-- 1. For each department, list its name along with the highest salary made by students who work for it.
select w.deptName, w.salary from employedBy w where w.salary >= all (select w1.salary from employedBy w1 where w1.deptName = w.deptName) order by 1;

-- 2. Find the sid and sname of students who earn more than each of their friends in the same department.
select s.sid, s.sname from Student s where exists (select 1 from employedBy w where w.sid = s.sid and exists (select 1 from employedBy w1 where w1.deptName = w.deptName and exists (select 1 from hasFriend f where f.sid1 = w.sid and f.sid2 = w1.sid) and w.salary > all (select w1.salary from employedBy w1 where (w.sid, w1.sid) in (select f.sid1, f.sid2 from hasFriend f) and w1.deptName = w.deptName))) order by 1;

-- 3. Find the deptName of each department that employs at least one student whose home city is not Indianapolis.
select d.deptName from Department d where exists (select w.sid from employedBy w where w.deptName = d.deptName and w.sid not in (select s.sid from student s where s.homeCity = 'Indianapolis'));

-- 4. Find the sid and sname of students from Bloomington who earn more than 20000 and have at least one friend.
select s.sid, s.sname from Student s where s.homeCity = 'Bloomington' and s.sid in (select w.sid from employedBy w where w.salary > 20000) and exists (select 1 from hasFriend f where f.sid1 = s.sid) order by 1,2;

-- 5. Find pairs of department names whose main offices are in the same building.
select d1.deptName, d2.deptName from Department d1, Department d2 where d1.deptName <> d2.deptName and d1.mainOffice = d2.mainOffice order by 1,2;
-- 6. Find the sid and sname of students whose home city is different from their friends' home city.
select s.sid, s.sname from Student s where exists (select 1 from hasFriend f where f.sid1 = s.sid) and s.homeCity not in (select f.homeCity from Student f where (s.sid, f.sid) in (select f.sid1, f.sid2 from hasFriend f)) order by 1;

-- 7. Find each major that has at most two students.
select m.major from Major m except select m.major from Major m where exists (select 1 from studentMajor sm1, studentMajor sm2, studentMajor sm3 where sm1.major = m.major and sm2.major = m.major and sm3.major = m.major and sm1.sid <> sm2.sid and sm2.sid <> sm3.sid and sm1.sid <> sm3.sid) order by 1;

-- 8. Find the sid, sname, and salary of students with at least two friends who share a common major (not Mathematics).
select s.sid, s.sname, w.salary from Student s, employedBy w where s.sid = w.sid and exists (select 1 from hasFriend hf1, hasFriend hf2 where hf1.sid1 = s.sid and hf2.sid1 = s.sid and hf1.sid2 <> hf2.sid2 and (hf1.sid2, hf2.sid2) in (select sm1.sid, sm2.sid from studentMajor sm1, studentMajor sm2 where sm1.major = sm2.major and sm1.major <> 'Mathematics')) order by 1;

TRC TO SQL
-- 10. {(s1.sid, s1.sname) | Student(s1) ∧ ∃d ∈ Department, w ∈ employedBy (d.deptName = w.deptName ∧ s1.sid = w.sid ∧ d.mainOffice = LuddyHall ∧ ∃s2 ∈ Student (hasFriend(s1.sid, s2.sid) ∧ s2.homeCity ≠ Bloomington))}.
select s1.sid, s1.sname from Student s1 where exists (select 1 from Department d, employedBy w where d.deptName = w.deptName and s1.sid = w.sid and d.mainOffice = 'LuddyHall' and exists (select 1 from Student s2 where (s1.sid, s2.sid) in (select * from hasFriend) and s2.homeCity <> 'Bloomington'));

-- 11. {s1.sid | Student(s1) ∧ ∀s2 ∈ Student(hasFriend(s1.sid, s2.sid) → ∃sm1 ∈ studentMajor, sm2 ∈ studentMajor (sm1.sid = s1.sid ∧ sm2.sid = s2.sid ∧ sm1.major = sm2.major ∧ sm1.sid ≠ sm2.sid))}.
select s1.sid from Student s1 where true = all (select true = some (select sm1.sid = s1.sid and sm2.sid = s2.sid and sm1.major = sm2.major and sm1.sid <> sm2.sid from studentMajor sm1, studentMajor sm2) from Student s2 where (s1.sid, s2.sid) in (select * from hasFriend));

-- 12. {(s1.sid, s2.sid) | Student(s1) ∧ Student(s2) ∧ s1.sid ≤ s2.sid ∧ ∀f1 ∈ hasFriend (s1.sid = f1.sid1 → ∃f2 ∈ hasFriend (f2.sid1 = s2.sid ∧ f1.sid2 = f2.sid2))}.
select s1.sid as "sid1", s2.sid as "sid2" from Student s1, Student s2 where s1.sid <> s2.sid and true = all (select true = some (select f2.sid1 = s2.sid and f1.sid2 = f2.sid2 from hasFriend f2) from hasFriend f1 where s1.sid = f1.sid1);

-- 13. {m.major | Major(m) ∧ ¬(∃s ∈ Student ∃sm ∈ studentMajor(s.sid = sm.sid ∧ sm.major = m.major ∧ s.homeCity = 'Bloomington'))}.
select m.major from Major m where not exists (select 1 from Student s, studentMajor sm where s.sid = sm.sid and sm.major = m.major and s.homeCity = 'Bloomington') order by 1;

TRC
-- 14 a) Find each pair (d, m) where d is the name of a department and m is a major of a student who is employed by that department and who earns a salary of at least 20000.
{(d.deptName, m.major) | Department(d) ∧ Major(m) ∧ ∃s ∈ Student (studentMajor(s.sid,m.major) ∧ ∃w ∈ employedBy (w.deptName = d.deptName ∧ w.sid = s.sid ∧ w.salary ≥ 20000))}.

-- 14 b) Find each pair (s1, s2) of sids of different students who have the same (set of) friends who work for the CS department.
{(s1.sid, s2.sid) | Student(s1) ∧ Student(s2) ∧ s1.sid ≠ s2.sid ∧ ∀w ∈ employedBy (w.deptName = CS → ((hasFriend(s1.sid, w.sid) → hasFriend(s2.sid, w.sid)) ∧ (hasFriend(s2.sid, w.sid) → hasFriend(s1.sid, w.sid))))}

-- 15 a) Find each major for which there exists a student with that major and who does not only have friends who also have that major.
{m.major | Major(m) ∧ ∃s ∈ Student(studentMajor(s.sid, m.major) ∧ ¬(∀s1 ∈ Student(hasFriend(s.sid, s1.sid) → studentMajor(s1, m.major))))}.

-- 15 b) Find the sid and sname of each student whose home city is different than those of his or her friends.
{(s.sid, s.sname, s.city) | Student(s) ∧ ∃f(hasFriend(f) ∧ f.sid1 = s.sid) ∧ ¬∃s1(Student(s1) ∧ s.city = s1.city ∧ hasFriend(s.sid, s1.sid))}

-- 16 a) Find the sid, sname, and salary of each student who has at least two friends such that these friends have a common major but provided that it is not the 'Maths' major
{s.sid| Student(s) ∧ ∃f1 ∃f2(hasFriend(f1) ∧ hasFriend(f2) ∧ f1.sid1 = s.sid ∧ f2.sid1 = s.sid ∧ f1.sid2 ≠ f2.sid2 ∧ ∃sm1 ∃sm2(studentMajor(sm1) ∧ studentMajor(sm2) ∧ f1.mid=sm1.sid ∧ f2.mid=sm2.sid ∧ sm1.major = sm2.major ∧ sm1.major ≠ 'Mathematics'))}.

-- 16 b) For each department, list its name along with the highest salary made by students who are employed by it.
{(d.deptName, w.salary) | Department(d) ∧ employedBy(w) ∧ w.deptName = d.deptName ∧ ¬∃w1(employedBy(w1) ∧ w1.deptName = d.deptName ∧ w1.salary > w.salary)}.

-- 17 Find the sid, sname of each student who has home city Bloomington, works for a dept where he or she earns a salary that is higher than 20k, & has at least one friend.
{s.sid, s.sname | Student(s) ∧ s.city = 'Bloomington' ∧ ∃w(employedBy(w) ∧ w.sid = s.sid ∧ w.salary > 20000) ∧ ∃f(hasFriend(f) ∧ f.sid1 = s.sid)}.

TRC and BOOLEAN SQL
-- 18 Some major has fewer than 2 students with that major
a) select true = some (select true = all (select s1.sid = s2.sid
from Student s1, Student s2 where (s1.sid, m.major) in (select *
from studentMajor) and (s2.sid, m.major) in (select * from studentMajor)) from Major m);
b) ∃ m (Major(m) ∧ ¬ ∃s1 ∈ Student ∃s2 ∈ Student (s1.sid ≠s2.sid ∧ studentMajor(s1.sid, m.major) ∧ studentMajor(s2.sid, m.major)

-- 19 Each student is employed by a department and has at least two majors
a) select not exists (select 1 from Student s where not(exists (select 1
from employedBy w where w.sid = s.sid) and not exists (select 1
from studentMajor sm1, studentMajor sm2 where sm1.sid = s.sid and sm2.sid = s.sid and
sm1.major <> sm2.major)));
b) ∀Student(s) →(∃w(employedBy(w) ∧ w.sid=s.sid) ∧ ∃sm1sm2(studentMajor(sm1) ∧ studentMajor(sm2) ∧ sm1.sid = s.sid ∧ sm2.sid = s.sid ∧ sm1.major ≠ sm2.major))

-- 20 Each student and his or her friends work for the same department
a) select not exists (select 1 from hasFriend f, employedBy wf1, employedBy wf2
where f.sid1 = wf1.sid and f.sid2 = wf2.sid and wf1.deptName <> wf2.deptName);
b) ∀f ∀w1 ∀w2((hasFriend(f) ∧employedBy(w1) ∧employedBy(w2) ∧ f.sid1 = w1.sid ∧ f.sid2 = w2.sid ∧ w1.dname ≠ w2.dname)

CONSTRAINTS
Eg.1 Each student has at least two skills
πsid(S) ⊆ πsS1.sid(σsS1.skill≠sS2.skill(sS1 ⋈ sS1.sid=sS2.sid sS2))

Eg.2 Some student works for Google
πsid(σcname=Google(worksF or)) ≠ ∅

7. Each manager knows all of his/her students
πmid(M) ⊆ πmid(σM.eid=K.sid2(M ⋈ M.mid=K.sid1 K))

8. No student who works at Amazon knows at-most 2 people
πw1.sid(σw1.cname="Amazon"(W1))−πw1.sid(σw1.cname="Amazon"(W1 ⋈ w1.sid=k1.sid K1 ⋈ w1.sid=k2.sid K2 ⋈ w1.sid=k3.sid K3)) = ∅

9. Some student who works for a company headquarted at Cupertino has a salary less than
studentWithNoSkill(A) = πs.sid(S) – πss.sid(SS)
StudentWithNoSkill Salary(B) = πw.salary (W ⋈ w.sid=a.sid A)
πw.sid(W ⋈ (w.salary < b.salary)B ⋈ (w.cname = c.cname)σc.headquarter="Cupertino"(C)) ≠∅

PURE SQL a – exists, not exists b – in, not in, some, all c – intersect, union, except

-- 1. Find each triple (c, s, sl) where c is the cname of a company, s is the sid of a student who earns the lowest salary at that company and knows at least someone who has Operating Systems skill, and sl is the salary of s."
a) SELECT DISTINCT W1.CNAME AS C, W1.SID AS S, W1.salary AS SL FROM WORKSFOR W1,KNOWS K WHERE NOT EXISTS (SELECT 1 FROM WORKSFOR W2 WHERE W1.SID != W2.SID AND W1.CNAME=W2.CNAME AND W1.salary > W2.salary) AND W1.SID=K.SID1 AND EXISTS(SELECT 1 FROM STUDENTSKILL SS WHERE SS.SID=K.SID2 AND SS.SKILL='OperatingSystems') ORDER BY W1.CNAME;

b) SELECT DISTINCT W1.CNAME AS C, W1.SID AS S , W1.salary AS SL FROM WORKSFOR W1 , KNOWS K WHERE W1.CNAME NOT IN (SELECT W2.CNAME FROM WORKSFOR W2 WHERE W1.SID<>W2.SID AND W1.SALARY > W2.SALARY) AND W1.SID=K.SID1 AND K.SID2 IN (SELECT SS.SID FROM STUDENTSKILL SS WHERE SS.SKILL ='OperatingSystems') ORDER BY W1.CNAME;

c) SELECT W.CNAME AS C, W.SID AS S, W.SALARY SL FROM (SELECT W1.CNAME, W1.SID , W1.salary FROM WORKSFOR W1,KNOWS K , STUDENTSKILL SS WHERE W1.SID=K.SID1 AND SS.SID=K.SID2 AND SS.SKILL='OperatingSystems' EXCEPT SELECT DISTINCT W1.CNAME, W1.SID , W1.salary FROM WORKSFOR W1, WORKSFOR W2 WHERE W1.SID != W2.SID AND W1.CNAME=W2.CNAME AND W1.salary > W2.salary) W ORDER BY W.CNAME;

-- 2. Find the name, salary and city of each student who (a) lives in a city where no one has the Networks skill and (b) earns the highest salary in his/her company.
a) SELECT S1.SNAME,W.SALARY,S1.CITY FROM STUDENT S1,WORKSFOR W WHERE W.SID=S1.SID AND NOT EXISTS (SELECT 1 FROM STUDENT S2,STUDENTSKILL SS WHERE S2.SID = SS.SID AND S1.CITY = S2.CITY AND SS.SKILL = 'Networks') AND NOT EXISTS (SELECT 1 FROM WORKSFOR W1, WORKSFOR W2 WHERE W1.SID <> W2.SID AND W1.CNAME=W2.CNAME AND W1.SALARY < W2.SALARY AND W1.SID=S1.SID) ORDER BY S1.SNAME;

b) SELECT S1.SNAME,W.SALARY,S1.CITY FROM STUDENT S1,WORKSFOR W WHERE W.SID=S1.SID AND S1.CITY NOT IN (SELECT S2.CITY FROM STUDENT S2,STUDENTSKILL SS WHERE S2.SID = SS.SID AND SS.SKILL = 'Networks') AND S1.SID NOT IN (SELECT W1.SID FROM WORKSFOR W1, WORKSFOR W2 WHERE W1.SID <> W2.SID AND W1.CNAME=W2.CNAME AND W1.SALARY < W2.SALARY) ORDER BY S1.SNAME;

c) SELECT S.SNAME AS SNAME ,S.SALARY AS SALARY ,S.CITY AS CITY FROM (SELECT DISTINCT S1.SNAME,W.SALARY,S1.CITY FROM STUDENT S1,STUDENT S2,WORKSFOR W WHERE W.SID=S1.SID AND S1.CITY = S2.CITY EXCEPT SELECT DISTINCT S1.SNAME,W.SALARY,S1.CITY FROM STUDENT S1,WORKSFOR W,STUDENTSKILL SS WHERE W.SID=S1.SID AND S1.SID = SS.SID AND SS.SKILL = 'Networks' EXCEPT SELECT DISTINCT S1.SNAME,W1.SALARY,S1.CITY FROM STUDENT S1, WORKSFOR W1, WORKSFOR W2 WHERE W1.SID=S1.SID AND W1.SID <> W2.SID AND W1.CNAME=W2.CNAME AND W1.SALARY < W2.SALARY) S ORDER BY S.SNAME;

-- 3. Find each pair (c1, c2) of cnames of different companies such that no employee of c1 and no employee of c2 live in Chicago
a) SELECT C1.CNAME ,C2.CNAME FROM COMPANY C1,COMPANY C2 WHERE C1.CNAME<>C2.CNAME AND EXISTS(SELECT * FROM STUDENT S1,WORKSFOR W1 WHERE S1.CITY= 'Chicago' AND S1.SID=W1.SID AND C1.CNAME<>W1.CNAME AND C2.CNAME<>W1.CNAME) ORDER BY C1.CNAME,C2.CNAME;

b) SELECT C1.CNAME ,C2.CNAME FROM COMPANY C1,COMPANY C2 WHERE C1.CNAME<>C2.CNAME AND C1.CNAME NOT IN (SELECT W1.CNAME FROM STUDENT S1, WORKSFOR W1 WHERE S1.CITY='Chicago' AND S1.SID=W1.SID) AND C2.CNAME NOT IN (SELECT W1.CNAME FROM STUDENT S1, WORKSFOR W1 WHERE S1.CITY='Chicago' AND S1.SID=W1.SID) ORDER BY C1.CNAME,C2.CNAME;

c) SELECT C1.CNAME ,C2.CNAME FROM COMPANY C1,COMPANY C2 WHERE C1.CNAME<>C2.CNAME EXCEPT SELECT DISTINCT C1.CNAME ,C2.CNAME FROM COMPANY C1,COMPANY C2, STUDENT S1, WORKSFOR W1 WHERE C1.CNAME<>C2.CNAME AND S1.CITY='Chicago' AND S1.SID=W1.SID AND W1.CNAME=C1.CNAME EXCEPT SELECT DISTINCT C2.CNAME , C1.CNAME FROM COMPANY C1,COMPANY C2, STUDENT S1, WORKSFOR W1 WHERE C1.CNAME<>C2.CNAME AND S1.CITY='Chicago' AND S1.SID=W1.SID AND W1.CNAME=C1.CNAME;

RA and RA SQL
4 a) π cname, sid, salary (π w1.cname, w1.sid, w1.salary (W1 ⋈ w1.sid=k.sid1 K ⋈ sS.sid=k.sid2 (σ sS.skill='OperatingSystems'(sS))) – π w1.cname, w1.sid, w1.salary (W1 ⋈ w1.sid ≠ w2.sid AND w1.cname=w2.cname AND w1.salary ≤ w2.salary W2))

4 b) SELECT W.CNAME AS C, W.SID AS P, W.SALARY FROM (SELECT W1.CNAME, W1.SID , W1.salary FROM WORKSFOR W1 JOIN KNOWS K ON W1.SID=K.SID1 JOIN STUDENTSKILL SS ON SS.SID=K.SID2 AND SS.SKILL='OperatingSystems' EXCEPT SELECT DISTINCT W1.CNAME, W1.SID , W1.salary FROM WORKSFOR W1 JOIN WORKSFOR W2 ON W1.SID != W2.SID AND W1.CNAME=W2.CNAME AND W1.salary > W2.salary) W ORDER BY W.CNAME;

5 a) π sname, salary, city (π s1.sname, w.salary, s1.city (S1 ⋈ s1.city=s2.city S2 ⋈ w.sid=s1.sid W1)
– π s1.sname, w.salary, s1.city (S1 ⋈ w.sid=s1.sid W1 ⋈ s1.sid=ps.sid AND ps.skill='Networks' pS)
– π s1.sname, w1.salary, s1.city (S1 ⋈ w1.sid=s1.sid W1 ⋈ w1.sid ≠ w2.sid AND w1.cname=w2.cname AND w1.salary < w2.salary W2))

5 b) SELECT S.SNAME AS SNAME ,S.SALARY AS SALARY ,S.CITY AS CITY FROM (SELECT DISTINCT S1.SNAME,W.SALARY,S1.CITY FROM STUDENT S1 JOIN STUDENT S2 ON S1.CITY = S2.CITY JOIN WORKSFOR W ON W.SID=S1.SID EXCEPT SELECT DISTINCT S1.SNAME,W.SALARY,S1.CITY FROM STUDENT S1 JOIN WORKSFOR W ON W.SID=S1.SID JOIN STUDENTSKILL SS ON S1.SID = SS.SID AND SS.SKILL = 'Networks' EXCEPT SELECT DISTINCT S1.SNAME,W1.SALARY,S1.CITY FROM STUDENT S1 JOIN WORKSFOR W1 ON W1.SID=S1.SID JOIN WORKSFOR W2 ON W1.SID <> W2.SID AND W1.CNAME=W2.CNAME AND W1.SALARY < W2.SALARY) S ORDER BY S.SNAME;

6 a) π c1.cname, c2.cname (π c1.cname, c2.cname (C1 ⋈ c1.cname ≠ c2.cname C2)
– π c1.cname, c2.cname (C1 ⋈ c1.cname ≠ c2.cname C2 ⋈ s1.city='Chicago' S1 ⋈ s1.sid=w1.sid AND w1.cname=c1.cname W1) – π c2.cname, c1.cname (C1 ⋈ c1.cname ≠ c2.cname C2 ⋈ s1.city='Chicago' S1 ⋈ s1.sid=w1.sid AND w1.cname=c1.cname W1))

6 b) SELECT C1.CNAME ,C2.CNAME FROM COMPANY C1 JOIN COMPANY C2 ON C1.CNAME<>C2.CNAME EXCEPT SELECT DISTINCT C1.CNAME ,C2.CNAME FROM COMPANY C1 JOIN COMPANY C2 ON C1.CNAME<>C2.CNAME JOIN STUDENT S1 ON S1.CITY='Chicago' JOIN WORKSFOR W1 ON S1.SID=W1.SID AND W1.CNAME=C1.CNAME EXCEPT SELECT DISTINCT C2.CNAME , C1.CNAME FROM COMPANY C1 JOIN COMPANY C2 ON C1.CNAME<>C2.CNAME JOIN STUDENT S1 ON S1.CITY='Chicago' JOIN WORKSFOR W1 ON S1.SID=W1.SID AND W1.CNAME=C1.CNAME;

10. Create a materialized view CompanyKnownStudent such that, for each company, the view returns the sid of Student who are known by at least two different student (other than pid) from the same company and the sid earns more salary than them.

CREATE MATERIALIZED VIEW CompanyKnownStudent AS
SELECT DISTINCT W1.SID
FROM WORKSFOR W1, KNOWS K1,KNOWS K2
WHERE W1.SID=K1.SID2 AND W1.SID=K2.SID2 AND

```
K1.SID1<>K1.SID2 AND K2.SID1<>K2.SID2 AND K1.SID1<>K2.SID1
AND EXISTS (SELECT 1 FROM WORKSFOR W2,WORKSFOR W3
        WHERE W2.SID=K1.SID1 AND W3.SID=K2.SID1 AND W1.CNAME=W2.CNAME AND W1.CNAME=W3.CNAME
        AND W1.SALARY>W2.SALARY AND W1.SALARY>W3.SALARY);
SELECT * FROM CompanyKnownStudent;
```

11. Create a parameterized view SkillOnlyOneStudent (skill1 text) that returns pair of different Students sid1, sid2 such that sid1 should have the skill identified by skill1 and sid2 should not have the skill identified by skill1. Note that sid2 should have at least one skill.

```
CREATE FUNCTION SkillOnlyOneStudent(skill1 TEXT)
RETURNS TABLE(SID1 TEXT,SID2 TEXT) AS
$$
SELECT DISTINCT SS1.SID AS SID1 ,SS2.SID AS SID2
FROM STUDENTSKILL SS1, STUDENTSKILL SS2
WHERE SS1.SID<>SS2.SID AND SS1.SKILL <> SS2.SKILL AND SS1.SKILL= skill1
ORDER BY SS1.SID,SS2.SID;
$$ LANGUAGE SQL;

SELECT * FROM SkillOnlyOneStudent('WebDevelopment');
```

12. Let P C(parent : integer, child : integer) be a rooted parentchild tree. So a pair (n, m) in P C indicates that node n is a parent of node m. The sameGeneration(n1, n2) binary relation is inductively defined using the following two rules: • If n is a node in P C, then the pair (n, n) is in the sameGeneration relation. (Base rule) • If n1 is the parent of m1 in P C and n2 is the parent of m2 in T ree and (n1, n2) is a pair in the sameGeneration relation then (m1, m2) is a pair in the sameGeneration relation. (Inductive Rule)

```
CREATE OR REPLACE VIEW sameGeneration AS
WITH RECURSIVE samegeneration(n1, n2) AS (
   -- Base rule: every node is in the same generation as itself
   SELECT parent AS n1, parent AS n2
   FROM PC
   UNION
   SELECT child AS n1, child AS n2
   FROM PC
   UNION
   -- Inductive rule
   SELECT PC1.child AS n1, PC2.child AS n2
   FROM PC PC1
   JOIN sameGen SG ON PC1.parent = SG.n1
   JOIN PC PC2 ON PC2.parent = SG.n2
   WHERE PC1.child <> PC2.child
)
SELECT DISTINCT * FROM samegeneration
ORDER BY n1, n2;
SELECT * FROM sameGeneration;
```

AGGREGATE FUNCTIONS

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

```
create or replace function lowest_salary_at(city_ TEXT)
returns table (salary INT) as
$$    with students_at_city as ( select p.* from student p where p.city = city_   )
    select pac.sid from students_at_city pac join worksfor w on
    w.sid = pac.sid and  w.salary = ( select min(w1.salary) from worksfor w1  join students_at_city pac1
        on pac1.sid = w1.sid     );        $$ language sql;
```

2. Find sid & name of each Stud who has less than 2 of the combined set of job skills of Studs who work for Netflix

```
WITH netflix_job_skills  AS(  SELECT    s.skill   FROM      Skill s    JOIN     studentSkill ps   ON       ps.skill = s.skill
    JOIN worksFor w     ON        ps.sid = w.sid AND w.cname = 'Netflix'   GROUP BY s.skill)
SELECT    p.sid,p.pname FROM student p  GROUP      BY p.sid, p.pname
HAVING    (SELECT COUNT(*) FROM ( SELECT    ps.skill FROM studentSkill ps
            WHERE     ps.sid = p.sid  INTERSECT SELECT * FROM netflix_job_skills)q) < 2;
select distinct p.city "c" ,  lowest_salary_at(p.city) "p" from student p;
```

3. Find each pairs (s1; s2) of skills such that the set of Studs with skill s1 is the same as the set of Studs with skill s2

```
SELECT DISTINCT     s1.skill AS s1,s2.skill AS s2     FROM Skill s1,Skill s2  WHERE     (
    SELECT COUNT(q.sid) FROM(  SELECT  q1.sid FROM (    SELECT ps1.sid FROM studentSkill ps1 WHERE
ps1.skill=s1.skill
            EXCEPT   SELECT ps2.sid FROM studentSkill ps2 WHERE ps2.skill = s2.skill)q1)q) = 0;
```

4. Find each sid of a Student who knows at least two people who (a) work for Apple and (b) who make less than 55000.You must not use set predicates in this query

```
SELECT    p.sid FROM student p WHERE (      SELECT COUNT(*) FROM (      SELECT    k.sid2      FROM knows k
        JOIN (SELECT ws.sid FROM worksFor ws WHERE ws.salary < 55000 AND ws.cname = 'Apple' ) w
        ON        k.sid2 = w.sid          AND       k.sid1 = p.sid)q) >= 2;
```

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.You must not use set predicates in this query

```
create function AmazonEmployees() returns table(sid int) as
$$    select w.sid from worksFor w where w.cname='Amazon'; $$ language sql;

create function getNumEmpKnowingAtLeastQuarter(companyName text)
returns int as $$     select count(*)  from worksFor w   where w.cname=companyName   and    4 * (select count(*)
 from knows k where k.sid1=w.sid and k.sid2 in (select * from AmazonEmployees())) >= (select count(*) from
AmazonEmployees())    $$ language sql;

select distinct c.cname  from company c   where getNumEmpKnowingAtLeastQuarter(c.cname) > 0;
```

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. You must not use set predicates

```
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.cnameORDER BY mc.cname;
```

7. Using the GROUP BY count method, define a function 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 INT, salary INT, numberOfSkills
BIGINT) AS
$$      SELECT    w.sid, w.salary, COUNT(*) AS n      FROM (SELECT * FROM worksFor ws WHERE ws.cname = c) w
    JOIN studentSkill ps ON w.sid = ps.sid     GROUP     BY w.sid,w.salary  UNION      SELECT    w.sid, w.salary, 0 AS n
    FROM     worksFor w     WHERE     w.cname = c  AND  w.sid NOT IN (SELECT ps.sid FROM studentSkill ps)
    ORDER     BY n, sid;         $$ LANGUAGE SQL;
SELECT * FROM numberOfSkills('Apple');
```

Without using the GROUP BY clause

```
CREATE OR REPLACE FUNCTION numberOfSkills(c TEXT) RETURNS TABLE (sid INT, salary INT, numberOfSkills
BIGINT) AS
$$    SELECT    w.sid, w.salary, (SELECT COUNT(ps.sid) FROM studentSkill ps WHERE ps.sid = w.sid) AS n
  FROM     worksFor w   WHERE     w.cname = c  UNION SELECT    q.sid, q.salary, 0 AS n -- NOT IN to EXCEPT
  FROM ( SELECT w.* FROM worksFor w  WHERE     w.cname = c EXCEPT  SELECT w.* FROM worksFor w,studentSkill ps
        WHERE     w.cname = c AND w.sid = ps.sid)q  ORDER BY n,sid;      $$ LANGUAGE SQL;
```

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

```
SELECT    c.cname, nos.sid FROM Company c, numberOfSkills(c.cname) nos WHERE     nos.salary > 50000
AND   nos.numberOfSkills = (SELECT  MAX(q.numberOfSkills) FROM ( SELECT    nos1.numberOfSkills
FROM Company c1, numberOfSkills(c1.cname) nos1 WHERE     c1.cname = c.cname)q );
```

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 (without count)

```
CREATE OR REPLACE FUNCTION Knows(sid int)    RETURNS TABLE (sid int) AS   $$     SELECT k.sid2  FROM   knows
k  WHERE  k.sid1 = Knows.sid;   $$ LANGUAGE SQL;

CREATE OR REPLACE VIEW CONDITIONAL AS    SELECT distinct p.sid    FROM   student p    join worksfor w on
p.sid=w.sid and w.salary>=55000   join studentskill ps on p.sid = ps.sid    WHERE p.city ='Bloomington';

SELECT sid, PNAME  FROM student P  WHERE NOT EXISTS(SELECT * FROM CONDITIONAL EXCEPT SELECT *
FROM KNOWS(P.sid));
```

QUANTIFIERS

9. Find the cname of each company who only employs managers who make more than 50000 (Without COUNT)

```
CREATE OR REPLACE FUNCTION Managers(CNAME text)  RETURNS TABLE (mid int) AS     $$  SELECT distinct
sid  FROM worksfor w JOIN hasmanager h on h.mid = w.sid   WHERE  w.cname = Managers.cname;    $$ LANGUAGE
SQL;

CREATE OR REPLACE FUNCTION makesmore(CNAME text)    RETURNS TABLE (eid int) AS  $$    SELECT distinct sid
FROM worksfor w  WHERE  w.cname = makesmore.cname and w.salary > 50000;   $$ LANGUAGE SQL;

SELECT distinct c.CNAME  FROM company C WHERE NOT EXISTS(SELECT mid FROM Managers(c.cname)  where
mid not in(            SELECT eid FROM makesmore(c.cname)));
```

10. Find the sid and name of each stud who knows at least 3 ppl who each have at most 2 managers (with COUNT)

```
CREATE OR REPLACE VIEW AtMostMan AS  SELECT distinct h.eid FROM   hasmanager h group by h.eid having
count(mid)<=2;

SELECT sid, PNAME FROM student P WHERE (Select count(1)  from (SELECT * FROM KNOWS(P.sid)   INTERSECT
            SELECT * FROM AtMostMan) q)>=3;
```

11. Find the cname of each company that employs an even number of Studs who have at least 2 skills (USE COUNT)

```
CREATE OR REPLACE VIEW AtLeastSkill AS  SELECT distinct ps.sid FROM   studentskill ps  group by ps.sid
  having count(skill)>=2;

CREATE OR REPLACE FUNCTION Employees(CNAME text)  RETURNS TABLE (eid int) AS  $$ SELECT distinct sid
  FROM worksfor w   WHERE  w.cname = Employees.cname;  $$ LANGUAGE SQL;

SELECT distinct c.CNAME  FROM company C WHERE mod((Select count(1)   from (SELECT * FROM
Employees(C.CNAME)
            INTERSECT    SELECT * FROM AtLeastSkill) q), 2) = 0;
```

12. Find pairs (p1, p2) of diffrnt stud sids so that the stud with sid p1 & the stud with sid p2 have the same no. of skills (USE COUNT)

```
CREATE OR REPLACE FUNCTION countskill(sid int) RETURNS TABLE (cnt int) AS  $$  SELECT count(skill)  FROM
studentskill p  WHERE  p.sid = countskill.sid    group by p.sid;   $$ LANGUAGE SQL;

SELECT Distinct P.sid as sid1, C.sid as sid2 FROM studentSKILL P  join studentSKILL C
on P.sid <> C.sid WHERE NOT EXISTS(SELECT * FROM countskill(P.sid) EXCEPT  SELECT * FROM countskill(C.sid));
```

TRIGGERS (no postgres cascade)

13. Explain how triggers can be used to implement the Primary key Constraint, with an example

```
CREATE OR REPLACE FUNCTION CHECK_COMPANY_KEY_CONSTRAINT() RETURNS TRIGGER AS $$
BEGIN IF NEW.cname IN (SELECT cname FROM Company) THEN  RAISE EXCEPTION 'cname already exists';
END IF;  RETURN NEW;  END;  $$ LANGUAGE 'plpgsql';

CREATE TRIGGER CHECK_COMPANY_KEY_  BEFORE  INSERT ON COMPANY  FOR EACH ROW EXECUTE
PROCEDURE CHECK_COMPANY_KEY_CONSTRAINT();
INSERT INTO COMPANY VALUES ('Apple', 'Cupertino');
```

14. Explain how triggers can be used to implement the Referential Integrity Constraint, with an example

```
CREATE OR REPLACE FUNCTION CHECK_COMPANYLOC_FKEY_CONSTRAINT() RETURNS TRIGGER AS $$
BEGIN IF NEW.cname NOT IN (SELECT cname FROM Company)  THEN  RAISE EXCEPTION 'cname does not exist in
Company'; END IF; RETURN NEW; END; $$ LANGUAGE 'plpgsql';

CREATE TRIGGER CHECK_COMPANYLOC_FKEY_ BEFORE INSERT ON COMPANYLOCATION
FOR EACH ROW EXECUTE PROCEDURE CHECK_COMPANYLOC_FKEY_CONSTRAINT();

INSERT INTO COMPANYLOCATION VALUES ('META', 'MenloPark');
```

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 func- tions on the relations R and S that implement this mate- rialized view. Write your trigger functions in the language 'plpgsql.' Make sure that your trigger functions act in an incremen- tal 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 OR REPLACE FUNCTION INSERT_R() RETURNS TRIGGER AS $$ BEGIN
   INSERT INTO V   (SELECT A FROM R    WHERE A > 10 AND B IN (SELECT B FROM S)   AND A NOT IN (SELECT A
FROM V));
RETURN NULL;  END;  $$ LANGUAGE PLPGSQL;


CREATE TRIGGER INSERT_R AFTER  INSERT ON R  FOR EACH ROW EXECUTE PROCEDURE INSERT_R();

    CREATE OR REPLACE FUNCTION INSERT_S() RETURNS TRIGGER AS $$ BEGIN    INSERT INTO V   (SELECT A
FROM R              WHERE B IN (SELECT B FROM S)    AND A > 10 AND A NOT IN (SELECT A FROM V));  RETURN
NULL;    END;
$$ LANGUAGE PLPGSQL;

CREATE TRIGGER INSERT_S AFTER   INSERT ON S   FOR EACH ROW EXECUTE PROCEDURE INSERT_S();
```

PURE SQL:

Select, from, where
Union, intersect, except,
Exists, not exists
In, not in, not, all, some
Views, with
No aggregates & joins

RA SQL:

select, from, where - (only with constants), no commas
union, intersect, except
join, cross join, natural join
views, with
no aggregates & predicates