Slip no-1

Q.1) Create the following database in 3NF using PostgreSQL. [10M]

Consider the following Bank database which maintains information about its branches, customers and their loan applications.

Branch (Bid integer, brname varchar (30), brcity varchar (10))

Customer (Cno integer, cname varchar (20), caddr varchar (35), city varchar (15))

Loan_application (Lno integer,l_amt_required money,lamtapproved money, l_date date)

Relationship:

Branch, Customer, Loan_application are related with ternary relationship as follows:

Ternary (Bid integer, Cno integer, Lno integer)

Constraints: Primary key, l_amt_required should be greater than zero.

Draw ER diagram and Normalization diagram for above relational schema

Answer.

A. Creating Views:

1. Display names of customers for the 'Pune' branch:

```sql
CREATE VIEW Pune_Customers AS
SELECT c.cname
FROM Customer c
JOIN Ternary t ON c.Cno = t.Cno
JOIN Branch b ON t.Bid = b.Bid
WHERE b.brcity = 'Pune';
```

2. Display names of customers who have taken a loan from the branch in the same city they live:

```sql
CREATE VIEW SameCityLoanCustomers AS
```

```
SELECT c.cname
FROM Customer c
JOIN Ternary t ON c.Cno = t.Cno
JOIN Branch bc ON t.Bid = bc.Bid
JOIN Loan_application la ON t.Lno = la.Lno
WHERE c.city = bc.brcity;
```

B. Writing a Stored Function:

Assuming the function is meant to count the number of customers for a particular branch based on the branch name provided as an input parameter:

```sql
CREATE FUNCTION CountCustomersForBranch(branch_name VARCHAR(30))
RETURNS INT
BEGIN
    DECLARE branch_id INT;
    DECLARE customer_count INT;

    SELECT Bid INTO branch_id FROM Branch WHERE brname = branch_name;

    SELECT COUNT(*) INTO customer_count
    FROM Ternary t
    WHERE t.Bid = branch_id;

    RETURN customer_count;
END;
```

Slip no 2

Answer.

Certainly! Let's solve these tasks step by step:

A. Creating Views:

1. To display customer details who have applied for a loan of Rs. 3,00,000:

```sql
CREATE VIEW Loan300k_Customers AS
SELECT c.*
FROM Customer c
JOIN Ternary t ON c.Cno = t.Cno
JOIN Loan_application la ON t.Lno = la.Lno
WHERE la.l_amt_required = 300000;
```

2. To display loan details from the 'Aundh' branch:

```sql
CREATE VIEW Aundh_Loans AS
SELECT la.*
FROM Loan_application la
JOIN Ternary t ON la.Lno = t.Lno
JOIN Branch b ON t.Bid = b.Bid
WHERE b.brname = 'Aundh';
```

B. Writing a Trigger:

Here's a trigger that validates the loan amount approved against the loan amount required:

```sql
CREATE TRIGGER CheckLoanApprovalAmount
BEFORE INSERT ON Loan_application
FOR EACH ROW
BEGIN
    IF NEW.lamtapproved > NEW.l_amt_required THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Loan amount approved must be less than or equal to the loan amount required.';
    END IF;
END;
```

This trigger will activate before inserting a new row into the `Loan_application` table. It checks if the `lamtapproved` value is greater than the `l_amt_required` value for the new row being inserted. If it is, it raises an error with the appropriate message.

Please adjust the SQL syntax based on the specific database system you're using, as there might be slight variations in syntax between different platforms.

Slip no -3

Q.1) Create the following database in 3NF using PostgresSQL. [10M]

Consider the following database of Bus-Transport System. Many buses run on one route. Drivers are allotted to buses shift-wise.

Bus (Bus_no int , capacity int , depot_name varchar (20))

Route (Route_no int, source varchar (20), destination varchar (20), no_of_stations int)

Driver (Driver_no int, driver_name varchar (20), license_no int, address varchar (20),

age int , salary float)

Relationship:

Bus and Route related with many to one relationship.

Bus and Driver related with many to many relationship with descriptive attributes, Shift – it can be 1 (Morning) or 2 (Evening) and Date_of_duty_allotted.

Constraints: Primary key, License_no must be unique, Bus capacity should not be null.

Draw ER diagram and Normalization diagram for above relational schema.

To create the database in 3NF and address your requirements, we'll start by defining the schema, drawing the ER diagram, and then proceeding to solve the queries and implement the trigger for PostgreSQL.

Answer.

Schema Definition:

```sql
CREATE TABLE Bus (
    Bus_no INT PRIMARY KEY,
    capacity INT NOT NULL,
    depot_name VARCHAR(20)
);

CREATE TABLE Route (
    Route_no INT PRIMARY KEY,
```

```
    source VARCHAR(20),
    destination VARCHAR(20),
    no_of_stations INT
);

CREATE TABLE Driver (
    Driver_no INT PRIMARY KEY,
    driver_name VARCHAR(20),
    license_no INT UNIQUE,
    address VARCHAR(20),
    age INT,
    salary FLOAT
);

CREATE TABLE Bus_Route (
    Bus_no INT REFERENCES Bus(Bus_no),
    Route_no INT REFERENCES Route(Route_no),
    PRIMARY KEY (Bus_no, Route_no)
);

CREATE TABLE Bus_Driver (
    Bus_no INT REFERENCES Bus(Bus_no),
    Driver_no INT REFERENCES Driver(Driver_no),
    Shift INT,
    Date_of_duty_allotted DATE,
    PRIMARY KEY (Bus_no, Driver_no),
    CHECK (Shift IN (1, 2))
);
```

ER Diagram:

```
+--------------+       +--------------+
|    Bus     |       |    Route     |
+--------------+       +--------------+
| Bus_no (PK)  |       | Route_no (PK)|
| capacity     |       | source       |
| depot_name   |       | destination  |
+--------------+       | no_of_stations|
                       +--------------+
     |                    |
     |                    |
     |                    |
     |                    |
+--------------+       +-------------+
|   Driver     |       | Bus_Route   |
+--------------+       +-------------+
| Driver_no(PK)|       | Bus_no (PK, |
| driver_name  |       | Route_no,   |
| license_no   |       | Date_of_duty|
| address      |       |_allotted)   |
| age          |       +-------------+
| salary       |
+--------------+
     |
     |
     |
+--------------+
|  Bus_Driver  |
+--------------+
| Bus_no (PK,  |
| Driver_no,   |
```

```
        | Shift,      |
        | Date_of_duty|
        | _allotted)  |
        +-------------+
```

A. Creating Views:

1. To display driver details working in Morning shift:

```sql
CREATE VIEW MorningShift_Drivers AS
SELECT d.*
FROM Driver d
JOIN Bus_Driver bd ON d.Driver_no = bd.Driver_no
WHERE bd.Shift = 1;
```

2. To display driver details having salary > 20,000:

```sql
CREATE VIEW HighSalary_Drivers AS
SELECT *
FROM Driver
WHERE salary > 20000;
```

B. Writing a Trigger:

```sql
CREATE OR REPLACE FUNCTION CheckDriverAge()
RETURNS TRIGGER AS $$
BEGIN
```

```
    IF NEW.age < 21 OR NEW.age > 30 THEN
        RAISE EXCEPTION 'Invalid Age';
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER ValidateDriverAge
BEFORE INSERT ON Driver
FOR EACH ROW
EXECUTE FUNCTION CheckDriverAge();
```

This trigger ensures that before inserting a new record into the `Driver` table, it checks if the age is between 21 and 30. If not, it raises an exception displaying 'Invalid Age'.

Ensure the PostgreSQL syntax is compatible and adjust if necessary based on your specific PostgreSQL version.

Slip no. 4

Answer.

A. Creating Views:

1. To display details of Bus_no 102 along with details of all drivers who have driven that bus:

```sql
CREATE VIEW Bus102_Drivers AS
SELECT b.*, d.*
FROM Bus b
JOIN Bus_Driver bd ON b.Bus_no = bd.Bus_no
JOIN Driver d ON bd.Driver_no = d.Driver_no
WHERE b.Bus_no = 102;
```

2. To display the route details on which buses of capacity 30 runs:

```sql
CREATE VIEW Route_Capacity30 AS
SELECT r.*
FROM Route r
JOIN Bus_Route br ON r.Route_no = br.Route_no
JOIN Bus b ON br.Bus_no = b.Bus_no
WHERE b.capacity = 30;
```

B. Writing a Stored Function:

```sql
CREATE OR REPLACE FUNCTION GetBusDrivers(bus_number INT, check_date DATE)
RETURNS VOID AS $$
```

```
DECLARE
    driver_details TEXT;
BEGIN
    SELECT STRING_AGG(d.driver_name, ', ')
    INTO driver_details
    FROM Driver d
    JOIN Bus_Driver bd ON d.Driver_no = bd.Driver_no
    WHERE bd.Bus_no = bus_number AND bd.Date_of_duty_allotted = check_date;

    IF driver_details IS NULL THEN
        RAISE EXCEPTION 'Invalid Bus Number';
    ELSE
        RAISE NOTICE 'Drivers for Bus % on %: %', bus_number, check_date, driver_details;
    END IF;
END;
$$ LANGUAGE plpgsql;
```

This function `GetBusDrivers` accepts a bus number and a date as input parameters. It retrieves the details of drivers allotted to the specified bus on the given date and prints their names. If the bus number is invalid or no drivers are found for that bus on that date, it raises an exception.

Ensure the PostgreSQL syntax is compatible and adjust if necessary based on your specific PostgreSQL version.

Slip no 5

Create the following database in 3NF using PostgresSQL. [10M]

Consider the following Project-Employee database, which is managed by a company and stores the details of projects assigned to employees.

Project (Pno int, pname varchar (30), ptype varchar (20), duration integer)

Employee (Eno integer, ename varchar (20), qualification char (15), joining_date date)

Relationship:

Project-Employee related with many-to-many relationship, with descriptive attributes as start_date_of_Project, no_of_hours_worked.

Constraints: Primary key, pname should not be null.

Draw ER diagram and Normalization diagram for above relational schema.

Using above database solve the following questions.

A. Create a View [4M]

1. To display the project name, project type, and project start date, sorted by project start date.

2. To display the name and qualification of the employee, sorted by the employee name.

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

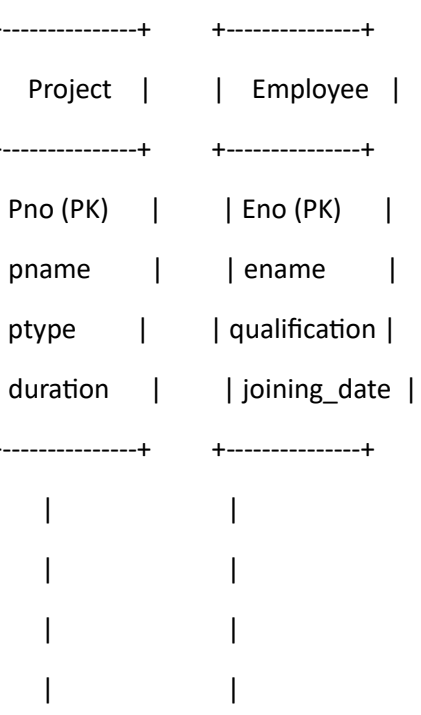
Answer.

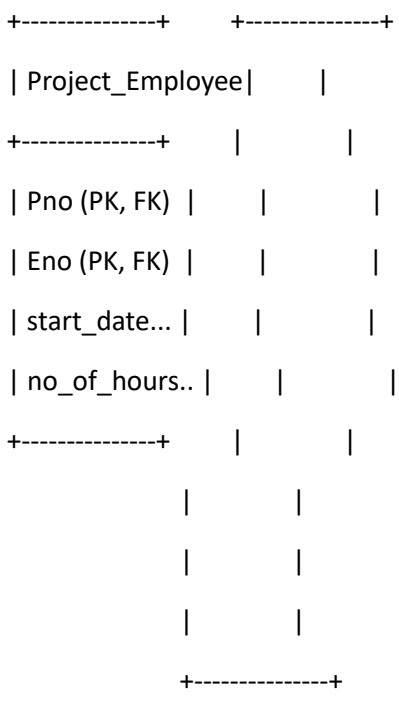
Schema Definition:

```sql
CREATE TABLE Project (
    Pno INT PRIMARY KEY,
    pname VARCHAR(30) NOT NULL,
    ptype VARCHAR(20),
    duration INTEGER
);
```

```
CREATE TABLE Employee (
   Eno INTEGER PRIMARY KEY,
   ename VARCHAR(20),
   qualification CHAR(15),
   joining_date DATE
);

CREATE TABLE Project_Employee (
   Pno INT REFERENCES Project(Pno),
   Eno INT REFERENCES Employee(Eno),
   start_date_of_Project DATE,
   no_of_hours_worked INTEGER,
   PRIMARY KEY (Pno, Eno)
);
```

ER Diagram:





A. Creating Views:

1. To display the project name, project type, and project start date, sorted by project start date:

```sql
CREATE VIEW ProjectDetails AS
SELECT pname, ptype, start_date_of_Project
FROM Project_Employee pe
JOIN Project p ON pe.Pno = p.Pno
ORDER BY start_date_of_Project;
```

2. To display the name and qualification of the employee, sorted by the employee name:

```sql
CREATE VIEW EmployeeDetails AS
SELECT ename, qualification
FROM Employee
ORDER BY ename;
```

```

### B. Writing a Trigger:

```sql
CREATE OR REPLACE FUNCTION CheckJoiningDate()
RETURNS TRIGGER AS $$
BEGIN
   IF NEW.joining_date <= CURRENT_DATE THEN
      RAISE EXCEPTION 'Joining date should be greater than current date';
   END IF;
   RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER ValidateJoiningDate
BEFORE INSERT ON Employee
FOR EACH ROW
EXECUTE FUNCTION CheckJoiningDate();
```

This trigger ensures that before inserting a new record into the `Employee` table, it checks if the joining date is greater than the current date. If not, it raises an exception displaying the appropriate message.

Ensure the PostgreSQL syntax is compatible and adjust if necessary based on your specific PostgreSQL version.
```

Slip no 6

Certainly! Let's handle these tasks based on the provided database schema.

Answer.

A. Creating Views:

1. To display employee details and sort by employee’s joining date:

```sql
CREATE VIEW EmployeeDetailsByJoiningDate AS
SELECT *
FROM Employee
ORDER BY joining_date;
```

2. To display project details sorted by project type:

```sql
CREATE VIEW ProjectDetailsByType AS
SELECT *
FROM Project
ORDER BY ptype;
```

B. Writing a Function using Cursor:

```sql
CREATE OR REPLACE FUNCTION GetEmployeesOnProject(project_name VARCHAR)
RETURNS TABLE (employee_name VARCHAR) AS $$
DECLARE
    emp_name VARCHAR;
    proj_name ALIAS FOR $1;
```

```
    emp_cursor CURSOR FOR
        SELECT e.ename
        FROM Employee e
        JOIN Project_Employee pe ON e.Eno = pe.Eno
        JOIN Project p ON pe.Pno = p.Pno
        WHERE p.pname = proj_name;
BEGIN
    OPEN emp_cursor;
    LOOP
        FETCH emp_cursor INTO emp_name;
        EXIT WHEN NOT FOUND;
        RETURN NEXT (emp_name);
    END LOOP;
    CLOSE emp_cursor;
END;
$$ LANGUAGE plpgsql;
```

Slip no 7

Q.1) Create the following database in 3NF using PostgresSQL. [10M]

Consider the following Student-Teacher database maintained by a college. It also gives information of the subject taught by teachers.

Student (Sno integer, sname varchar (20), sclass varchar (10), saddr varchar(30))

Teacher (Tno integer, tname varchar (20), qualification char (15), experience integer)

Relationship: Student-Teacher related with many to many relationship with descriptive attribute Subject.

Constraints: Primary Key, student and teacher name should not be null.

Draw ER diagram and Normalization diagram for above relational schema.

Using above database solve the following questions.

A. Create a View [4M]

1. To display student names who are taught by most experienced teacher.

2. To display subjects taught by each teacher.

B. Write a trigger before update a student's class from student table. Display appropriate message. [6M]

Answer.

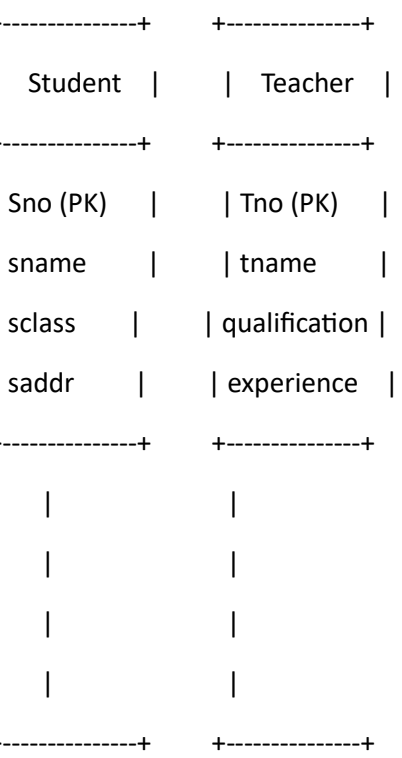
Schema Definition:

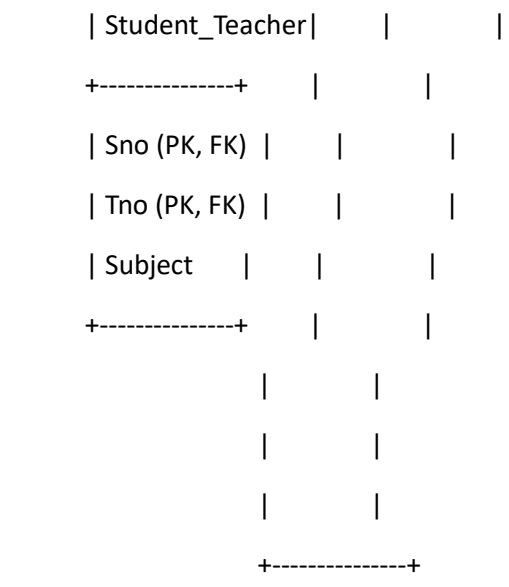
```sql
CREATE TABLE Student (
    Sno INTEGER PRIMARY KEY,
    sname VARCHAR(20) NOT NULL,
    sclass VARCHAR(10),
    saddr VARCHAR(30)
);
```

```
CREATE TABLE Teacher (
    Tno INTEGER PRIMARY KEY,
    tname VARCHAR(20) NOT NULL,
    qualification CHAR(15),
    experience INTEGER
);

CREATE TABLE Student_Teacher (
    Sno INTEGER REFERENCES Student(Sno),
    Tno INTEGER REFERENCES Teacher(Tno),
    Subject VARCHAR(30),
    PRIMARY KEY (Sno, Tno),
    CHECK (Subject IS NOT NULL)
);
```

ER Diagram:






```

### A. Creating Views:

#### 1. To display student names who are taught by the most experienced teacher:

```sql
CREATE VIEW StudentsTaughtByMostExperienced AS
SELECT s.sname
FROM Student s
JOIN Student_Teacher st ON s.Sno = st.Sno
JOIN Teacher t ON st.Tno = t.Tno
WHERE t.experience = (SELECT MAX(experience) FROM Teacher);
```

2. To display subjects taught by each teacher:

```sql
CREATE VIEW SubjectsTaughtByTeachers AS
SELECT t.tname AS teacher_name, ARRAY_AGG(st.Subject) AS subjects_taught
FROM Teacher t
LEFT JOIN Student_Teacher st ON t.Tno = st.Tno
GROUP BY t.tname;
```
```

```

### B. Writing a Trigger:

```sql
CREATE OR REPLACE FUNCTION PreventUpdateStudentClass()
RETURNS TRIGGER AS $$
BEGIN
   IF NEW.sclass <> OLD.sclass THEN
      RAISE EXCEPTION 'Class update not allowed.';
   END IF;
   RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER PreventStudentClassUpdate
BEFORE UPDATE ON Student
FOR EACH ROW
EXECUTE FUNCTION PreventUpdateStudentClass();
```

This trigger `PreventStudentClassUpdate` ensures that an update to the student's class in the `Student` table is not allowed. If an attempt is made to update the student's class, it raises an exception with an appropriate message.

Ensure the PostgreSQL syntax is compatible and adjust if necessary based on your specific PostgreSQL version.
```

Slip no 9

Using above database solve the following questions.

A. Create a View [4M]

1. To display details of teachers having experience > 5 years.

2. To display details of teachers whose name start with the letter ‘S’.

B. Write a function to count the number of the teachers who are teaching to a student named ‘_____’. (Accept student name as input parameter). [6M]

Answer.

A. Creating Views:

1. To display details of teachers having experience > 5 years:

```sql
CREATE VIEW ExperiencedTeachers AS
SELECT *
FROM Teacher
WHERE experience > 5;
```

2. To display details of teachers whose name starts with the letter 'S':

```sql
CREATE VIEW TeachersWithNameStartingS AS
SELECT *
FROM Teacher
WHERE tname LIKE 'S%';
```

B. Writing a Function:

```sql
CREATE OR REPLACE FUNCTION CountTeachersForStudent(student_name VARCHAR)
RETURNS INTEGER AS $$
DECLARE
    teacher_count INTEGER;
BEGIN
    SELECT COUNT(*) INTO teacher_count
    FROM Teacher t
    JOIN Student_Teacher st ON t.Tno = st.Tno
    JOIN Student s ON st.Sno = s.Sno
    WHERE s.sname = student_name;

    RETURN teacher_count;
END;
$$ LANGUAGE plpgsql;
```

This function `CountTeachersForStudent` accepts a student's name as input and counts the number of teachers teaching that particular student by joining the `Teacher`, `Student_Teacher`, and `Student` tables. It returns the count of teachers associated with the specified student.

Please ensure the PostgreSQL syntax compatibility and adjust if necessary according to your specific PostgreSQL version.

Slip no. 9

Q.1) Create the following database in 3NF using PostgreSQL. [10M]

Consider the following Student –Marks database

Student (Rollno integer, sname varchar(30), address varchar(50), class varchar(10))

Subject (Scode varchar(10), subject_name varchar(20))

Relationship:

Student-Subject related with many-to-many relationship with attributes marks_scored.

Constraints: Primary key, sname should not be null.

Draw ER diagram and Normalization diagram for above relational schema.

Using above database solve the following questions.

A. Create a View [4M]

1. To display student names of class ‘FYMScCA’.

2. To display students name, subject and marks who scored more than 90 marks.

B. Write a function using cursor, to calculate total marks of each student and display it.

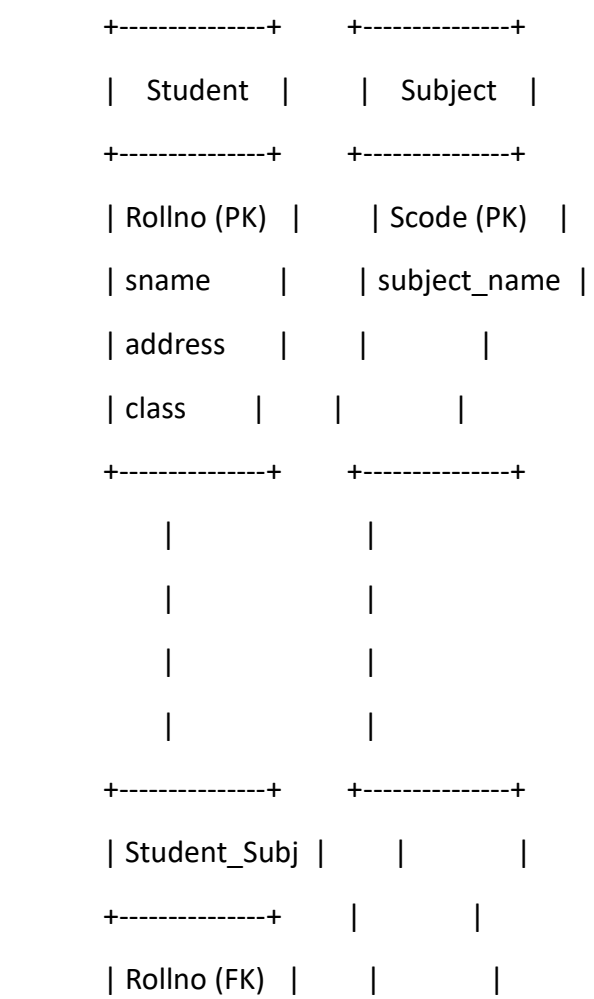
Answer.

Schema Definition:

```sql
CREATE TABLE Student (
    Rollno INTEGER PRIMARY KEY,
    sname VARCHAR(30) NOT NULL,
    address VARCHAR(50),
    class VARCHAR(10)
);

CREATE TABLE Subject (
    Scode VARCHAR(10) PRIMARY KEY,
    subject_name VARCHAR(20)
```

```
);

CREATE TABLE Student_Subject (
    Rollno INTEGER REFERENCES Student(Rollno),
    Scode VARCHAR(10) REFERENCES Subject(Scode),
    marks_scored INTEGER,
    PRIMARY KEY (Rollno, Scode),
    CHECK (marks_scored >= 0 AND marks_scored <= 100)
);
```

ER Diagram:



```
        | Scode (FK)    |        |             |
        | marks_scored  |        |             |
        +---------------+        |             |
                                 |             |
                                 |             |
                                 |             |
                                 +---------------+
```

A. Creating Views:

1. To display student names of class 'FYMScCA':

```sql
CREATE VIEW FYMScCA_Students AS
SELECT sname
FROM Student
WHERE class = 'FYMScCA';
```

2. To display students' names, subjects, and marks who scored more than 90 marks:

```sql
CREATE VIEW HighScorers AS
SELECT s.sname, subj.subject_name, ss.marks_scored
FROM Student_Subject ss
JOIN Student s ON ss.Rollno = s.Rollno
JOIN Subject subj ON ss.Scode = subj.Scode
WHERE ss.marks_scored > 90;
```

B. Writing a Function using Cursor:

```sql
CREATE OR REPLACE FUNCTION CalculateTotalMarks()
RETURNS TABLE (Rollno INTEGER, TotalMarks INTEGER) AS $$
DECLARE
   student_rollno INTEGER;
   total_marks INTEGER;
BEGIN
   FOR student_rollno IN SELECT Rollno FROM Student LOOP
      total_marks := 0;

      FOR subject_mark IN SELECT marks_scored FROM Student_Subject WHERE Rollno = student_rollno LOOP
         total_marks := total_marks + subject_mark;
      END LOOP;

      RETURN NEXT (student_rollno, total_marks);
   END LOOP;
END;
$$ LANGUAGE plpgsql;
```

This function `CalculateTotalMarks` calculates the total marks for each student by iterating over the `Student_Subject` table using cursors and returns a table containing the roll number and total marks for each student.

Ensure the PostgreSQL syntax is compatible and adjust if necessary based on your specific PostgreSQL version.

Slip No. 10

Using above database solve the following questions.

A. Create a View [4M]

1. To display details of students whose name starts with the letter ‘A’.

2. To display student names, subject and marks who scored less than 40 marks.

B. Write a trigger to ensure that the marks entered for a student, with respect to a subject is never < 0 and greater than 100.

Answer.

A. Creating Views:

1. To display details of students whose name starts with the letter 'A':

```sql
CREATE VIEW StudentsWithNameStartingA AS
SELECT *
FROM Student
WHERE sname LIKE 'A%';
```

2. To display student names, subjects, and marks who scored less than 40 marks:

```sql
CREATE VIEW LowScorers AS
SELECT s.sname, subj.subject_name, ss.marks_scored
FROM Student_Subject ss
JOIN Student s ON ss.Rollno = s.Rollno
JOIN Subject subj ON ss.Scode = subj.Scode
WHERE ss.marks_scored < 40;
```

B. Writing a Trigger:

```sql
CREATE OR REPLACE FUNCTION CheckMarks()
RETURNS TRIGGER AS $$
BEGIN
   IF NEW.marks_scored < 0 OR NEW.marks_scored > 100 THEN
      RAISE EXCEPTION 'Marks should be between 0 and 100.';
   END IF;
   RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER CheckMarksBeforeInsertUpdate
BEFORE INSERT OR UPDATE ON Student_Subject
FOR EACH ROW
EXECUTE FUNCTION CheckMarks();
```

Slip No. 11

Q.1) Create the following database in 3NF using PostgreSQL. [10M]

Consider the following database of Movie_Actor_Producer .

Movie (m_name varchar (25), release_year integer, budget money)

Actor (a_name char (30), role char (30), city varchar (30))

Producer (producer_id integer, pname char (30), p_address varchar (30))

Relationship:

Movie and Actor related with many-to-many relationship with descriptive attribute charges.

Producer and Movie related with many-to-many relationship.

Constraints: Primary key, release_year should not be null.

Draw ER diagram and Normalization diagram for above relational schema.

Using above database solve the following questions.

A. Create a View [4M]

1. To display actor names who lives in ‘Mumbai’.

2. To display the actors in each movie.

B. Write a trigger before inserting budget into a movie table. Budget should be minimum 50 lakh. Display appropriate message

Answer.

Schema Definition:

```sql
CREATE TABLE Movie (
    m_name VARCHAR(25) PRIMARY KEY,
    release_year INTEGER NOT NULL,
    budget MONEY CHECK (budget >= 5000000)
);

CREATE TABLE Actor (
```

```
    a_name CHAR(30),
    role CHAR(30),
    city VARCHAR(30)
);

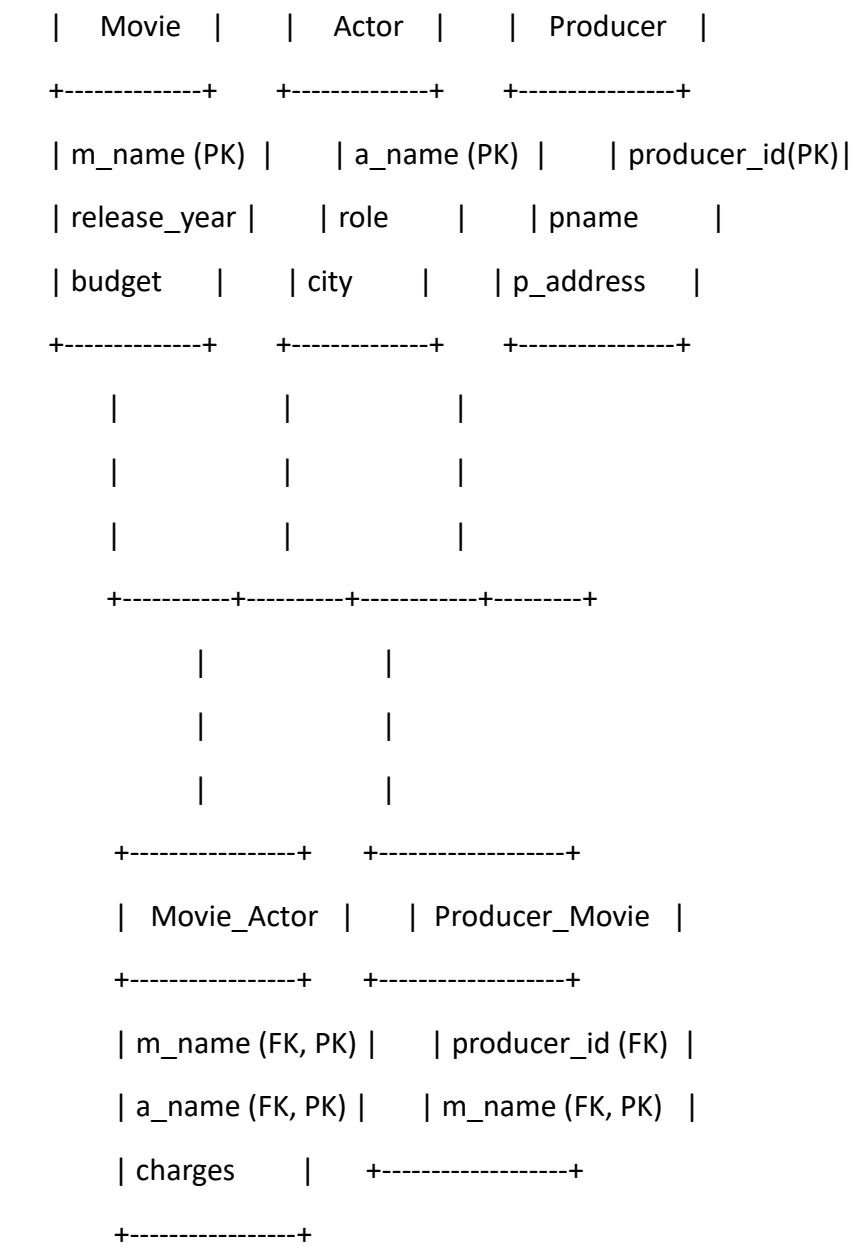
CREATE TABLE Producer (
    producer_id INTEGER,
    pname CHAR(30),
    p_address VARCHAR(30),
    PRIMARY KEY (producer_id)
);

CREATE TABLE Movie_Actor (
    m_name VARCHAR(25) REFERENCES Movie(m_name),
    a_name CHAR(30) REFERENCES Actor(a_name),
    charges MONEY,
    PRIMARY KEY (m_name, a_name)
);

CREATE TABLE Producer_Movie (
    producer_id INTEGER REFERENCES Producer(producer_id),
    m_name VARCHAR(25) REFERENCES Movie(m_name),
    PRIMARY KEY (producer_id, m_name)
);
```

ER Diagram:

```
    +--------------+       +--------------+       +----------------+
```




```

### A. Creating Views:

#### 1. To display actor names who live in 'Mumbai':

```sql
CREATE VIEW ActorsInMumbai AS
SELECT a_name
FROM Actor
```
```

WHERE city = 'Mumbai';
```

#### 2. To display the actors in each movie:

```sql
CREATE VIEW ActorsInMovies AS
SELECT m.m_name, ma.a_name
FROM Movie m
JOIN Movie_Actor ma ON m.m_name = ma.m_name;
```

B. Writing a Trigger:

```sql
CREATE OR REPLACE FUNCTION CheckMinimumBudget()
RETURNS TRIGGER AS $$
BEGIN
   IF NEW.budget < 5000000 THEN
      RAISE EXCEPTION 'Budget should be minimum 50 lakh.';
   END IF;
   RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER CheckBudgetBeforeInsert
BEFORE INSERT ON Movie
FOR EACH ROW
EXECUTE FUNCTION CheckMinimumBudget();
```
```

Slip no. 12

Using above database solve the following questions.

A. Create a View [4M]

1. To display movie names produced by ‘Mr. Subhash Ghai’.

2. To display actor names who do not live in Mumbai or Pune city.

B. Write a function to list movie-wise charges of ‘Amitabh Bachchan’. [6M]

Answer.

A. Creating Views:

1. To display movie names produced by 'Mr. Subhash Ghai':

```sql
CREATE VIEW MoviesBySubhashGhai AS
SELECT m.m_name
FROM Movie m
JOIN Producer_Movie pm ON m.m_name = pm.m_name
JOIN Producer p ON pm.producer_id = p.producer_id
WHERE p.pname = 'Mr. Subhash Ghai';
```

2. To display actor names who do not live in Mumbai or Pune city:

```sql
CREATE VIEW ActorsNotInMumbaiPune AS
SELECT a_name
FROM Actor
WHERE city NOT IN ('Mumbai', 'Pune');
```

B. Writing a Function:

```sql
CREATE OR REPLACE FUNCTION AmitabhBachchanCharges()
RETURNS TABLE (MovieName VARCHAR(25), Charges MONEY) AS $$
BEGIN
   RETURN QUERY SELECT ma.m_name, ma.charges
   FROM Movie_Actor ma
   JOIN Actor a ON ma.a_name = a.a_name
   WHERE a.a_name = 'Amitabh Bachchan';
END;
$$ LANGUAGE plpgsql;
```

Slip no 13

Q.1) Create the following database in 3NF using PostgreSQL. [10M]

Consider the following Person–Area database

Person (pnumber integer, pname varchar (20), birthdate date, income money)

Area (aname varchar (20), area_type varchar (5))

An area can have one or more persons living in it, but a person belongs to exactly one area. The

attribute ‘area_type’ can have values either ‘urban’ or ‘rural’. Create the above database in

PostgreSQL.

Draw ER diagram and Normalization diagram for above relational schema.

Using above database solve the following questions.

A. Create view [4M]

1. List the names of all people living in ‘_____’ area.

2. Display person details sorted by person name.

B. Write a cursor to update the income of all people living in ‘Urban’ area by 10%

Answer.

Schema Definition:

```sql
CREATE TABLE Person (
    pnumber INTEGER PRIMARY KEY,
    pname VARCHAR(20),
    birthdate DATE,
    income MONEY,
    aname VARCHAR(20) REFERENCES Area(aname)
);
```

```
CREATE TABLE Area (
   aname VARCHAR(20) PRIMARY KEY,
   area_type VARCHAR(5)
);
```

ER Diagram:

```
    +--------------+       +--------------+
    |    Person    |       |     Area     |
    +--------------+       +--------------+
    | pnumber (PK) |       | aname (PK)   |
    | pname        |       | area_type    |
    | birthdate    |       +--------------+
    | income       |
    | aname (FK)   |
    +--------------+
```

This ER diagram showcases a one-to-many relationship between `Area` and `Person`, with the `aname` attribute in `Person` being a foreign key referencing `Area`.

A. Creating Views:

1. List the names of all people living in '_____' area:

```sql
CREATE VIEW PeopleInSpecificArea AS
SELECT pname
FROM Person p
JOIN Area a ON p.aname = a.aname
WHERE a.aname = '_____'; -- Replace '_____' with the desired area name
```

```

#### 2. Display person details sorted by person name:

```sql
CREATE VIEW PersonDetailsSorted AS
SELECT *
FROM Person
ORDER BY pname;
```

B. Writing a Cursor to Update Incomes:

```sql
CREATE OR REPLACE FUNCTION UpdateIncomeUrbanArea() RETURNS VOID AS $$
DECLARE
  person_record RECORD;
  urban_area CURSOR FOR SELECT * FROM Person WHERE aname IN (SELECT aname FROM Area WHERE area_type = 'urban');
BEGIN
  OPEN urban_area;
  LOOP
    FETCH urban_area INTO person_record;
    EXIT WHEN NOT FOUND;
    UPDATE Person SET income = income * 1.1 WHERE CURRENT OF urban_area;
  END LOOP;
  CLOSE urban_area;
END;
$$ LANGUAGE plpgsql;
```
```

Slip No 14 .

Using above database solve the following questions.

A. Create view [4M]

1. To display details of all the peoples Whose birthday falls in the month of_____.

2. List names of people according to area_type having minimum income.

B. Write a function to print total number of persons of a particular area. (Accept area_name as

input parameter). Display appropriate message.

Answer.

A. Creating Views:

1. To display details of all the people Whose birthday falls in the month of '_____':

```sql
CREATE VIEW PeopleByBirthdayMonth AS
SELECT *
FROM Person
WHERE EXTRACT(MONTH FROM birthdate) = /* specify month number */;
```

2. List names of people according to area_type having minimum income:

```sql
CREATE VIEW PeopleByAreaMinIncome AS
SELECT pname, area_type, MIN(income) AS min_income
FROM Person p
JOIN Area a ON p.aname = a.aname
GROUP BY pname, area_type;
```

B. Writing a Function:

```sql
CREATE OR REPLACE FUNCTION CountPersonsInArea(area_name VARCHAR)
RETURNS VOID AS $$
DECLARE
   area_count INTEGER;
BEGIN
   SELECT COUNT(*)
   INTO area_count
   FROM Person p
   JOIN Area a ON p.aname = a.aname
   WHERE a.aname = area_name;

   IF area_count > 0 THEN
      RAISE NOTICE 'Total persons in % area: %', area_name, area_count;
   ELSE
      RAISE NOTICE 'No persons found in % area.', area_name;
   END IF;
END;
$$ LANGUAGE plpgsql;
```

Slip No 15

Q.1) Create the following database in 3NF using PostgreSQL. [10M]

Consider the following database maintained by a school about students and competitions.

STUDENT (sreg_no int , name char(30), class char(10))

COMPETITION (c_no int , name char(20), C_type char(15))

The relationship is as follows:

STUDENT-COMPETITION: M-M with described attributes rank and year.

Draw ER diagram and Normalization diagram for above relational schema.

Using above database solve the following questions.

A. Create a view [4M]

1. To display details of competitions and it should be sorted on competition type.

2. To display student name, class, competition name, rank and year. The list should be

sorted by student name.

B. Define a trigger on the relationship table. If the year entered is greater than current year, it

should be changed to current year.

Answer.

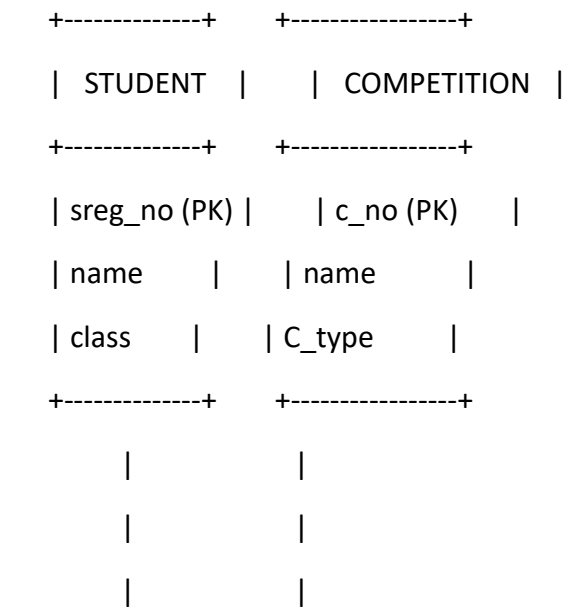
Schema Definition:

```sql
CREATE TABLE STUDENT (
    sreg_no INT PRIMARY KEY,
    name CHAR(30),
    class CHAR(10)
);

CREATE TABLE COMPETITION (
```

```
    c_no INT PRIMARY KEY,
    name CHAR(20),
    C_type CHAR(15)
);

CREATE TABLE STUDENT_COMPETITION (
    sreg_no INT REFERENCES STUDENT(sreg_no),
    c_no INT REFERENCES COMPETITION(c_no),
    rank INT,
    year INT,
    PRIMARY KEY (sreg_no, c_no)
);
```

This schema includes three tables: `STUDENT`, `COMPETITION`, and `STUDENT_COMPETITION`. The latter serves as the relationship table between students and competitions, storing additional attributes like rank and year.

ER Diagram:



```
          +-----------+----------+
                      |
                      |
           +---------+------------+
           | STUDENT_COMPETITION |
           +---------------------+
           | sreg_no (FK, PK)    |
           | c_no (FK, PK)       |
           | rank                |
           | year                |
           +---------------------+
```

A. Creating Views:

1. To display details of competitions and sorted by competition type:

```sql
CREATE VIEW CompetitionsSortedByType AS
SELECT *
FROM COMPETITION
ORDER BY C_type;
```

2. To display student name, class, competition name, rank, and year sorted by student name:

```sql
CREATE VIEW StudentCompetitionDetails AS
SELECT s.name AS student_name, s.class, c.name AS competition_name, sc.rank, sc.year
FROM STUDENT s
JOIN STUDENT_COMPETITION sc ON s.sreg_no = sc.sreg_no
```

```
JOIN COMPETITION c ON sc.c_no = c.c_no
ORDER BY s.name;
```

B. Defining a Trigger:

```sql
CREATE OR REPLACE FUNCTION UpdateYearIfFuture()
RETURNS TRIGGER AS $$
BEGIN
   IF NEW.year > EXTRACT(YEAR FROM CURRENT_DATE) THEN
      NEW.year := EXTRACT(YEAR FROM CURRENT_DATE);
   END IF;
   RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER CheckYearBeforeInsertUpdate
BEFORE INSERT OR UPDATE ON STUDENT_COMPETITION
FOR EACH ROW
EXECUTE FUNCTION UpdateYearIfFuture();
```