR(25) PRIMARY KEY,
   release_year INTEGER,
    budget MONEY
);

-- Create Actor table
CREATE TABLE Actor (
    a_name VARCHAR(30) PRIMARY KEY,
    role VARCHAR(30),
    charges MONEY,
    a_address VARCHAR(30)
);

-- Create junction table for many-to-many relationship
CREATE TABLE Movie_Actor (
    m_name VARCHAR(25) REFERENCES Movie(m_name) ON DELETE CASCADE,
    a_name VARCHAR(30) REFERENCES Actor(a_name) ON DELETE CASCADE,
    PRIMARY KEY (m_name, a_name)
);

-- Insert movies
INSERT INTO Movie (m_name, release_year, budget) VALUES
('Inception', 2010, 160000000),
('Titanic', 1997, 200000000),
('Interstellar', 2014, 165000000),
('Indie Movie', 2025, 50000);  -- Low budget example

-- Insert actors
INSERT INTO Actor (a_name, role, charges, a_address) VALUES
('Leonardo DiCaprio', 'Lead', 20000000, 'LA'),
('Joseph Gordon-Levitt', 'Supporting', 5000000, 'LA'),
('Kate Winslet', 'Lead', 15000000, 'UK'),
('Matthew McConaughey', 'Lead', 10000000, 'TX');

-- Link movies and actors
INSERT INTO Movie_Actor (m_name, a_name) VALUES
('Inception', 'Leonardo DiCaprio'),
('Inception', 'Joseph Gordon-Levitt'),
('Titanic', 'Leonardo DiCaprio'),
('Titanic', 'Kate Winslet'),
('Interstellar', 'Matthew McConaughey');
```

a. Write a trigger which will be executed whenever an actor is deleted from the actor table, display appropriate message.

Answer

```
-- Create the function
CREATE OR REPLACE FUNCTION actor_delete_trigger()
RETURNS TRIGGER AS $$
BEGIN
    RAISE NOTICE 'Actor "%" will be deleted!', OLD.a_name;
    RETURN OLD;
END;
```

```
$$ LANGUAGE plpgsql;

-- Create the trigger
CREATE TRIGGER before_actor_delete
BEFORE DELETE ON Actor
FOR EACH ROW
EXECUTE FUNCTION actor_delete_trigger();
```

//To Call It://

```
DELETE FROM Actor WHERE a_name='Joseph Gordon-Levitt';
```

b. Write a trigger which will be executed whenever a movie is deleted from the movie table, display appropriate message.

Answer

```
-- Create the function
CREATE OR REPLACE FUNCTION movie_delete_trigger()
RETURNS TRIGGER AS $$
BEGIN
   RAISE NOTICE 'Movie "%" will be deleted!', OLD.m_name;
   RETURN OLD;
END;
$$ LANGUAGE plpgsql;

-- Create the trigger
CREATE TRIGGER before_movie_delete
BEFORE DELETE ON Movie
FOR EACH ROW
EXECUTE FUNCTION movie_delete_trigger();
```

//To Call It://

```
DELETE FROM Movie WHERE m_name='Inception';
```

c. Write a trigger which will be executed whenever insertion is made to the movie table. If the budget is less than 1,00,000 do not allow the insertion. Give appropriate message

Answer

```
-- Create the function
CREATE OR REPLACE FUNCTION movie_insert_trigger()
RETURNS TRIGGER AS $$
BEGIN
   IF NEW.budget < 100000 THEN
      RAISE EXCEPTION 'Cannot insert movie "%" with budget less than 100,000',
NEW.m_name;
   END IF;
   RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

```
-- Create the trigger
CREATE TRIGGER before_movie_insert
BEFORE INSERT ON Movie
FOR EACH ROW
EXECUTE FUNCTION movie_insert_trigger();

//To Call It://

-- This will fail
INSERT INTO Movie (m_name, release_year, budget) VALUES ('Low Budget Film', 2026,
50000);
-- ERROR:  Cannot insert movie "Low Budget Film" with budget less than 100,000
```

SET B

Doctor – Hospital Database
Consider the following database
Doctor (d_no int, d_name varchar(30), specialization varchar(35), charges int)
Hospital (h_no int, h_name varchar(20), city varchar(10))
Doctor and Hospital are related with many to one relationship.

```
-- Drop tables if they exist (for clean run)
DROP TABLE IF EXISTS Doctor CASCADE;
DROP TABLE IF EXISTS Hospital CASCADE;

-- Hospital Table
CREATE TABLE Hospital (
   h_no INT PRIMARY KEY,
   h_name VARCHAR(20),
   city VARCHAR(10)
);

-- Doctor Table (Many-to-One relationship with Hospital)
CREATE TABLE Doctor (
   d_no INT PRIMARY KEY,
   d_name VARCHAR(30),
   specialization VARCHAR(35),
   charges INT,
   h_no INT REFERENCES Hospital(h_no)
);
-- Insert Hospitals
INSERT INTO Hospital (h_no, h_name, city) VALUES
(101, 'City Hospital', 'NYC'),
(102, 'Greenwood Clinic', 'LA'),
(103, 'Sunrise Medical', 'Chicago');

-- Insert Doctors
INSERT INTO Doctor (d_no, d_name, specialization, charges, h_no) VALUES
(1, 'Dr. Smith', 'Cardiology', 300, 101),
(2, 'Dr. Johnson', 'Neurology', 350, 102),
```

(3, 'Dr. Lee', 'Pediatrics', 250, 103);

Create the above database in PostGreSQL and insert sufficient records.
a. Write a trigger before insert/update on Doctor table. Raise exception if charges are <0.

Answer

```
CREATE OR REPLACE FUNCTION check_charges_nonnegative()
RETURNS TRIGGER AS $$
BEGIN
   IF NEW.charges < 0 THEN
      RAISE EXCEPTION 'Charges cannot be negative: %', NEW.charges;
   END IF;
   RETURN NEW;
END;
$$ LANGUAGE plpgsql;

-- Trigger before insert or update
CREATE TRIGGER before_doctor_insert_update
BEFORE INSERT OR UPDATE ON Doctor
FOR EACH ROW
EXECUTE FUNCTION check_charges_nonnegative();


//To Call It://

INSERT INTO Doctor (d_no, d_name, specialization, charges, h_no)
VALUES (4, 'Dr. Adams', 'Orthopedics', -50, 101);
-- ERROR: Charges cannot be negative: -50
```

b. Write a trigger that restricts insertion of charges value greater than 400.

Answer

```
CREATE OR REPLACE FUNCTION restrict_charges_max()
RETURNS TRIGGER AS $$
BEGIN
   IF NEW.charges > 400 THEN
      RAISE EXCEPTION 'Charges cannot exceed 400: %', NEW.charges;
   END IF;
   RETURN NEW;
END;
$$ LANGUAGE plpgsql;

-- Trigger before insert or update
CREATE TRIGGER before_doctor_insert_update_max
BEFORE INSERT OR UPDATE ON Doctor
FOR EACH ROW
EXECUTE FUNCTION restrict_charges_max();


//To Call It://
```

```
INSERT INTO Doctor (d_no, d_name, specialization, charges, h_no)
VALUES (5, 'Dr. Brown', 'Dermatology', 450, 102);
-- ERROR: Charges cannot exceed 400: 450
```

SET C

Student – Subject database
Consider the following database :
Student (rollno integer, name varchar(30),city varchar(50),class varchar(10))
Subject(Scode varchar(10),subject name varchar(20))
Student and subject are related with M-M relationship with attributes marks_scored.
Create the above database in PostGreSQL and insert sufficient records

Answer

```
-- Drop tables if they exist
DROP TABLE IF EXISTS Student_Subject CASCADE;
DROP TABLE IF EXISTS Student CASCADE;
DROP TABLE IF EXISTS Subject CASCADE;

-- Student Table
CREATE TABLE Student (
   rollno INTEGER PRIMARY KEY,
   name VARCHAR(30),
   city VARCHAR(50),
   class VARCHAR(10)
);

-- Subject Table
CREATE TABLE Subject (
   Scode VARCHAR(10) PRIMARY KEY,
   subject_name VARCHAR(20)
);

-- Junction Table for Many-to-Many relationship with marks_scored
CREATE TABLE Student_Subject (
   rollno INTEGER REFERENCES Student(rollno) ON DELETE CASCADE,
   Scode VARCHAR(10) REFERENCES Subject(Scode) ON DELETE CASCADE,
   marks_scored INTEGER,
   PRIMARY KEY (rollno, Scode)
);

-- Insert Students
INSERT INTO Student (rollno, name, city, class) VALUES
(1, 'Alice', 'Mumbai', '10-A'),
(2, 'Bob', 'Delhi', '10-B'),
(3, 'Charlie', 'Pune', '10-A');

-- Insert Subjects
INSERT INTO Subject (Scode, subject_name) VALUES
```

```
('MATH101', 'Mathematics'),
('PHY101', 'Physics'),
('CHEM101', 'Chemistry');

-- Insert Student_Subject relationships with marks
INSERT INTO Student_Subject (rollno, Scode, marks_scored) VALUES
(1, 'MATH101', 85),
(1, 'PHY101', 90),
(2, 'MATH101', 78),
(2, 'CHEM101', 88),
(3, 'PHY101', 92);
```

a. Write a trigger before insert/update the marks_scored. Raise exception if Marks are negative.

Answer

```
CREATE OR REPLACE FUNCTION check_marks_nonnegative()
RETURNS TRIGGER AS $$
BEGIN
   IF NEW.marks_scored < 0 THEN
      RAISE EXCEPTION 'Marks cannot be negative: %', NEW.marks_scored;
   END IF;
   RETURN NEW;
END;
$$ LANGUAGE plpgsql;

-- Trigger before insert or update on Student_Subject
CREATE TRIGGER before_marks_insert_update
BEFORE INSERT OR UPDATE ON Student_Subject
FOR EACH ROW
EXECUTE FUNCTION check_marks_nonnegative();
```

b. Write a trigger which is executed when insertion is made in the student-subject table. If marks_scored is less than 0, give appropriate message and do not allow the insertion.

Answer

```
CREATE OR REPLACE FUNCTION restrict_negative_marks_insert()
RETURNS TRIGGER AS $$
BEGIN
   IF NEW.marks_scored < 0 THEN
      RAISE EXCEPTION 'Insertion failed: Marks cannot be negative (%).',
NEW.marks_scored;
   END IF;
   RETURN NEW;
END;
```

```
$$ LANGUAGE plpgsql;

-- Trigger before insert on Student_Subject
CREATE TRIGGER before_student_subject_insert
BEFORE INSERT ON Student_Subject
FOR EACH ROW
EXECUTE FUNCTION restrict_negative_marks_insert();
```

c. Write a trigger which will prevent deleting students from ‘Mumbai’ city.

Answer

```
CREATE OR REPLACE FUNCTION prevent_mumbai_student_delete()
RETURNS TRIGGER AS $$
BEGIN
   IF OLD.city = 'Mumbai' THEN
      RAISE EXCEPTION 'Cannot delete students from Mumbai: %', OLD.name;
   END IF;
   RETURN OLD;
END;
$$ LANGUAGE plpgsql;

-- Trigger before delete on Student
CREATE TRIGGER before_student_delete
BEFORE DELETE ON Student
FOR EACH ROW
EXECUTE FUNCTION prevent_mumbai_student_delete();

//Test cases//

-- Test negative marks insertion
INSERT INTO Student_Subject (rollno, Scode, marks_scored) VALUES (2, 'PHY101', -10);
-- ERROR: Insertion failed: Marks cannot be negative (-10)

-- Test deleting Mumbai student
DELETE FROM Student WHERE rollno = 1;
-- ERROR: Cannot delete students from Mumbai: Alice
```

SET D

Customer – Account database
Consider the following database
Customer (cno integer, cname varchar(20), city varchar(20))
Account (a_no int, a_type varchar(10), opening_date date, balance money) Customer
and Account are related with one to many relationship
Create the above database in PostGreSQL and insert sufficient records.

```
-- Drop tables if they exist
DROP TABLE IF EXISTS Account CASCADE;
DROP TABLE IF EXISTS Customer CASCADE;
```

```
-- Customer Table
CREATE TABLE Customer (
    cno INTEGER PRIMARY KEY,
    cname VARCHAR(20),
    city VARCHAR(20)
);

-- Account Table (One-to-Many with Customer)
CREATE TABLE Account (
    a_no INTEGER PRIMARY KEY,
    a_type VARCHAR(10),
    opening_date DATE,
    balance MONEY,
    cno INTEGER REFERENCES Customer(cno)
);

-- Insert Customers
INSERT INTO Customer (cno, cname, city) VALUES
(1, 'Alice', 'Mumbai'),
(2, 'Bob', 'Delhi'),
(3, 'Charlie', 'Pune');

-- Insert Accounts
INSERT INTO Account (a_no, a_type, opening_date, balance, cno) VALUES
(101, 'Saving', '2023-01-10', 2000, 1),
(102, 'Current', '2023-02-15', 5000, 2),
(103, 'Saving', '2023-03-20', 3000, 3);
```

a. Write a trigger which is executed whenever update is made to the account table. If the balance becomes less than 1000, print an error message that balance cannot be less than 1000.

Answer

```
CREATE OR REPLACE FUNCTION prevent_low_balance_update()
RETURNS TRIGGER AS $$
BEGIN
    IF NEW.balance::numeric < 1000 THEN
        RAISE EXCEPTION 'Balance cannot be less than 1000. Current balance: %',
NEW.balance;
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

-- Trigger before UPDATE
CREATE TRIGGER before_account_update
BEFORE UPDATE ON Account
FOR EACH ROW
EXECUTE FUNCTION prevent_low_balance_update();

//To Call It://
```

```
UPDATE Account SET balance = 500 WHERE a_no = 101;
-- ERROR: Balance cannot be less than 1000. Current balance: 500
```

b. Write a trigger before deleting an account record from Account table. Raise a notice and display the message "Account record is being deleted."

Answer

```
CREATE OR REPLACE FUNCTION account_delete_notice()
RETURNS TRIGGER AS $$
BEGIN
   RAISE NOTICE 'Account record % is being deleted', OLD.a_no;
   RETURN OLD;
END;
$$ LANGUAGE plpgsql;

-- Trigger before DELETE
CREATE TRIGGER before_account_delete
BEFORE DELETE ON Account
FOR EACH ROW
EXECUTE FUNCTION account_delete_notice();
```

//To Call It://

```
DELETE FROM Account WHERE a_no = 102;
-- NOTICE: Account record 102 is being deleted
```

c. Write a trigger before inserting an account record in Account table and raise exception if balance is <500.

Answer

```
CREATE OR REPLACE FUNCTION prevent_low_balance_insert()
RETURNS TRIGGER AS $$
BEGIN
   IF NEW.balance::numeric < 500 THEN
      RAISE EXCEPTION 'Cannot insert account: balance must be at least 500. Current: %',
NEW.balance;
   END IF;
   RETURN NEW;
END;
$$ LANGUAGE plpgsql;

-- Trigger before INSERT
CREATE TRIGGER before_account_insert
BEFORE INSERT ON Account
FOR EACH ROW
EXECUTE FUNCTION prevent_low_balance_insert();
```

//To Call It://

```
INSERT INTO Account (a_no, a_type, opening_date, balance, cno)
VALUES (104, 'Saving', '2024-01-01', 400, 1);
-- ERROR: Cannot insert account: balance must be at least 500. Current: 400
```

SET E

Project-Employee Database
Consider the following Entities and their Relationships for Project-Employee database
Project (pno integer, pname char (30), ptype char (20), duration integer)
Employee (eno integer, ename char (20), qualification char (15), joining_date date)
Relationship between Project and Employee is many to many with descriptive attribute start_date date, no_of_hours_worked integer.
Constraints: Primary Key, pname should not be null.
Create trigger for the following:

Answer

```
-- Drop tables if they exist
DROP TABLE IF EXISTS Project_Employee CASCADE;
DROP TABLE IF EXISTS Employee CASCADE;
DROP TABLE IF EXISTS Project CASCADE;

-- Project Table
CREATE TABLE Project (
   pno INTEGER PRIMARY KEY,
   pname CHAR(30) NOT NULL,
   ptype CHAR(20),
   duration INTEGER
);

-- Employee Table
CREATE TABLE Employee (
   eno INTEGER PRIMARY KEY,
   ename CHAR(20),
   qualification CHAR(15),
   joining_date DATE
);

-- Many-to-Many Relationship Table with descriptive attributes
CREATE TABLE Project_Employee (
   pno INTEGER REFERENCES Project(pno) ON DELETE CASCADE,
   eno INTEGER REFERENCES Employee(eno) ON DELETE CASCADE,
   start_date DATE,
   no_of_hours_worked INTEGER,
   PRIMARY KEY (pno, eno)
);

-- Insert Projects
INSERT INTO Project (pno, pname, ptype, duration) VALUES
```

```
(1, 'Website Redesign', 'IT', 6),
(2, 'Marketing Campaign', 'Marketing', 3);

-- Insert Employees
INSERT INTO Employee (eno, ename, qualification, joining_date) VALUES
(101, 'Alice', 'MBA', '2022-01-15'),
(102, 'Bob', 'B.Tech', '2023-06-10');

-- Insert Project_Employee relationships
INSERT INTO Project_Employee (pno, eno, start_date, no_of_hours_worked) VALUES
(1, 101, '2022-02-01', 120),
(2, 102, '2023-07-01', 60);
```

a. Write a trigger before inserting into an employee table to check current date should be always greater than joining date. Display appropriate message.

Answer

```
CREATE OR REPLACE FUNCTION check_joining_date()
RETURNS TRIGGER AS $$
BEGIN
   IF NEW.joining_date > CURRENT_DATE THEN
      RAISE EXCEPTION 'Joining date (%) cannot be in the future.', NEW.joining_date;
   END IF;
   RETURN NEW;
END;
$$ LANGUAGE plpgsql;

-- Trigger before INSERT on Employee
CREATE TRIGGER before_employee_insert
BEFORE INSERT ON Employee
FOR EACH ROW
EXECUTE FUNCTION check_joining_date();

//To Call It://

-- Example test:
INSERT INTO Employee (eno, ename, qualification, joining_date) VALUES
(103, 'Charlie', 'MCA', '2026-05-01');
-- ERROR: Joining date (2026-05-01) cannot be in the future.
```

b. Write a trigger before inserting into a project table to check duration should be always greater than zero. Display appropriate message.

Answer

```
CREATE OR REPLACE FUNCTION check_project_duration()
RETURNS TRIGGER AS $$
BEGIN
   IF NEW.duration <= 0 THEN
```

```
        RAISE EXCEPTION 'Project duration (%) must be greater than zero.', NEW.duration;
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

-- Trigger before INSERT on Project
CREATE TRIGGER before_project_insert
BEFORE INSERT ON Project
FOR EACH ROW
EXECUTE FUNCTION check_project_duration();
```

//To Call It://

```
INSERT INTO Project (pno, pname, ptype, duration) VALUES
(3, 'Test Project', 'IT', 0);
-- ERROR: Project duration (0) must be greater than zero.
```

c. Write a trigger before deleting an employee record from employee table.
Raise a notice and display the message “Employee record is being
deleted”

Answer

```
CREATE OR REPLACE FUNCTION employee_delete_notice()
RETURNS TRIGGER AS $$
BEGIN
    RAISE NOTICE 'Employee record % is being deleted.', OLD.eno;
    RETURN OLD;
END;
$$ LANGUAGE plpgsql;

-- Trigger before DELETE on Employee
CREATE TRIGGER before_employee_delete
BEFORE DELETE ON Employee
FOR EACH ROW
EXECUTE FUNCTION employee_delete_notice();
```

//To Call It://

```
DELETE FROM Employee WHERE eno = 101;
-- NOTICE: Employee record 101 is being deleted.
```