```
-- Script for Assignment 4

-- Creating database with full name

CREATE DATABASE subashreevs;

-- Connecting to database
\c subashreevs

-- Relation schemas and instances for assignment 2

CREATE TABLE student(sid integer,
                  pname text,
                  city text,
                  primary key (sid));

CREATE TABLE Company(cname text,
                   headquarter text,
                   primary key (cname));

CREATE TABLE Skill(skill text,
                 primary key (skill));


CREATE TABLE worksFor(sid integer,
                    cname text,
                    salary integer,
                    primary key (sid),
                    foreign key (sid) references student (sid),
                    foreign key (cname) references
Company(cname));


CREATE TABLE companyLocation(cname text,
                           city text,
                           primary key (cname, city),
                           foreign key (cname) references
Company (cname));


CREATE TABLE studentSkill(sid integer,
                        skill text,
                        primary key (sid, skill),
                        foreign key (sid) references student
(sid) on delete cascade,
```

```
                        foreign key (skill) references Skill
(skill) on delete cascade);


CREATE TABLE hasManager(eid integer,
                       mid integer,
                       primary key (eid, mid),
                       foreign key (eid) references student
(sid),
                       foreign key (mid) references student
(sid));

CREATE TABLE Knows(sid1 integer,
                  sid2 integer,
                  primary key(sid1, sid2),
                  foreign key (sid1) references student (sid),
                  foreign key (sid2) references student (sid));



INSERT INTO student VALUES
    (1001,'Jean','Cupertino'),
    (1002,'Vidya', 'Cupertino'),
    (1003,'Anna', 'Seattle'),
    (1004,'Qin', 'Seattle'),
    (1005,'Megan', 'MountainView'),
    (1006,'Ryan', 'Chicago'),
    (1007,'Danielle','LosGatos'),
    (1008,'Emma', 'Bloomington'),
    (1009,'Hasan', 'Bloomington'),
    (1010,'Linda', 'Chicago'),
    (1011,'Nick', 'MountainView'),
    (1012,'Eric', 'Cupertino'),
    (1013,'Lisa', 'Indianapolis'),
    (1014,'Deepa', 'Bloomington'),
    (1015,'Chris', 'Denver'),
    (1016,'YinYue', 'Chicago'),
    (1017,'Latha', 'LosGatos'),
    (1018,'Arif', 'Bloomington'),
    (1019,'John', 'NewYork');

INSERT INTO Company VALUES
    ('Apple', 'Cupertino'),
    ('Amazon', 'Seattle'),
    ('Google', 'MountainView'),
```

```
    ('Netflix', 'LosGatos'),
    ('Microsoft', 'Redmond'),
    ('IBM', 'NewYork'),
    ('ACM', 'NewYork'),
    ('Yahoo', 'Sunnyvale');


INSERT INTO worksFor VALUES
    (1001,'Apple', 65000),
    (1002,'Apple', 45000),
    (1003,'Amazon', 55000),
    (1004,'Amazon', 55000),
    (1005,'Google', 60000),
    (1006,'Amazon', 55000),
    (1007,'Netflix', 50000),
    (1008,'Amazon', 50000),
    (1009,'Apple',60000),
    (1010,'Amazon', 55000),
    (1011,'Google', 70000),
    (1012,'Apple', 50000),
    (1013,'Yahoo', 55000),
    (1014,'Apple', 50000),
    (1015,'Amazon', 60000),
    (1016,'Amazon', 55000),
    (1017,'Netflix', 60000),
    (1018,'Apple', 50000),
    (1019,'Microsoft', 50000);

INSERT INTO companyLocation VALUES
  ('Apple', 'Bloomington'),
  ('Amazon', 'Chicago'),
  ('Amazon', 'Denver'),
  ('Amazon', 'Columbus'),
  ('Google', 'NewYork'),
  ('Netflix', 'Indianapolis'),
  ('Netflix', 'Chicago'),
  ('Microsoft', 'Bloomington'),
  ('Apple', 'Cupertino'),
  ('Amazon', 'Seattle'),
  ('Google', 'MountainView'),
  ('Netflix', 'LosGatos'),
  ('Microsoft', 'Redmond'),
  ('IBM', 'NewYork'),
  ('Yahoo', 'Sunnyvale');
```

```
INSERT INTO Skill VALUES
  ('Programming'),
  ('AI'),
  ('Networks'),
  ('OperatingSystems'),
  ('Databases');

INSERT INTO studentSkill VALUES
(1001,'Programming'),
(1001,'AI'),
(1002,'Programming'),
(1002,'AI'),
(1004,'AI'),
(1004,'Programming'),
(1005,'AI'),
(1005,'Programming'),
(1005,'Networks'),
(1006,'Programming'),
(1006,'OperatingSystems'),
(1007,'OperatingSystems'),
(1007,'Programming'),
(1009,'OperatingSystems'),
(1009,'Networks'),
(1010,'Networks'),
(1011,'Networks'),
(1011,'OperatingSystems'),
(1011,'AI'),
(1011,'Programming'),
(1012,'AI'),
(1012,'OperatingSystems'),
(1012,'Programming'),
(1013,'Programming'),
(1013,'OperatingSystems'),
(1013,'Networks'),
(1014,'OperatingSystems'),
(1014,'AI'),
(1014,'Networks'),
(1015,'Programming'),
(1015,'AI'),
(1016,'OperatingSystems'),
(1016,'AI'),
(1017,'Networks'),
(1017,'Programming'),
(1018,'AI'),
(1019,'Networks'),
```

```
(1010,'Databases'),
(1011,'Databases'),
(1013,'Databases'),
(1014,'Databases'),
(1017,'Databases'),
(1019,'Databases'),
(1005,'Databases'),
(1006,'AI'),
(1009,'Databases');


INSERT INTO hasManager VALUES
(1004, 1003),
(1006, 1003),
(1015, 1003),
(1016, 1004),
(1016, 1006),
(1008, 1015),
(1010, 1008),
(1013, 1007),
(1017, 1013),
(1002, 1001),
(1009, 1001),
(1014, 1012),
(1011, 1005);


INSERT INTO Knows VALUES
(1011,1009),
(1007,1016),
(1011,1010),
(1003,1004),
(1006,1004),
(1002,1014),
(1009,1005),
(1018,1009),
(1007,1017),
(1017,1019),
(1019,1013),
(1016,1015),
(1001,1012),
(1015,1011),
(1019,1006),
(1013,1002),
(1018,1004),
```

(1013,1007),
(1014,1006),
(1004,1014),
(1001,1014),
(1010,1013),
(1010,1014),
(1004,1019),
(1018,1007),
(1014,1005),
(1015,1018),
(1014,1017),
(1013,1018),
(1007,1008),
(1005,1015),
(1017,1014),
(1015,1002),
(1018,1013),
(1018,1010),
(1001,1008),
(1012,1011),
(1002,1015),
(1007,1013),
(1008,1007),
(1004,1002),
(1015,1005),
(1009,1013),
(1004,1012),
(1002,1011),
(1004,1013),
(1008,1001),
(1008,1019),
(1019,1008),
(1001,1019),
(1019,1001),
(1004,1003),
(1006,1003),
(1015,1003),
(1016,1004),
(1016,1006),
(1008,1015),
(1010,1008),
(1017,1013),
(1002,1001),
(1009,1001),
(1011,1005),

```
(1014,1012),
(1010,1002),
(1010,1012),
(1010,1018);



\qecho 'Problem 1'
/*Find each pair (c, p) where c is the city and p is the sid of
the student
that lives in c, and earns the lowest salary among all students
living in c. You must not use set
predicates in this query*/

-- Step 1: Find the minimum salary for each city
CREATE VIEW CityMinSalary AS
SELECT s.city, MIN(w.salary) AS min_salary
FROM student s
JOIN worksFor w ON s.sid = w.sid
GROUP BY s.city;

-- Step 2: Join the minimum salary view with the worksFor and
student tables to get students with the lowest salary in each
city
SELECT s.city, w.sid
FROM student s
JOIN worksFor w ON s.sid = w.sid
JOIN CityMinSalary cms ON s.city = cms.city AND w.salary =
cms.min_salary
ORDER BY s.city, w.sid;



\qecho 'Problem 2'
/*Find the sid and name of each student who has fewer than 2 of
the com-
bined set of job skills of students who work for Netflix. By
combined set
of jobskills we mean the set
{s | s is a jobskill of an employee of Netflix }*/

-- Step 1: Create a view for the combined set of job skills of
students working at Netflix
```

```
CREATE VIEW NetflixSkills AS
SELECT DISTINCT ss.skill
FROM studentSkill ss
JOIN worksFor w ON ss.sid = w.sid
WHERE w.cname = 'Netflix';

-- Step 2: Count the number of Netflix skills each student has
CREATE VIEW StudentNetflixSkillCount AS
SELECT s.sid, s.pname, COUNT(DISTINCT n.skill) AS skill_count
FROM student s
LEFT JOIN studentSkill ss ON s.sid = ss.sid
LEFT JOIN NetflixSkills n ON ss.skill = n.skill
GROUP BY s.sid, s.pname;

-- Step 3: Final query to find students with fewer than 2 Netflix
skills
SELECT sid, pname
FROM StudentNetflixSkillCount
WHERE skill_count < 2;



\qecho 'Problem 3'
/*Find each pairs (s1; s2) of skills such that the set of
students with skill s1
is the same as the set of students with skill s2.You must not use
set predicates in this query.*/

-- Step 1: Create a view that represents the unique student set
associated with each skill
CREATE VIEW SkillStudentList AS
SELECT skill, STRING_AGG(CAST(sid AS TEXT), ',' ORDER BY sid) AS
student_set
FROM studentSkill
GROUP BY skill;

-- Step 2: Self-join to find pairs of skills with identical
student sets
SELECT s1.skill AS s1, s2.skill AS s2
FROM SkillStudentList s1
JOIN SkillStudentList s2 ON s1.student_set = s2.student_set
ORDER BY s1.skill, s2.skill;
```

```
\qecho 'Problem 4'
SELECT k.sid1 AS sid
FROM Knows k
JOIN worksFor w ON k.sid2 = w.sid
WHERE w.cname = 'Apple' AND w.salary < 55000
GROUP BY k.sid1
HAVING COUNT(DISTINCT k.sid2) >= 2;



\qecho 'Problem 5'
/*Find the cname of each company, such that some student that
works there
knows at-least quarter of the people that work at Amazon.*/
-- Step 1: Count the total number of Amazon employees
WITH AmazonCount AS (
   SELECT COUNT(*) AS total_amazon_employees
   FROM worksFor
   WHERE cname = 'Amazon'
),

-- Step 2: Create a CTE that calculates how many Amazon employees
each student knows, grouped by company
CompanyKnows AS (
   SELECT w.cname, k.sid1 AS student_id, COUNT(DISTINCT k.sid2)
AS known_amazon_employees
   FROM worksFor w
   JOIN Knows k ON w.sid = k.sid1
   JOIN worksFor amazon_emp ON k.sid2 = amazon_emp.sid AND
amazon_emp.cname = 'Amazon'
   GROUP BY w.cname, k.sid1
)

-- Step 3: Filter companies with students who know at least a
quarter of Amazon employees
SELECT DISTINCT cname
FROM CompanyKnows, AmazonCount
WHERE known_amazon_employees > total_amazon_employees / 4;
```

```
\qecho 'Problem 6'
/*
Find each pair (c, a) where c is the cname of each company that
has at
least one manager, and a is the average salary of all employees
working at
the company who are not managers
*/
WITH ManagerCompanies AS (
 SELECT DISTINCT w.cname
 FROM worksFor w
 JOIN hasManager hm ON w.sid = hm.mid
),
NonManagerSalaries AS (
 SELECT w.cname, w.salary
 FROM worksFor w
 LEFT JOIN hasManager hm ON w.sid = hm.mid
 WHERE hm.mid IS NULL
)
SELECT mc.cname AS c, round(AVG(nms.salary)) AS a
FROM ManagerCompanies mc
LEFT JOIN NonManagerSalaries nms ON mc.cname = nms.cname
GROUP BY mc.cname
ORDER BY mc.cname;



\qecho 'Problem 7'
\qecho '(a)'
/*Using the GROUP BY count method, define a function
create or replace function numberOfSkills(c text)
returns table (sid integer, salary int, numberOfSkills bigint) as
$$
...
$$ language sql;
that returns for a company identified by its cname, each triple
(p,
s, n) where (1) p is the sid of a student who is employed by that
company, (2) s is the salary of p, and (3) n is the number of job
skills
of p. (Note that a student may not have any job skills.)*/
CREATE OR REPLACE FUNCTION numberOfSkills(c TEXT)
RETURNS TABLE (sid INTEGER, salary INT, numberOfSkills BIGINT) AS
$$
```

```
BEGIN
   RETURN QUERY
   SELECT w.sid, w.salary, COUNT(ss.skill) AS numberOfSkills
   FROM worksFor w
   LEFT JOIN studentSkill ss ON w.sid = ss.sid
   WHERE w.cname = c
   GROUP BY w.sid, w.salary;
END;
$$ LANGUAGE plpgsql;


\qecho '(b)'
/*Test your function for Problem 7a for the companies Apple,
Amazon,
and ACM.*/
SELECT * FROM numberOfSkills('Apple');
SELECT * FROM numberOfSkills('Amazon');
SELECT * FROM numberOfSkills('ACM');


\qecho '(c)'
/*Write the same function numberOfSkills as in Problem 7a but
this
time without using the GROUP BY clause.*/
CREATE OR REPLACE FUNCTION numberOfSkills_noGroupBy(c TEXT)
RETURNS TABLE (student_sid INTEGER, student_salary INT,
numberOfSkills BIGINT) AS $$
DECLARE
   student RECORD;
BEGIN
   FOR student IN
       SELECT w.sid, w.salary
       FROM worksFor w
       WHERE w.cname = c
   LOOP
       -- Count the number of skills for each student
       SELECT COUNT(ss.skill) INTO numberOfSkills
       FROM studentSkill ss
       WHERE ss.sid = student.sid;

       -- Assign values to the output columns
       student_sid := student.sid;
       student_salary := student.salary;

       -- Return the row
```

```
        RETURN NEXT;
    END LOOP;
END;
$$ LANGUAGE plpgsql;


\qecho '(d)'
/*Test your function for Problem 7c for the companies Apple,
Amazon,
and ACM.*/
SELECT * FROM numberOfSkills_noGroupBy('Apple');
SELECT * FROM numberOfSkills_noGroupBy('Amazon');
SELECT * FROM numberOfSkills_noGroupBy('ACM');


\qecho '(e)'
/*Using the function numberOfSkills but without using set
predicates,
write the following query: "Find each pair (c; p) where c is the
name
of a company and where p is the sid of a student who (1) works
for
company c, (2) makes more than 50000 and (3) has the most job
skills among all the employees who work for company c."*/

-- Function to find the student with the most skills for a given
company
CREATE OR REPLACE FUNCTION max_skills_student(company_name TEXT)
RETURNS TABLE (max_sid INTEGER, max_skills BIGINT) AS $$
DECLARE
    max_skill_count BIGINT := 0;
    current_sid INTEGER;
    current_skills BIGINT;
BEGIN
    -- Loop through all students working for the company and count
their skills
    FOR current_sid, current_skills IN
        SELECT w.sid, COUNT(ss.skill) AS numberOfSkills
        FROM worksFor w
        LEFT JOIN studentSkill ss ON w.sid = ss.sid
        WHERE w.cname = company_name
        GROUP BY w.sid
    LOOP
        -- If current student has more skills than previous
maximum, update max values
```

```
        IF current_skills > max_skill_count THEN
            max_skill_count := current_skills;
            max_sid := current_sid;
            max_skills := current_skills;
        END IF;
    END LOOP;

    -- Return the student with the maximum skills count
    RETURN QUERY SELECT max_sid, max_skill_count;
END;
$$
LANGUAGE plpgsql;

-- Main query to find pairs (c, p) where c is the company and p is
the student ID
-- who has the most skills in that company and makes more than
50000.
SELECT
    c.cname AS c,
    COALESCE(ns.sid, max_skills.max_sid) AS p
FROM
    Company c
LEFT JOIN LATERAL (
    -- Subquery to get students who work for the company and make
more than 50000
    SELECT w.sid, w.salary, COUNT(ss.skill) AS numberOfSkills
    FROM worksFor w
    LEFT JOIN studentSkill ss ON w.sid = ss.sid
    WHERE w.cname = c.cname AND w.salary > 50000
    GROUP BY w.sid, w.salary
) ns ON true
LEFT JOIN LATERAL (
    -- Subquery to get the student with the most skills for each
company using our function
    SELECT * FROM max_skills_student(c.cname)
) max_skills ON true
WHERE
    ns.sid IS NOT NULL
    AND ns.numberOfSkills = max_skills.max_skills;




\qecho 'Problem 8'
/*Find the sid and name of each student who knows all the
```

```
students who (a) live in Bloomington, (b) make at least
55000, and (c) have at least one skill.*/
-- Define view A: Students in Bloomington with specific criteria

-- Step 1: Create a view for students who live in Bloomington,
make at least 55000, and have at least one skill (Set B)

CREATE OR REPLACE VIEW Students_Bloomington AS
SELECT s.sid
FROM student s
JOIN worksFor w ON s.sid = w.sid
JOIN studentSkill ss ON s.sid = ss.sid
WHERE s.city = 'Bloomington'
 AND w.salary >= 55000;

-- Step 2: Create a view for students who know all students in
Set B (Knows_All_Bloomington)
CREATE OR REPLACE VIEW Knows_All_Bloomington AS
SELECT s1.sid
FROM student s1
WHERE NOT EXISTS (
   -- Find any student in Set B that is not known by s1
   SELECT sb.sid
   FROM Students_Bloomington sb
   WHERE NOT EXISTS (
       -- Check if s1 knows sb
       SELECT 1
       FROM Knows k
       WHERE k.sid1 = s1.sid AND k.sid2 = sb.sid
   )
);

-- Step 3: Final query to get the sid and name of each student in
Knows_All_Bloomington
SELECT s.sid, s.pname
FROM student s
JOIN Knows_All_Bloomington kab ON s.sid = kab.sid;




\qecho 'Problem 9'
/*Find the cname of each company who only employs man-
agers who make more than 50000.*/
```

```
-- View A: All companies
CREATE OR REPLACE VIEW AllCompanies AS
SELECT DISTINCT cname
FROM Company;

-- View B: Companies that employ managers making 50000 or less
CREATE OR REPLACE VIEW CompaniesWithLowPaidManagers AS
SELECT DISTINCT w.cname
FROM worksFor w
JOIN hasManager hm ON w.sid = hm.mid
WHERE w.salary <= 50000;

-- Main query to find companies that only employ managers who make
more than 50000
SELECT ac.cname
FROM AllCompanies ac
WHERE ac.cname NOT IN (
   SELECT cname
   FROM CompaniesWithLowPaidManagers
);

\qecho 'THIS WILL ALSO CONTAIN THE COMPANIES WHICH DO NOT HAVE
ANY MANAGERS EMPLOYED.'




\qecho 'Problem 10'
/*Find the sid and name of each student who knows at least
3 people who each have at most 2 managers.*/
-- View for students with 2 or fewer managers
-- View A: Students who have at most 2 managers
CREATE VIEW Students_with_2_or_fewer_managers AS
SELECT hm.eid
FROM hasManager hm
GROUP BY hm.eid
HAVING COUNT(hm.mid) <= 2;

-- View B: Students who know at least 3 people from view A
CREATE VIEW Students_knowing_3_with_2_or_fewer_managers AS
SELECT s.sid, s.pname
FROM student s
JOIN Knows k ON s.sid = k.sid1
WHERE k.sid2 IN (SELECT eid FROM
Students_with_2_or_fewer_managers)
```

```
GROUP BY s.sid, s.pname
HAVING COUNT(DISTINCT k.sid2) >= 3;

-- Final query to display the results
SELECT * FROM Students_knowing_3_with_2_or_fewer_managers;




\qecho 'Problem 11'
/*Find the cname of each company that employs an even
number of students who have at least 2 skills*/
-- Create a view for students with at least 2 skills
CREATE OR REPLACE VIEW StudentsWithAtLeastTwoSkills AS
SELECT sid
FROM studentSkill
GROUP BY sid
HAVING COUNT(DISTINCT skill) >= 2;

-- Create a function to get companies with an even number of
qualified employees
CREATE OR REPLACE FUNCTION CompaniesWithEvenEmployees()
RETURNS TABLE (cname TEXT) AS $$
BEGIN
   RETURN QUERY
   SELECT w.cname
   FROM worksFor w
   JOIN StudentsWithAtLeastTwoSkills s ON w.sid = s.sid
   GROUP BY w.cname
   HAVING COUNT(w.sid) % 2 = 0
   UNION
   SELECT c.cname
   FROM Company c
   WHERE NOT EXISTS (
       SELECT 1
       FROM worksFor w
       JOIN StudentsWithAtLeastTwoSkills s ON w.sid = s.sid
       WHERE w.cname = c.cname
   )
   ORDER BY cname;
END;
$$
LANGUAGE plpgsql;

-- Call the function to get the results
```

```
SELECT * FROM CompaniesWithEvenEmployees();

\qecho 'THIS WILL ALSO CONTAIN THE COMPANIES WHICH DO NOT HAVE
ANY EMPLOYEEs.'



\qecho 'Problem 12'
/*Find the pairs (p1, p2) of different student sids such that
the student with sid p1 and the student with sid p2 have the
same number of skills.*/
-- Step 1: Create a view to calculate the number of skills each
student has
CREATE OR REPLACE VIEW StudentSkillCount AS
SELECT sid, COUNT(skill) AS skill_count
FROM studentSkill
GROUP BY sid;

-- Step 2: Generate pairs of students with the same number of
skills
SELECT s1.sid AS p1, s2.sid AS p2
FROM StudentSkillCount s1
JOIN StudentSkillCount s2 ON s1.skill_count = s2.skill_count AND
s1.sid <> s2.sid;



\qecho 'Problem 13'
/*Explain how triggers can be used to implement the
Primary key Constraint, with an example.*/
-- Trigger function to enforce the primary key constraint on
'cname' in the existing Company table
CREATE OR REPLACE FUNCTION check_company_key_constraint()
RETURNS TRIGGER AS $$
BEGIN
   -- Check if cname already exists in the Company table
   IF EXISTS (SELECT 1 FROM Company WHERE cname = NEW.cname) THEN
       RAISE EXCEPTION 'Primary key constraint violated: cname %
already exists', NEW.cname;
   END IF;
   RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

```
-- Create a trigger to enforce the primary key constraint before
insertion or update
CREATE TRIGGER enforce_company_primary_key
BEFORE INSERT OR UPDATE ON Company
FOR EACH ROW
EXECUTE FUNCTION check_company_key_constraint();


-- Insert a new company with a unique cname
INSERT INTO Company (cname, headquarter) VALUES ('Amazon', 'San
Francisco');




\qecho 'Problem 14'
/*Explain how triggers can be used to implement the Referential
Integrity
Constraint, with an example.(You are not allowed to use postgres
cascade).*/
-- Trigger function to check referential integrity on INSERT and
UPDATE
CREATE OR REPLACE FUNCTION check_company_exists()
RETURNS TRIGGER AS $$
BEGIN
   -- Check if cname exists in the Company table
   IF NOT EXISTS (SELECT 1 FROM Company WHERE cname = NEW.cname)
THEN
       RAISE EXCEPTION 'Referential integrity constraint
violated: cname % does not exist in Company', NEW.cname;
   END IF;
   RETURN NEW;
END;
$$ LANGUAGE plpgsql;

-- Create a trigger for INSERT and UPDATE on companyLocation
CREATE TRIGGER enforce_company_foreign_key
BEFORE INSERT OR UPDATE ON companyLocation
FOR EACH ROW
EXECUTE FUNCTION check_company_exists();


-- Trigger function to handle deletion of related rows in
companyLocation
```

```
CREATE OR REPLACE FUNCTION delete_referencing_locations()
RETURNS TRIGGER AS $$
BEGIN
   -- Delete rows from companyLocation where cname matches the
deleted cname in Company
   DELETE FROM companyLocation WHERE cname = OLD.cname;
   RETURN OLD;
END;
$$ LANGUAGE plpgsql;

-- Create a trigger for DELETE on Company
CREATE TRIGGER cascade_delete_company_location
AFTER DELETE ON Company
FOR EACH ROW
EXECUTE FUNCTION delete_referencing_locations();

INSERT INTO companyLocation (cname, city) VALUES ('Amadeus',
'Chicago');




\qecho 'Problem 15'
/*Consider two relations R(A:integer,B:integer) and S(B:integer)
and a view with the following definition:*/

/*select distinct r.A
from R r, S s
where r.A > 10 and r.B = s.B;*/

/*Suppose we want to maintain this view as a materialized
view called V(A:integer) upon the insertion of tuples in R
and in S. (You do not have to consider deletions in this
question.)

Define SQL insert triggers and their associated trigger functions
on the relations R and S that implement this materialized view.
Write your trigger functions in the language

Make sure that your trigger functions act in an incremental way
and that no duplicates appear in the materialized view.*/


CREATE TABLE IF NOT EXISTS R(A INT, B INT);
CREATE TABLE IF NOT EXISTS S(B INT);
```

```
CREATE TABLE IF NOT EXISTS V(A INT);


/* -----------*/
--Create TRIGGERS Insert_R and Insert_S
/* -----------*/

-- Trigger function for insertions into R
CREATE OR REPLACE FUNCTION update_view_on_insert_R()
RETURNS TRIGGER AS $$
BEGIN
   -- Insert into V if NEW.A > 10 and there exists a matching B
in S
   IF NEW.A > 10 THEN
       INSERT INTO V (A)
       SELECT NEW.A
       FROM S
       WHERE S.B = NEW.B
       ON CONFLICT DO NOTHING;  -- Prevent duplicates
   END IF;
   RETURN NEW;
END;
$$ LANGUAGE plpgsql;

-- Create the trigger for insertion on R
CREATE TRIGGER insert_trigger_on_R
AFTER INSERT ON R
FOR EACH ROW
EXECUTE FUNCTION update_view_on_insert_R();


-- Trigger function for insertions into S
CREATE OR REPLACE FUNCTION update_view_on_insert_S()
RETURNS TRIGGER AS $$
BEGIN
   -- Insert into V if there exists an A > 10 in R with a
matching B
   INSERT INTO V (A)
   SELECT DISTINCT R.A
   FROM R
   WHERE R.B = NEW.B AND R.A > 10
   ON CONFLICT DO NOTHING;  -- Prevent duplicates
   RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

```
-- Create the trigger for insertion on S
CREATE TRIGGER insert_trigger_on_S
AFTER INSERT ON S
FOR EACH ROW
EXECUTE FUNCTION update_view_on_insert_S();



--TEST YOUR TRIGGERS ACROSS THE BELOW RECORDS.

/* -----------*/

INSERT INTO R VALUES(15,35);
INSERT INTO S VALUES(35);
SELECT * FROM V;


INSERT INTO R VALUES(4,12);
INSERT INTO S VALUES(12);
SELECT * FROM V;

INSERT INTO R VALUES(26,13);
INSERT INTO S VALUES(11);
SELECT * FROM V;


INSERT INTO R VALUES(101,106);
INSERT INTO S VALUES(106);
SELECT * FROM V;


DROP TABLE IF EXISTS R;
DROP TABLE IF EXISTS S;
DROP TABLE IF EXISTS V;

/* -----------*/


-- Connect to default database
\c postgres

-- Drop database created for this assignment
DROP DATABASE subashreevs    ;
```