B561 Advanced Database Concepts
Assignment 3
Fall 2022
Solutions

Dirk Van Gucht

This assignment relies on the lectures

- SQL Part 1 and SQL Part 2 (Pure SQL);
- Tuple Relational Calculus;
- 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-2022-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.

The problems that need to be included in the `assignment3.sql` are marked with a blue bullet •.

There are also practice problems that you should not submit. They are marked with a red bullet •.

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:

   $$\texttt{if } F = \emptyset \texttt{ then return } E_1 \quad \texttt{else return } E_2$$

   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  true = all (select false from F)
union
select e2.*
from   E2 e2
where  true = some (select true from F);
```

   (a) Now for the problem. Write an RA expression in standard notation that expresses this `if-then-else` query.[3]
   Solution:

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

   In RA SQL,

```
(select e1.*
 from   E1 e1
 except
 select e1.*
 from   E1 e1 cross join (select row()
                          from   F) f)
 union
 select e2.*
```

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

```
from   E2 e2 cross join (select row()
                         from   F) f;
```

(b) Test you solution for

$$\begin{array}{lcl} E_1 & = & \{(1),(2)\} \\ E_1 & = & \{(3),(4)\} \\ F & & \emptyset \end{array}$$

(c) Test you solution for

$$\begin{array}{lcl} E_1 & = & \{(1),(2)\} \\ E_1 & = & \{(3),(4)\} \\ F & & \{(\texttt{'a'}),(\texttt{'b'}),(\texttt{'c'})\} \end{array}$$

2. • Let `F(x integer, y integer)` be relation that can store pairs of integers. Consider the following boolean SQL query:

```
select true = all (select p1 = p2
                   from   F p1, F p2
                   where  p1.x = p2.x)  "isaFunction";
```

This boolean query returns the constant "`true`" if F is a function and returns the constant "`false`" otherwise.

(a) Using the insights you gained from Problem 1, write an RA SQL query that expresses the above boolean SQL query.[4]

(b) Test your query for $F = \emptyset$.

(c) Test your query for $F = \{(1,10),(2,20)\}$.

(d) Test your query for $F = \{(1,10),(1,20),(2,20)\}$.

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

[4]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 formulated in RA SQL a '`select true as "A"`' and '`select false as "A"`', respectively.

Prove, from first principles[5], 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

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

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

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

4. • 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 3.

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

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

[5] In particular, do not use the rewrite rule of pushing projections over joins. Rather, use TRC to provide a proof.

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

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

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

5. • 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.

```
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
       (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

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

Solution (`not exists` case)

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

```
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

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

Solution (General `exists` case)

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

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)

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

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

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