B561 Advanced Database Concepts
Assignment 3
Fall 2022

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**
- **Aggregate functions**

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.
- Subquery expressions with [`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` subquery expressions
`IN` and `not IN` subquery expressions
`ALL` and `SOME` subquery expressions
`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

Full SQL

---

all the features of Pure SQL and RA SQL
user-defined functions
aggregate functions
`group ... by ...`
`having ...`

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 express this query in Pure SQL as follows

```
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 SQL query that expresses this `if-then-else` query.[2]

   (b) Test you solution for

   $$\begin{array}{lll} E_1 & = & \{(1),(2)\} \\ E_2 & = & \{(3),(4)\} \\ F & & \emptyset \end{array}$$

   (c) Test you solution for

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

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

2. • Let `F(x integer, y integer)` be relation that can store a binary relation F of pairs of integers (x, y). 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.[3]
   (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).

   Prove, from first principles[4], the correctness of the following rewrite rule:
$$\pi_{a,d}(R \bowtie_{c=d} S) = \pi_{a,d}(\pi_{a,c}(R) \bowtie_{c=d} \pi_d(S)).$$

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.

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

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

5. • In the translation algorithm from Pure SQL to RA we tacitly assumed that the argument of each subquery expression 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 subquery expressions [`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` subquery expression.

```
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: [5]

```
select L(r)
from   R r
```

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

```
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))
```

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`
hM, hM_1, $hM_2, \cdots$	`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);
```

which is translated to[6]

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

```
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
        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[7]

```
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[8]

```
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 the RA SQL query[9]

```
select q.pid, q.cname
from   (select w.*
        from   worksFor w
```

[7] Translation of 'in' and '<= ALL'.

[8] Move 'constant' condition.

[9] Introduction of natural join and join.

```
 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;
```

We can now commence with optimization. We outline two approaches:

Approach 1: Optimization in RA in standard notation

The non-optimized translated RA SQL query can be expressd as an RA expression in standard notation as

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

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.

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.,

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

Approach 2: Optimization in RA SQL

Since RA SQL is essentially RA simulated directly in SQL, all the optimization rules for RA can be applied in RA SQL. The benefit of this approach is that we can progressively run the RA SQL queries that result as intermediate steps during the optimization.

Step 1 *Observing what we did in step 1 in Approach 1, the query can be expressed as*

```
with
E as (select w.*
      from   worksFor w
      natural join (select cl.*
                    from   companyLocation cl
                    where  cl.city = 'Bloomington') cl),
F as (select w.*
      from   worksFor w join worksFor w1 on (w.cname=w1.cname and
                                             w.salary > w1.salary))
select w.pid, w.cname
from   (select e.*
        from   E e
        except
        select f.*
        from   F f) w;
```

Step 2, Step 3 *We can now optimize 'E' and 'F' and get the fully optimized query*

```
with
E_opt as (select w.*
          from   worksFor w
                 natural join (select cl.cname
                               from   companyLocation cl
                               where  cl.city = 'Bloomington') cl),
F_opt as (select w.*
      from   worksFor w join (select w1.cname, w1.salary
                              from   worksFor w1) w1
                        on (w.cname = w1.cname and w.salary > w1.salary))
select w.pid, w.cname
from   (select e.*
        from   E_opt e
        except
        select f.*
        from   F_opt f) w;
```

We now turn to the problems in this section.

6. Consider the query "*Find the pid and pname of each persons who knows no-one who works for the Apple company.*"
   A possible way to write this query in Pure SQL is

```
select p.pid, p.pname
from   Person p
where  false = all (select exists (select 1
                                   from   worksFor w
                                   where  p1.pid = w.pid and w.cname = 'Apple') and
                           (p.pid,p1.pid) = some (select k.pid1, k.pid2
                                                  from   Knows k)
                    from   Person p1);
```

   (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) • Using Approach 2, optimize this RA SQL query and provide the optimized expression in RA SQL. Specify at least three conceptually different rewrite rules that you used during the optimization.

7. Consider the query "*Find each pair* (c, p) *where* c *is the cname of a company and* p *is the pid of a person who works for that company and who earns strictly more than all other persons who work for that company and who earns more than 60000.*"

A possible way to write this query in Pure SQL is

```
select c.cname, p.pid
from   Company c, Person p
where  p.pid in (select w.pid
                 from   worksFor w
                 where  w.cname = c.cname and
                        true = all (select w1.salary <= 60000
                                    from   worksFor w1
                                    where  p.pid != w1.pid and
                                           w1.cname = c.cname and
                                           w.salary <= w1.salary));
```

(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) • Using Approach 2, optimize this RA SQL query and provide the optimized expression in RA SQL. Specify at least three conceptually different rewrite rules that you used during the optimization.

8. 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  not (p.pid, s.skill) in (select ps.pid, ps.skill
                                               from   personSkill ps)) and
       true = all (select s1 = s2
                   from   skill s1, skill s2
                   where  (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) • Using Approach 2, optimize this RA SQL query and provide the optimized expression in RA SQL. Specify at least three conceptually different rewrite rules that you used during the optimization.

9. Consider the query "*Find the cname of each company that (1) is not located in Chicago, (2) employs at least one person and (3) whose workers who make strictly less 60000 neither have the programming skill nor the AI skill.*"

```
select c.cname
from   Company c
where  c.cname in (select w.cname
                   from   worksFor w
                   where  not exists (select 1
                                      from   companyLocation cl
                                      where  w.cname = cl.cname and
                                             cl.city = 'Chicago')) and
       true = all (select p.pid not in (select ps.pid
                                        from   personSkill ps
                                        where  ps.skill = 'Programming' or
                                               ps.skill = 'AI')
                   from   Person p
                   where  p.pid in (select w.pid
                                    from   worksFor w
                                    where  w.cname = c.cname and
                                           w.salary < 60000));
```

(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) • Using Approach 2, optimize this RA SQL query and provide the optimized expression in RA SQL. Specify at least three conceptually different rewrite rules that you used during the optimization.

10. Consider the following Pure SQL query.

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

This query returns a pair $(p, \texttt{t})$ if p is the pid of a person who manages at least two persons and returns the pair $(p, \texttt{f})$ otherwise.[10]

(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) • Using Approach 2, optimize this RA SQL query and provide the optimized expression in RA SQL. Specify at least three conceptually different rewrite rules that you used during the optimization.

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

3 Solving queries using aggregate functions

Express the following queries in Full SQL. You will need to use aggregate functions to solve these queries. You can use views, temporary views, parameterized views, and user-defined functions.

11. • Find each pair $(c, \mathbf{a}, l, u)$ where 'c' is the cname of a company that pays each of its employees a salary between 50000 and 60000, '$\mathbf{a}'$ is the average salary of the employees who work for company 'c', 'l' is the number of employees who earn a salary strictly below this average, and 'u' is the number of employees who earn at least this average.

12. • Find each skill that is the skill of at least 3 persons who each know at least 2 persons who work for Apple and whose salary is at most 50000.

13. • Find the pid and name of each person p who has at least 3 job skills in the combined set of job skills of the persons who are managed by p[11].

14. • Find the cname of each company that employs at least 4 persons and that pays the highest average salary among such companies.

15. • Without using subquery expressions, find each pid of a person who knows each person who earns the highest salary at company Amazon.

16. • Without using subquery expressions, find each pairs (p_1, p_2) of pids of different persons such that if s is a job skill of p_1 then s is not a job skill of person p_2.

17. • Find each pairs (p_1, p_2) of different pids of persons p_1 and p_2 and such that (1) the number of skills of person s_1 is strictly less than the number of skills of person s_2 and (2) such that the gap between these numbers is the largest among all such possible gaps.

[11] This set of combined job skills is the set '$\{s|s$ is a job skill of a person managed by $p\}$'.

18. • Consider three types of salaries:

    - '`low`' is a salary below 50000
    - '`medium`' is a salary between 50001 and 60000
    - '`high`' is a salary above 60001

    Write a SQL query that returns each triple (c, t, n) where c is the name of a company, t is a salary type (i.e., one of '`low`', '`medium`', or '`high`', and n is the number of employees who work for company c and who have a salary of type t.

    (You can think of this problem as that of creating a histogram.)

19. •

    (a) Using the GROUP BY counting method, define a function

    ```
    create or replace function numberOfSkills(c text)
      returns table (pid integer, salary int, numberOfSkills bigint) as
      $$
      ...
      $$ language sql;
    ```

    that returns for a company identified by its cname, each triple (p, s, n) where (1) p is the pid of a person 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 person may not have any job skills.)

    (b) Test your function for Problem 19a for the companies Apple, Amazon, and ACM.

    (c) Write the same function `numberOfSkills` as in Problem 19c but this time without using the `GROUP BY` clause.

    (d) Test your function for Problem 19c for the companies Apple, Amazon, and ACM.

    (e) Using the function `numberOfSkills` but without using subquery expressions, write the following query: "*Find each pair (c, p) where c is the name of a company and where p is the pid of a person 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.*"