**Q1) Consider the following database**

**Doctor (dno, dname, city)**

**Patient (pno, pat\_name, city, disease)**

**The relationship is as follows: Doctor-Patient: many-to-many with Fee as a descriptive attribute.**

**Normalize the above relation schema in 3NF and insert minimum 5 records in each table.**

**Given Relation Schema**

**We are provided with two entities: Doctor and Patient, having a many-to-many relationship, with Fee as the descriptive attribute. The schema details are as follows:**

* **Doctor: (dno, dname, city)**
* **Patient: (pno, pat\_name, city, disease)**
* **Many-to-many relationship with a descriptive attribute Fee.**

**Normalization to 3NF**

**To normalize this schema to 3NF, we will create a separate Doctor-Patient relationship table for the many-to-many relationship with the Fee attribute. The tables in 3NF will be as follows:**

1. **Doctor Table**
   * **Attributes: dno (PK), dname, city**
2. **Patient Table**
   * **Attributes: pno (PK), pat\_name, city, disease**
3. **Doctor\_Patient Table**
   * **Attributes: dno (FK), pno (FK), Fee**
   * **Composite Primary Key: (dno, pno)**

**Tables After Normalization:**

1. **Doctor (dno, dname, city)**
2. **Patient (pno, pat\_name, city, disease)**
3. **Doctor\_Patient (dno, pno, Fee)**

**Explanation**

* **The Doctor and Patient tables store individual entity information.**
* **The Doctor\_Patient table handles the many-to-many relationship between doctors and patients, with Fee as a descriptive attribute.**

**This structure is now in Third Normal Form (3NF):**

* **All non-key attributes in the Doctor and Patient tables depend solely on their respective primary keys.**
* **The composite table Doctor\_Patient ensures that there are no transitive dependencies.**

**Q.2) Execute the following queries in PostGreSQL [any 4]**

**i. List the details of all the doctors from city.**

**ii. Display count of patients.**

**iii. List the names of the patients suffering from 'cancer' disease.**

**iv. Change the city of 'Dr. Patil' to Pune.**

**v. List the names of patients that starts with alphabet 'A'**

**Step 1: Create the Tables**

sql

Copy code

-- Create Doctor Table

CREATE TABLE Doctor (

dno SERIAL PRIMARY KEY,

dname VARCHAR(100) NOT NULL,

city VARCHAR(100) NOT NULL

);

-- Create Patient Table

CREATE TABLE Patient (

pno SERIAL PRIMARY KEY,

pat\_name VARCHAR(100) NOT NULL,

city VARCHAR(100) NOT NULL,

disease VARCHAR(100) NOT NULL

);

-- Create Doctor\_Patient Table (Many-to-Many with Fee)

CREATE TABLE Doctor\_Patient (

dno INT REFERENCES Doctor(dno),

pno INT REFERENCES Patient(pno),

Fee DECIMAL(10, 2),

PRIMARY KEY (dno, pno)

);

-- Insert sample data into Doctor table

INSERT INTO Doctor (dname, city)

VALUES

('Dr. Smith', 'New York'),

('Dr. Johnson', 'Los Angeles'),

('Dr. Lee', 'Chicago'),

('Dr. Brown', 'Houston'),

('Dr. Patil', 'Mumbai'); -- For Update Query

-- Insert sample data into Patient table

INSERT INTO Patient (pat\_name, city, disease)

VALUES

('John Doe', 'New York', 'Flu'),

('Jane Roe', 'Los Angeles', 'Diabetes'),

('Mark Twain', 'Chicago', 'Hypertension'),

('Lucy Hale', 'Houston', 'Asthma'),

('Alice Smith', 'Miami', 'Cancer'), -- For Query iii

('Alan Walker', 'San Diego', 'Cancer'), -- For Query v

('Aiden Murphy', 'Miami', 'Heart Disease'); -- For Query v

-- Insert sample data into Doctor\_Patient table

INSERT INTO Doctor\_Patient (dno, pno, Fee)

VALUES

(1, 1, 200),

(2, 2, 300),

(3, 3, 150),

(4, 4, 180),

(5, 5, 250);

**Step 2: Queries in PostgreSQL**

**i. List the details of all doctors from a specific city (e.g., New York)**

sql

Copy code

SELECT \*

FROM Doctor

WHERE city = 'New York';

This will retrieve all the doctors whose city is **New York**.

**ii. Display the count of patients**

sql

Copy code

SELECT COUNT(\*) AS total\_patients

FROM Patient;

This will return the total number of patients in the **Patient** table.

**iii. List the names of patients suffering from 'Cancer' disease**

sql

Copy code

SELECT pat\_name

FROM Patient

WHERE disease = 'Cancer';

This will retrieve the names of patients who are suffering from **Cancer**.

**iv. Change the city of 'Dr. Patil' to 'Pune'**

sql

Copy code

UPDATE Doctor

SET city = 'Pune'

WHERE dname = 'Dr. Patil';

This query will update the city of **Dr. Patil** to **Pune**.

**v. List the names of patients whose names start with the alphabet 'A'**

sql

Copy code

SELECT pat\_name

FROM Patient

WHERE pat\_name LIKE 'A%';

This will return the names of patients whose names start with the letter 'A' (e.g., **Alice Smith**, **Alan Walker**, **Aiden Murphy**).

**Output Example:**

For the given queries:

* Query (i) would list all doctors from New York.
* Query (ii) would return a count of patients.
* Query (iii) would list patients suffering from cancer.
* Query (iv) would change the city of Dr. Patil to Pune.
* Query (v) would list patients whose names start with 'A'.

Q.1) Consider the following database.

Student-Teacher database Student (sno, s\_name, s\_class) s\_class can be either "FY", "SY" or "TY" Teacher (tno, t\_name, yrs\_experience)

The relationship is as follows: Student-Teacher: M-M with descriptive attribute Subject.

Normalize the above relation schema in 3NF and insert minimum 5 records in each table

**Given Database Schema:**

We have two entities: **Student** and **Teacher**, with a many-to-many relationship. The descriptive attribute is **Subject**. The schema details are:

* **Student**: (sno, s\_name, s\_class)
* **Teacher**: (tno, t\_name, yrs\_experience)
* Many-to-many relationship with a descriptive attribute **Subject**.

**Normalization to 3NF:**

To achieve **Third Normal Form (3NF)**, we need to break the many-to-many relationship into three separate tables: **Student**, **Teacher**, and a relationship table with **Subject** as the descriptive attribute.

After normalization, the schema will be as follows:

1. **Student Table**
   * Attributes: sno (PK), s\_name, s\_class
2. **Teacher Table**
   * Attributes: tno (PK), t\_name, yrs\_experience
3. **Student\_Teacher Table**
   * Attributes: sno (FK), tno (FK), Subject
   * Composite Primary Key: (sno, tno)

**Tables After Normalization:**

1. **Student (sno, s\_name, s\_class)**
2. **Teacher (tno, t\_name, yrs\_experience)**
3. **Student\_Teacher (sno, tno, Subject)**

**Explanation**

1. **Student** table holds unique student information, with each student having a sno (Primary Key).
2. **Teacher** table holds unique teacher information, with each teacher having a tno (Primary Key).
3. **Student\_Teacher** table manages the many-to-many relationship between students and teachers, with **Subject** as a descriptive attribute. The composite primary key (sno, tno) ensures no duplication in the relationships between the same student and teacher.

This schema is in **3NF** because:

* All non-key attributes depend only on the primary key (no partial dependencies).
* There are no transitive dependencies between attributes.

Q.2) Execute the following queries in PostGreSQL [any 4]

i. List teachers having > 5 yrs of experience.

ii. List all students studying in "FY.

iii. Count the number of teachers who teach subject

iv. Delete record of student whose sno"

v. Add designation column in teacher table.

**Step 1: Create the Tables**

sql

Copy code

-- Create Student Table

CREATE TABLE Student (

sno SERIAL PRIMARY KEY,

s\_name VARCHAR(100) NOT NULL,

s\_class VARCHAR(10) CHECK (s\_class IN ('FY', 'SY', 'TY'))

);

-- Create Teacher Table

CREATE TABLE Teacher (

tno SERIAL PRIMARY KEY,

t\_name VARCHAR(100) NOT NULL,

yrs\_experience INT

);

-- Create Student\_Teacher Table (Many-to-Many with Subject)

CREATE TABLE Student\_Teacher (

sno INT REFERENCES Student(sno),

tno INT REFERENCES Teacher(tno),

Subject VARCHAR(100),

PRIMARY KEY (sno, tno)

);

-- Insert sample data into Student table

INSERT INTO Student (s\_name, s\_class)

VALUES

('Alice White', 'FY'),

('Bob Green', 'SY'),

('Charlie Blue', 'TY'),

('Daisy Brown', 'FY'),

('Ethan Black', 'SY');

-- Insert sample data into Teacher table

INSERT INTO Teacher (t\_name, yrs\_experience)

VALUES

('Mr. Roberts', 10),

('Ms. Smith', 8),

('Dr. Johnson', 15),

('Prof. Lee', 12),

('Mrs. Taylor', 9);

-- Insert sample data into Student\_Teacher table

INSERT INTO Student\_Teacher (sno, tno, Subject)

VALUES

(1, 1, 'Mathematics'),

(2, 2, 'Physics'),

(3, 3, 'Chemistry'),

(4, 4, 'English'),

(5, 5, 'History'),

(1, 2, 'Physics'),

(2, 3, 'Chemistry'),

(3, 4, 'English');

**Step 2: Queries in PostgreSQL**

**i. List teachers having more than 5 years of experience**

sql

Copy code

SELECT t\_name, yrs\_experience

FROM Teacher

WHERE yrs\_experience > 5;

This will return a list of teachers who have more than 5 years of experience.

**ii. List all students studying in "FY"**

sql

Copy code

SELECT s\_name, s\_class

FROM Student

WHERE s\_class = 'FY';

This will return the names of students who are in the **FY** class.

**iii. Count the number of teachers who teach a subject**

sql

Copy code

SELECT COUNT(DISTINCT tno) AS teacher\_count

FROM Student\_Teacher

WHERE Subject IS NOT NULL;

This query counts the number of distinct teachers who are teaching any subject.

**iv. Delete the record of a student whose sno = 3**

sql

Copy code

DELETE FROM Student

WHERE sno = 3;

This query deletes the record of the student whose sno is 3 (e.g., **Charlie Blue** in this case).

**v. Add a "designation" column to the Teacher table**

sql

Copy code

ALTER TABLE Teacher

ADD COLUMN designation VARCHAR(100);

-- Optionally, update the designation values for existing teachers

UPDATE Teacher

SET designation = 'Professor'

WHERE yrs\_experience > 10;

UPDATE Teacher

SET designation = 'Assistant Professor'

WHERE yrs\_experience <= 10;

This query adds a new column called **designation** to the **Teacher** table and assigns values based on their years of experience.

**Output Example**

For the given queries:

* Query (i) lists teachers with more than 5 years of experience.
* Query (ii) returns students who are in the FY class.
* Query (iii) counts how many teachers are teaching any subject.
* Query (iv) deletes the student record with sno = 3.
* Query (v) adds a new column **designation** and populates it based on years of experience.

Q.1)Consider the following database (primary keys are underlined).

Person (pnumber, pname, birthdate, income),

Area(aname,area\_type).

There exists a one-to-many relationship between Area and Person.

The attribute 'area\_type' can have values as either urban or rural.

Normalize the above relation schema in 3NF and insert minimum 5 records in each table

**Given Database Schema**

* **Person**: (pnumber, pname, birthdate, income)
* **Area**: (aname, area\_type)

**Relationship**

There exists a one-to-many relationship between **Area** and **Person**, meaning that each area can have multiple people associated with it, but each person belongs to only one area.

**Normalization to 3NF**

In 3NF, we need to ensure that:

1. The table is in **Second Normal Form (2NF)**.
2. There are no transitive dependencies; that is, all attributes are fully functional and depend only on the primary key.

**Tables After Normalization:**

1. **Area Table**
   * Attributes: aname (PK), area\_type
2. **Person Table**
   * Attributes: pnumber (PK), pname, birthdate, income, aname (FK)

**Explanation of the SQL Code**

1. **Create Tables**:
   * The **Area** table has aname as its primary key and includes a CHECK constraint on area\_type to ensure it can only be 'urban' or 'rural'.
   * The **Person** table has a primary key pnumber, and aname acts as a foreign key referencing the **Area** table.
2. **Insert Records**:
   * Sample data is inserted into both the **Area** and **Person** tables, representing different areas and people living in those areas.

**Summary**

After normalization:

* The **Area** table is normalized and contains area information.
* The **Person** table stores individual records and references the area they belong to, maintaining the one-to-many relationship.

Q.2) Execute the following queries in PostGreSQL [any 4]

i. List the details of persons having income > 10000.

ii. Transfer all people living in 'Pune' to 'Delhi'.

iii. Count number of area's having type 'urban'.

iv. Give the count of people who are born on

v. List the names of all people living in area.

**Step 1: Create the Tables and Insert Sample Data**

sql

Copy code

-- Create Area Table

CREATE TABLE Area (

aname VARCHAR(100) PRIMARY KEY,

area\_type VARCHAR(10) CHECK (area\_type IN ('urban', 'rural'))

);

-- Create Person Table

CREATE TABLE Person (

pnumber SERIAL PRIMARY KEY,

pname VARCHAR(100) NOT NULL,

birthdate DATE NOT NULL,

income DECIMAL(10, 2),

aname VARCHAR(100) REFERENCES Area(aname)

);

-- Insert sample data into Area table

INSERT INTO Area (aname, area\_type) VALUES

('Downtown', 'urban'),

('Countryside', 'rural'),

('Uptown', 'urban'),

('Village', 'rural'),

('Suburb', 'urban'),

('Pune', 'urban'),

('Delhi', 'urban');

-- Insert sample data into Person table

INSERT INTO Person (pname, birthdate, income, aname) VALUES

('Alice Johnson', '1990-05-14', 55000.00, 'Downtown'),

('Bob Smith', '1985-07-22', 48000.00, 'Countryside'),

('Charlie Brown', '2000-03-01', 32000.00, 'Uptown'),

('Daisy Clark', '1995-12-31', 72000.00, 'Village'),

('Ethan Hunt', '1993-11-18', 65000.00, 'Suburb'),

('Fiona Green', '1992-01-10', 12000.00, 'Pune'),

('George White', '1988-03-15', 15000.00, 'Delhi');

**Step 2: Queries in PostgreSQL**

**i. List the details of persons having income > 10000**

sql

Copy code

SELECT \*

FROM Person

WHERE income > 10000;

This query retrieves all details of persons with an income greater than **10,000**.

**ii. Transfer all people living in 'Pune' to 'Delhi'**

sql

Copy code

UPDATE Person

SET aname = 'Delhi'

WHERE aname = 'Pune';

This query updates the aname field for all persons currently living in **Pune**, changing it to **Delhi**.

**iii. Count the number of areas having type 'urban'**

sql

Copy code

SELECT COUNT(\*) AS urban\_area\_count

FROM Area

WHERE area\_type = 'urban';

This query counts the number of areas classified as **urban**.

**iv. Count of people who are born on a specific date (e.g., '1990-05-14')**

sql

Copy code

SELECT COUNT(\*) AS count\_of\_people

FROM Person

WHERE birthdate = '1990-05-14';

This query counts the number of people born on **May 14, 1990**. You can replace this date with any specific date you want to check.

**v. List the names of all people living in a specific area (e.g., 'Delhi')**

sql

Copy code

SELECT pname

FROM Person

WHERE aname = 'Delhi';

This query retrieves the names of all persons living in **Delhi**.

**Summary**

* The **full SQL code** includes the creation of tables, sample data insertion, and the execution of the specified queries.
* You can run this code in a PostgreSQL environment to create the database schema and execute the queries. Adjust the specific dates or area names in the queries as needed for your requirements.

Q.1) Consider the following database

Student (sno, s name, s class) s\_class can be either "FY", "SY" or "TY"

Teacher (tno, t name, yrs\_experience)

The relationship is as follows: Student-Teacher: M-M with descriptive attribute Subject.

Normalize the above relation schema in 3NF and insert minimum 5 records in each table

**Given Database Schema**

* **Student**: (sno, s\_name, s\_class)
* **Teacher**: (tno, t\_name, yrs\_experience)
* There is a many-to-many relationship between **Student** and **Teacher** with a descriptive attribute **Subject**.

**Step 1: Normalize to 3NF**

To achieve **3NF**, we need to ensure that:

1. The table is in **Second Normal Form (2NF)**.
2. There are no transitive dependencies.

**Tables After Normalization**

1. **Student Table**
   * Attributes: sno (PK), s\_name, s\_class
2. **Teacher Table**
   * Attributes: tno (PK), t\_name, yrs\_experience
3. **Student\_Teacher Table**
   * Attributes: sno (FK), tno (FK), Subject
   * Composite Primary Key: (sno, tno)

**Explanation of the SQL Code**

1. **Create Tables**:
   * The **Student** table includes a primary key (sno) and a CHECK constraint on s\_class to ensure values can only be **FY**, **SY**, or **TY**.
   * The **Teacher** table includes a primary key (tno).
   * The **Student\_Teacher** table manages the many-to-many relationship and includes a composite primary key made up of sno and tno.
2. **Insert Records**:
   * Sample data is inserted into the **Student**, **Teacher**, and **Student\_Teacher** tables.

**Summary**

* The **Student** table contains student information.
* The **Teacher** table contains teacher information.
* The **Student\_Teacher** table represents the many-to-many relationship between students and teachers, with **Subject** as a descriptive attribute.

Q.2)Execute the following queries in PostGreSQL [any 4]

i. Give class-wise number of students.

ii. List all students studying in class "TY".

iii. Count the number of students who have taken subject "

iv. Delete record of student whose sno

v. Add designation column in teacher table.

**Step 1: Create Tables and Insert Sample Data**

sql

Copy code

-- Create Student Table

CREATE TABLE Student (

sno SERIAL PRIMARY KEY,

s\_name VARCHAR(100) NOT NULL,

s\_class VARCHAR(10) CHECK (s\_class IN ('FY', 'SY', 'TY'))

);

-- Create Teacher Table

CREATE TABLE Teacher (

tno SERIAL PRIMARY KEY,

t\_name VARCHAR(100) NOT NULL,

yrs\_experience INT

);

-- Create Student\_Teacher Table (Many-to-Many with Subject)

CREATE TABLE Student\_Teacher (

sno INT REFERENCES Student(sno),

tno INT REFERENCES Teacher(tno),

Subject VARCHAR(100),

PRIMARY KEY (sno, tno)

);

-- Insert sample data into Student table

INSERT INTO Student (s\_name, s\_class)

VALUES

('Alice Johnson', 'FY'),

('Bob Smith', 'SY'),

('Charlie Brown', 'TY'),

('Daisy Clark', 'FY'),

('Ethan Hunt', 'SY');

-- Insert sample data into Teacher table

INSERT INTO Teacher (t\_name, yrs\_experience)

VALUES

('Mr. Roberts', 10),

('Ms. Smith', 8),

('Dr. Johnson', 15),

('Prof. Lee', 12),

('Mrs. Taylor', 9);

-- Insert sample data into Student\_Teacher table

INSERT INTO Student\_Teacher (sno, tno, Subject)

VALUES

(1, 1, 'Mathematics'),

(2, 2, 'Physics'),

(3, 3, 'Chemistry'),

(4, 4, 'English'),

(5, 5, 'History'),

(1, 2, 'Physics'),

(2, 3, 'Chemistry'),

(3, 4, 'English'),

(4, 5, 'Mathematics');

**Step 2: Queries in PostgreSQL**

**i. Give class-wise number of students**

sql

Copy code

SELECT s\_class, COUNT(\*) AS number\_of\_students

FROM Student

GROUP BY s\_class;

This query counts the number of students in each class (**FY**, **SY**, **TY**) and groups the results by class.

**ii. List all students studying in class "TY"**

sql

Copy code

SELECT \*

FROM Student

WHERE s\_class = 'TY';

This query retrieves all details of students who are in the **TY** class.

**iii. Count the number of students who have taken a specific subject (e.g., 'Physics')**

sql

Copy code

SELECT COUNT(DISTINCT sno) AS number\_of\_students

FROM Student\_Teacher

WHERE Subject = 'Physics';

This query counts the number of distinct students who have taken the subject **Physics**. You can replace **'Physics'** with any other subject as needed.

**iv. Delete record of a student whose sno = 3**

sql

Copy code

DELETE FROM Student

WHERE sno = 3;

This query deletes the record of the student whose sno is **3** (e.g., **Charlie Brown** in this case).

**v. Add a "designation" column to the Teacher table**

sql

Copy code

ALTER TABLE Teacher

ADD COLUMN designation VARCHAR(100);

-- Optionally, you can update the designation values for existing teachers

UPDATE Teacher

SET designation = 'Professor'

WHERE yrs\_experience > 10;

UPDATE Teacher

SET designation = 'Assistant Professor'

WHERE yrs\_experience <= 10;

This query adds a new column called **designation** to the **Teacher** table and assigns values based on their years of experience.

**Summary**

* The **full SQL code** includes the creation of tables, sample data insertion, and the execution of the specified queries.
* You can run this code in a PostgreSQL environment to create the database schema and execute the queries as specified. Adjust the subject or student sno in the queries as needed for your requirements.

Q.1) Consider the following database

Employee(eno,ename, designation,salary)

Department(dno,dname, location)

The relationship is as follows: Employee-Department: many-to-one.

Normalize the above relation schema in 3NF and insert minimum 5 records in each table.

**Given Database Schema**

* **Employee**: (eno, ename, designation, salary)
* **Department**: (dno, dname, location)
* There is a **many-to-one** relationship from **Employee** to **Department**. This means that multiple employees can belong to a single department, but each employee belongs to only one department.

**Step 1: Normalize to 3NF**

To achieve **3NF**, we must ensure that:

1. The tables are in **Second Normal Form (2NF)**.
2. There are no transitive dependencies, meaning all attributes depend only on the primary key.

**Tables After Normalization**

1. **Department Table**
   * Attributes: dno (PK), dname, location
2. **Employee Table**
   * Attributes: eno (PK), ename, designation, salary, dno (FK)
   * The foreign key dno references the **Department** table.

**Explanation of the SQL Code**

1. **Create Tables**:
   * The **Department** table includes a primary key (dno) and stores the department name and location.
   * The **Employee** table includes a primary key (eno), employee name, designation, salary, and a foreign key (dno) referencing the **Department** table.
2. **Insert Records**:
   * Sample data is inserted into both the **Department** and **Employee** tables, representing different departments and employees working in those departments.

**Summary**

After normalization:

* The **Department** table contains information about different departments.
* The **Employee** table contains details about employees, including their association with departments through the foreign key.

Q.2) Execute the following queries in PostGreSQL [any 4]

i. Give a 5% raise in salary to all the employees.

ii. Display average salary of an employee.

iii. List the details of all the departments located at city.

iv. Display the details of employees whose names ends with an alphabet.

v. Display the location of 'HR' department.

C Write a query to List the details of employees who do not work in any of the departments located at 'Calcutta'.

OR

Create a view to list the names of employees whose salary is greater than all the employees working in 'HR' department.

**Step 1: Create Tables and Insert Sample Data**

sql

Copy code

-- Create Department Table

CREATE TABLE Department (

dno SERIAL PRIMARY KEY,

dname VARCHAR(100) NOT NULL,

location VARCHAR(100) NOT NULL

);

-- Create Employee Table

CREATE TABLE Employee (

eno SERIAL PRIMARY KEY,

ename VARCHAR(100) NOT NULL,

designation VARCHAR(100) NOT NULL,

salary DECIMAL(10, 2) NOT NULL,

dno INT REFERENCES Department(dno)

);

-- Insert sample data into Department table

INSERT INTO Department (dname, location)

VALUES

('Human Resources', 'New York'),

('Finance', 'San Francisco'),

('Engineering', 'Seattle'),

('Marketing', 'Los Angeles'),

('Sales', 'Chicago');

-- Insert sample data into Employee table

INSERT INTO Employee (ename, designation, salary, dno)

VALUES

('Alice Johnson', 'HR Manager', 75000.00, 1),

('Bob Smith', 'Financial Analyst', 65000.00, 2),

('Charlie Brown', 'Software Engineer', 85000.00, 3),

('Daisy Clark', 'Marketing Specialist', 60000.00, 4),

('Ethan Hunt', 'Sales Executive', 70000.00, 5);

**Step 2: Queries in PostgreSQL**

**i. Give a 5% raise in salary to all the employees**

sql

Copy code

UPDATE Employee

SET salary = salary \* 1.05;

This query updates the salary of all employees by increasing it by **5%**.

**ii. Display average salary of an employee**

sql

Copy code

SELECT AVG(salary) AS average\_salary

FROM Employee;

This query calculates the average salary of all employees in the **Employee** table.

**iii. List the details of all the departments located in a specific city (e.g., 'New York')**

sql

Copy code

SELECT \*

FROM Department

WHERE location = 'New York';

This query retrieves all details of departments located in **New York**. You can replace **'New York'** with any other city as needed.

**iv. Display the details of employees whose names end with an alphabet**

sql

Copy code

SELECT \*

FROM Employee

WHERE ename ILIKE '%[A-Za-z]';

This query retrieves the details of employees whose names end with an alphabet. Note that ILIKE is used for case-insensitive matching.

**v. Display the location of the 'HR' department**

sql

Copy code

SELECT location

FROM Department

WHERE dname = 'Human Resources';

This query retrieves the location of the **Human Resources** department.

**Optional Query or View Creation**

**List the details of employees who do not work in any of the departments located at 'Calcutta'**

sql

Copy code

SELECT \*

FROM Employee

WHERE dno NOT IN (

SELECT dno

FROM Department

WHERE location = 'Calcutta'

);

This query retrieves details of employees who are not associated with any departments located in **Calcutta**.

**OR Create a view to list the names of employees whose salary is greater than all the employees working in 'HR' department**

sql

Copy code

CREATE VIEW HighSalaryEmployees AS

SELECT ename

FROM Employee

WHERE salary > ALL (

SELECT salary

FROM Employee

WHERE dno = (

SELECT dno

FROM Department

WHERE dname = 'Human Resources'

)

);

This query creates a view called **HighSalaryEmployees** that lists the names of employees whose salary is greater than any employee in the **Human Resources** department.

**Summary**

* The **full SQL code** includes the creation of tables, sample data insertion, and execution of specified queries.
* You can run this code in a PostgreSQL environment to create the database schema and execute the queries as specified. Adjust city names or any specific values in the queries as needed for your requirements.

Q.1) Consider the following database

Employee(eno,ename, designation,salary)

Department(dno,dname, location)

The relationship is as follows: Employee-Department: many-to-one.

Normalize the above relation schema in 3NF and insert minimum 5 records in each table

**Given Database Schema**

1. **Employee**: (eno, ename, designation, salary)
2. **Department**: (dno, dname, location)

**Relationship**: There is a **many-to-one** relationship from **Employee** to **Department**, meaning multiple employees can belong to one department.

**Step 1: Normalize to 3NF**

To achieve **3NF**, we need to ensure that:

1. The tables are in **Second Normal Form (2NF)**.
2. There are no transitive dependencies.

**Normalization Process**

1. **Department Table**:
   * Attributes: dno (PK), dname, location
   * No normalization needed as it contains only atomic attributes.
2. **Employee Table**:
   * Attributes: eno (PK), ename, designation, salary, dno (FK)
   * The foreign key dno references the **Department** table.

**Final Table Structures**

1. **Department Table**
   * **Primary Key**: dno
   * Attributes:
     + dno (INT)
     + dname (VARCHAR)
     + location (VARCHAR)
2. **Employee Table**
   * **Primary Key**: eno
   * **Foreign Key**: dno
   * Attributes:
     + eno (INT)
     + ename (VARCHAR)
     + designation (VARCHAR)
     + salary (DECIMAL)
     + dno (INT)

**Explanation of the SQL Code**

1. **Create Tables**:
   * The **Department** table includes a primary key (dno), along with the department name and location.
   * The **Employee** table includes a primary key (eno), employee name, designation, salary, and a foreign key (dno) that references the **Department** table.
2. **Insert Records**:
   * Sample data is inserted into both the **Department** and **Employee** tables, representing different departments and their employees.

**Summary**

After normalization, we have:

* The **Department** table that contains information about different departments.
* The **Employee** table that contains details about employees, including their association with departments through the foreign key.

Q.2) Execute the following queries in PostGreSQL [any 4]

i. Give a 5% raise in salary to all the employees.

ii. Display average salary of an employee.

iii. List the details of all the departments located at city.

iv. Display the details of employees whose names ends with an alphabet.

v. Display the location of 'HR' department.

C Write a query to List the details of employees who do not work in any of the departments located at 'Calcutta'.

OR

Create a view to list the names of employees whose salary is greater than all the employees working in 'HR' department.

**Step 1: Create Tables and Insert Sample Data**

sql

Copy code

-- Create Department Table

CREATE TABLE Department (

dno INT AUTO\_INCREMENT PRIMARY KEY,

dname VARCHAR(100) NOT NULL,

location VARCHAR(100) NOT NULL

);

-- Create Employee Table

CREATE TABLE Employee (

eno INT AUTO\_INCREMENT PRIMARY KEY,

ename VARCHAR(100) NOT NULL,

designation VARCHAR(100) NOT NULL,

salary DECIMAL(10, 2) NOT NULL,

dno INT,

FOREIGN KEY (dno) REFERENCES Department(dno)

);

-- Insert sample data into Department table

INSERT INTO Department (dname, location)

VALUES

('Human Resources', 'New York'),

('Finance', 'San Francisco'),

('Engineering', 'Seattle'),

('Marketing', 'Los Angeles'),

('Sales', 'Chicago'),

('HR', 'Calcutta');

-- Insert sample data into Employee table

INSERT INTO Employee (ename, designation, salary, dno)

VALUES

('Alice Johnson', 'HR Manager', 75000.00, 1),

('Bob Smith', 'Financial Analyst', 65000.00, 2),

('Charlie Brown', 'Software Engineer', 85000.00, 3),

('Daisy Clark', 'Marketing Specialist', 60000.00, 4),

('Ethan Hunt', 'Sales Executive', 70000.00, 5);

**Step 2: Execute Queries in MySQL**

**i. Give a 5% raise in salary to all the employees**

sql

Copy code

UPDATE Employee

SET salary = salary \* 1.05;

This query updates the salary of all employees by increasing it by **5%**.

**ii. Display average salary of an employee**

sql

Copy code

SELECT AVG(salary) AS average\_salary

FROM Employee;

This query calculates the average salary of all employees in the **Employee** table.

**iii. List the details of all the departments located in a specific city (e.g., 'Calcutta')**

sql

Copy code

SELECT \*

FROM Department

WHERE location = 'Calcutta';

This query retrieves all details of departments located in **Calcutta**.

**iv. Display the details of employees whose names end with an alphabet**

sql

Copy code

SELECT \*

FROM Employee

WHERE ename REGEXP '[a-zA-Z]$';

This query retrieves the details of employees whose names end with an alphabet. The REGEXP operator checks if the last character is an alphabet.

**v. Display the location of the 'HR' department**

sql

Copy code

SELECT location

FROM Department

WHERE dname = 'Human Resources';

This query retrieves the location of the **Human Resources** department.

**Optional Query or View Creation**

**List the details of employees who do not work in any of the departments located at 'Calcutta'**

sql

Copy code

SELECT \*

FROM Employee

WHERE dno NOT IN (

SELECT dno

FROM Department

WHERE location = 'Calcutta'

);

This query retrieves details of employees who are not associated with any departments located in **Calcutta**.

**OR Create a view to list the names of employees whose salary is greater than all the employees working in 'HR' department**

sql

Copy code

CREATE VIEW HighSalaryEmployees AS

SELECT ename

FROM Employee

WHERE salary > ALL (

SELECT salary

FROM Employee

WHERE dno = (

SELECT dno

FROM Department

WHERE dname = 'Human Resources'

)

);

This query creates a view called **HighSalaryEmployees** that lists the names of employees whose salary is greater than any employee in the **Human Resources** department.

**Summary**

* The **full SQL code** includes the creation of tables, sample data insertion, and execution of specified queries.
* You can run this code in a MySQL environment to create the database schema and execute the queries as specified. Adjust city names or any specific values in the queries as needed for your requirements.

Q.1) Consider the following database

Person (pnumber, pname, birthdate, income)

Area (aname, area\_type, pincode)

The relationship is as follows: Person-Area: many-to-one. The 'area\_type' can have values as either 'urban' or 'rural'.

Normalize the above relation schema in 3NF and insert minimum 5 records in each table

**Given Database Schema**

1. **Person**: (pnumber, pname, birthdate, income)
2. **Area**: (aname, area\_type, pincode)

**Relationship**: There is a **many-to-one** relationship from **Person** to **Area**, meaning multiple persons can belong to one area.

**Step 1: Normalize to 3NF**

To achieve **3NF**, we need to ensure that:

1. The tables are in **Second Normal Form (2NF)**.
2. There are no transitive dependencies.

**Normalization Process**

1. **Area Table**:
   * Attributes: aname (PK), area\_type, pincode
   * No normalization needed as it contains only atomic attributes.
2. **Person Table**:
   * Attributes: pnumber (PK), pname, birthdate, income, aname (FK)
   * The foreign key aname references the **Area** table.

**Final Table Structures**

1. **Area Table**
   * **Primary Key**: aname
   * Attributes:
     + aname (VARCHAR)
     + area\_type (ENUM: 'urban', 'rural')
     + pincode (VARCHAR)
2. **Person Table**
   * **Primary Key**: pnumber
   * **Foreign Key**: aname
   * Attributes:
     + pnumber (INT)
     + pname (VARCHAR)
     + birthdate (DATE)
     + income (DECIMAL)
     + aname (VARCHAR)

**Explanation of the SQL Code**

1. **Create Tables**:
   * The **Area** table includes a primary key (aname), area type (with allowed values of 'urban' and 'rural'), and a pincode.
   * The **Person** table includes a primary key (pnumber), person name, birthdate, income, and a foreign key (aname) that references the **Area** table.
2. **Insert Records**:
   * Sample data is inserted into both the **Area** and **Person** tables, representing different areas and their respective residents.

**Summary**

After normalization, we have:

* The **Area** table that contains information about different areas, including type and pincode.
* The **Person** table that contains details about individuals, including their association with areas through the foreign key.

Q.2) Execute the following queries in PostGreSQL [any 4]

i. List the details of all people whose name starts with the alphabet 'R'.

ii. List the names of all people whose birthday falls in the month of 'July'.

iii. Display the details of people in the sorted order of their income.

iv. Display the count of areas of 'urban' type.

v. Change the pincode of 'kalyaninagar' to 411036.

vi. Write a query to List the names of people who live in 'Camp' area and have income less than at least one person who lives in 'kalyaninagar' area.

**Step 1: Create Tables and Insert Sample Data**

sql

Copy code

-- Create Area Table

CREATE TABLE Area (

aname VARCHAR(100) PRIMARY KEY,

area\_type VARCHAR(10) CHECK (area\_type IN ('urban', 'rural')),

pincode VARCHAR(10) NOT NULL

);

-- Create Person Table

CREATE TABLE Person (

pnumber SERIAL PRIMARY KEY,

pname VARCHAR(100) NOT NULL,

birthdate DATE NOT NULL,

income DECIMAL(10, 2) NOT NULL,

aname VARCHAR(100),

FOREIGN KEY (aname) REFERENCES Area(aname)

);

-- Insert sample data into Area table

INSERT INTO Area (aname, area\_type, pincode)

VALUES

('Greenwood', 'urban', '12345'),

('Sunnydale', 'rural', '67890'),

('Lakeside', 'urban', '11223'),

('Hilltop', 'rural', '44556'),

('Riverbend', 'urban', '78901'),

('Camp', 'urban', '54321'),

('Kalyaninagar', 'urban', '411037');

-- Insert sample data into Person table

INSERT INTO Person (pname, birthdate, income, aname)

VALUES

('John Doe', '1990-01-15', 50000.00, 'Greenwood'),

('Jane Smith', '1985-07-22', 60000.00, 'Sunnydale'),

('Sam Brown', '1992-06-10', 55000.00, 'Lakeside'),

('Emily White', '1988-07-05', 75000.00, 'Hilltop'),

('Michael Johnson', '1995-09-30', 48000.00, 'Riverbend'),

('Rachel Green', '1992-07-15', 70000.00, 'Camp'),

('Rita Blue', '1990-07-25', 72000.00, 'Kalyaninagar');

**Step 2: Execute Queries**

**i. List the details of all people whose name starts with the alphabet 'R'**

sql

Copy code

SELECT \*

FROM Person

WHERE pname LIKE 'R%';

This query retrieves all details of people whose names start with 'R'.

**ii. List the names of all people whose birthday falls in the month of 'July'**

sql

Copy code

SELECT pname

FROM Person

WHERE EXTRACT(MONTH FROM birthdate) = 7;

This query retrieves the names of all people whose birthdays are in **July**.

**iii. Display the details of people in the sorted order of their income**

sql

Copy code

SELECT \*

FROM Person

ORDER BY income;

This query retrieves details of people sorted in ascending order of their **income**.

**iv. Display the count of areas of 'urban' type**

sql

Copy code

SELECT COUNT(\*) AS urban\_area\_count

FROM Area

WHERE area\_type = 'urban';

This query counts the number of areas that are of **'urban'** type.

**v. Change the pincode of 'Kalyaninagar' to 411036**

sql

Copy code

UPDATE Area

SET pincode = '411036'

WHERE aname = 'Kalyaninagar';

This query updates the pincode for the area **Kalyaninagar** to **411036**.

**vi. List the names of people who live in 'Camp' area and have income less than at least one person who lives in 'Kalyaninagar' area.**

sql

Copy code

SELECT pname

FROM Person

WHERE aname = 'Camp'

AND income < (

SELECT MIN(income)

FROM Person

WHERE aname = 'Kalyaninagar'

);

This query retrieves the names of people living in **Camp** area who have an income less than at least one person living in **Kalyaninagar** area.

**Summary**

* The **full SQL code** includes the creation of the Area and Person tables, sample data insertion, and execution of the specified queries.
* You can run this code in a PostgreSQL environment to create the database schema and execute the queries as specified. Adjust names or any specific values in the queries as needed for your requirements.

Q.1) Consider the following database

Student (rno,name,city)

Teacher(tno,tname,phone\_no,salary)

The relationship is as follows: Student-Teacher: many-to-many with subject as a descriptive attribute. Normalize the above relation schema in 3NF and insert minimum 5 records in each table.

**Given Database Schema**

1. **Student**: (rno, name, city)
2. **Teacher**: (tno, tname, phone\_no, salary)
3. **Relationship**: Many-to-Many between **Students** and **Teachers** with a descriptive attribute subject.

**Step 1: Normalize to 3NF**

**Identify Entities and Their Attributes**

1. **Student Table**:
   * **Primary Key**: rno
   * Attributes: name, city
2. **Teacher Table**:
   * **Primary Key**: tno
   * Attributes: tname, phone\_no, salary
3. **Relationship Table** (to handle the many-to-many relationship):
   * **Primary Key**: Composite key of (rno, tno)
   * Attributes: subject

**Final Table Structures**

1. **Student Table**
   * **Primary Key**: rno
   * Attributes:
     + rno (INT)
     + name (VARCHAR)
     + city (VARCHAR)
2. **Teacher Table**
   * **Primary Key**: tno
   * Attributes:
     + tno (INT)
     + tname (VARCHAR)
     + phone\_no (VARCHAR)
     + salary (DECIMAL)
3. **Student\_Teacher Table** (to manage the many-to-many relationship)
   * **Primary Key**: Composite key of (rno, tno)
   * Attributes:
     + rno (INT, FK)
     + tno (INT, FK)
     + subject (VARCHAR)

Q.2) Execute the following queries in PostGreSQL. [any 4]

i. List the names of students from city.

ii. Display the count of students from 'Mumbai'city.

iii. Change the phone number of 'Prof. Patil' to '9822131226'

iv. List the details of the teachers in the sorted order of their name.

v. List the names of the teachers who have salary less than 50000

**Step 1: Create Tables and Insert Sample Data**

sql

Copy code

-- Create Student Table

CREATE TABLE Student (

rno INT PRIMARY KEY,

name VARCHAR(100) NOT NULL,

city VARCHAR(100) NOT NULL

);

-- Create Teacher Table

CREATE TABLE Teacher (

tno INT PRIMARY KEY,

tname VARCHAR(100) NOT NULL,

phone\_no VARCHAR(15) NOT NULL,

salary DECIMAL(10, 2) NOT NULL

);

-- Create Student\_Teacher Table

CREATE TABLE Student\_Teacher (

rno INT,

tno INT,

subject VARCHAR(100) NOT NULL,

PRIMARY KEY (rno, tno),

FOREIGN KEY (rno) REFERENCES Student(rno),

FOREIGN KEY (tno) REFERENCES Teacher(tno)

);

-- Insert sample data into Student table

INSERT INTO Student (rno, name, city)

VALUES

(1, 'Alice', 'Mumbai'),

(2, 'Bob', 'Delhi'),

(3, 'Charlie', 'Mumbai'),

(4, 'Daisy', 'Bangalore'),

(5, 'Ethan', 'Chennai');

-- Insert sample data into Teacher table

INSERT INTO Teacher (tno, tname, phone\_no, salary)

VALUES

(1, 'Prof. Patil', '123-456-7890', 80000.00),

(2, 'Ms. Johnson', '234-567-8901', 75000.00),

(3, 'Mr. Brown', '345-678-9012', 72000.00),

(4, 'Ms. White', '456-789-0123', 40000.00),

(5, 'Dr. Black', '567-890-1234', 85000.00);

**Step 2: Execute Queries**

**i. List the names of students from the city 'Mumbai'**

sql

Copy code

SELECT name

FROM Student

WHERE city = 'Mumbai';

This query retrieves the names of all students residing in **Mumbai**.

**ii. Display the count of students from 'Mumbai'**

sql

Copy code

SELECT COUNT(\*) AS student\_count

FROM Student

WHERE city = 'Mumbai';

This query counts the number of students residing in **Mumbai**.

**iii. Change the phone number of 'Prof. Patil' to '9822131226'**

sql

Copy code

UPDATE Teacher

SET phone\_no = '9822131226'

WHERE tname = 'Prof. Patil';

This query updates the phone number for **Prof. Patil**.

**iv. List the details of the teachers in the sorted order of their name**

sql

Copy code

SELECT \*

FROM Teacher

ORDER BY tname;

This query retrieves the details of all teachers, sorted alphabetically by their names.

**v. List the names of the teachers who have a salary less than 50000**

sql

Copy code

SELECT tname

FROM Teacher

WHERE salary < 50000;

This query retrieves the names of teachers whose salaries are less than **50000**.

**Summary**

* The **full SQL code** includes creating the Student and Teacher tables, inserting sample data, and executing the specified queries.
* You can run this code in a PostgreSQL environment to create the database schema and execute the queries as specified. Adjust names or any specific values in the queries as needed for your requirements.

Q.1) Consider the following database

Item (item\_no, name, quantity)

Supplier (s\_no, name, city)

The relationship is as follows: Item-Supplier: many-to-many with rate as a descriptive attribute. Normalize the above relation schema in 3NF and insert minimum 5 records in each table

**Given Database Schema**

1. **Item**: (item\_no, name, quantity)
2. **Supplier**: (s\_no, name, city)
3. **Relationship**: Many-to-Many between **Items** and **Suppliers** with a descriptive attribute rate.

**Step 1: Normalize to 3NF**

**Identify Entities and Their Attributes**

1. **Item Table**:
   * **Primary Key**: item\_no
   * Attributes: name, quantity
2. **Supplier Table**:
   * **Primary Key**: s\_no
   * Attributes: name, city
3. **Relationship Table** (to handle the many-to-many relationship):
   * **Primary Key**: Composite key of (item\_no, s\_no)
   * Attributes: rate

**Final Table Structures**

1. **Item Table**
   * **Primary Key**: item\_no
   * Attributes:
     + item\_no (INT)
     + name (VARCHAR)
     + quantity (INT)
2. **Supplier Table**
   * **Primary Key**: s\_no
   * Attributes:
     + s\_no (INT)
     + name (VARCHAR)
     + city (VARCHAR)
3. **Item\_Supplier Table** (to manage the many-to-many relationship)
   * **Primary Key**: Composite key of (item\_no, s\_no)
   * Attributes:
     + item\_no (INT, FK)
     + s\_no (INT, FK)
     + rate (DECIMAL)

**Explanation of the SQL Code**

1. **Create Tables**:
   * The **Item** table includes the primary key (item\_no) and attributes for name and quantity.
   * The **Supplier** table includes the primary key (s\_no) and attributes for name and city.
   * The **Item\_Supplier** table includes a composite primary key formed by (item\_no, s\_no) to manage the many-to-many relationship, along with the rate attribute.
2. **Insert Records**:
   * Sample data is inserted into the **Item**, **Supplier**, and **Item\_Supplier** tables to illustrate the relationships.

**Summary**

After normalization:

* The **Item** table contains information about items.
* The **Supplier** table contains information about suppliers.
* The **Item\_Supplier** table manages the many-to-many relationships between items and suppliers, along with the rates.

Q.2) Execute the following queries in PostGreSQL [any 4]

i. Change the quantity for item 'Mouse' to 80.

ii. List the details of the suppliers whose name begins with the alphabet 'M'.

iii. Display the count of items.

iv. List the names of suppliers who do not live in

city.

v. List the names of items with quantity less than 10.

**Step 1: Create Tables and Insert Sample Data**

sql

Copy code

-- Create Item Table

CREATE TABLE Item (

item\_no INT PRIMARY KEY,

name VARCHAR(100) NOT NULL,

quantity INT NOT NULL

);

-- Create Supplier Table

CREATE TABLE Supplier (

s\_no INT PRIMARY KEY,

name VARCHAR(100) NOT NULL,

city VARCHAR(100) NOT NULL

);

-- Create Item\_Supplier Table

CREATE TABLE Item\_Supplier (

item\_no INT,

s\_no INT,

rate DECIMAL(10, 2) NOT NULL,

PRIMARY KEY (item\_no, s\_no),

FOREIGN KEY (item\_no) REFERENCES Item(item\_no),

FOREIGN KEY (s\_no) REFERENCES Supplier(s\_no)

);

-- Insert sample data into Item table

INSERT INTO Item (item\_no, name, quantity)

VALUES

(1, 'Laptop', 10),

(2, 'Mobile', 50),

(3, 'Tablet', 30),

(4, 'Mouse', 5),

(5, 'Keyboard', 100);

-- Insert sample data into Supplier table

INSERT INTO Supplier (s\_no, name, city)

VALUES

(1, 'Supplier A', 'New York'),

(2, 'Supplier B', 'Los Angeles'),

(3, 'Supplier C', 'Chicago'),

(4, 'Supplier D', 'Houston'),

(5, 'Supplier M', 'Phoenix');

-- Insert sample data into Item\_Supplier table

INSERT INTO Item\_Supplier (item\_no, s\_no, rate)

VALUES

(1, 1, 1500.00),

(1, 2, 1550.00),

(2, 1, 700.00),

(2, 3, 720.00),

(3, 2, 300.00),

(4, 4, 250.00),

(5, 5, 20.00),

(3, 4, 320.00);

**Step 2: Execute Queries**

**i. Change the quantity for item 'Mouse' to 80**

sql

Copy code

UPDATE Item

SET quantity = 80

WHERE name = 'Mouse';

This query updates the quantity of the item named **Mouse** to **80**.

**ii. List the details of the suppliers whose name begins with the alphabet 'M'**

sql

Copy code

SELECT \*

FROM Supplier

WHERE name LIKE 'M%';

This query retrieves the details of suppliers whose names start with the letter **M**.

**iii. Display the count of items**

sql

Copy code

SELECT COUNT(\*) AS item\_count

FROM Item;

This query counts the total number of items in the **Item** table.

**iv. List the names of suppliers who do not live in a specific city (e.g., 'Chicago')**

sql

Copy code

SELECT name

FROM Supplier

WHERE city <> 'Chicago';

This query retrieves the names of suppliers who do not live in **Chicago**. You can replace **'Chicago'** with any other city of your choice.

**v. List the names of items with quantity less than 10**

sql

Copy code

SELECT name

FROM Item

WHERE quantity < 10;

This query retrieves the names of items whose quantities are less than **10**.

**Summary**

* The **full SQL code** includes creating the Item, Supplier, and Item\_Supplier tables, inserting sample data, and executing the specified queries.
* You can run this code in a PostgreSQL environment to create the database schema and execute the queries as specified. Adjust names or any specific values in the queries as needed for your requirements.

Q.1) Consider the following database

Policy (pno,pname,premium\_amt,policy\_type)

Customer(cno,cname.city, agent\_name)

The relationship is as follows: Policy-Customer: many-to-one. The 'policy\_type' can have values as 'Yearly', 'Half-yearly' or 'Monthly'.

Normalize the above relation schema in 3NF and insert minimum 5 records in each table

**Given Database Schema**

1. **Policy**: (pno, pname, premium\_amt, policy\_type)
2. **Customer**: (cno, cname, city, agent\_name)
3. **Relationship**: Many-to-One from **Policy** to **Customer**.

**Step 1: Normalize to 3NF**

**Identify Entities and Their Attributes**

1. **Policy Table**:
   * **Primary Key**: pno
   * Attributes: pname, premium\_amt, policy\_type
2. **Customer Table**:
   * **Primary Key**: cno
   * Attributes: cname, city, agent\_name

**Final Table Structures**

1. **Policy Table**
   * **Primary Key**: pno
   * Attributes:
     + pno (INT)
     + pname (VARCHAR)
     + premium\_amt (DECIMAL)
     + policy\_type (VARCHAR)
2. **Customer Table**
   * **Primary Key**: cno
   * Attributes:
     + cno (INT)
     + cname (VARCHAR)
     + city (VARCHAR)
     + agent\_name (VARCHAR)
3. **Policy\_Customer Table** (to handle the many-to-one relationship):
   * **Primary Key**: pno (FK)
   * Attributes:
     + pno (INT, FK)
     + cno (INT, FK)

**Explanation of the SQL Code**

1. **Create Tables**:
   * The **Policy** table includes the primary key (pno) and attributes for name, premium amount, and policy type.
   * The **Customer** table includes the primary key (cno) and attributes for name, city, and agent name.
   * The **Policy\_Customer** table links policies to customers through foreign keys.
2. **Insert Records**:
   * Sample data is inserted into the **Policy**, **Customer**, and **Policy\_Customer** tables to illustrate the relationships.

**Summary**

After normalization:

* The **Policy** table contains information about various insurance policies.
* The **Customer** table contains information about customers.
* The **Policy\_Customer** table manages the many-to-one relationships between policies and customers.

Q.2) Execute the following queries in PostGreSQL [any 4]

i. List the details of all customers who live in city.

ii. Display the average premium amount.

iii. Count the number of customers who have taken 'Jeevan Anand' policy.

iv. Increases the premium amount for 'Monthly 'policies by 10%.

v. Display the policy\_type wise count of policies.

**Step 1: Create Tables and Insert Sample Data**

sql

Copy code

-- Create Policy Table

CREATE TABLE Policy (

pno INT PRIMARY KEY,

pname VARCHAR(100) NOT NULL,

premium\_amt DECIMAL(10, 2) NOT NULL,

policy\_type VARCHAR(20) CHECK (policy\_type IN ('Yearly', 'Half-yearly', 'Monthly'))

);

-- Create Customer Table

CREATE TABLE Customer (

cno INT PRIMARY KEY,

cname VARCHAR(100) NOT NULL,

city VARCHAR(100) NOT NULL,

agent\_name VARCHAR(100) NOT NULL

);

-- Create Policy\_Customer Table

CREATE TABLE Policy\_Customer (

pno INT,

cno INT,

PRIMARY KEY (pno, cno),

FOREIGN KEY (pno) REFERENCES Policy(pno),

FOREIGN KEY (cno) REFERENCES Customer(cno)

);

-- Insert sample data into Policy table

INSERT INTO Policy (pno, pname, premium\_amt, policy\_type)

VALUES

(1, 'Health Insurance', 5000.00, 'Yearly'),

(2, 'Car Insurance', 3000.00, 'Half-yearly'),

(3, 'Jeevan Anand', 2500.00, 'Monthly'),

(4, 'Travel Insurance', 1500.00, 'Yearly'),

(5, 'Life Insurance', 7000.00, 'Yearly');

-- Insert sample data into Customer table

INSERT INTO Customer (cno, cname, city, agent\_name)

VALUES

(1, 'John Doe', 'New York', 'Agent Smith'),

(2, 'Jane Smith', 'Los Angeles', 'Agent Johnson'),

(3, 'Emily Davis', 'Chicago', 'Agent Brown'),

(4, 'Michael Wilson', 'Houston', 'Agent Taylor'),

(5, 'Mary Johnson', 'Phoenix', 'Agent Anderson');

-- Insert sample data into Policy\_Customer table

INSERT INTO Policy\_Customer (pno, cno)

VALUES

(1, 1),

(2, 1),

(3, 2),

(4, 3),

(5, 4),

(3, 5); -- Assuming this customer also took 'Jeevan Anand'

**Step 2: Execute Queries**

**i. List the details of all customers who live in a specific city (e.g., 'New York')**

sql

Copy code

SELECT \*

FROM Customer

WHERE city = 'New York';

This query retrieves the details of all customers living in **New York**. You can change **'New York'** to any other city as needed.

**ii. Display the average premium amount**

sql

Copy code

SELECT AVG(premium\_amt) AS average\_premium

FROM Policy;

This query calculates the average premium amount across all policies in the **Policy** table.

**iii. Count the number of customers who have taken the 'Jeevan Anand' policy**

sql

Copy code

SELECT COUNT(DISTINCT c.cno) AS customer\_count

FROM Policy\_Customer pc

JOIN Policy p ON pc.pno = p.pno

WHERE p.pname = 'Jeevan Anand';

This query counts the number of distinct customers who have taken the **'Jeevan Anand'** policy.

**iv. Increase the premium amount for 'Monthly' policies by 10%**

sql

Copy code

UPDATE Policy

SET premium\_amt = premium\_amt \* 1.10

WHERE policy\_type = 'Monthly';

This query updates the premium amount for all **Monthly** policies by increasing it by 10%.

**v. Display the policy\_type wise count of policies**

sql

Copy code

SELECT policy\_type, COUNT(\*) AS policy\_count

FROM Policy

GROUP BY policy\_type;

This query retrieves the count of policies grouped by their **policy\_type**.

**Summary**

* The **full SQL code** includes creating the **Policy**, **Customer**, and **Policy\_Customer** tables, inserting sample data, and executing the specified queries.
* You can run this code in a PostgreSQL environment to create the database schema and execute the queries as specified. Adjust names or any specific values in the queries as needed for your requirements.

Q.1) Consider the database

Room (room\_no,room\_name,room\_type,charges)

Guest(Guest\_code, Gname,city)

The relationship is as follows: Room-Guest: one-to-one, room\_type can have values as either 'AC' or 'NonAC'.

Normalize the above relation schema in 3NF and insert minimum 5 records in each table.

**Given Database Schema**

1. **Room**: (room\_no, room\_name, room\_type, charges)
2. **Guest**: (Guest\_code, Gname, city)
3. **Relationship**: One-to-One from **Room** to **Guest**.

**Step 1: Normalize to 3NF**

**Identify Entities and Their Attributes**

1. **Room Table**:
   * **Primary Key**: room\_no
   * Attributes: room\_name, room\_type, charges
2. **Guest Table**:
   * **Primary Key**: Guest\_code
   * Attributes: Gname, city

**Final Table Structures**

1. **Room Table**
   * **Primary Key**: room\_no
   * Attributes:
     + room\_no (INT)
     + room\_name (VARCHAR)
     + room\_type (VARCHAR)
     + charges (DECIMAL)
2. **Guest Table**
   * **Primary Key**: Guest\_code
   * Attributes:
     + Guest\_code (INT)
     + Gname (VARCHAR)
     + city (VARCHAR)

**Explanation of the SQL Code**

1. **Create Tables**:
   * The **Room** table includes the primary key (room\_no) and attributes for the room name, type, and charges.
   * The **Guest** table includes the primary key (Guest\_code) and attributes for the guest's name and city.
2. **Insert Records**:
   * Sample data is inserted into both the **Room** and **Guest** tables to illustrate the structure and relationships.

**Summary**

After normalization:

* The **Room** table contains information about various rooms.
* The **Guest** table contains information about guests.

Q2)Execute the following queries in PostGreSQI. (any 4)

a) List the details of the rooms having charges between 5000 and 10000.

b) List the names of the guests in the sorted order by city name.

c) List the minimum charges of a room.

d) Increase the charges of all AC rooms by 15%.

e) List the names of all the NONAC rooms whose charges are more than 10000.

**Step 1: Create Tables and Insert Sample Data**

sql

Copy code

-- Create Room Table

CREATE TABLE Room (

room\_no INT PRIMARY KEY,

room\_name VARCHAR(100) NOT NULL,

room\_type VARCHAR(20) CHECK (room\_type IN ('AC', 'NonAC')),

charges DECIMAL(10, 2) NOT NULL

);

-- Create Guest Table

CREATE TABLE Guest (

Guest\_code INT PRIMARY KEY,

Gname VARCHAR(100) NOT NULL,

city VARCHAR(100) NOT NULL

);

-- Insert sample data into Room table

INSERT INTO Room (room\_no, room\_name, room\_type, charges)

VALUES

(1, 'Room A', 'AC', 1500.00),

(2, 'Room B', 'NonAC', 800.00),

(3, 'Room C', 'AC', 1200.00),

(4, 'Room D', 'NonAC', 600.00),

(5, 'Room E', 'AC', 1800.00),

(6, 'Room F', 'NonAC', 20000.00), -- Example for charges > 10000

(7, 'Room G', 'AC', 10000.00);

-- Insert sample data into Guest table

INSERT INTO Guest (Guest\_code, Gname, city)

VALUES

(1, 'John Doe', 'New York'),

(2, 'Jane Smith', 'Los Angeles'),

(3, 'Emily Davis', 'Chicago'),

(4, 'Michael Wilson', 'Houston'),

(5, 'Mary Johnson', 'Phoenix'),

(6, 'Chris Brown', 'New York'),

(7, 'Anna Taylor', 'Los Angeles');

**Step 2: Execute Queries**

**a) List the details of the rooms having charges between 5000 and 10000**

sql

Copy code

SELECT \*

FROM Room

WHERE charges BETWEEN 5000 AND 10000;

This query retrieves details of rooms with charges between 5000 and 10000. Since the sample data does not have such entries, you may want to adjust the values for testing.

**b) List the names of the guests in the sorted order by city name**

sql

Copy code

SELECT Gname, city

FROM Guest

ORDER BY city;

This query retrieves the names of guests and sorts them by city name.

**c) List the minimum charges of a room**

sql

Copy code

SELECT MIN(charges) AS minimum\_charges

FROM Room;

This query retrieves the minimum charge among all rooms.

**d) Increase the charges of all AC rooms by 15%**

sql

Copy code

UPDATE Room

SET charges = charges \* 1.15

WHERE room\_type = 'AC';

This query updates the charges for all AC rooms, increasing them by 15%.

**e) List the names of all the NONAC rooms whose charges are more than 10000**

sql

Copy code

SELECT room\_name

FROM Room

WHERE room\_type = 'NonAC' AND charges > 10000;

This query retrieves the names of all Non-AC rooms where the charges are greater than 10000. The sample data includes a Non-AC room (Room F) that meets this criterion.

**Summary**

The SQL code above includes:

* Creation of **Room** and **Guest** tables.
* Insertion of sample data into each table.
* Execution of specified queries based on the requirements.

Q.1) Consider the following database

Emp(eno, ename, designation, sal)

Dept(dno, dname, dloc)

There exists a one-to-many relationship between Dept and Emp.

Normalize the above relation schema in 3NF and insert minimum 5 records in each table.

**Given Database Schema**

1. **Emp**: (eno, ename, designation, sal)
2. **Dept**: (dno, dname, dloc)
3. **Relationship**: One-to-Many from **Dept** to **Emp**.

**Step 1: Normalize to 3NF**

**Identify Entities and Their Attributes**

1. **Dept Table**:
   * **Primary Key**: dno
   * Attributes: dname, dloc
2. **Emp Table**:
   * **Primary Key**: eno
   * Attributes: ename, designation, sal
   * **Foreign Key**: dno (to link the employee to their department)

**Final Table Structures**

1. **Dept Table**
   * **Primary Key**: dno
   * Attributes:
     + dno (INT)
     + dname (VARCHAR)
     + dloc (VARCHAR)
2. **Emp Table**
   * **Primary Key**: eno
   * Attributes:
     + eno (INT)
     + ename (VARCHAR)
     + designation (VARCHAR)
     + sal (DECIMAL)
     + **Foreign Key**: dno (to reference the department)

Q.2) Execute the following queries in PostGreSQL [any 4]

i. Increase salary of "managers" by 15%;

ii. Delete all employees of department no 30;

iii. Count the number of employees in dept no

iv. List the details of employees having salary between 30000 and 50000.

v. Display the average salary of employee.

**Step 1: Create Tables and Insert Sample Data**

sql

Copy code

-- Create Dept Table

CREATE TABLE Dept (

dno INT PRIMARY KEY,

dname VARCHAR(100) NOT NULL,

dloc VARCHAR(100) NOT NULL

);

-- Create Emp Table

CREATE TABLE Emp (

eno INT PRIMARY KEY,

ename VARCHAR(100) NOT NULL,

designation VARCHAR(100) NOT NULL,

sal DECIMAL(10, 2) NOT NULL,

dno INT,

FOREIGN KEY (dno) REFERENCES Dept(dno) -- Establishing the relationship with Dept

);

-- Insert sample data into Dept table

INSERT INTO Dept (dno, dname, dloc)

VALUES

(1, 'Human Resources', 'New York'),

(2, 'Finance', 'Los Angeles'),

(3, 'Development', 'Chicago'),

(4, 'Marketing', 'Houston'),

(5, 'Sales', 'Phoenix');

-- Insert sample data into Emp table

INSERT INTO Emp (eno, ename, designation, sal, dno)

VALUES

(1, 'John Doe', 'Manager', 75000.00, 1),

(2, 'Jane Smith', 'Finance Analyst', 65000.00, 2),

(3, 'Emily Davis', 'Software Engineer', 80000.00, 3),

(4, 'Michael Wilson', 'Manager', 60000.00, 4),

(5, 'Mary Johnson', 'Sales Representative', 55000.00, 5),

(6, 'Chris Brown', 'Manager', 50000.00, 1),

(7, 'Anna Taylor', 'Intern', 25000.00, 2);

**Step 2: Execute Queries**

**i) Increase salary of "managers" by 15%**

sql

Copy code

UPDATE Emp

SET sal = sal \* 1.15

WHERE designation = 'Manager';

This query increases the salary of all employees with the designation "Manager" by 15%.

**ii) Delete all employees of department no 30**

sql

Copy code

DELETE FROM Emp

WHERE dno = 30;

This query deletes all employees who belong to the department with the number 30. Note that in the provided sample data, there is no department with the number 30, so this won't affect any records.

**iii) Count the number of employees in department no 1**

sql

Copy code

SELECT COUNT(\*) AS employee\_count

FROM Emp

WHERE dno = 1;

This query counts the number of employees who belong to department number 1.

**iv) List the details of employees having salary between 30000 and 50000**

sql

Copy code

SELECT \*

FROM Emp

WHERE sal BETWEEN 30000 AND 50000;

This query retrieves the details of employees whose salary is between 30,000 and 50,000.

**v) Display the average salary of employees**

sql

Copy code

SELECT AVG(sal) AS average\_salary

FROM Emp;

This query calculates the average salary of all employees in the **Emp** table.

**Summary**

The SQL code above includes:

* Creation of the **Dept** and **Emp** tables.
* Insertion of sample data into each table.
* Execution of the specified queries based on the requirements.

Q.1) Consider the following database.

Student (stud\_reg\_no, stud\_name, class) Competition (cno, cname, ctype) Relation between Student and Competition is Many to Many with rank and year as descriptive attribute. Constraint: Primary key, class must be("FY,SY,TY").

Normalize the above relation schema in 3NF and insert minimum 5 records in each table

**Given Database Schema**

1. **Student**: (stud\_reg\_no, stud\_name, class)
2. **Competition**: (cno, cname, ctype)
3. **Relationship**: Many-to-Many with additional attributes rank and year.

**Step 1: Normalize to 3NF**

**Identify Entities and Their Attributes**

1. **Student Table**:
   * **Primary Key**: stud\_reg\_no
   * Attributes: stud\_name, class
2. **Competition Table**:
   * **Primary Key**: cno
   * Attributes: cname, ctype
3. **Student\_Competition Table** (for Many-to-Many Relationship):
   * **Primary Key**: Composite key of (stud\_reg\_no, cno)
   * Attributes: rank, year
   * **Foreign Keys**:
     + stud\_reg\_no (references **Student**)
     + cno (references **Competition**)

**Final Table Structures**

1. **Student Table**
   * **Primary Key**: stud\_reg\_no
   * Attributes:
     + stud\_reg\_no (INT)
     + stud\_name (VARCHAR)
     + class (ENUM with values: "FY", "SY", "TY")
2. **Competition Table**
   * **Primary Key**: cno
   * Attributes:
     + cno (INT)
     + cname (VARCHAR)
     + ctype (VARCHAR)
3. **Student\_Competition Table**
   * **Primary Key**: (stud\_reg\_no, cno)
   * Attributes:
     + stud\_reg\_no (INT)
     + cno (INT)
     + rank (VARCHAR)
     + year (INT)

**Explanation of the SQL Code**

1. **Create Tables**:
   * The **Student** table includes the primary key (stud\_reg\_no) and attributes for the student's name and class.
   * The **Competition** table includes the primary key (cno) and attributes for the competition name and type.
   * The **Student\_Competition** table links students to competitions, including their rank and year of participation.
2. **Insert Records**:
   * Sample data is inserted into the **Student**, **Competition**, and **Student\_Competition** tables to illustrate the structure and relationships.

**Summary**

After normalization:

* The **Student** table maintains student-related information.
* The **Competition** table maintains competition-related information.
* The **Student\_Competition** table maintains the many-to-many relationship along with descriptive attributes (rank and year).

Q.2) Execute the following queries. [any 4]

a) b)Count total no students class wise.

b) Find the number of student for programming competition.

c) Display total number of competition held in the year 2022.

d) Display the competition name in which more than 5 students have participated.

**Step 1: Table Creation and Data Insertion**

sql

Copy code

-- Create Student Table

CREATE TABLE Student (

stud\_reg\_no INT PRIMARY KEY,

stud\_name VARCHAR(100) NOT NULL,

class ENUM('FY', 'SY', 'TY') NOT NULL

);

-- Create Competition Table

CREATE TABLE Competition (

cno INT PRIMARY KEY,

cname VARCHAR(100) NOT NULL,

ctype VARCHAR(100) NOT NULL

);

-- Create Student\_Competition Table (Many-to-Many Relationship with Descriptive Attributes)

CREATE TABLE Student\_Competition (

stud\_reg\_no INT,

cno INT,

rank VARCHAR(10),

year INT,

PRIMARY KEY (stud\_reg\_no, cno),

FOREIGN KEY (stud\_reg\_no) REFERENCES Student(stud\_reg\_no),

FOREIGN KEY (cno) REFERENCES Competition(cno)

);

-- Insert sample data into Student table

INSERT INTO Student (stud\_reg\_no, stud\_name, class)

VALUES

(1, 'Alice Johnson', 'FY'),

(2, 'Bob Smith', 'SY'),

(3, 'Charlie Brown', 'TY'),

(4, 'Diana Prince', 'FY'),

(5, 'Ethan Hunt', 'SY'),

(6, 'Frank White', 'FY');

-- Insert sample data into Competition table

INSERT INTO Competition (cno, cname, ctype)

VALUES

(1, 'Math Olympiad', 'Academic'),

(2, 'Programming Competition', 'Academic'),

(3, 'Art Contest', 'Cultural'),

(4, 'Sports Meet', 'Sports'),

(5, 'Debate Competition', 'Cultural');

-- Insert sample data into Student\_Competition table

INSERT INTO Student\_Competition (stud\_reg\_no, cno, rank, year)

VALUES

(1, 1, 'Gold', 2023),

(1, 2, 'Silver', 2023),

(2, 2, 'Gold', 2023),

(3, 4, 'Bronze', 2022),

(4, 1, 'Silver', 2023),

(5, 5, 'Gold', 2022),

(6, 2, 'Bronze', 2023);

**Step 2: Execute Queries**

**a) Count the total number of students class-wise.**

sql

Copy code

SELECT class, COUNT(\*) AS total\_students

FROM Student

GROUP BY class;

This query will count the number of students in each class (FY, SY, TY).

**b) Find the number of students who participated in the 'Programming Competition'.**

sql

Copy code

SELECT COUNT(DISTINCT sc.stud\_reg\_no) AS total\_students

FROM Student\_Competition sc

JOIN Competition c ON sc.cno = c.cno

WHERE c.cname = 'Programming Competition';

This query counts the number of unique students who participated in the **Programming Competition**.

**c) Display the total number of competitions held in the year 2022.**

sql

Copy code

SELECT COUNT(DISTINCT cno) AS total\_competitions

FROM Student\_Competition

WHERE year = 2022;

This query counts the total number of unique competitions that were held in the year 2022.

**d) Display the competition name in which more than 5 students have participated.**

sql

Copy code

SELECT c.cname, COUNT(sc.stud\_reg\_no) AS total\_participants

FROM Student\_Competition sc

JOIN Competition c ON sc.cno = c.cno

GROUP BY c.cname

HAVING COUNT(sc.stud\_reg\_no) > 5;

This query lists the names of the competitions where more than 5 students participated.

**Summary**

This MySQL code:

1. Creates the **Student**, **Competition**, and **Student\_Competition** tables.
2. Inserts sample data into each table.
3. Executes the specified queries, including counting students class-wise, finding the number of students in a specific competition, and checking competition details based on participation and year.

Q.1) Consider the following database

Person (pnumber, pname, birthdate, income), Area(aname, area\_type).

There exists a one-to-many relationship between Area and Person. The attribute 'area\_type' can have values as

either urban or rural

Normalize the above relation schema in 3NF and insert minimum 5 records in each table

**Given Database Schema**

1. **Person**: (pnumber, pname, birthdate, income)
2. **Area**: (aname, area\_type)
3. **Relationship**: One-to-Many between **Area** and **Person**.

**Step 1: Normalize to 3NF**

**Identify Entities and Their Attributes**

1. **Area Table**:
   * **Primary Key**: aname
   * Attributes: area\_type (values can be urban or rural)
2. **Person Table**:
   * **Primary Key**: pnumber
   * Attributes: pname, birthdate, income, aname (Foreign Key referencing **Area**)

**Final Table Structures**

1. **Area Table**:
   * **Primary Key**: aname
   * Attributes:
     + aname (VARCHAR)
     + area\_type (ENUM with values: 'urban', 'rural')
2. **Person Table**:
   * **Primary Key**: pnumber
   * Attributes:
     + pnumber (INT)
     + pname (VARCHAR)
     + birthdate (DATE)
     + income (DECIMAL)
     + aname (VARCHAR) (Foreign Key referencing Area)

**Explanation of the SQL Code**

1. **Create Tables**:
   * The **Area** table includes the primary key (aname) and the area\_type attribute, which is either 'urban' or 'rural'.
   * The **Person** table includes the primary key (pnumber) and attributes for name, birthdate, income, and a foreign key (aname) that references the **Area** table.
2. **Insert Records**:
   * Sample data is inserted into the **Area** and **Person** tables to demonstrate the structure and relationships.

**Summary**

After normalization:

* The **Area** table holds information about different areas, including whether they are urban or rural.
* The **Person** table holds information about individuals, including their personal details and the area they live in (linked via a foreign key to the **Area** table).

Q.2)Execute the following queries in PostGreSQL [any 4]

i. Delete the record of "Mr. Kumar".

ii. Transfer all people living in 'Mumbai' to 'Pune'.

iii. Count number of area's having type 'urban'.

iv. Give the count of people who are living in "

v. Calculate the average income of people in "Pune".

**Step 1: Creating Tables and Inserting Data**

sql

Copy code

-- Create the Area Table

CREATE TABLE Area (

aname VARCHAR(100) PRIMARY KEY,

area\_type ENUM('urban', 'rural') NOT NULL

);

-- Create the Person Table

CREATE TABLE Person (

pnumber INT PRIMARY KEY,

pname VARCHAR(100) NOT NULL,

birthdate DATE NOT NULL,

income DECIMAL(10, 2) NOT NULL,

aname VARCHAR(100),

FOREIGN KEY (aname) REFERENCES Area(aname)

);

-- Insert sample data into the Area table

INSERT INTO Area (aname, area\_type)

VALUES

('Kalyaninagar', 'urban'),

('Camp', 'urban'),

('Wakad', 'urban'),

('Baner', 'urban'),

('Mulshi', 'rural'),

('Mumbai', 'urban'),

('Pune', 'urban');

-- Insert sample data into the Person table

INSERT INTO Person (pnumber, pname, birthdate, income, aname)

VALUES

(1, 'John Doe', '1985-04-12', 50000, 'Kalyaninagar'),

(2, 'Jane Smith', '1990-11-05', 45000, 'Camp'),

(3, 'Mike Johnson', '1978-02-23', 60000, 'Wakad'),

(4, 'Sara Lee', '1988-07-15', 55000, 'Baner'),

(5, 'David Brown', '1995-09-08', 40000, 'Mulshi'),

(6, 'Mr. Kumar', '1982-03-10', 47000, 'Mumbai'),

(7, 'Alice Green', '1992-05-22', 51000, 'Mumbai');

**Step 2: Execute Queries**

**i. Delete the record of "Mr. Kumar".**

sql

Copy code

DELETE FROM Person

WHERE pname = 'Mr. Kumar';

This query deletes the record of "Mr. Kumar" from the Person table.

**ii. Transfer all people living in 'Mumbai' to 'Pune'.**

sql

Copy code

UPDATE Person

SET aname = 'Pune'

WHERE aname = 'Mumbai';

This query updates the aname column in the Person table, changing all instances of 'Mumbai' to 'Pune'.

**iii. Count the number of areas having type 'urban'.**

sql

Copy code

SELECT COUNT(\*) AS urban\_area\_count

FROM Area

WHERE area\_type = 'urban';

This query counts the number of areas in the Area table that are classified as 'urban'.

**iv. Give the count of people who are living in 'Pune'.**

sql

Copy code

SELECT COUNT(\*) AS people\_in\_pune

FROM Person

WHERE aname = 'Pune';

This query counts the number of people in the Person table who live in 'Pune'.

**v. Calculate the average income of people in 'Pune'.**

sql

Copy code

SELECT AVG(income) AS average\_income\_in\_pune

FROM Person

WHERE aname = 'Pune';

This query calculates the average income of all people living in 'Pune'.

**Summary**

1. Created the **Area** and **Person** tables in MySQL.
2. Inserted sample records into both tables.
3. Provided queries to:
   * Delete a specific record.
   * Update records for a city transfer.
   * Count the number of areas by type.
   * Calculate the number of people living in a specific area.
   * Calculate the average income of people living in 'Pune'.