BANK DATABASE

CREATE TABLE Branch (Bid INTEGER PRIMARY KEY, brname VARCHAR(30) NOT NULL, brcity VARCHAR(10) NOT NULL);

CREATE TABLE Customer (Cno INTEGER PRIMARY KEY, cname VARCHAR(20) NOT NULL, caddr VARCHAR(35), city VARCHAR(15));

CREATE TABLE Loan_application (Lno INTEGER PRIMARY KEY, l_amt_required INT CHECK (l_amt_required > 0), lamtapproved INT, l_date DATE);

CREATE TABLE Ternary (Bid INTEGER, Cno INTEGER, Lno INTEGER, PRIMARY KEY (Bid, Cno, Lno), FOREIGN KEY (Bid) REFERENCES Branch(Bid), FOREIGN KEY (Cno) REFERENCES Customer(Cno), FOREIGN KEY (Lno) REFERENCES Loan_application(Lno));

INSERT INTO Branch (Bid, brname, brcity) VALUES (1, 'Pimpri', 'Pimpri'), (2, 'Aundh', 'Aundh');

INSERT INTO Customer (Cno, cname, caddr, city) VALUES (1, 'Rahul', '123 Street', 'Pimpri'), (2, 'Neha', '456 Avenue', 'Aundh'), (3, 'Raj', '789 Boulevard', 'Pune');

INSERT INTO Loan_application (Lno, l_amt_required, lamtapproved, l_date) VALUES (101, 500000, 450000, '2024-09-01'), (102, 200000, 150000, '2024-09-05'), (103, 600000, 550000, '2024-09-10');

INSERT INTO Ternary (Bid, Cno, Lno) VALUES (1, 1, 101), (2, 2, 102), (2, 3, 103);

Q.1) Create a View:

CREATE VIEW Customers_Pimpri_Branch 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 = 'Pimpri';

SELECT * FROM Customers_Pimpri_Branch;

CREATE VIEW Customers_Loan_Same_City AS SELECT c.cname FROM Customer c JOIN Ternary t ON c.Cno = t.Cno JOIN Branch b ON t.Bid = b.Bid WHERE c.city = b.brcity;

SELECT * FROM Customers_Loan_Same_City;

Q.2) Using above database solve following questions:

```
        CREATE OR REPLACE FUNCTION
prevent_customer_number_update()
        RETURNS TRIGGER AS $$
        BEGIN
          RAISE EXCEPTION 'You can’t change existing customer number';
          RETURN NULL;
        END;
        $$ LANGUAGE plpgsql;

        CREATE TRIGGER trigger_prevent_customer_number_update
        BEFORE UPDATE OF Cno ON Customer
        FOR EACH ROW
        EXECUTE FUNCTION prevent_customer_number_update();

        UPDATE Customer SET Cno = 2 WHERE Cno = 1
```

```
        CREATE OR REPLACE FUNCTION
get_loan_info_by_branch(branch_name VARCHAR)
        RETURNS VOID AS $$
        BEGIN
          FOR rec IN
            SELECT l.Lno, l.l_amt_required, l.lamtapproved, l.l_date
            FROM Loan_application l
            JOIN Ternary t ON l.Lno = t.Lno
            JOIN Branch b ON t.Bid = b.Bid
            WHERE b.brname = branch_name
          LOOP
            RAISE NOTICE 'Loan No: %, Amount Required: %, Approved
Amount: %, Date: %', rec.Lno, rec.l_amt_required, rec.lamtapproved,
rec.l_date;
          END LOOP;
        END;
        $$ LANGUAGE plpgsql;


        SELECT get_loan_info_by_branch('Pimpri');
```

BANK DATABASE

CREATE TABLE Branch (Bid INTEGER PRIMARY KEY, brname VARCHAR(30) NOT NULL, brcity VARCHAR(10) NOT NULL);

CREATE TABLE Customer (Cno INTEGER PRIMARY KEY, cname VARCHAR(20) NOT NULL, caddr VARCHAR(35), city VARCHAR(15));

CREATE TABLE Loan_application (Lno INTEGER PRIMARY KEY, l_amt_required INT CHECK (l_amt_required > 0), lamtapproved INT, l_date DATE);

CREATE TABLE Ternary (Bid INTEGER, Cno INTEGER, Lno INTEGER, PRIMARY KEY (Bid, Cno, Lno), FOREIGN KEY (Bid) REFERENCES Branch(Bid), FOREIGN KEY (Cno) REFERENCES Customer(Cno), FOREIGN KEY (Lno) REFERENCES Loan_application(Lno));

INSERT INTO Branch (Bid, brname, brcity) VALUES (1, 'Pimpri', 'Pimpri'), (2, 'Aundh', 'Aundh');

INSERT INTO Customer (Cno, cname, caddr, city) VALUES (1, 'Rahul', '123 Street', 'Pimpri'), (2, 'Neha', '456 Avenue', 'Aundh'), (3, 'Raj', '789 Boulevard', 'Pune');

INSERT INTO Loan_application (Lno, l_amt_required, lamtapproved, l_date) VALUES (101, 500000, 450000, '2024-09-01'), (102, 200000, 150000, '2024-09-05'), (103, 600000, 550000, '2024-09-10');

INSERT INTO Ternary (Bid, Cno, Lno) VALUES (1, 1, 101), (2, 2, 102), (2, 3, 103);

Q.1) Create a View:

CREATE VIEW Customers_Loan_500k AS SELECT c.* FROM Customer c JOIN Ternary t ON c.Cno = t.Cno JOIN Loan_application l ON t.Lno = l.Lno WHERE l.l_amt_required = 500000;

SELECT * FROM Customers_Loan_500k

CREATE VIEW Loans_From_Aundh_Branch AS SELECT l.* FROM Loan_application l JOIN Ternary t ON l.Lno = t.Lno JOIN Branch b ON t.Bid = b.Bid WHERE b.brcity = 'Aundh';

SELECT * FROM Loans_From_Aundh_Branch;

Q.2) Using above database solve following questions:

```
CREATE OR REPLACE FUNCTION validate_loan_amount()
RETURNS TRIGGER AS $$
BEGIN
  IF NEW.lamtapproved > NEW.l_amt_required THEN
    RAISE EXCEPTION 'Loan amount approved must be less than or equal to the loan amount required';
  END IF;
  RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trigger_validate_loan_amount
BEFORE INSERT OR UPDATE ON Loan_application
FOR EACH ROW
EXECUTE FUNCTION validate_loan_amount();

INSERT INTO Loan_application VALUES (104, 300000, 400000, '2024-09-15');
```

```
CREATE OR REPLACE FUNCTION
count_customers_in_branch(branch_name VARCHAR)
RETURNS INT AS $$
DECLARE
  customer_count INT;
BEGIN
  SELECT COUNT(c.Cno) INTO customer_count
  FROM Customer c
  JOIN Ternary t ON c.Cno = t.Cno
  JOIN Branch b ON t.Bid = b.Bid
  WHERE b.brname = branch_name;

  IF customer_count IS NULL OR customer_count = 0 THEN
    RAISE NOTICE 'Invalid branch name';
    RETURN 0;
  ELSE
    RETURN customer_count;
  END IF;
END;
$$ LANGUAGE plpgsql;

SELECT count_customers_in_branch('Aundh');
```

BANK DATABASE

```
CREATE TABLE Branch (Bid INTEGER PRIMARY KEY, brname VARCHAR(30) NOT NULL, brcity VARCHAR(10) NOT NULL);

CREATE TABLE Customer (Cno INTEGER PRIMARY KEY, cname VARCHAR(20) NOT NULL, caddr VARCHAR(35), city VARCHAR(15));

CREATE TABLE Loan_application (Lno INTEGER PRIMARY KEY, l_amt_required INT CHECK (l_amt_required > 0), lamtapproved INT, l_date DATE);

CREATE TABLE Ternary (Bid INTEGER, Cno INTEGER, Lno INTEGER, PRIMARY KEY (Bid, Cno, Lno), FOREIGN KEY (Bid) REFERENCES Branch(Bid), FOREIGN KEY (Cno) REFERENCES Customer(Cno), FOREIGN KEY (Lno) REFERENCES Loan_application(Lno));
```

```
INSERT INTO Branch (Bid, brname, brcity) VALUES (1, 'Pimpri', 'Pimpri'), (2, 'Aundh', 'Aundh');

INSERT INTO Customer (Cno, cname, caddr, city) VALUES (1, 'Rahul', '123 Street', 'Pimpri'), (2, 'Neha', '456 Avenue', 'Aundh'), (3, 'Raj', '789 Boulevard', 'Pune');

INSERT INTO Loan_application (Lno, l_amt_required, lamtapproved, l_date) VALUES (101, 500000, 450000, '2024-09-01'), (102, 200000, 150000, '2024-09-05'), (103, 600000, 550000, '2024-09-10');

INSERT INTO Ternary (Bid, Cno, Lno) VALUES (1, 1, 101), (2, 2, 102), (2, 3, 103);
```

Q.1) Create a View:

```
CREATE VIEW Customers_Loan_Above_200k AS SELECT c.cname FROM Customer c JOIN Ternary t ON c.Cno = t.Cno JOIN Loan_application l ON t.Lno = l.Lno WHERE l.l_amt_required > 200000;

SELECT * FROM Customers_Loan_Above_200k;

CREATE VIEW Branch_Wise_Customers AS SELECT b.brname, c.cname FROM Branch b JOIN Ternary t ON b.Bid = t.Bid JOIN Customer c ON t.Cno = c.Cno;

SELECT * FROM Branch_Wise_Customers;
```

Q.2) Using above database solve following questions:

```
CREATE OR REPLACE FUNCTION validate_customer_number()
RETURNS TRIGGER AS $$
BEGIN
   IF NEW.Cno <= 0 THEN
      RAISE EXCEPTION 'Customer number must be greater than zero';
   END IF;
   RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trigger_validate_customer_number
BEFORE INSERT ON Customer
FOR EACH ROW
EXECUTE FUNCTION validate_customer_number();

INSERT INTO Customer VALUES (-1, 'John', '101 Main St', 'Pune');
```

```
CREATE OR REPLACE FUNCTION display_customer_loan_details()
RETURNS VOID AS $$
DECLARE
   customer_cursor CURSOR FOR
      SELECT c.cname, c.caddr, l.lamtapproved
      FROM Customer c
      JOIN Ternary t ON c.Cno = t.Cno
      JOIN Loan_application l ON t.Lno = l.Lno;

   customer_record RECORD;
BEGIN
   OPEN customer_cursor;

   LOOP
      FETCH customer_cursor INTO customer_record;

      EXIT WHEN NOT FOUND;

      RAISE NOTICE 'Customer Name: %, Address: %, Approved Loan Amount: %', customer_record.cname, customer_record.caddr, customer_record.lamtapproved;
   END LOOP;

   CLOSE customer_cursor;
END;
$$ LANGUAGE plpgsql;

SELECT display_customer_loan_details();
```

BUS-TRANSPORT SYSTEM

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 DECIMAL);

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

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

INSERT INTO Route VALUES (1, 'Mumbai', 'Pune', 5), (2, 'Nashik', 'Mumbai', 7), (3, 'Ahmednagar', 'Aurangabad', 6), (4, 'Pune', 'Nagpur', 10), (5, 'Mumbai', 'Goa', 8);

INSERT INTO Bus VALUES (101, 30, 'Depot A'), (102, 40, 'Depot B'), (103, 50, 'Depot C'), (104, 35, 'Depot D'), (105, 30, 'Depot E');

INSERT INTO Driver VALUES (1, 'Rajesh', 123456, 'Mumbai', 45, 25000), (2, 'Amit', 654321, 'Pune', 35, 18000), (3, 'Sunil', 789456, 'Nashik', 50, 22000), (4, 'Suresh', 111222, 'Aurangabad', 40, 24000), (5, 'Mahesh', 222333, 'Nagpur', 29, 26000), (6, 'Anil', 333444, 'Goa', 55, 27000);

INSERT INTO Bus_Route VALUES (101, 1), (102, 2), (103, 3), (104, 4), (105, 5);

INSERT INTO Bus_Driver VALUES (101, 1, 1, '2024-10-10'), (101, 1, 2, '2024-10-10'), (102, 2, 1, '2024-10-11'), (102, 3, 2, '2024-10-11'), (103, 4, 1, '2024-10-12'), (103, 5, 2, '2024-10-12'), (104, 5, 1, '2024-10-13'), (104, 6, 2, '2024-10-13'), (105, 1, 1, '2024-10-14'), (105, 6, 2, '2024-10-14');

Q.1) Create a View:

1. CREATE VIEW Morning_Shift_Drivers AS SELECT d.Driver_no, d.driver_name, d.license_no, d.address, d.age, d.salary FROM Driver d JOIN Bus_Driver bd ON d.Driver_no = bd.Driver_no WHERE bd.Shift = 1;
   SELECT * FROM Morning_Shift_Drivers;

2. CREATE VIEW High_Salary_Drivers AS SELECT * FROM Driver WHERE salary > 20000;
   SELECT * FROM High_Salary_Drivers;

Q.2) Using above database solve following questions:

1.
```
CREATE OR REPLACE FUNCTION check_driver_age()
RETURNS TRIGGER AS $$
BEGIN
IF NEW.age < 18 OR NEW.age > 35 THEN
RAISE EXCEPTION 'Invalid input: Driver age must be between 18 and 35.';
END IF;
RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trigger_check_age
BEFORE INSERT ON Driver
FOR EACH ROW
EXECUTE FUNCTION check_driver_age();

INSERT INTO Driver VALUES (6, 'Manoj', 998877, 'Nagpur', 40, 20000);

INSERT INTO Driver VALUES (5, 'Suresh', 112233, 'Delhi', 30, 25000);
```

2.
```
CREATE OR REPLACE FUNCTION get_buses_by_route(route_id INT)
RETURNS VOID AS $$
DECLARE
  bus_record RECORD;
BEGIN
  RAISE NOTICE 'Function get_buses_by_route called with route_id: %', route_id;

  IF NOT EXISTS (SELECT 1 FROM Route WHERE Route_no = route_id) THEN
    RAISE EXCEPTION 'Route ID % does not exist', route_id;
  END IF;

  FOR bus_record IN
    SELECT b.Bus_no, b.capacity, b.depot_name
    FROM Bus b
    JOIN Bus_Route br ON b.Bus_no = br.Bus_no
    WHERE br.Route_no = route_id
  LOOP
    RAISE NOTICE 'Bus_no: %, Capacity: %, Depot_name: %',
           bus_record.Bus_no, bus_record.capacity, bus_record.depot_name;
  END LOOP;
END;
$$ LANGUAGE plpgsql;

SELECT get_buses_by_route(1);
```

BUS-TRANSPORT SYSTEM

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 DECIMAL);

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

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

INSERT INTO Route VALUES (1, 'Mumbai', 'Pune', 5), (2, 'Nashik', 'Mumbai', 7), (3, 'Ahmednagar', 'Aurangabad', 6), (4, 'Pune', 'Nagpur', 10), (5, 'Mumbai', 'Goa', 8);

INSERT INTO Bus VALUES (101, 30, 'Depot A'), (102, 40, 'Depot B'), (103, 50, 'Depot C'), (104, 35, 'Depot D'), (105, 30, 'Depot E');

INSERT INTO Driver VALUES (1, 'Rajesh', 123456, 'Mumbai', 45, 25000), (2, 'Amit', 654321, 'Pune', 35, 18000), (3, 'Sunil', 789456, 'Nashik', 50, 22000), (4, 'Suresh', 111222, 'Aurangabad', 40, 24000), (5, 'Mahesh', 222333, 'Nagpur', 29, 26000), (6, 'Anil', 333444, 'Goa', 55, 27000);

INSERT INTO Bus_Route VALUES (101, 1), (102, 2), (103, 3), (104, 4), (105, 5);

INSERT INTO Bus_Driver VALUES (101, 1, 1, '2024-10-10'), (101, 1, 2, '2024-10-10'), (102, 2, 1, '2024-10-11'), (102, 3, 2, '2024-10-11'), (103, 4, 1, '2024-10-12'), (103, 5, 2, '2024-10-12'), (104, 5, 1, '2024-10-13'), (104, 6, 2, '2024-10-13'), (105, 1, 1, '2024-10-14'), (105, 6, 2, '2024-10-14');

Q.1) Create a View:

1. CREATE VIEW Bus_102_Drivers AS SELECT b.Bus_no, b.capacity, b.depot_name, d.Driver_no, d.driver_name, d.license_no, bd.Shift, bd.Date_of_duty_allotted 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;
   SELECT * FROM Bus_102_Drivers;

2. CREATE VIEW Route_Bus_Capacity_30 AS SELECT r.Route_no, r.source, r.destination, r.no_of_stations FROM Route r JOIN Bus_Route br ON r.Route_no = br.Route_no JOIN Bus b ON br.Bus_no = b.Bus_no WHERE b.capacity = 30;
   SELECT * FROM Route_Bus_Capacity_30;

Q.2) Using above database solve following questions:

1.
```
CREATE OR REPLACE FUNCTION check_driver_salary()
RETURNS TRIGGER AS $$
BEGIN
  IF NEW.salary <= 0 THEN
    RAISE EXCEPTION 'Invalid Salary: Salary must be greater than
zero.';
  END IF;
  RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trigger_check_salary
BEFORE INSERT ON Driver
FOR EACH ROW
EXECUTE FUNCTION check_driver_salary();

INSERT INTO Driver VALUES (7, 'Karan', 445566, 'Chennai', 28, 0);

INSERT INTO Driver VALUES (7, 'Vijay', 223344, 'Goa', 32, 18000);
```

2.
```
CREATE OR REPLACE FUNCTION get_driver_dates(d_name VARCHAR)
RETURNS VOID AS $$
DECLARE
  rec RECORD;
BEGIN
  FOR rec IN
    SELECT bd.Date_of_duty_allotted
    FROM Bus_Driver bd
    JOIN Driver d ON bd.Driver_no = d.Driver_no
    WHERE d.driver_name = d_name
  LOOP
    RAISE NOTICE 'Date of duty: %', rec.Date_of_duty_allotted;
  END LOOP;

  IF NOT FOUND THEN
    RAISE NOTICE 'No records found for driver: %', d_name;
  END IF;
END;
$$ LANGUAGE plpgsql;

SELECT get_driver_dates('Rajesh');
```

BUS-TRANSPORT SYSTEM

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 DECIMAL);

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

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

INSERT INTO Route VALUES (1, 'Mumbai', 'Pune', 5), (2, 'Nashik', 'Mumbai', 7), (3, 'Ahmednagar', 'Aurangabad', 6), (4, 'Pune', 'Nagpur', 10), (5, 'Mumbai', 'Goa', 8);

INSERT INTO Bus VALUES (101, 30, 'Depot A'), (102, 40, 'Depot B'), (103, 50, 'Depot C'), (104, 35, 'Depot D'), (105, 30, 'Depot E');

INSERT INTO Driver VALUES (1, 'Rajesh', 123456, 'Mumbai', 45, 25000), (2, 'Amit', 654321, 'Pune', 35, 18000), (3, 'Sunil', 789456, 'Nashik', 50, 22000), (4, 'Suresh', 111222, 'Aurangabad', 40, 24000), (5, 'Mahesh', 222333, 'Nagpur', 29, 26000), (6, 'Anil', 333444, 'Goa', 55, 27000);

INSERT INTO Bus_Route VALUES (101, 1), (102, 2), (103, 3), (104, 4), (105, 5);

INSERT INTO Bus_Driver VALUES (101, 1, 1, '2024-10-10'), (101, 1, 2, '2024-10-10'), (102, 2, 1, '2024-10-11'), (102, 3, 2, '2024-10-11'), (103, 4, 1, '2024-10-12'), (103, 5, 2, '2024-10-12'), (104, 5, 1, '2024-10-13'), (104, 6, 2, '2024-10-13'), (105, 1, 1, '2024-10-14'), (105, 6, 2, '2024-10-14');

Q.1) Create a View:

1. CREATE VIEW Drivers_Both_Shifts AS SELECT d.driver_name FROM Driver d JOIN Bus_Driver bd1 ON d.Driver_no = bd1.Driver_no AND bd1.Shift = 1 JOIN Bus_Driver bd2 ON d.Driver_no = bd2.Driver_no AND bd2.Shift = 2 GROUP BY d.driver_name;
   SELECT * FROM Drivers_Both_Shifts;

2. CREATE VIEW Route_Bus_101 AS SELECT r.Route_no, r.source, r.destination, r.no_of_stations FROM Route r JOIN Bus_Route br ON r.Route_no = br.Route_no WHERE br.Bus_no = 101;
   SELECT * FROM Route_Bus_101;

Q.2) Using above database solve following questions:

1.

```
CREATE OR REPLACE FUNCTION after_delete_bus()
RETURNS TRIGGER AS $$
BEGIN
   IF OLD.capacity < 20 THEN
      RAISE NOTICE 'Bus with capacity less than 20 has been
deleted.';
   END IF;
   RETURN OLD;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trigger_after_delete_bus
AFTER DELETE ON Bus
FOR EACH ROW
EXECUTE FUNCTION after_delete_bus();

INSERT INTO Bus VALUES (103, 15, 'Depot C');

DELETE FROM Bus WHERE Bus_no = 103;

DELETE FROM Bus WHERE Bus_no = 101;
```

2.

```
CREATE OR REPLACE FUNCTION display_buses_on_route_1()
RETURNS VOID AS $$
DECLARE
   rec RECORD;
BEGIN
   FOR rec IN
      SELECT b.Bus_no, b.capacity, b.depot_name
      FROM Bus b
      JOIN Bus_Route br ON b.Bus_no = br.Bus_no
      WHERE br.Route_no = 1
   LOOP
      RAISE NOTICE 'Bus No: %, Capacity: %, Depot: %',
rec.Bus_no, rec.capacity, rec.depot_name;
   END LOOP;

   IF NOT FOUND THEN
      RAISE NOTICE 'No buses found on Route No: 1';
   END IF;
END;
$$ LANGUAGE plpgsql;

SELECT display_buses_on_route_1();
```

RAILWAY RESERVATION

```
CREATE TABLE Train (Train_no INTEGER PRIMARY KEY, train_name VARCHAR(20), depart_time TIME, arrival_time TIME, source_stn VARCHAR(20), dest_stn VARCHAR(20), no_of_res_bogies INTEGER, bogie_capacity INTEGER);

CREATE TABLE Passenger (Passenger_id INTEGER PRIMARY KEY, passenger_name VARCHAR(20), address VARCHAR(30), age INTEGER, gender CHAR(1));

CREATE TABLE Ticket (Ticket_no INTEGER PRIMARY KEY, Train_no INTEGER, Passenger_id INTEGER, bogie_no INTEGER, no_of_berths INTEGER, tdate DATE, ticket_amt DECIMAL(7, 2), ticket_status CHAR(1) CHECK (ticket_status IN ('W', 'C')), FOREIGN KEY (Train_no) REFERENCES Train (Train_no), FOREIGN KEY (Passenger_id) REFERENCES Passenger (Passenger_id));

INSERT INTO Train VALUES (101, 'Shatabdi Express', '08:00', '14:00', 'Mumbai', 'Delhi', 10, 72),(102, 'Rajdhani Express', '06:00', '12:00', 'Delhi', 'Chennai', 12, 70);

INSERT INTO Passenger VALUES(1, 'Rahul', 'Mumbai', 30, 'M'),(2, 'Anjali', 'Pune', 25, 'F'),(3, 'Amit', 'Delhi', 35, 'M'),(4, 'Priya', 'Bangalore', 28, 'F'),(5, 'Suresh', 'Hyderabad', 40, 'M');

INSERT INTO Ticket VALUES(1001, 101, 1, 1, 1, '2022-03-02', 1500.00, 'W'),(1002, 101, 2, 1, 1, '2022-03-02', 1500.00, 'C'),(1003, 101, 3, 1, 1, '2022-03-02', 1500.00, 'C'),(1004, 102, 4, 2, 1, '2021-05-04', 2000.00, 'C'),(1005, 102, 5, 2, 1, '2021-05-04', 2000.00, 'C'),(1006, 102, 1, 2, 1, '2021-05-04', 2000.00, 'W'),(1007, 102, 3, 2, 1, '2022-01-01', 2000.00, 'C');
```

Q.1) Create a View:

1.
   ```
   CREATE VIEW Shatabdi_Waiting AS SELECT P.passenger_name FROM Passenger P JOIN Ticket T ON P.Passenger_id = T.Passenger_id JOIN Train TR ON T.Train_no = TR.Train_no WHERE TR.train_name = 'Shatabdi Express' AND T.ticket_status = 'W' AND T.tdate = '2022-03-02';
   SELECT * FROM Shatabdi_Waiting;
   ```

2.
   ```
   CREATE VIEW Rajdhani_Bookings AS SELECT T.Ticket_no, P.passenger_name, T.ticket_amt FROM Passenger P JOIN Ticket T ON P.Passenger_id = T.Passenger_id JOIN Train TR ON T.Train_no = TR.Train_no WHERE TR.train_name = 'Rajdhani Express' AND T.tdate = '2021-05-04' ORDER BY T.Ticket_no LIMIT 3;
   SELECT * FROM Rajdhani_Bookings;
   ```

Q.2) Using above database solve following questions:

```
CREATE OR REPLACE FUNCTION restrict_bogie_capacity() RETURNS TRIGGER AS $$
BEGIN
  IF NEW.bogie_capacity > 25 THEN
    RAISE EXCEPTION 'Bogie capacity cannot exceed 30';
  END IF;
  RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER check_bogie_capacity
BEFORE INSERT OR UPDATE ON Train
FOR EACH ROW EXECUTE FUNCTION restrict_bogie_capacity();

 INSERT INTO Train VALUES (103, 'Duronto Express', '09:00', '15:00', 'Kolkata', 'Mumbai', 12, 35);
```

```
CREATE OR REPLACE FUNCTION display_train_wise_confirmed_bookings()
RETURNS VOID AS $$
DECLARE
  train_record RECORD;
  ticket_record RECORD;
  booking_date DATE := '2022-04-19';
BEGIN
  FOR train_record IN SELECT Train_no, train_name FROM Train
  LOOP
    RAISE NOTICE 'Train: %, Train Name: %', train_record.Train_no, train_record.train_name;

    FOR ticket_record IN
      SELECT T.Ticket_no, P.passenger_name, T.ticket_amt
      FROM Ticket T
      JOIN Passenger P ON T.Passenger_id = P.Passenger_id
      WHERE T.Train_no = train_record.Train_no
      AND T.ticket_status = 'C'
      AND T.tdate = booking_date
    LOOP
      RAISE NOTICE 'Ticket No: %, Passenger: %, Amount: %', ticket_record.Ticket_no, ticket_record.passenger_name, ticket_record.ticket_amt;
    END LOOP;

  END LOOP;
END;
$$ LANGUAGE plpgsql;

SELECT display_train_wise_confirmed_bookings();
```

RAILWAY RESERVATION

CREATE TABLE Train (Train_no INTEGER PRIMARY KEY, train_name VARCHAR(20), depart_time TIME, arrival_time TIME, source_stn VARCHAR(20), dest_stn VARCHAR(20), no_of_res_bogies INTEGER, bogie_capacity INTEGER);

CREATE TABLE Passenger (Passenger_id INTEGER PRIMARY KEY, passenger_name VARCHAR(20), address VARCHAR(30), age INTEGER, gender CHAR(1));

CREATE TABLE Ticket (Ticket_no INTEGER PRIMARY KEY, Train_no INTEGER, Passenger_id INTEGER, bogie_no INTEGER, no_of_berths INTEGER, tdate DATE, ticket_amt DECIMAL(7, 2), ticket_status CHAR(1) CHECK (ticket_status IN ('W', 'C')), FOREIGN KEY (Train_no) REFERENCES Train (Train_no), FOREIGN KEY (Passenger_id) REFERENCES Passenger (Passenger_id));

INSERT INTO Train VALUES (101, 'Shatabdi Express', '08:00', '14:00', 'Mumbai', 'Delhi', 10, 72),(102, 'Rajdhani Express', '06:00', '12:00', 'Delhi', 'Chennai', 12, 70);

INSERT INTO Passenger VALUES(1, 'Rahul', 'Mumbai', 30, 'M'),(2, 'Anjali', 'Pune', 25, 'F'),(3, 'Amit', 'Delhi', 35, 'M'),(4, 'Priya', 'Bangalore', 28, 'F'),(5, 'Suresh', 'Hyderabad', 40, 'M');

INSERT INTO Ticket VALUES(1001, 101, 1, 1, 1, '2022-03-02', 1500.00, 'W'),(1002, 101, 2, 1, 1, '2022-03-02', 1500.00, 'C'),(1003, 101, 3, 1, 1, '2022-03-02', 1500.00, 'C'),(1004, 102, 4, 2, 1, '2021-05-04', 2000.00, 'C'),(1005, 102, 5, 2, 1, '2021-05-04', 2000.00, 'C'),(1006, 102, 1, 2, 1, '2021-05-04', 2000.00, 'W'),(1007, 102, 3, 2, 1, '2022-01-01', 2000.00, 'C');

Q.1) Create a View:

1. CREATE VIEW Shatabdi_Confirmed_Passengers AS SELECT P.passenger_name FROM Passenger P JOIN Ticket T ON P.Passenger_id = T.Passenger_id JOIN Train TR ON T.Train_no = TR.Train_no WHERE TR.train_name = 'Shatabdi Express' AND T.ticket_status = 'C' AND T.tdate = '2022-03-02';
   SELECT * FROM Shatabdi_Confirmed_Passengers;
2. CREATE VIEW Rajdhani_Confirmed_Bookings_Count AS SELECT COUNT(*) AS confirmed_booking_count FROM Ticket T JOIN Train TR ON T.Train_no = TR.Train_no WHERE TR.train_name = 'Rajdhani Express' AND T.ticket_status = 'C' AND T.tdate = '2022-01-01';
   SELECT * FROM Rajdhani_Confirmed_Bookings_Count;

Q.2) Using above database solve following questions:

```
CREATE OR REPLACE FUNCTION check_age()
RETURNS TRIGGER AS $$
BEGIN
  IF NEW.age > 5 THEN
    RAISE NOTICE 'Age above 5 years will be charged the full fare';
  END IF;
  RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER age_check_trigger
AFTER INSERT ON Passenger
FOR EACH ROW EXECUTE FUNCTION check_age();

INSERT INTO Passenger  VALUES (6, 'Vikram', 'Delhi', 6, 'M');
INSERT INTO Passenger VALUES (7, 'Sita', 'Mumbai', 5, 'F');
```

```
CREATE OR REPLACE FUNCTION display_train_wise_waiting_bookings()
RETURNS VOID AS $$
DECLARE
  train_record RECORD;
  ticket_record RECORD;
  booking_date DATE := '2020-05-02';
BEGIN
  FOR train_record IN SELECT Train_no, train_name FROM Train
  LOOP
    RAISE NOTICE 'Train: %, Train Name: %', train_record.Train_no, train_record.train_name;

    FOR ticket_record IN
      SELECT T.Ticket_no, P.passenger_name
      FROM Ticket T
      JOIN Passenger P ON T.Passenger_id = P.Passenger_id
      WHERE T.Train_no = train_record.Train_no
      AND T.ticket_status = 'W'
      AND T.tdate = booking_date
    LOOP
      RAISE NOTICE 'Ticket No: %, Passenger: %', ticket_record.Ticket_no, ticket_record.passenger_name;
    END LOOP;

  END LOOP;
END;
$$ LANGUAGE plpgsql;

SELECT display_train_wise_waiting_bookings();
```

PROJECT-EMPLOYEE DATABASE

CREATE TABLE Project (Pno INTEGER 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 INTEGER, Eno INTEGER, start_date_of_project DATE, no_of_hours_worked INTEGER, PRIMARY KEY (Pno, Eno), FOREIGN KEY (Pno) REFERENCES Project(Pno), FOREIGN KEY (Eno) REFERENCES Employee(Eno));

INSERT INTO Project VALUES (1, 'Robotics', 'Research', 24), (2, 'ERP', 'Development', 18), (3, 'AI Model', 'Research', 12), (4, 'Web Application', 'Development', 9);

INSERT INTO Employee VALUES (101, 'Amit', 'B.Tech', '2020-01-10'), (102, 'Priya', 'MCA', '2021-03-15'), (103, 'Rahul', 'B.Sc', '2019-07-22'), (104, 'Sneha', 'M.Tech', '2022-06-10');

INSERT INTO Project_Employee VALUES (1, 101, '2022-05-01', 120), (2, 102, '2022-04-15', 90), (1, 103, '2021-08-10', 50), (3, 104, '2023-01-12', 60), (2, 101, '2022-08-22', 130), (4, 102, '2022-09-05', 70);

Q.1) Create a View:

CREATE OR REPLACE VIEW Project_Details AS SELECT p.pname, p.ptype, pe.start_date_of_project FROM Project p JOIN Project_Employee pe ON p.Pno = pe.Pno ORDER BY pe.start_date_of_project;

SELECT * FROM Project_Details;

CREATE OR REPLACE VIEW Employees_On_Robotics AS SELECT e.Eno, e.ename, e.qualification, e.joining_date FROM Employee e JOIN Project_Employee pe ON e.Eno = pe.Eno JOIN Project p ON p.Pno = pe.Pno WHERE p.pname = 'Robotics';

SELECT * FROM Employees_On_Robotics;

Q.2) Using above database solve following questions:

```
CREATE OR REPLACE FUNCTION check_duration()
RETURNS TRIGGER AS $$
BEGIN
  IF NEW.duration <= 0 THEN
    RAISE EXCEPTION 'Duration must be greater than zero';
  END IF;
  RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER before_insert_duration
BEFORE INSERT ON Project
FOR EACH ROW EXECUTE FUNCTION check_duration();
```

INSERT INTO Project VALUES (101, 'AI Research', 'Research', 0);

```
CREATE OR REPLACE FUNCTION
get_employees_by_project(p_name VARCHAR)
RETURNS VOID AS $$
DECLARE
  emp_record RECORD;
  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 = p_name;
BEGIN
  OPEN emp_cursor;

  LOOP
    FETCH emp_cursor INTO emp_record;
    EXIT WHEN NOT FOUND;
    RAISE NOTICE 'Employee Name: %', emp_record.ename;
  END LOOP;

  CLOSE emp_cursor;
END;
$$ LANGUAGE plpgsql;
```

SELECT get_employees_by_project('Robotics');

PROJECT-EMPLOYEE DATABASE

```
CREATE TABLE Project (Pno INTEGER 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 INTEGER, Eno INTEGER, start_date_of_project DATE, no_of_hours_worked INTEGER, PRIMARY KEY (Pno, Eno), FOREIGN KEY (Pno) REFERENCES Project(Pno), FOREIGN KEY (Eno) REFERENCES Employee(Eno));

INSERT INTO Project VALUES (1, 'Robotics', 'Research', 24), (2, 'ERP', 'Development', 18), (3, 'AI Model', 'Research', 12), (4, 'Web Application', 'Development', 9);

INSERT INTO Employee VALUES (101, 'Amit', 'B.Tech', '2020-01-10'), (102, 'Priya', 'MCA', '2021-03-15'), (103, 'Rahul', 'B.Sc', '2019-07-22'), (104, 'Sneha', 'M.Tech', '2022-06-10');

INSERT INTO Project_Employee VALUES (1, 101, '2022-05-01', 120), (2, 102, '2022-04-15', 90), (1, 103, '2021-08-10', 50), (3, 104, '2023-01-12', 60), (2, 101, '2022-08-22', 130), (4, 102, '2022-09-05', 70);
```

Q.1) Create a View:

```
CREATE OR REPLACE VIEW Employee_Details AS SELECT Eno, ename, qualification, joining_date FROM Employee ORDER BY joining_date;

SELECT * FROM Employee_Details;

CREATE OR REPLACE VIEW Employees_Worked_Less_Than_100_Hours AS SELECT e.Eno, e.ename, p.pname, pe.no_of_hours_worked FROM Employee e JOIN Project_Employee pe ON e.Eno = pe.Eno JOIN Project p ON p.Pno = pe.Pno WHERE pe.no_of_hours_worked < 100;

SELECT * FROM Employees_Worked_Less_Than_100_Hours;
```

Q.2) Using above database solve following questions:

```
CREATE OR REPLACE FUNCTION check_joining_date()
RETURNS TRIGGER AS $$
BEGIN
   IF NEW.joining_date >= CURRENT_DATE THEN
      RAISE EXCEPTION 'Joining date must be before the current date.';
   END IF;
   RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER validate_joining_date
BEFORE INSERT OR UPDATE ON Employee
FOR EACH ROW
EXECUTE FUNCTION check_joining_date();

INSERT INTO Employee VALUES (101, 'Vijay', 'Bachelors', '2022-09-01');

INSERT INTO Employee VALUES (102, 'Anil', 'Masters', CURRENT_DATE);
```

```
CREATE OR REPLACE FUNCTION get_employee_count_by_project(p_project_name VARCHAR)
RETURNS INTEGER AS $$
DECLARE
   employee_count INTEGER;
BEGIN
   SELECT COUNT(*)
   INTO employee_count
   FROM Project
   WHERE pname = p_project_name;

   IF employee_count = 0 THEN
      RAISE EXCEPTION 'Invalid project name: %', p_project_name;
   END IF;

   SELECT COUNT(*)
   INTO employee_count
   FROM Project_Employee
   WHERE project_name = p_project_name;

   RETURN employee_count;
END;
$$ LANGUAGE plpgsql;

SELECT get_employee_count_by_project('Robotics');
```

PROJECT-EMPLOYEE DATABASE

CREATE TABLE Project (Pno INTEGER 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 INTEGER, Eno INTEGER, start_date_of_project DATE, no_of_hours_worked INTEGER, PRIMARY KEY (Pno, Eno), FOREIGN KEY (Pno) REFERENCES Project(Pno), FOREIGN KEY (Eno) REFERENCES Employee(Eno));

INSERT INTO Project VALUES (1, 'Robotics', 'Research', 24), (2, 'ERP', 'Development', 18), (3, 'AI Model', 'Research', 12), (4, 'Web Application', 'Development', 9);

INSERT INTO Employee VALUES (101, 'Amit', 'B.Tech', '2020-01-10'), (102, 'Priya', 'MCA', '2021-03-15'), (103, 'Rahul', 'B.Sc', '2019-07-22'), (104, 'Sneha', 'M.Tech', '2022-06-10');

INSERT INTO Project_Employee VALUES (1, 101, '2022-05-01', 120), (2, 102, '2022-04-15', 90), (1, 103, '2021-08-10', 50), (3, 104, '2023-01-12', 60), (2, 101, '2022-08-22', 130), (4, 102, '2022-09-05', 70);

Q.1) Create a View:

CREATE OR REPLACE VIEW Employees_On_ERP AS SELECT e.Eno, e.ename, e.qualification, e.joining_date FROM Employee e JOIN Project_Employee pe ON e.Eno = pe.Eno JOIN Project p ON p.Pno = pe.Pno WHERE p.pname = 'ERP';

SELECT * FROM Employees_On_ERP;

CREATE OR REPLACE VIEW Employees_Worked_More_Than_100_Hours AS SELECT e.Eno, e.ename, p.pname, pe.no_of_hours_worked FROM Employee e JOIN Project_Employee pe ON e.Eno = pe.Eno JOIN Project p ON p.Pno = pe.Pno WHERE pe.no_of_hours_worked > 100;

SELECT * FROM Employees_Worked_More_Than_100_Hours;

Q.2) Using above database solve following questions:

```
CREATE OR REPLACE FUNCTION after_project_delete()
RETURNS TRIGGER AS $$
BEGIN
  RAISE NOTICE 'Project record is being deleted';
  RETURN OLD;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER project_delete_trigger
AFTER DELETE ON Project
FOR EACH ROW EXECUTE FUNCTION after_project_delete();


DELETE FROM Project WHERE Pno = 1;
```

```
      CREATE OR REPLACE FUNCTION
count_employees_before_joining_date(target_date DATE)
      RETURNS INTEGER AS $$
      DECLARE
        employee_count INTEGER;
      BEGIN
        SELECT COUNT(*)
        INTO employee_count
        FROM Employee
        WHERE joining_date < target_date;

        IF employee_count = 0 THEN
          RAISE EXCEPTION 'No employees found who joined before %',
target_date;
        END IF;

        RETURN employee_count;
      END;
      $$ LANGUAGE plpgsql;

      SELECT count_employees_before_joining_date('2022-10-03');
```

STUDENT-TEACHER DATABASE

```
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));

INSERT INTO Student (Sno, sname, sclass, saddr) VALUES (1, 'Rahul', '10th', 'Pune'), (2, 'Sneha', '12th', 'Mumbai'), (3, 'Amit', '11th', 'Pune'), (4, 'Vijay', '10th', 'Nashik');

INSERT INTO Teacher (Tno, tname, qualification, experience) VALUES (1, 'Sharma', 'Ph.D.', 10), (2, 'Joshi', 'M.Sc.', 4), (3, 'Singh', 'Ph.D.', 7), (4, 'Gupta', 'M.A.', 5);

INSERT INTO Student_Teacher (Sno, Tno, subject) VALUES (1, 1, 'Mathematics'), (1, 3, 'Physics'), (2, 2, 'Chemistry'), (3, 1, 'Mathematics'), (4, 3, 'Biology');
```

Q.1) Create a View:

```
CREATE VIEW MostExperiencedTeacher 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);

SELECT * FROM MostExperiencedTeacher;

CREATE VIEW SubjectsByTeacher AS SELECT t.tname, st.subject FROM Teacher t JOIN Student_Teacher st ON t.Tno = st.Tno;

SELECT * FROM SubjectsByTeacher;
```

Q.2) Using above database solve following questions:

```
CREATE OR REPLACE FUNCTION validate_student_number()
RETURNS TRIGGER AS $$
BEGIN
   IF NEW.Sno <= 0 THEN
      RAISE EXCEPTION 'Invalid student number';
   END IF;
   RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER before_insert_student
BEFORE INSERT ON Student
FOR EACH ROW
EXECUTE FUNCTION validate_student_number();

INSERT INTO Student VALUES (0, 'John Doe', '10th', 'Mumbai');
```

```
CREATE OR REPLACE FUNCTION count_students_by_subject(subject_name VARCHAR)
RETURNS INTEGER AS $$
DECLARE
   student_count INTEGER;
BEGIN
   IF subject_name IS NULL OR subject_name = '' THEN
      RAISE EXCEPTION 'Invalid subject name';
   END IF;

   SELECT COUNT(DISTINCT s.Sno) INTO student_count
   FROM Student s
   JOIN Student_Teacher st ON s.Sno = st.Sno
   WHERE st.subject = subject_name;

   RETURN student_count;
END;
$$ LANGUAGE plpgsql;

SELECT count_students_by_subject('Mathematics');
```

STUDENT-TEACHER DATABASE

```
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));

INSERT INTO Student (Sno, sname, sclass, saddr) VALUES (1, 'Rahul', '10th', 'Pune'), (2, 'Sneha', '12th', 'Mumbai'), (3, 'Amit', '11th', 'Pune'), (4, 'Vijay', '10th', 'Nashik');

INSERT INTO Teacher (Tno, tname, qualification, experience) VALUES (1, 'Sharma', 'Ph.D.', 10), (2, 'Joshi', 'M.Sc.', 4), (3, 'Singh', 'Ph.D.', 7), (4, 'Gupta', 'M.A.', 5);

INSERT INTO Student_Teacher (Sno, Tno, subject) VALUES (1, 1, 'Mathematics'), (1, 3, 'Physics'), (2, 2, 'Chemistry'), (3, 1, 'Mathematics'), (4, 3, 'Biology');
```

Q.1) Create a View:

```
CREATE VIEW PhDTeachers AS SELECT * FROM Teacher WHERE qualification = 'Ph.D.';

SELECT * FROM PhDTeachers;

CREATE VIEW StudentsInPune AS SELECT * FROM Student WHERE saddr = 'Pune';

SELECT * FROM StudentsInPune;
```

Q.2) Using above database solve following questions:

```
CREATE OR REPLACE FUNCTION validate_teacher_experience()
RETURNS TRIGGER AS $$
BEGIN
  IF NEW.experience < 5 THEN
    RAISE EXCEPTION 'Experience should be a minimum of 5
years';
  END IF;
  RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER before_insert_teacher_experience
BEFORE INSERT ON Teacher
FOR EACH ROW
EXECUTE FUNCTION validate_teacher_experience();


INSERT INTO Teacher VALUES (1, 'Mr. Smith', 'M.Sc.', 3);
```

```
CREATE OR REPLACE FUNCTION
list_teachers_for_student(student_name VARCHAR)
RETURNS SETOF Teacher AS $$
DECLARE
  teacher_record Teacher%ROWTYPE;
  cur CURSOR FOR
    SELECT t.*
    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;

BEGIN
  OPEN cur;
  LOOP
    FETCH cur INTO teacher_record;
    EXIT WHEN NOT FOUND;
    RETURN NEXT teacher_record;
  END LOOP;
  CLOSE cur;
  RETURN;
END;
$$ LANGUAGE plpgsql;

SELECT * FROM list_teachers_for_student('Rahul');
```

STUDENT-TEACHER DATABASE

```
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));

INSERT INTO Student (Sno, sname, sclass, saddr) VALUES (1, 'Rahul', '10th', 'Pune'), (2, 'Sneha', '12th', 'Mumbai'), (3, 'Amit', '11th', 'Pune'), (4, 'Vijay', '10th', 'Nashik');

INSERT INTO Teacher (Tno, tname, qualification, experience) VALUES (1, 'Sharma', 'Ph.D.', 10), (2, 'Joshi', 'M.Sc.', 4), (3, 'Singh', 'Ph.D.', 7), (4, 'Gupta', 'M.A.', 5);

INSERT INTO Student_Teacher (Sno, Tno, subject) VALUES (1, 1, 'Mathematics'), (1, 3, 'Physics'), (2, 2, 'Chemistry'), (3, 1, 'Mathematics'), (4, 3, 'Biology');
```

Q.1) Create a View:

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

SELECT * FROM ExperiencedTeachers;

CREATE VIEW TeachersStartingWithS AS SELECT * FROM Teacher WHERE tname LIKE 'S%';

SELECT * FROM TeachersStartingWithS;
```

Q.2) Using above database solve following questions:

```
CREATE OR REPLACE FUNCTION before_update_student_class()
RETURNS TRIGGER AS $$
BEGIN
   IF OLD.sclass IS DISTINCT FROM NEW.sclass THEN
      RAISE NOTICE 'Updating class for student: %', OLD.sname;
   END IF;
   RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER before_update_student_class_trigger
BEFORE UPDATE ON Student
FOR EACH ROW
EXECUTE FUNCTION before_update_student_class();

UPDATE Student SET sclass = '12th' WHERE sname = 'Rahul';
```

```
CREATE OR REPLACE FUNCTION
count_teachers_for_student(student_name VARCHAR)
RETURNS INTEGER AS $$
DECLARE
   teacher_count INTEGER;
BEGIN
   SELECT COUNT(DISTINCT t.Tno) 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;

SELECT count_teachers_for_student('Rahul');
```

STUDENT – MARKS DATABASE

CREATE TABLE Students (Rollno INTEGER PRIMARY KEY, sname VARCHAR(30) NOT NULL, city VARCHAR(50), class VARCHAR(10));

CREATE TABLE Subjects (Scode VARCHAR(10) PRIMARY KEY, subject_name VARCHAR(20));

CREATE TABLE Students_Subjects (Rollno INTEGER, Scode VARCHAR(10), marks_scored INTEGER, PRIMARY KEY (Rollno, Scode), FOREIGN KEY (Rollno) REFERENCES Students(Rollno), FOREIGN KEY (Scode) REFERENCES Subjects(Scode));

INSERT INTO Students (Rollno, sname, city, class) VALUES (1, 'Amit', 'Mumbai', 'FYBCA'), (2, 'Anjali', 'Pune', 'SYBCA'), (3, 'Rahul', 'Nagpur', 'TYBCA'), (4, 'Arjun', 'Nashik', 'FYBCA');

INSERT INTO Subjects (Scode, subject_name) VALUES ('S101', 'DBMS'), ('S102', 'Math'), ('S103', 'Networking');

INSERT INTO Students_Subjects (Rollno, Scode, marks_scored) VALUES (1, 'S101', 95), (1, 'S102', 85), (2, 'S101', 78), (2, 'S102', 88), (3, 'S101', 92), (3, 'S103', 65), (4, 'S101', 81), (4, 'S102', 38);

Q.1) Create a View:

CREATE VIEW Students_FYBCA AS SELECT sname FROM Students WHERE class = 'FYBCA';

SELECT * FROM Students_FYBCA;

CREATE VIEW Students_Scored_Above_90 AS SELECT s.sname, sub.subject_name, ss.marks_scored FROM Students s JOIN Students_Subjects ss ON s.Rollno = ss.Rollno JOIN Subjects sub ON ss.Scode = sub.Scode WHERE ss.marks_scored > 90;

SELECT * FROM Students_Scored_Above_90;

Q.2) Using above database solve following questions:

```
CREATE OR REPLACE FUNCTION check_rollno() RETURNS TRIGGER AS $$
BEGIN
  IF NEW.Rollno <= 0 THEN
    RAISE EXCEPTION 'Error: Roll number must be greater than zero';
  END IF;
  RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER check_rollno_before_insert
BEFORE INSERT ON Students
FOR EACH ROW
EXECUTE FUNCTION check_rollno();

INSERT INTO Students VALUES (101, 'Rahul', 'Pune', 'FYBCA');

INSERT INTO Students VALUES (0, 'Priya', 'Mumbai', 'SYBCA');
```

```
CREATE OR REPLACE FUNCTION calculate_total_marks() RETURNS VOID
AS $$
DECLARE
  student_rec RECORD;
  total INTEGER;
BEGIN
  FOR student_rec IN SELECT DISTINCT Rollno FROM Students_Subjects
  LOOP
    SELECT SUM(marks_scored) INTO total
    FROM Students_Subjects
    WHERE Rollno = student_rec.Rollno;

    RAISE NOTICE 'Rollno: %, Total Marks: %', student_rec.Rollno, total;
  END LOOP;
END;
$$ LANGUAGE plpgsql;

SELECT calculate_total_marks();
```

STUDENT – MARKS DATABASE

```
CREATE TABLE Students (Rollno INTEGER PRIMARY KEY, sname VARCHAR(30) NOT NULL, city VARCHAR(50), class VARCHAR(10));

CREATE TABLE Subjects (Scode VARCHAR(10) PRIMARY KEY, subject_name VARCHAR(20));

CREATE TABLE Students_Subjects (Rollno INTEGER, Scode VARCHAR(10), marks_scored INTEGER, PRIMARY KEY (Rollno, Scode), FOREIGN KEY (Rollno) REFERENCES Students(Rollno), FOREIGN KEY (Scode) REFERENCES Subjects(Scode));

INSERT INTO Students (Rollno, sname, city, class) VALUES (1, 'Amit', 'Mumbai', 'FYBCA'), (2, 'Anjali', 'Pune', 'SYBCA'), (3, 'Rahul', 'Nagpur', 'TYBCA'), (4, 'Arjun', 'Nashik', 'FYBCA');

INSERT INTO Subjects (Scode, subject_name) VALUES ('S101', 'DBMS'), ('S102', 'Math'), ('S103', 'Networking');

INSERT INTO Students_Subjects (Rollno, Scode, marks_scored) VALUES (1, 'S101', 95), (1, 'S102', 85), (2, 'S101', 78), (2, 'S102', 88), (3, 'S101', 92), (3, 'S103', 65), (4, 'S101', 81), (4, 'S102', 38);
```

Q.1) Create a View:

```
CREATE VIEW Students_DBMS_Above_80 AS SELECT s.sname FROM Students s JOIN Students_Subjects ss ON s.Rollno = ss.Rollno JOIN Subjects sub ON ss.Scode = sub.Scode WHERE sub.subject_name = 'DBMS' AND ss.marks_scored > 80;

SELECT * FROM Students_DBMS_Above_80;

CREATE VIEW Students_TYBCA_Details AS SELECT Rollno, sname, city, class FROM Students WHERE class = 'TYBCA';

SELECT * FROM Students_TYBCA_Details;
```

Q.2) Using above database solve following questions:

```
CREATE OR REPLACE FUNCTION notify_delete_student() RETURNS TRIGGER AS $$
BEGIN
   RAISE NOTICE 'Student record is being deleted';
   RETURN OLD;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER notify_student_delete
AFTER DELETE ON Students
FOR EACH ROW
EXECUTE FUNCTION notify_delete_student();

DELETE FROM Students WHERE Rollno = 101;
```

```
CREATE OR REPLACE FUNCTION get_subject_info(student_name VARCHAR) RETURNS VOID AS $$
DECLARE
   subject_rec RECORD;
BEGIN
   FOR subject_rec IN
      SELECT sub.subject_name, ss.marks_scored
      FROM Students s
      JOIN Students_Subjects ss ON s.Rollno = ss.Rollno
      JOIN Subjects sub ON ss.Scode = sub.Scode
      WHERE s.sname = student_name
   LOOP
      RAISE NOTICE 'Subject: %, Marks: %',
subject_rec.subject_name, subject_rec.marks_scored;
   END LOOP;
END;
$$ LANGUAGE plpgsql;

SELECT get_subject_info('Amit');
```

STUDENT – MARKS DATABASE

CREATE TABLE Students (Rollno INTEGER PRIMARY KEY, sname VARCHAR(30) NOT NULL, city VARCHAR(50), class VARCHAR(10));

CREATE TABLE Subjects (Scode VARCHAR(10) PRIMARY KEY, subject_name VARCHAR(20));

CREATE TABLE Students_Subjects (Rollno INTEGER, Scode VARCHAR(10), marks_scored INTEGER, PRIMARY KEY (Rollno, Scode), FOREIGN KEY (Rollno) REFERENCES Students(Rollno), FOREIGN KEY (Scode) REFERENCES Subjects(Scode));

INSERT INTO Students (Rollno, sname, city, class) VALUES (1, 'Amit', 'Mumbai', 'FYBCA'), (2, 'Anjali', 'Pune', 'SYBCA'), (3, 'Rahul', 'Nagpur', 'TYBCA'), (4, 'Arjun', 'Nashik', 'FYBCA');

INSERT INTO Subjects (Scode, subject_name) VALUES ('S101', 'DBMS'), ('S102', 'Math'), ('S103', 'Networking');

INSERT INTO Students_Subjects (Rollno, Scode, marks_scored) VALUES (1, 'S101', 95), (1, 'S102', 85), (2, 'S101', 78), (2, 'S102', 88), (3, 'S101', 92), (3, 'S103', 65), (4, 'S101', 81), (4, 'S102', 38);

Q.1) Create a View:

CREATE VIEW Students_Name_Starts_A AS SELECT Rollno, sname, city, class FROM Students WHERE sname LIKE 'A%';

SELECT * FROM Students_Name_Starts_A;

CREATE VIEW Students_Scored_Below_40 AS SELECT s.Rollno, s.sname, s.city, s.class, sub.subject_name, ss.marks_scored FROM Students s JOIN Students_Subjects ss ON s.Rollno = ss.Rollno JOIN Subjects sub ON ss.Scode = sub.Scode WHERE ss.marks_scored < 40;

SELECT * FROM Students_Scored_Below_40;

Q.2) Using above database solve following questions:

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

CREATE TRIGGER check_marks_before_insert
BEFORE INSERT OR UPDATE ON Students_Subjects
FOR EACH ROW
EXECUTE FUNCTION check_marks_range();

         INSERT INTO Students_Subjects (101, 'CS101', 85);

         INSERT INTO Students_Subjects VALUES (101, 'CS102', -5);

         INSERT INTO Students_Subjects VALUES (101, 'CS103', 105);
```

```
CREATE OR REPLACE FUNCTION
get_students_by_city(student_city VARCHAR) RETURNS VOID AS
$$
DECLARE
   student_rec RECORD;
BEGIN
   FOR student_rec IN
      SELECT Rollno, sname, city, class
      FROM Students
      WHERE city = student_city
   LOOP
      RAISE NOTICE 'Rollno: %, Name: %, City: %, Class: %',
student_rec.Rollno, student_rec.sname, student_rec.city,
student_rec.class;
   END LOOP;
END;
$$ LANGUAGE plpgsql;

SELECT get_students_by_city('Mumbai');
```

MOVIE_ACTOR_PRODUCER

CREATE TABLE Movie (m_name VARCHAR(25), release_year INTEGER NOT NULL, budget DECIMAL, PRIMARY KEY (m_name, release_year));

CREATE TABLE Actor (a_name CHAR(30), city VARCHAR(30), PRIMARY KEY (a_name));

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), release_year INTEGER, a_name CHAR(30), role VARCHAR(50), charges DECIMAL, PRIMARY KEY (m_name, release_year, a_name), FOREIGN KEY (m_name, release_year) REFERENCES Movie(m_name, release_year), FOREIGN KEY (a_name) REFERENCES Actor(a_name));

CREATE TABLE Movie_Producer (m_name VARCHAR(25), release_year INTEGER, producer_id INTEGER, PRIMARY KEY (m_name, release_year, producer_id), FOREIGN KEY (m_name, release_year) REFERENCES Movie(m_name, release_year), FOREIGN KEY (producer_id) REFERENCES Producer(producer_id));

INSERT INTO Movie VALUES ('Sholey', 1975, 5000000), ('Lagaan', 2001, 3000000), ('Taal', 1999, 2000000);

INSERT INTO Actor VALUES ('Amitabh Bachchan', 'Mumbai'), ('Aamir Khan', 'Mumbai'), ('Dharmendra', 'Pune'), ('Hema Malini', 'Delhi');

INSERT INTO Producer VALUES (1, 'Mr. Subhash Ghai', 'Mumbai'), (2, 'Yash Chopra', 'Pune');

INSERT INTO Movie_Actor VALUES ('Sholey', 1975, 'Amitabh Bachchan', 'Jai', 1000000), ('Sholey', 1975, 'Dharmendra', 'Veeru', 800000), ('Lagaan', 2001, 'Aamir Khan', 'Bhuvan', 1200000);

INSERT INTO Movie_Producer VALUES ('Sholey', 1975, 1), ('Lagaan', 2001, 2), ('Lagaan', 2001, 1) ('Taal', 1999, 1);

Q.1) Create a View:

CREATE VIEW Actors_In_Mumbai AS SELECT a_name FROM Actor WHERE city = 'Mumbai';

SELECT * FROM Actors_In_Mumbai;

CREATE VIEW Actors_In_Movies AS SELECT ma.m_name, ma.release_year, a.a_name, ma.role, ma.charges FROM Movie_Actor ma JOIN Actor a ON ma.a_name = a.a_name;

SELECT * FROM Actors_In_Movies;

Q.2) Using above database solve following questions:

```
CREATE OR REPLACE FUNCTION check_budget()
RETURNS TRIGGER AS $$
BEGIN
  IF NEW.budget < 6000000 THEN
    RAISE EXCEPTION 'Budget must be at least 60 lakhs.';
  END IF;
  RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER before_insert_budget
BEFORE INSERT ON Movie
FOR EACH ROW EXECUTE FUNCTION check_budget();


INSERT INTO Movie VALUES ('New Movie', 2023, 5000000);
```

```
    CREATE OR REPLACE FUNCTION
count_movies_by_producer(producer_name CHAR(30))
    RETURNS INTEGER AS $$
    DECLARE
      movie_count INTEGER;
    BEGIN
      SELECT COUNT(*) INTO movie_count
      FROM Movie_Producer mp
      JOIN Producer p ON mp.producer_id = p.producer_id
      WHERE p.pname = producer_name;

      RETURN movie_count;
    END;
    $$ LANGUAGE plpgsql;

    SELECT count_movies_by_producer('Mr. Subhash Ghai');
```

MOVIE_ACTOR_PRODUCER

```
CREATE TABLE Movie (m_name VARCHAR(25), release_year INTEGER NOT NULL, budget DECIMAL, PRIMARY KEY (m_name, release_year));

CREATE TABLE Actor (a_name CHAR(30), city VARCHAR(30), PRIMARY KEY (a_name));

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), release_year INTEGER, a_name CHAR(30), role VARCHAR(50), charges DECIMAL, PRIMARY KEY (m_name, release_year, a_name), FOREIGN KEY (m_name, release_year) REFERENCES Movie(m_name, release_year), FOREIGN KEY (a_name) REFERENCES Actor(a_name));

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

```
INSERT INTO Movie VALUES ('Sholey', 1975, 5000000), ('Lagaan', 2001, 3000000), ('Taal', 1999, 2000000);

INSERT INTO Actor VALUES ('Amitabh Bachchan', 'Mumbai'), ('Aamir Khan', 'Mumbai'), ('Dharmendra', 'Pune'), ('Hema Malini', 'Delhi');

INSERT INTO Producer VALUES (1, 'Mr. Subhash Ghai', 'Mumbai'), (2, 'Yash Chopra', 'Pune');

INSERT INTO Movie_Actor VALUES ('Sholey', 1975, 'Amitabh Bachchan', 'Jai', 1000000), ('Sholey', 1975, 'Dharmendra', 'Veeru', 800000), ('Lagaan', 2001, 'Aamir Khan', 'Bhuvan', 1200000);

INSERT INTO Movie_Producer VALUES ('Sholey', 1975, 1), ('Lagaan', 2001, 2), ('Lagaan', 2001, 1) ('Taal', 1999, 1);
```

Q.1) Create a View:

```
CREATE VIEW Actors_In_Sholey AS SELECT ma.a_name, ma.role, ma.charges FROM Movie_Actor ma WHERE ma.m_name = 'Sholey' AND ma.release_year = 1975;

SELECT * FROM Actors_In_Sholey;

CREATE VIEW Producers_More_Than_Two_Movies AS SELECT p.pname FROM Producer p JOIN Movie_Producer mp ON p.producer_id = mp.producer_id GROUP BY p.pname HAVING COUNT(mp.m_name) > 2;

SELECT * FROM Producers_More_Than_Two_Movies;
```

Q.2) Using above database solve following questions:

```
CREATE OR REPLACE FUNCTION check_charges()
RETURNS TRIGGER AS $$
BEGIN
  IF NEW.charges > 3000000 THEN
    RAISE EXCEPTION 'Charges cannot be more than 30 lakhs.';
  END IF;
  RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER before_insert_charges
BEFORE INSERT ON Movie_Actor
FOR EACH ROW EXECUTE FUNCTION check_charges();

INSERT INTO Movie_Actor VALUES ('Sholey', 1975, 'Amitabh Bachchan', 'Lead', 4000000);
```

```
CREATE OR REPLACE FUNCTION movies_by_actor(actor_name CHAR(30))
RETURNS VOID AS $$
DECLARE
  movie_name VARCHAR(25);
BEGIN
  FOR movie_name IN
    SELECT ma.m_name
    FROM Movie_Actor ma
    WHERE ma.a_name = actor_name
  LOOP
    RAISE NOTICE 'Movie: %', movie_name;
  END LOOP;

  IF NOT FOUND THEN
    RAISE EXCEPTION 'Invalid actor name: %', actor_name;
  END IF;
END;
$$ LANGUAGE plpgsql;

SELECT movies_by_actor('Amitabh Bachchan');
```

MOVIE_ACTOR_PRODUCER

```
CREATE TABLE Movie (m_name VARCHAR(25), release_year INTEGER NOT NULL, budget DECIMAL, PRIMARY KEY (m_name, release_year));

CREATE TABLE Actor (a_name CHAR(30), city VARCHAR(30), PRIMARY KEY (a_name));

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), release_year INTEGER, a_name CHAR(30), role VARCHAR(50), charges DECIMAL, PRIMARY KEY (m_name, release_year, a_name), FOREIGN KEY (m_name, release_year) REFERENCES Movie(m_name, release_year), FOREIGN KEY (a_name) REFERENCES Actor(a_name));

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

```
INSERT INTO Movie VALUES ('Sholey', 1975, 5000000), ('Lagaan', 2001, 3000000), ('Taal', 1999, 2000000);

INSERT INTO Actor VALUES ('Amitabh Bachchan', 'Mumbai'), ('Aamir Khan', 'Mumbai'), ('Dharmendra', 'Pune'), ('Hema Malini', 'Delhi');

INSERT INTO Producer VALUES (1, 'Mr. Subhash Ghai', 'Mumbai'), (2, 'Yash Chopra', 'Pune');

INSERT INTO Movie_Actor VALUES ('Sholey', 1975, 'Amitabh Bachchan', 'Jai', 1000000), ('Sholey', 1975, 'Dharmendra', 'Veeru', 800000), ('Lagaan', 2001, 'Aamir Khan', 'Bhuvan', 1200000);

INSERT INTO Movie_Producer VALUES ('Sholey', 1975, 1), ('Lagaan', 2001, 2), ('Lagaan', 2001, 1) ('Taal', 1999, 1);
```

Q.1) Create a View:

```
CREATE VIEW Movies_Produced_By_Subhash_Ghai AS SELECT mp.m_name, mp.release_year FROM Movie_Producer mp JOIN Producer p ON mp.producer_id = p.producer_id WHERE p.pname = 'Mr. Subhash Ghai';

SELECT * FROM Movies_Produced_By_Subhash_Ghai;

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

SELECT * FROM Actors_Not_In_Mumbai_Or_Pune;
```

Q.2) Using above database solve following questions:

```
CREATE OR REPLACE FUNCTION check_release_year()
RETURNS TRIGGER AS $$
BEGIN
  IF NEW.release_year > EXTRACT(YEAR FROM CURRENT_DATE) THEN
    RAISE EXCEPTION 'Release year cannot be greater than the current year.';
  END IF;
  RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER before_insert_release_year
BEFORE INSERT ON Movie
FOR EACH ROW EXECUTE FUNCTION check_release_year();

INSERT INTO Movie VALUES ('Tere Naam', 2025, 6000000);
```

```
CREATE OR REPLACE FUNCTION charges_of_amitabh_bachchan()
RETURNS VOID AS $$
DECLARE
  rec RECORD;
  cur CURSOR FOR
    SELECT ma.m_name, ma.release_year, ma.charges
    FROM Movie_Actor ma
    WHERE ma.a_name = 'Amitabh Bachchan';

BEGIN
  OPEN cur;
  LOOP
    FETCH cur INTO rec;
    EXIT WHEN NOT FOUND;
    RAISE NOTICE 'Movie: %, Year: %, Charges: %', rec.m_name, rec.release_year, rec.charges;
  END LOOP;
  CLOSE cur;
END;
$$ LANGUAGE plpgsql;

SELECT charges_of_amitabh_bachchan();
```