B561 Advanced Database Concepts
Assignment 3
Fall 2021

Dirk Van Gucht

This assignment relies on the lectures

- SQL Part 1 and SQL Part 2 (Pure SQL);
- Relational Algebra (RA);
- joins and semijoins;
- **Translating Pure SQL queries into RA expressions**; and
- **Query optimization**

with particular focus on the last two lectures.

To turn in your assignment, you will need to upload to Canvas a single file with name `assignment3.sql` which contains the necessary SQL statements that solve the problems in this assignment. The `assignment3.sql` file must be so that the AI's can run it in their PostgreSQL environment. You should use the `Assignment-Script-2021-Fall-Assignment3.sql` file to construct the `assignment3.sql` file. (note that the data to be used for this assignment is included in this file.) In addition, you will need to upload a separate `assignment3.txt` file that contains the results of running your queries. Finally, you need to upload a file `assignment3.pdf` that contains the solutions to the problems that require it.

The problems that need to be included in the `assignment3.sql` are marked with a blue bullet •. The problems that need to be included in the `assignment3.pdf` are marked with a red bullet •. (You should aim to use Latex to construct your .pdf file.)

Database schema and instances

For the problems in this assignment we will use the following database schema:[1]

```
Person(pid, pname, city)
Company(cname, headquarter)
Skill(skill)
worksFor(pid, cname, salary)
companyLocation(cname, city)
personSkill(pid, skill)
hasManager(eid, mid)
Knows(pid1, pid2)
```

In this database we maintain a set of persons (`Person`), a set of companies (`Company`), and a set of (job) skills (`Skill`). The `pname` attribute in `Person` is the name of the person. The `city` attribute in `Person` specifies the city in which the person lives. The `cname` attribute in `Company` is the name of the company. The `headquarter` attribute in `Company` is the name of the city wherein the company has its headquarter. The `skill` attribute in `Skill` is the name of a (job) skill.

A person can work for at most one company. This information is maintained in the `worksFor` relation. (We permit that a person does not work for any company.) The `salary` attribute in `worksFor` specifies the salary made by the person.

The `city` attribute in `companyLocation` indicates a city in which the company is located. (Companies may be located in multiple cities.)

A person can have multiple job skills. This information is maintained in the `personSkill` relation. A job skill can be the job skill of multiple persons. (A person may not have any job skills, and a job skill may have no persons with that skill.)

A pair (e, m) in `hasManager` indicates that person e has person m as one of his or her managers. We permit that an employee has multiple managers and that a manager may manage multiple employees. (It is possible that an employee has no manager and that an employee is

[1] The primary key, which may consist of one or more attributes, of each of these relations is underlined.

not a manager.) We further require that an employee and his or her managers must work for the same company.

The relation `Knows` maintains a set of pairs (p_1, p_2) where p_1 and p_2 are pids of persons. The pair (p_1, p_2) indicates that the person with pid p_1 knows the person with pid p_2. We do not assume that the relation `Knows` is symmetric: it is possible that (p_1, p_2) is in the relation but that (p_2, p_1) is not.

The domain for the attributes `pid`, `pid1`, `pid2`, `salary`, `eid`, and `mid` is `integer`. The domain for all other attributes is `text`.

We assume the following foreign key constraints:

- `pid` is a foreign key in `worksFor` referencing the primary key `pid` in `Person`;
- `cname` is a foreign key in `worksFor` referencing the primary key `cname` in `Company`;
- `cname` is a foreign key in `companyLocation` referencing the primary key `cname` in `Company`;
- `pid` is a foreign key in `personSkill` referencing the primary key `pid` in `Person`;
- `skill` is a foreign key in `personSkill` referencing the primary key `skill` in `Skill`;
- `eid` is a foreign key in `hasManager` referencing the primary key `pid` in `Person`;
- `mid` is a foreign key in `hasManager` referencing the primary key `pid` in `Person`;
- `pid1` is a foreign key in `Knows` referencing the primary key `pid` in `Person`; and
- `pid2` is a foreign key in `Knows` referencing the primary key `pid` in `Person`

Pure SQL and RA SQL

In this assignemt, we distinguish between Pure SQL and RA SQL. Below we list the **only** features that are allowed in Pure SQL and in RA SQL.

In particular notice that

- `join`, `NATURAL join`, and `CROSS join` are **not** allowed in Pure SQL.
- The predicates [`not`] `IN`, `SOME`, `ALL`, [`not`] `exists` are **not** allowed in RA SQL.

The only features allowed in Pure SQL

`select ... from ... where`
`WITH ...`
`union, intersect, except` operations
`exists` and `not exists` predicates
`IN` and `not IN` predicates
`ALL` and `SOME` predicates
`VIEW`s that can only use the above RA SQL features

The only features allowed in RA SQL

`select ... from ... where`
`WITH ...`
`union, intersect, except` operations
`join ... ON ...`, `natural join`, and `CROSS join` operations
`VIEW`s that can only use the above RA SQL features
commas in the `from` clause are **not** allowed

1 Theoretical problems related to query translation and optimization

1. Consider two RA expressions E_1 and E_2 over the same schema. Furthermore, consider an RA expression F with a schema that is not necessarily the same as that of E_1 and E_2.

   Consider the following `if-then-else` query:

   $$\begin{array}{lll} \texttt{if } F = \emptyset & \texttt{then} & \texttt{return } E_1 \\ & \texttt{else} & \texttt{return } E_2 \end{array}$$

   So this query evaluates to the expression E_1 if $F = \emptyset$ and to the expression E_2 if $F \neq \emptyset$.

   We can formulate this query in SQL as follows[2]:

```
select e1.*
from   E1 e1
where  not exists (select distinct row() from F)
union
select e2.*
from   E2 e1
where  exists (select distinct row() from F);
```

   Incidentally, the query

```
select distinct row() from F
```

   returns the empty set if $F = \emptyset$ and returns the tuple `()` if $F \neq \emptyset$.[3] In RA, this query can be written as

   $$\pi_{()}(F).$$

   I.e., the projection of F on an empty list of attributes.

   • In function of E_1, E_2, and F, write an RA expression in standard notation that expresses this `if-then-else` query.[4]

   Solution:

   $$(E_1 - E_1 \times \pi_{()}(F)) \cup (E_2 \times \pi_{()}(F)).$$

[2] In this SQL query `E1`, `E2`, and `F` denote SQL queries corresponding to the RA expressions E_1, E_2, and F, respectively.

[3] The tuple `()` is often times referred to as the *empty tuple*, i.e., the tuple without components. It is akin to the empty string ϵ in the theory of formal languages. I.e., the string wihout alphabet characters.

[4] Hint: consider using the Pure SQL to RA SQL translation algorithm.

2. Let `R(x)` be a unary relation that can store a set of integers R. Consider the following boolean SQL query:

```
select not exists(select 1
                  from   R r1, R 2
                  where  r1.x <> r2.x) as fewerThanTwo;
```

This boolean query returns the constant "`true`" if R has fewer than two elements and returns the constant "`false`" otherwise.

• Using the insights you gained from Problem 1, write an RA expression in standard notation that expresses the above boolean SQL query.[5]

Solution

We can consider the `if-then-else` query

$$\texttt{if } (R_1 \bowtie_{R_1.x \neq R_2.x} R_2) = \emptyset \quad \begin{array}{ll} \texttt{then} & \texttt{return}\ (\mathrm{A} : \mathrm{true}) \\ \texttt{else} & \texttt{return}\ (\mathrm{A} : \mathrm{false}) \end{array}$$

Consequently, an RA expression for the boolean query is

$$((\mathrm{A} : \mathrm{true}) - (\mathrm{A} : \mathrm{true}) \times \pi_{()}(R_1 \bowtie_{R_1.x \neq R_2.x} R_2)) \cup ((\mathrm{A} : \mathrm{false}) \times \pi_{()}(R_1 \bowtie_{R_1.x \neq R_2.x} R_2)).$$

3. In the translation algorithm from Pure SQL to RA we tacitly assumed that the argument of each set predicate was a (possibly parameterized) Pure SQL query that did not use a `union`, `intersect`, nor an `except` operation.

   In this problem, you are asked to extend the translation algorithm from Pure SQL to RA such that the set predicates [`not`] `exists` are eliminated that have as an argument a Pure SQL query (possibly with parameters) that uses a `union`, `intersect`, or `except` operation.

   More specifically, consider the following types of queries using the [`not`] `exists` set predicate.

[5]Hint: recall that, in general, a constant value "`a`" can be represented in RA by an expression of the form `(A: a)`. (Here, `A` is some arbitrary attribute name.) Furthermore, recall that we can express `(A: a)` in SQL as "`select a as A`". Thus RA expressions for the constants "`true`" and "`false`" can be the expressions `(A: true)` and `(A: false)`, respectively.

```
select L(r1,...,rn)
from   R1 r1, ..., Rn rn
where  C1(r1,...,rn) and
              [not] exists (select L(s1,...,sm)
                            from   S1 s1,..., S1 sm
                            where  C2(s1,...,sm,r1,...,rn)
                            [union | intersect | except]
                            select distinct 1
                            from   T1 t1, ..., Tk tk
                            where  C3(t1,...,tk,r1,...,rn))
```

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

To get practice, first consider the following special case where $n = 1$, $m = 1$, and $k = 1$. I.e., the following case: [6]

```
select L(r)
from   R r
where  C1(r) [not] and exists (select distinct 1
                               from   S s
                               where  C2(s,r)
                               [union|intersect|except]
                               select distinct 1
                               from   T t
                               where  C3(t,r))
```

Solution (`exists` case)

```
select distinct L(r)
from   (select r.* from R r where C1(r)) r
        natural join
```

[6] Once you can handle this case, the general case is a similar.

```
(select r.*, s.*
 from   R r join S s ON C2(s,r)
 [union|intersect|except]
 select r.*, t.*
 from   R r join T t1 ON C3(t,r)) q
```

where the `natural join` is a semijoin.

I.e., in standard RA notation

$$\pi_{L(R_1)}(\sigma_{C_1}(R) \ltimes (R \bowtie_{C_2(S,R)} S\,[\cup \mid \cap \mid -]\,R \bowtie_{C_3(T,R)} T)).$$

Solution (`not exists` case)

```
select distinct L(r)
from (select r.*
      from   R r
      where  C1(r)
      except
      select r.*
      from   (select r.* from R r where C1) r
              natural join
              (select r.*, s.*
               from   R r join S s ON C2(s,r)
               [union|intersect|except]
               select r.*, t.*
               from   R r join T t ON C3(t,r)) q) p
```

where the `natural join` is a semijoin.

I.e., in standard RA notation

$$\pi_{L(R)}(\sigma_{C_1}(R) - \sigma_{C_1}(R) \ltimes (R \bowtie_{C_2(S,R)} S\,[\cup \mid \cap \mid -]\,R \bowtie_{C_3(T,R)} T)).$$

Solution (General `exists` case)

$$\pi_{L(R_1,\ldots,R_k)}(\sigma_{C_1}(\mathbf{R}) \ltimes (\mathbf{R} \bowtie_{C_2(\mathbf{S},\mathbf{R})} \mathbf{S}\,[\cup \mid \cap \mid -]\,\mathbf{R} \bowtie_{C_3(\mathbf{T},\mathbf{R})} \mathbf{T}))$$

where

$$\begin{array}{rcl}\mathbf{R} & = & R_1 \times \cdots \times R_k \\ \mathbf{S} & = & S_1 \times \cdots \times S_m \\ \mathbf{T} & = & T_1 \times \cdots \times T_n\end{array}$$

Solution (General `not exists` case)

$$\pi_{L(R_1,\ldots,R_k)}(\sigma_{C_1}(\mathbf{R}) - \sigma_{C_1}(\mathbf{R}) \ltimes (\mathbf{R} \bowtie_{C_2(\mathbf{S},\mathbf{R})} \mathbf{S}\, [\cup \mid \cap \mid -]\, \mathbf{R} \bowtie_{C_3(\mathbf{T},\mathbf{R})} \mathbf{T}))$$

where

$$\begin{array}{rcl}\mathbf{R} & = & R_1 \times \cdots \times R_k \\ \mathbf{S} & = & S_1 \times \cdots \times S_m \\ \mathbf{T} & = & T_1 \times \cdots \times T_n\end{array}$$

Remark: In the case of union, we can do better

$$\pi_{L(R_1,\ldots,R_k)}((\sigma_{C_1}(\mathbf{R}) - \sigma_{C_1}(\mathbf{R}) \bowtie_{C_2(\mathbf{S},\mathbf{R})} \mathbf{S}) \cap (\sigma_{C_1}(\mathbf{R}) - \sigma_{C_1}(\mathbf{R}) \bowtie_{C_3(\mathbf{T},\mathbf{R})} \mathbf{T}))$$

4. • Let R be a relation with schema (a, b, c) and let S be a relation with schema (d, e).

   Prove, from first principles, the correctness of the following rewrite rule:

$$\pi_{a,d}(R \bowtie_{c=d} S) = \pi_{a,d}(\pi_{a,c}(R) \bowtie_{c=d} \pi_d(S))$$

Solution: In other words, you have to prove that

$$\pi_{a,d}(R \bowtie_{c=d} S) \subseteq \pi_{a,d}(\pi_{a,c}(R) \bowtie_{c=d} \pi_d(S))$$

and

$$\pi_{a,d}(R \bowtie_{c=d} S) \supseteq \pi_{a,d}(\pi_{a,c} \bowtie_{c=d} \pi_d(S))$$

Solution: We translate the RA expression in Predicate Logic and we use logical equivalences

$$\begin{aligned}
\pi_{a,d}(R \bowtie_{c=d} S) &= \{(a,d) | \exists b \exists c \exists e (R(a,b,c) \wedge c = d \wedge S(d,e))\} \\
&= \{(a,d) | \exists b \exists c (R(a,b,c) \wedge c = d \wedge \exists e S(d,e))\} \\
&= \{(a,d) | \exists c ((\exists b R(a,b,c)) \wedge c = d \wedge (\exists e S(d,e)))\} \\
&= \{(a,d) | \exists c ((a,c) \in \pi_{a,c}(R) \wedge c = d \wedge (d \in \pi_d(S)))\} \\
&= \{(a,d) | \exists c ((a,c,d) \in \pi_{a,c}(R) \bowtie_{c=d} \pi_d(S))\} \\
&= \pi_{a,d}(\pi_{a,c}(R) \bowtie_{c=d} \pi_d(S))
\end{aligned}$$

5. • Consider the same rewrite rule

$$\pi_{a,d}(R \bowtie_{c=d} S) = \pi_{a,d}(\pi_{a,c}(R) \bowtie_{c=d} \pi_d(S))$$

as in problem 4.

Furthermore assume that S has primary key d and that R has foreign key c referencing this primary key in S.

How can you simplify this rewrite rule? Argue why this rewrite rule is correct.

Solution: We claim that in the presence of the foreign key constraint

$$\pi_{a,d}(R \bowtie_{c=d} S) = \pi_{a,c}(R).$$

We can express the foreign key constraint as follows:

$$\forall a \forall b \forall c (R(a,b,c) \rightarrow \exists e S(c,e)) \qquad (1)$$

We already proved that

$$\pi_{a,d}(R \bowtie_{c=d} S) = \pi_{a,d}(\pi_{a,c}(R) \bowtie_{c=d} \pi_d(S)).$$

We claim that

$$\pi_{a,d}(\pi_{a,c}(R) \bowtie_{c=d} \pi_d(S)) = \pi_{a,c}(R)$$

$$\begin{aligned}
\pi_{a,c}(R) &= \{(a,c) | \exists b (R(a,b,c)\} \\
&= \{(a,d) | \exists c \exists b (R(a,b,c) \wedge c = d)\} \\
&= \{(a,d) | \exists c (\exists b (R(a,b,c) \wedge c = d \wedge \exists e S(c,e)))\} \quad \text{by (1)} \\
&= \{(a,d) | \exists c (\exists b R(a,b,c) \wedge c = d \wedge \exists e S(d,e)))\} \quad \text{since } c = d \\
&= \{(a,d) | \exists c ((a,c,d) \in \pi_{a,c}(R) \bowtie_{c=d} \pi_d(S))\} \\
&= \pi_{a,d}(\pi_{a,c}(R) \bowtie_{c=d} \pi_d(S))
\end{aligned}$$

2 Translating Pure SQL queries to RA expressions and optimized RA expressions

In this section, you are asked to *translate* Pure SQL queries into RA SQL queries as well as in standard RA expressions using the *translation algorithm given in class*. You are required to show the intermediate steps that you took during the translation. After the translation, you are asked to *optimize* the resulting RA expressions.

You can use the following letters, or indexed letters, to denote relation names in RA expressions:

$P, P_1, P_2, \cdots$	`Person`
$C, C_1, C_2, \cdots$	`Company`
$S, S_1, S_2, \cdots$	`Skill`
$W, W_1, W_2, \cdots$	`worksFor`
$cL, cL_1, cL_2, \cdots$	`companyLocation`
$pS, pS_1, pS_2, \cdots$	`personSkill`
$M, M_1, M_2, \cdots$	`hasManager`
$K, K_1, K_2, \cdots$	`Knows`

We illustrate what is expected using an example.

Example 1 *Consider the query* "Find each (p, c) pair where p is the pid of a person who works for a company c located in Bloomington and whose salary is the lowest among the salaries of persons who work for that company.

A possible formulation of this query in Pure SQL is

```
select w.pid, w.cname
from   worksfor w
where  w.cname in  (select cl.cname
                    from   companyLocation cl
                    where  cl.city = 'Bloomington') and
       w.salary <= ALL (select w1.salary
                        from   worksfor w1
                        where  w1.cname = w.cname) order by 1,2;
```

which is translated to[7]

```
select q.pid, q.cname
from   (select w.*
        from   worksfor w
        where  w.cname in (select cl.cname
                           from   companyLocation cl
                           where  cl.city = 'Bloomington')
        intersect
```

[7] Translation of '`and`' in the '`where`' clause.

```
        select w.*
        from   worksfor w
        where  w.salary <= ALL (select w1.salary
                                from   worksfor w1
                                where  w1.cname = w.cname)) q;
```

which is translated to[8]

```
select q.pid, q.cname
from   (select w.*
        from   worksfor w, companyLocation cl
        where  w.cname = cl.cname and cl.city = 'Bloomington'
        intersect
        (select w.*
         from   worksfor w
         except
         select w.*
         from   worksfor w, worksfor w1
         where  w.salary > w1.salary and w1.cname = w.cname)) q;
```

which is translated to[9]

```
select q.pid, q.cname
from   (select w.*
        from   worksfor w, (select cl.* from companyLocation cl  where cl.city = 'Bloomington') cl
        where  w.cname = cl.cname
        intersect
        (select w.*
         from   worksfor w
         except
         select w.*
         from   worksfor w, worksfor w1
         where  w.salary > w1.salary and w1.cname = w.cname)) q;
```

which is translated to[10]

```
select q.pid, q.cname
from   (select w.*
from   worksfor w
       natural join (select cl.* from companyLocation cl  where cl.city = 'Bloomington') cl
       intersect
       (select w.*
        from   worksfor w
        except
        select w.*
        from   worksfor w join worksfor w1 on (w.salary > w1.salary and w1.cname = w.cname))) q;
```

This RA SQL query can be formulated as an RA expression in standard notation as follows:

$$\pi_{W.pid,W.cname}(\mathbf{E} \cap (W - \mathbf{F}))$$

where

$$\mathbf{E} = \pi_{W.*}(W \bowtie \sigma_{city=\mathbf{Bloomington}}(cL))$$

and

$$\mathbf{F} = \pi_{W.*}(W \bowtie_{W.salary>W_1.salary \wedge W_1.cname=W.cname} W_1).$$

We can now commence the optimization.

---

[8] Translation of '`in`' and '`<= ALL`'.

[9] Move 'constant' condition.

[10] Introduction of natural join and join.

Step 1 *Observe the expression* $\mathbf{E} \cap (W - \mathbf{F})$*. This expression is equivalent with* $(\mathbf{E} \cap W) - \mathbf{F}$*. Then observe that, in this case,* $\mathbf{E} \subseteq W$*. Therefore* $\mathbf{E} \cap W = \mathbf{E}$*, and therefore* $\mathbf{E} \cap (W - \mathbf{F})$ *can be replaced by* $\mathbf{E} - \mathbf{F}$*. So the expression for the query becomes*

$$\pi_{W.pid,W.cname}(\mathbf{E} - \mathbf{F}).$$

Step 2 *We now concentrate on the expression*

$$\mathbf{E} = \pi_{W.*}(W \bowtie \sigma_{city=\mathbf{Bloomington}}(cL)).$$

We can push the projection over the join and get

$$\pi_{W.*}(W \bowtie \pi_{cname}(\sigma_{city=\mathbf{Bloomington}}(cL))).$$

Which further simplifies to

$$W \ltimes \sigma_{city=\mathbf{Bloomington}}(cL).$$

We will call this expression $\mathbf{E}^{opt}$*.*

Step 3 *We now concentrate on the expression*

$$\mathbf{F} = \pi_{W.*}(W \bowtie_{W.salary > W_1.salary \wedge W_1.cname = W.cname} W_1).$$

We can push the projection over the join and get the expression

$$\pi_{W.*}(W \bowtie_{W.salary > W_1.salary \wedge W_1.cname = W.cname} \pi_{W_1.cname, W_1.salary}(W_1)).$$

We will call this expression $\mathbf{F}^{opt}$*.*

Therefore, the fully optimized RA expression is

$$\pi_{W.pid,W.cname}(\mathbf{E}^{opt} - \mathbf{F}^{opt}).$$

I.e.,

$$\pi_{W.pid,W.cname}(W \ltimes \sigma_{city=\mathbf{Bloomington}}(cL) - \pi_{W.*}(W \bowtie_{W.salary > W_1.salary \wedge W_1.cname = W.cname} \pi_{W_1.cname, W_1.salary}(W_1))).$$

We now turn to the problems in this section.

6. Consider the query "*Find the cname and headquarter of each company that employs persons who earns less than 55000 and who do not live in Bloomington.*"

   A possible way to write this query in Pure SQL is

```
select c.cname, c.headquarter
from   company c
where  c.cname in (select w.cname
                   from   worksfor w
                   where  w.salary < 55000 and
                          w.pid = SOME (select p.pid
                                        from   person p
                                        where  p.city <> 'Bloomington'));
```

(a) • Using the Pure SQL to RA SQL translation algorithm, translate this Pure SQL query to an equivalent RA SQL query. Show the translation steps you used to obtain your solution.

(b) • Optimize this RA SQL query and provide the optimized expression in standard RA notation. Specify at least three conceptually different rewrite rules that you used during the optimization.

Solution:

$$\pi_{C.*}(C \ltimes \pi_{cname}(\pi_{cname,pid}(\sigma_{salary<55000} W) \ltimes \pi_{pid}(\sigma_{city \neq \textbf{Bloomington}} P))).$$

- Push (constant) selections down over joins and projections
- Push projections down over joins
- Introduce semi-joins.

7. Consider the query "*Find the pid of each person who has all-but-one job skill.*"

A possible way to write this query in Pure SQL is

```
select p.pid
from   person p
where  exists (select 1
               from   skill s
               where  (p.pid, s.skill) not in (select ps.pid, ps.skill
                                                from   personSkill ps)) and
        not exists (select 1
                    from   skill s1, skill s2
                    where  s1.skill <> s2.skill and
                           (p.pid, s1.skill) not in (select ps.pid, ps.skill
                                                      from   personSkill ps) and
                           (p.pid, s2.skill) not in (select ps.pid, ps.skill
                                                      from   personSkill ps));
```

(a) • Using the Pure SQL to RA SQL translation algorithm, translate this Pure SQL query to an equivalent RA SQL query. Show the translation steps you used to obtain your solution.

(b) • Optimize this RA SQL query and provide the optimized expression in standard RA notation. Specify at least three conceptually different rewrite rules that you used during the optimization.

Solution

Let
$$P = \pi_{pid}(Person).$$

$\pi_{pid}((P\times S)-pS)-\pi_{pid}((P\times S_1 \bowtie_{S_1.skill\neq S_2.skill} S_2)-(pS\times S)-\pi_{pS.pid,S.skill,pS.skill}(pS\times S)).$

- Push projections down over joins
- Apply foreign key constraints
- Use succesive projections elimination
- Apply some set theoretical equalities.

8. Consider the query "*Find the pid and name of each person who works for a company located in Bloomington but who does not know any person who lives in Chicago.*"

A possible way to write this query in Pure SQL is

```
select p.pid, p.pname
from   person p
where  exists (select 1
               from   worksFor w, companyLocation c
               where  p.pid = w.pid and w.cname = c.cname and c.city = 'Bloomington') and
       p.pid not in (select k.pid1
                     from   knows k
                     where  exists (select 1
                                    from   person p
                                    where  k.pid2 = p.pid and  p.city = 'Chicago'));
```

(a) • Using the Pure SQL to RA SQL translation algorithm, translate this Pure SQL query to an equivalent RA SQL query. Show the translation steps you used to obtain your solution.

(b) • Optimize this RA SQL query and provide the optimized expression in standard RA notation. Specify at least three conceptually different rewrite rules that you used during the optimization.

$$E - F$$

where

$E = \pi_{pid,pname}(P \ltimes \pi_{pid}(\pi_{pid,cname}(W) \ltimes \pi_{cname}(\sigma_{city=\mathbf{Bloomington}}(cL))))$

and

$F = \pi_{pid,pname}(P \bowtie_{pid=pid1} \pi_{pid1}(K \bowtie_{pid2=pid} \pi_{pid}(\sigma_{city=\mathbf{Chicago}}(P))))$

- Push selections down over joins and projections
- Push projections down over joins
- Introduce semi-joins.

9. Consider the query "*Find the cname and headquarter of each company that (1) employs at least one person and (2) whose workers who make at most 70000 have both the programming and AI skills.*"
A possible way to write this query in Pure SQL is

```
select c.cname, c.headquarter
from   company c
where  exists (select 1 from worksfor w where w.cname = c.cname) and
       not exists (select 1
                   from   worksfor w
                   where  w.cname = c.cname and w.salary <= 70000 and
                          (w.pid not in (select ps.pid from personskill ps where skill = 'Programming') or
                           w.pid not in (select ps.pid from personskill ps where skill = 'AI')));
```

(a) • Using the Pure SQL to RA SQL translation algorithm, translate this Pure SQL query to an equivalent RA SQL query. Show the translation steps you used to obtain your solution.

(b) • Optimize this RA SQL query and provide the optimized expression in standard RA notation. Specify at least three conceptually different rewrite rules that you used during the optimization.

Solution Consider the following expressions

$$\begin{array}{rcl} E & = & C \bowtie_{C.cname=W.cname} \pi_{cname,pid}(\sigma_{salary \leq 70000}(W)) \\ W & = & \pi_{cname,pid}(W) \\ Programming & = & W \ltimes \pi_{pid}(\sigma_{skill=\mathbf{Programming}}(pS)) \\ AI & = & W \ltimes \pi_{pid}(\sigma_{skill=\mathbf{AI}}(pS)) \end{array}$$

Then, an optimized RA expression for the query is

$C \ltimes \pi_{cname}(W) - \pi_{cname,headquarter}(E - (E \ltimes (Programming \cap AI)))$.

- Push selections down over joins and projections
- Push projections down over joins
- Introduce semi-joins.
- Apply some logical and set-theoretical equivalences

10. Consider the following Pure SQL query.

```
select p.pid, exists (select 1
                      from   hasManager hm1, hasManager hm2
                      where  hm1.mid = p.pid and hm2.mid = p.pid and
                             hm1.eid <> hm2.eid)
from   Person p;
```

This query returns a pair (p, t) if p is the pid of a person who manages at least two persons and returns the pair (p, f) otherwise.[11]

(a) • Using the insights gained from Problem 2, translate this Pure SQL query to an equivalent RA SQL query.

(b) • Optimize this RA SQL query and provide the optimized expression in standard RA notation.

Solution:

Let

$$P = \pi_{pid}(Person)$$

and let

$$E = \pi_{hM_1.mid}(hM_1 \bowtie_{hM_1.mid = hM_2.mid \wedge hM_1.eid \neq hM_2.eid} hM_2)$$

Then the solution is

$$(E \times (\text{value} : \textbf{true})) \cup ((P - E) \times (\text{value} : \textbf{false})).$$

[11] `t` represent the boolean value `true` and `f` represents the boolean value `false`.