Assignment 5
Query Translation and Optimization

For this assignment you will need the material covered in the lectures

- Lecture 13: Translating SQL queries into RA expressions
- Lecture 14: Query optimization

For this assignment, you will need to submit a single .pdf file that contains your solutions for the problems on this assignment.

1 Theoretical Problems

1. In the translation algorithm from SQL to RA, when we eliminated set predicates, we tacitly assumed that the argument of each set predicate was a (possibly parameterized) SQL query that did not use a UNION, INTERSECT, or an EXCEPT operation.

   In this problem, you are asked to extend the translation algorithm from SQL to RA such that (possibly parameterized) set predicates [NOT] EXISTS are eliminated that have as an argument a SQL query that uses a UNION, INTERSECT, or EXCEPT operation.

   More specifically, consider the following types of queries using the [NOT] IN set predicate.

```
SELECT L(r1,...,rk)
FROM   R1 r1, ..., Rk rk
WHERE  C1(r1,...,rk) AND
                [NOT] EXISTS (SELECT L1(s1,...,sm)
                              FROM   S1 s1,..., S1 sm
                              WHERE  C2(s1,...,sm,r1,...,rk)
                              [UNION|INTERSECT|EXCEPT]
                              SELECT L2(t1,...,tn)
                              FROM   T1 t1, ..., Tn tn
                              WHERE  C3(t1,...,tn,r1,...,rk))
```

   Notice that there are six cases to consider:

   (a) `EXISTS (...  UNION ...)`

   (b) `EXISTS (...  INTERSECT ...)`

   (c) `EXISTS (...  EXCEPT ...)`

   (d) `NOT EXISTS (...  UNION ...)`

   (e) `NOT EXISTS (...  INTERSECT ...)`

   (f) `NOT EXISTS (...  EXCEPT ...)`

Show how such SQL queries can be translated to equivalent RA expressions for the special case where $k = m = n = 1$. In particular, for the case

```
SELECT L1(r)
FROM   R r
WHERE  C1(r) AND [NOT] EXISTS (SELECT L2(s)
                               FROM   S s
                               WHERE  C2(s,r)
                               [UNION|INTERSECT|EXCEPT]
                               SELECT L3(t)
                               FROM   T t
                               WHERE  C3(t,r))
```

Be careful in the translation since you should take into account that projections do not in general distribute over intersections or over set differences.

Solution: We translate towards an SQL RA query. It is then straighforward to translate this to an RA expression in standard notation.

The first thing we observe it that the NOT EXISTS cases can be reduced to the EXISTS cases. Notice that

```
SELECT L1(r)
FROM   R r
WHERE  C1(r) AND NOT EXISTS (SELECT L2(s)
                             FROM   S s
                             WHERE  C2(s,r)
                             [UNION|INTERSECT|EXCEPT]
                             SELECT L3(t)
                             FROM   T t
                             WHERE  C3(t,r))
```

can be translated to

```
SELECT L1(r)
FROM   R r
WHERE  C1(r)
EXCEPT
SELECT L1(r)
FROM   R r
WHERE  C1(r) AND EXISTS (SELECT L2(s)
                         FROM   S s
                         WHERE  C2(s,r)
                         [UNION|INTERSECT|EXCEPT]
                         SELECT L3(t)
                         FROM   T t
                         WHERE  C3(t,r))
```

So we can focus on the translation of

```
SELECT L1(r)
FROM   R r
WHERE  C1(r) AND EXISTS (SELECT L2(s)
                         FROM   S s
                         WHERE  C2(s,r)
                         [UNION|INTERSECT|EXCEPT]
                         SELECT L3(t)
                         FROM   T t
                         WHERE  C3(t,r))
```

We begin with the UNION case. It should be clear that

```
SELECT L1(r)
FROM   R r
WHERE  C1(r) AND EXISTS (SELECT L2(s)
                         FROM   S s
                         WHERE  C2(s,r)
                         UNION
                         SELECT L3(t)
                         FROM   T t
                         WHERE  C3(t,r))
```

is equivalent with

```
SELECT L1(r)
FROM   R r
WHERE  C1(r) AND (EXISTS (SELECT L2(s)
                          FROM   S s
                          WHERE  C2(s,r))
                          OR
                  EXISTS (SELECT L3(t)
                          FROM   T t
                          WHERE  C3(t,r)))
```

By the distribution law of AND over OR, this is equivalent with

```
SELECT L1(r)
FROM   R r
WHERE  C1(r) AND EXISTS (SELECT L2(s)
                         FROM   S s
                         WHERE  C2(s,r))
       OR
       C1(r1) AND EXISTS (SELECT L3(t)
                          FROM   T t
                          WHERE  C3(t,r))
```

We can replace the OR using a UNION operation as follows

```
SELECT L1(r)
FROM   R r
WHERE  C1(r) AND EXISTS (SELECT L2(s)
                         FROM   S s
                         WHERE  C2(s,r))
UNION
SELECT L1(r)
FROM   R r
WHERE  C1(r1) AND EXISTS (SELECT L3(t)
                          FROM   T t
                          WHERE  C3(t,r))
```

And this is equivalent with

```
SELECT L1(r)
FROM   R r JOIN S s ON (C1(r) AND C2(s,r))
UNION
SELECT L1(r)
FROM   R r JOIN S s ON (C1(r1) AND C3(t,r))
```

Next consider the INTERSECT case. This is more complex since EXISTS does not distribute over INTERSECT. Consider

```
SELECT L1(r)
FROM   R r
WHERE  C1(r) AND EXISTS (SELECT L2(s)
                         FROM   S s
                         WHERE  C2(s,r)
                         INTERSECT
                         SELECT L3(t)
                         FROM   T t
                         WHERE  C3(t,r))
```

To keep things simple, let us assume that `L2(s)` is `s.A` and that `L3(t)` is `t.A`. So we must translate

```
SELECT L1(r)
FROM   R r
WHERE  C1(r) AND EXISTS (SELECT s.A
                         FROM   S s
                         WHERE  C2(s,r)
                         INTERSECT
                         SELECT t.A
                         FROM   T t
                         WHERE  C3(t,r))
```

It is instructive to formulate this query in the Tuple Relational Calculus

$\{L1(r) \mid R(r) \wedge C1(r) \wedge \exists s \exists t(S(s) \wedge C2(s,r) \wedge s.A = t.A \wedge T(t) \wedge C3(t,r)\}$.

This is equal with

$$\{L1(r) \mid R(r) \wedge C1(r) \wedge \exists s \exists t(S(s) \bowtie_{s.A=t.A} T(t) \wedge C2(s,r) \wedge C3(t,r)\}.$$

which is equal with

$$\{L1(r) \mid R(r) \wedge C1(r) \wedge \exists s \exists t(R(r) \wedge C1(r) \wedge S(s) \bowtie_{s.A=t.A} T(t) \wedge C2(s,r) \wedge C3(t,r)\}.$$

which is equal with

$$\{L1(r) \mid R(r) \wedge C1(r) \wedge \exists s \exists t(R(r) \wedge C1(r) \wedge S(s) \bowtie_{s.A=t.A} T(t) \wedge C2(s,r) \wedge C3(t,r)\}.$$

which is equal with

$$\{L1(r) \mid R(r) \wedge C1(r) \wedge \exists s \exists t(R(r) \bowtie_{C1(r) \wedge C2(s,r) \wedge C3(t,r)} (S(s) \bowtie_{s.A=t.A} T(t))\}$$

and this is equal with

$$\pi_{L1(r)}(\sigma_{C1(r)}(R) \cap \pi_{R.*}(R \bowtie_{C1(r) \wedge C2(s,r) \wedge C3(t,r)} (S \bowtie_{S.A=T.A} T)))$$

or, with

$$\pi_{L1(r)}(\sigma_{C1(r)}(R) \cap \pi_{R.*}(\sigma_{C1(r)}(R) \bowtie_{C2(s,r) \wedge C3(t,r)} (S \bowtie_{S.A=T.A} T)))$$

In the most general case, the RA expression for the INTERSECT case is

$$\pi_{L1(r)}(\sigma_{C1(r)}(R) \cap \pi_{R.*}(\sigma_{C1(r)}(R) \bowtie_{C2(s,r) \wedge C3(t,r)} (S \bowtie_{L2(s)=L3(t)} T)))$$

In RA SQL, this becomes

```
SELECT L1(r)
FROM   (SELECT R.*
        FROM   R r
        WHERE  C1(r)
        INTERSECT
        SELECT R_q.*
        FROM (SELECT r.* AS R_q FROM R WHERE C1) r
               JOIN (S s JOIN T t ON (L2(s)=L3(t)) ON (C2(s,r) AND C3(t,r)))) q
```

A similar analysis for the EXCEPT case leads to the following RA expression and the RA SQL query

$$\pi_{L1(r)}(\sigma_{C1(r)}(R) - \pi_{R.*}(\sigma_{C1(r)}(R) \bowtie_{C2(s,r) \wedge C3(t,r)} (S \bowtie_{L2(s)=L3(t)} T)))$$

In RA SQL, this becomes

```
SELECT L1(r)
FROM   (SELECT R.*
        FROM   R r
        WHERE  C1(r)
        EXCEPT
        SELECT R_q.*
        FROM (SELECT r.* AS R_q FROM R WHERE C1) r
               JOIN (S s JOIN T t ON (L2(s)=L3(t)) ON (C2(s,r) AND C3(t,r)))) q
```

2. Let R be a relation with schema (a, b, c) and let S be a relation with schema (a, b, d). You may assume that the domains for the attributes a, b, and c are the same.

   Prove or disprove that

$$\pi_{a,b}(R \overline{\ltimes} S) = \pi_{a,b}(R) - \pi_{a,b}(S).$$

   Solution: This equation holds.

   We can formulate $\pi_{a,b}(R \overline{\ltimes} S)$ in TRC as follows

$$\{(a, b) \mid \exists c(R(a, b, c) \land \neg\exists d(R(a, b, c) \land S(a, b, d))\}.$$

   It can be verified that the formulas

$$R(a, b, c) \land \neg\exists d(R(a, b, c) \land S(a, b, d)) \Leftrightarrow R(a, b, c) \land \neg\exists d(S(a, b, d)).$$

   So,

$$\begin{array}{rcl}
\pi_{a,b}(R \overline{\ltimes} S) & = & \{(a, b) \mid \exists c(R(a, b, c) \land \neg\exists d(R(a, b, c) \land S(a, b, d))\} \\
& = & \{(a, b) \mid \exists c(R(a, b, c) \land \neg\exists d(S(a, b, d))\} \\
& = & \{(a, b) \mid \exists c(R(a, b, c) \land \neg(a, b) \in \pi_{a,b}(S))\} \\
& = & \{(a, b) \mid \exists c(R(a, b, c)) \land \neg(a, b) \in \pi_{a,b}(S))\} \\
& = & \{(a, b) \mid (a, b) \in \pi_{a,b}(R) \land \neg(a, b) \in \pi_{a,b}(S))\} \\
& = & \pi_{a,b}(R) - \pi_{a,b}(S).
\end{array}$$

2 Translating and Optimizing SQL Queries to Equivalent RA Expressions

Using the translation algorithm presented in class, translate each of the following SQL queries into equivalent RA expressions. For each query, provide its corresponding RA expression in your .pdf file. This RA expression needs to be formulated in the standard RA notation.

Then use rewrite rules to optimize each of these RA expressions into an equivalent but optimized RA expression. Include this optimized expression in you .pdf file is standard RA notation.

You are required to specify some, but not necessarily all, of the intermediate steps that you applied during the translations and optimizations. Use your own judgment to specify the most important steps.

During the optimization, take into account the primary keys and foreign key constraints that are assumed for the Person, Company, jobSkill, worksFor, Knows, and personSkill relations.

3. "Find the pid and name of each person who works for 'Google' and who knows a person who earns a lower salary.

```
select p1.pid, p1.name
from   person p1, worksfor w1
where  p1.pid = w1.pid and w1.cname = 'Google' and
       exists (select 1
               from  person p2, worksfor w2
               where  p2.pid = w2.pid and
                      (p1.pid,p2.pid) in (select k.pid1,k.pid2 from knows k) and
                      w1.salary < w2.salary);
```

   (a) Using the translation algorithm presented in the lectures, translate this SQL into and equivalent RA expression formulated in the standard RA syntax. Specify this RA expression in your .pdf file.
   Solution: The translation gives the following RA SQL query:

```
with   worksForGoogle as (select pid, cname, salary from worksfor where w.cname = 'Google')
select distinct p1.pid, p1.name
from   (worksForGoogle w1 join worksfor w2 on w1.salary < w2.salary)
       join
       (person p1
        join   knows k on p1.pid = k.pid1
        join   person p2 on p2.pid = k.pid2)
       on (p1.pid = w1.pid and p2.pid = w2.pid);
```

   Consider the following expressions:

$$\begin{aligned} worksForGoogle &= \pi_{pid,cname,salary}(\sigma_{cname=\text{Google}}(worksFor)) \\ E &= worksForGoogle \bowtie_{worksForGoogle.salary<w_2.salary} W_2 \\ F &= P_1 \bowtie_{P_1.pid=pid1} Knows \bowtie_{P_2.pid=pid2} P_2 \end{aligned}$$

   Then the RA expression becomes

$$\pi_{P_1.pid,name}(E \bowtie_{P_1.pid=workForGoogle.pid \wedge P_2.pid=W_2.pid} F).$$

(b) Using the optimization rewrite rules presented in the lectures, including those that rely on constraints, optimize this RA expression. Specify this optimized RA expression in your .pdf file.

Solution: The optimized version can be stated in RA SQL as follows:

```
select distinct pid, name
from
       (select distinct w1.pid as w1pid, w2.pid as w2pid
        from  (select pid, salary from worksfor  where cname = 'Google') w1
              join
              (select  pid, salary from worksfor) w2 on w1.salary < w2.salary) E
       join
       (select pid, name, pid2
        from   (select pid, name from person) p1
                join   knows k on pid = pid1) F
       on (pid = w1pid and pid2 = w2pid);
```

Consider the following expressions:

$$
\begin{aligned}
E &= \pi_{W_1.pid,W_2.pid}(\pi_{W_1.pid,W_1.salary}(\sigma_{cname=\text{Google}}(W_1)) \bowtie_{W_1.salary<W_2.salary} \pi_{W_2.pid,W_2.salary}(W_2)\\
F &= \pi_{P_1.pid,name,pid2}(\pi_{P_1.pid,name}(P_1) \bowtie_{P_1.pid=pid1} K)
\end{aligned}
$$

Then the optimized RA expression is

$$\pi_{P_1.pid,name}(E \bowtie_{P_1.pid=W_1.pid \wedge pid2=W_2.pid} F).$$

4. Find the pid of each person who

- has a ‘Programming’ skill or a ‘Networks’ skill.
- does not work for ‘Amazon’,
- does not know anyone who lives in ’Indianapolis’,

```
select p.pid
from   person p
where  p.pid = SOME (select ps.pid
                     from   personSkill ps
                     where  ps.skill = 'Programming' or ps.skill = 'Networks') and
        p.pid <> ALL (select w.pid
                      from   worksFor w
                      where  w.cname = 'Amazon') and
        not exists (select p1.pid
                    from   person p1
                    where  p1.city = 'Indianapolis' and
                           p1.pid in (select k.pid2 from knows k where k.pid1 = p.pid));
```

(a) Using the translation algorithm presented in the lectures, translate this SQL into and equivalent RA expression formulated in the standard RA syntax. Specify this RA expression in your .pdf file.
Solution. The translation gives the following RA SQL query:

```
with   personSkillProgrammingNetwork as
       (select pid, skill
        from   personSkill
        where  skill = 'Programming' or skill = 'Networks'),
       personIndianapolis as
       (select pid, name, city, birthyear
        from   person
        where  city = 'Indianapolis'),
       worksForAmazon as
       (select pid, cname, salary
        from   worksFor
        where  cname = 'Amazon')
select pid
from ((select pid, name, city, birthyear, skill
       from   person natural join personSkillProgrammingNetwork
       except
       select p.pid, name, city, birthyear, skill
       from   person p
              natural join personSkillProgrammingNetwork
              natural join worksForAmazon)
       intersect
       (select pid, name, city, birthyear, skill
        from   person natural join personSkillProgrammingNetwork
        except
        select p.pid, p.name, p.city, p.birthyear, skill
        from   person p
               natural join personSkillProgrammingNetwork
               join (personIndianapolis p1 join knows k on p1.pid = k.pid2) on k.pid1 = p.pid) ) q
order by 1;
```

Consider the expressions

$$\begin{aligned}
pSProgNet &= \sigma_{skill=\texttt{Programming} \lor skill=\texttt{Networks}}(personSkill)\\
pIndy &= \sigma_{city=\texttt{Indianapolis}}(Person)\\
workAmazon &= \sigma_{cname=\texttt{Amazon}}(worksFor)\\
E &= \pi_{P.*,skill}(P \bowtie pSkillProgNet)\\
F &= \pi_{P.*,skill}(E \bowtie workAmazon)\\
G &= \pi_{P.*,skill}(E \bowtie_{k.pid1=p.pid} (pIndy_1 \bowtie_{pIndy.pid=K.pid2} K))
\end{aligned}$$

Then the RA expression becomes

$$\pi_{P.pid}((E - F) \cap (E - G)).$$

Notice that this expression is also equivalent with

$$\pi_{P.pid}(E - (F \cup G)).$$

(b) Using the optimization rewrite rules in the lectures, including those that rely on constraints, optimize this RA expression. Specify this optimized RA expression in your .pdf file.

Solution.

```
with   personSkillProgrammingNetwork as
       (select pid, skill
        from   personSkill
        where  skill = 'Programming' or skill = 'Networks'),
       personIndianapolis as
       (select pid
        from   person
        where  city = 'Indianapolis'),
       worksForAmazon as
       (select pid
        from   worksFor
        where  cname = 'Amazon')
select pid
from (select pid, skill
      from   personSkillProgrammingNetwork
      except
      (select pid, skill
       from   personSkillProgrammingNetwork natural join worksForAmazon   -- semijojn
       union
       select distinct p.pid, skill
       from   personSkillProgrammingNetwork p
              join (select distinct pid1
                    from   personIndianapolis join knows k on pid = k.pid2) k on pid1 = p.pid)) q
order by 1;
```

Consider the expressions

$$\begin{array}{rcl}
pSProgNet & = & \pi_{pid,skill}(\sigma_{skill=\texttt{Programming} \vee skill=\texttt{Networks}}(personSkill)) \\
pIndy & = & \pi_{pid}(\sigma_{city=\texttt{Indianapolis}}(Person)) \\
workAmazon & = & \pi_{pid}(\sigma_{cname=\texttt{Amazon}}(worksFor)) \\
E & = & pSkillProgNet \\
F & = & \pi_{P.pid,skill}(E \ltimes workAmazon) \\
G & = & \pi_{P.pid,skill}(E \bowtie_{pid1=p.pid} (\pi_{pid}(pIndy \bowtie_{pid=K.pid2} K)))
\end{array}$$

$$\pi_{P.pid}(E - (F \cup G)).$$

5. Find each (p_1, p_2) pair where p_1 and p_2 are pids of persons and such that p_2 is among the oldest persons who are known by p_1.

```
select p1.pid, p2.pid
from   person p1, person p2
where (p1.pid, p2.pid) in (select k.pid1, k.pid2 from knows k) and
      not p2.birthyear > SOME (select p.birthyear
                               from   person p
                               where  p.pid in (select k.pid2
                                                from   knows k
                                                where  k.pid1 = p1.pid));
```

(a) Using the translation algorithm presented in the lectures, translate this SQL into and equivalent RA expression formulated in the standard RA syntax. Specify this RA expression in your .pdf file.

Solution:

The translation algorithm produces the SQL query

```
select p1pid, p2pid
from (select p1.pid as p1pid, p1.name, p1.city, p1.birthyear,
             k.*,
             p2.pid as p2pid, p2.name, p2.city, p2.birthyear
      from   person p1
             join knows k on p1.pid = k.pid1
             join person p2 on p2.pid = k.pid2
      except
      select p1.*, k.*, p2.*
      from   person p1
             join knows k on p1.pid = k.pid1
             join person p2 on p2.pid = k.pid2
             join (person p3 join knows k3 on (p3.pid = k3.pid2)) on
                    (p2.birthyear > p3.birthyear and k3.pid1 = p1.pid)) q
order by 1,2;
```

Consider the following expressions:

$$\begin{array}{lcl} E & = & \pi_{P_1.*,K.*,P_2.*}(P_1 \bowtie_{P_1.pid=K.pid1} \bowtie K \bowtie_{P_2.pid=K.pid2} P_2) \\ F & = & \pi_{P_1.*,K.*,P_2.*}(E \bowtie_{P_2.birthyear>P_3.birthyear \wedge K_3.pid1=P_1.pid} (P_3 \bowtie_{P_3.pid=K_3.pid2} K_3)) \end{array}$$

Then the RA expression for the query becomes

$$\pi_{P_1.pid,P_2.pid}(E - F)$$

(b) Using the optimization rewrite rules in the lectures, including those that rely on constraints, optimize this RA expression. Specify this optimized RA expression in your .pdf file.

Solution:

Solution:

```
with person as (select pid, birthyear from person),
     p1_Knows_p2 as (select p1.pid as pid1, p2.pid as pid2, p2.birthyear
                     from   person p1
                            join knows k on p1.pid = k.pid1
                            join person p2 on p2.pid = k.pid2)

select pid1, pid2
from (select p1_Knows_p2.*
      from   p1_Knows_p2
      except
      select p1_Knows_p2.*
      from   p1_Knows_p2
             join (select distinct birthyear, pid1
                   from   person p3 join knows k3 on p3.pid = k3.pid2) pk3
             on (p1_Knows_p2.birthyear > pk3.birthyear and pk3.pid1 = p1_Knows_p2.pid1)) q
order by 1,2;
```

Consider the following expressions:

$$
\begin{array}{rcl}
P & = & \pi_{pid,birthyear}(Person) \\
p1\_Knows\_p2 & = & \pi_{P_1.pid,P_2.pid,P_2.birthyear}(P_1 \bowtie_{P_1.pid=K.pid1} K \bowtie_{P_2.pid=K.pid2} P_2) \\
E & = & \pi_{birthyear,pid1}(P_3 \bowtie_{P_3.pid=K_3.pid2} K_3) \\
F & = & p1\_Knows\_p2 \bowtie_{p1\_Knows\_p2.birthyear > P_3.birthyear \wedge K_3.pid1 = p1\_Knows\_p2.pid1} (E)
\end{array}
$$

Then the optimized RA expression for the query becomes

$$\pi_{pid1,pid2}(p1\_Knows\_p2 - \pi_{p1\_Knows\_p2.*}(F))$$

6. Find the pid of each person who knows some person who has the 'Programming' and the 'Network' skills.

```
select p.pid
from   person p
where  exists (select 1
               from   person p1
               where  p1.pid in (select ps.pid from personSkill ps
                                 where  ps.skill = 'Programming'
                                 intersect
                                 select ps.pid from personSkill ps
                                 where  ps.skill = 'Databases') and
                      (p.pid, p1.pid) in (select k.pid1, k.pid2 from knows k));
```

(a) Using the translation algorithm presented in the lectures, translate this SQL into and equivalent RA expression formulated in the standard RA syntax. Specify this RA expression in your .pdf file.

Solution:

```
with
      personSkillProgramming as (select *
                                 from   personSkill
                                 where  skill = 'Programming'),
      personSkillDatabases as (select *
                               from   personSkill
                               where  skill = 'Databases')
select distinct pid
from (select p.*,
             p1.pid as pid1, p1.name, p1.city, p1.birthyear
      from   person p cross join
             person p1 join personSkillProgramming ps on (p1.pid = ps.pid)
      intersect
      select p.*, p1.*
      from   person p cross join
             person p1 join personSkillDatabases ps on (p1.pid = ps.pid)
      intersect
      select p.*, p1.*
```

```
      from   person p
             join knows k on (p.pid = k.pid1)
             join person p1 on (p1.pid = k.pid2)) q
order by 1;
```

Consider the expressions

$$
\begin{aligned}
pSkillProgramming &= \pi_{pSkill.*}(\sigma_{skill=\texttt{Programming}}(pSkill)) \\
pSkillDatabases &= \pi_{pSkill.*}(\sigma_{skill=\texttt{Databases}}(pSkill)) \\
E &= \pi_{P.*,P_1.*}(P \times (P_1 \bowtie_{P_1.pid=pSkillProgramming.pid} pSkillProgramming)) \\
F &= \pi_{P.*,P_1.*}(P \times (P_1 \bowtie_{P_1.pid=pSkillDatabases.pid} pSkillDatabases)) \\
G &= \pi_{P.*,P_1.*}(P \bowtie_{P.pid=K.pid1} K \bowtie_{P_1.pid=K.pid2} P_1)
\end{aligned}
$$

Then the RA expression for the query is

$$\pi_{P.pid}(E \cap F \cap G).$$

(b) Using the optimization rewrite rules in the lectures, including those that rely on constraints, optimize this RA expression. Specify this optimized RA expression in your .pdf file.

Solution:

```
select distinct pid1
from ( (select pid
        from   personSkill where skill = 'Programming'
        intersect
        select pid
        from   personSkill where skill = 'Databases') p
        join   knows on pid = pid2 ) k
order by 1;
```

Then the optimized RA expression is

$$\pi_{pid1}((\pi_{pid}(\sigma_{skill=\texttt{Programming}}(pSkill)) \cap \pi_{pi}(\sigma_{skill=\texttt{Databases}}(pSkill))) \bowtie_{pid=pid2} K).$$

3 Experiments to Test the Effectiveness of Query Optimization

In the following problems, you will conduct experiments to gain insight into whether or not query optimization can be effective. In other words, can it be determined experimentally if optimizing an SQL or an RA expression improves the time (and space) complexity of query evaluation?

You will need to use the PostgreSQL system to do you experiments. Recall that in SQL you can specify RA expression in a way that mimics it faithfully.

As part of the experiment, you might notice that PostgreSQL's query optimizer does not fully exploit all the optimization that is possible as discussed in Lecture 14.

In the following problems you will need to generate artificial data of increasing size and measure the time of evaluating non-optimized and optimized queries. The size of this data can be in the ten or hundreds of thousands of tuples. This is necessary because on very small data is it is not possible to gain sufficient insights into the quality (or lack of quality) of optimization.

Consider a binary relation `R(a int, b int)`. You can think of this relation as a graph, wherein each pair (a, b) represents and edge from a to b. (We work with directed graph. In other words edges (a, b) and (b, a) represent two different edges.) It is possible that `R` contains self-loops, i.e., edges of the form (a, a). Besides the relation R we will also use a unary relation `S(b int)`.

Along with this assignment, I have provided the code of two functions

`makerandomR(m integer, n integer, l integer)`

and

`makerandomS(n int, l int)`

```
create or replace function makerandomR(m integer, n integer, l integer)
returns void as
$$
declare  i integer; j integer;
begin
    drop table if exists Ra; drop table if exists Rb;
    drop table if exists R;
    create table Ra(a int); create table Rb(b int);
    create table R(a int, b int);

    for i in 1..m loop insert into Ra values(i); end loop;
    for j in 1..n loop insert into Rb values(j); end loop;
    insert into R select * from Ra a, Rb b order by random() limit(l);
end;
$$ LANGUAGE plpgsql;
```

```
create or replace function makerandomS(n integer, l integer)
returns void as
$$
declare i integer;
begin
    drop table if exists Sb;
    drop table if exists S;
    create table Sb(b int);
    create table S(b int);

    for i in 1..n loop insert into Sb values(i); end loop;
    insert into S select * from Sb order by random() limit (l);
end;
$$ LANGUAGE plpgsql;
```

When you run

`makerandomR(m,n,l);`

for some values m, n, and k, this function will generate a random relation instance for R with l tuples that is subset of $[1, m] \times [1, n]$. For example,

`makerandomR(3,3,4);`

might generate the relation instance

R

a	b
2	1
3	3
2	3
3	1

But, when you call

`makerandomR(3,3,4)`

again, it may now generate a different random relation such as

R

a	b
1	2
2	3
3	1
1	1

Notice that when you call

```
makerandomR(1000,1000,1000000)
```

it will make the entire relation $[1, 1000] \times [1, 1000]$ consisting of one million tuples.

The function `makerandomS(n, l)` will generate a random set of size l that is a subset of $[1, n]$.

Now consider the following simple query Q_1:

```
select distinct r1.a
from   R r1, R r2
where  r1.b = r2.a;
```

This query can be translated and optimized to the query Q_2:

```
select distinct r1.a
from   R r1 natural join (select distinct r2.a as b from R r2) r2;
```

Image that you have created a relation `R` using the function `makerandomR`. Then when you execute in PostgreSQL the following

```
explain analyze
select distinct r1.a
from   R r1, R r2
where  r1.b = r2.a;
```

the system will return its execution plan as well as the execution time to evaluate Q_1 measured in ms.

And, when you execute in PostgeSQL the following

```
select distinct r1.a
from   R r1 natural join (select distinct r2.a as b from R r2) r2;
```

the system will return its execution plan as well as the execution time to evaluate Q_2 measured in ms.

This permits us to compare the performance of the non-optimized query Q_1 with the optimized Q_2 for various-sized relations R.

Here are some of these comparisons for various different random relations `R`.

makerandomR	Q_1 (in ms)	Q_2 (in ms)
(100,100,1000)	4.9	1.5
(500,500,25000)	320.9	28.2
(1000,1000,100000)	2648.3	76.1
(2000,2000,400000)	23143.4	322.0
(5000,5000,2500000)	$--$	1985.8

The "$--$" symbol indicates that I had to stop the experiment because it was taken too long. (All the experiments where done on a MacBook pro.)

Notice the significant difference between the execution times of the non-optimized query Q_1 and the optimized query Q_2. So clearly, optimization works on query Q_1.

If you look at the query plan of PostgreSQL for Q_1, you will notice that it does a double nested loop and it therefore is $O(|R|^2)$ whereas for query Q_2 it runs in $O(|R|)$. Clearly, optimization has helped significantly.[1]

7. Now consider query Q_3:

```
select distinct r1.a
from   R r1, R r2, R r3
where  r1.b = r2.a and r2.b = r3.a;
```

(a) Translate and optimize this query and call it Q_4. Then write Q_4 as an SQL query with RA operations just as was done for query Q_2.

Solution:

```
select distinct a
from   R natural join (select distinct a as b
                       from   R natural join (select distinct a as b
                                              from   R) q) q order by 1;
```

In RA notation

$$\pi_a(R \bowtie_{R.b=a} \pi_a(R \bowtie_{R.b=a} \pi_a(R))).$$

(b) Compare queries Q_3 and Q_4 in a similar way as we did for Q_1 and Q_2.

You should experiment with different sizes for `R`. Incidentally, these relations do not need to use the same m,n, and l parameters as those shown in the above table for Q_1 and Q_2.

Solution:

Here are some experiments:

R	Q_3 (in ms)	Q_4 (in ms)
makerandomR(100,100,1000)	30	2
makerandomR(200,200,4000)	487	5
makerandomR(200,200,10000)	14253	22
makerandomR(500,500,50000)	248900	111

(c) What conclusions can you draw from the results of these experiments?

Solution:

The PostgreSQL engine runs Q_3 using a three-level nested loop over `R` and is thus $O(|R|^3)$.

[1] It is actually really surprising that the PostgreSQL system did not optimize query Q_1 any better.

The PostgreSQL engine runs Q_4 using hash-joins and therefore run in linear time $O(|R|)$.

Clearly, optimization in this case speeds up the query by two orders of magnitude in $|R|$.

8. Now consider query Q_5 which is an implementation of the ONLY set semijoin between R and S. (See the lecture on set semijoins for more information.)

   (Incidentally, if you look at the code for makerandomR you will see a relation Ra that provides the domain of all a values. You will need to use this relation in the queries. Analogously, in the code for makerandomS you will see the relation Sb that contains the domain of all b values.)

   In SQL, Q_5 can be expressed as follows:

```
select ra.a
from   Ra ra
where  not exists (select r.b
                   from   R r
                   where  r.a = ra.a and
                          r.b not in (select s.b from S s));
```

   (a) Translate and optimize this query and call it Q_6. Then write Q_6 as an SQL query with RA operations just as was done for Q_2 above.

   Solution:

```
select distinct a
from   Ra
except
select a
from   (select a, b
        from  R
        except
        select a, b
        from   R natural join S) q;
```

   (b) Compare queries Q_5 and Q_6 in a similar way as we did for Q_1 and Q_2.

   You should experiment with different sizes for R and S. (Vary the size of S from smaller to larger.) Also use the same value for the parameter n in makerandomR(m, n, l) and makerandomS(n, l) so that the maximum number of b values in R and S are the same.

   Solution:

R	S	Q_5 (in ms)	Q_6 (in ms)
makerandomR(1000,10,1000)	makerandomS(10,4)	1	6
makerandomR(10000,20,10000)	makerandomS(20,10)	8	31
makerandomR(10000,20,50000)	makerandomS(20,10)	23	102

(c) What conclusions can you draw from the results of these experiments?

Solution: For both queries, the performance is quite good. Optimization has not improved the performance. Looking at the PostgreSQL query plan, both queries are implemented with semijoin type operations and run in linear time $O(|R| + |S| + |Ra|)$.

9. Now consider query Q_7 which is an implementation of the ALL set semijoin between R and S. (See the lecture on set semijoins for more information.)

   In SQL, Q_7 can be expressed as follows:

```
select ra.a
from   Ra ra
where  not exists (select s.b
                   from   S s
                   where  s.b not in (select r.b
                                      from   R r
                                      where  r.a = ra.a));
```

(a) Translate and optimize this query and call it Q_8. Then write Q_8 as an SQL query with RA operations just as was done for query Q_2 above.

Solution:

```
select distinct a
from   Ra
except
select a
from   (select a, b
        from   Ra cross join S
        except
        select a, b
        from   R) q;
```

(b) Compare queries Q_7 and Q_8 in a similar way as we did for Q_1 and Q_2.

You should experiment with different sizes for R and S. (Vary the size of S from smaller to larger.) Also use the same value for the parameter n in makerandomR(m,n,l) and makerandomS(n,l) so that the maximum number of b values in R and S are the same.

Solution:

R	S	Q_7 (in ms)	Q_8 (in ms)
makerandomR(1000,10,1000)	makerandomS(10,4)	202	10
makerandomR(10000,20,10000)	makerandomS(20,10)	7264	209
makerandomR(10000,20,50000)	makerandomS(20,10)	39925	409

(c) What conclusions can you draw from the results of these experiments?

Solution: Clearly optimization has helped. Looking at the PostgreSQL query plan for Q_7 we observe a triple nested loop resulting in a time complexity of $O(|Ra| * |S| * |R|)$. On the other hand, the complexity of Q_8 is $O(|Ra| * |S| + |R|)$. This is an order of magnitude better in $|R|$

(d) Furthermore, what conclusion can you draw when you compare you experiment with those for the ONLY set semijoin in problem 8?

Solution: Given the time complexities, we expect that the ONLY query performs better than the ALL query. Indeed, for the optimized versions, the ONLY query runs in $O(|R| + |S| + |Ra|)$ whereas the ALL query runs in time $O(|S| * |Ra| + |R|)$.