B461 Database Concepts
Assignment 3
Fall 2023

This assignment tests the following concepts:

- Pure SQL
- Relational Algebra (RA)
- Joins and semi-joins
- Pure SQL to RA SQL and RA Expressions

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 script 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.

- Include all problems with the blue bullet •in assignment3.sql
- Include all problems with the red bullet •in assignment3.pdf

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 assignment, we distinguish between Pure SQL and RA SQL.

Pure SQL *Rules*:

- `SELECT, FROM, WHERE`
- Set operations: `UNION, INTERSECT, EXCEPT`
- Predicates: `EXISTS, NOT EXISTS`
- Predicates: `IN, NOT IN`
- Predicates: `[NOT] (ALL, SOME)`
- Window functions `PARTITION, RANK` etc. and aggregate functions `SUM, AVG` etc. are **not** allowed
- `VIEWs, WITH` clauses that obey the rules above

RA SQL *Rules*:

- `SELECT, FROM, WHERE`
- The `WHERE` clause can **only** be used with constants.
- Set operations: `UNION, INTERSECT, EXCEPT`
- `JOINs, CROSS JOINs, NATURAL JOINs`
- Commas are **not** allowed
- Window functions `PARTITION, RANK` etc. and aggregate functions `SUM, AVG` etc. are **not** allowed
- `VIEWs, WITH` clauses that obey the rules above

In particular, any other elements of SQL that are not mentioned in the lecture slides (like `LIMIT`) are **not** allowed for Pure SQL **or** RA SQL

1 Theoretical problems related to query translation and optimization

1. In the translation algorithm from Pure SQL to RA we 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 argument of set predicate is a (parameterized) pure SQL query containing `UNION , INTERSECT, EXCEPT` operations.
   More specifically, consider the following types of queries using the [`not`] `exists` set predicate.

```
select L1(r1,...,rn)
from   R1 r1, ..., Rn rn
where  C1(r1,...,rn) and
              [not] exists (select s1.*,s2.*,...,sm.*
                            from   S1 s1,..., S1 sm
                            where  C2(s1,...,sm,r1,...,rn)
                            [union | intersect | except]
                            select s1.*,s2.*,...,sm.*
                            from   S1 s1,..., S1 sm
                            where  C3(s1,...,sm,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 in standard notation. In the translation, you should take into account that projections do not in general distribute over intersections and over set differences. [`15 pts`]

   exists (... union ...)

   $$= \pi_{L1}(\sigma_{C1\wedge(C2\vee C3}(\text{S1 x S... x Sm x R1 x R... x Rn}))$$

exists (... intersect ...)

$$= \pi_{L1}(\sigma_{C1\wedge(C2\wedge C3}(\text{S1 x S... x Sm x R1 x R... x Rn}))$$

exists (... except ...)

$$= \pi_{L1}(\sigma_{C1\wedge(C2\neg C3}(\text{S1 x S... x Sm x R1 x R... x Rn}))$$

not exists (... union ...)

$$= \pi_{Q1}\,(\sigma_{C1}(R1) \bowtie (\pi_{r1,\ldots,rn}(S1 \times \ldots \times Sm)) \\ \neg\sigma_{C2\vee C3}\,(s1 \times \ldots \times sn)$$

not exists (... intersect ...)

$$= \pi_{Q1}\,(\sigma_{C1}(R1) \bowtie (\pi_{r1,\ldots,rn}(S1 \times \ldots \times Sm)) \\ \neg\sigma_{C2\wedge C3}\,(s1 \times \ldots \times sn)$$

not exists (... except ...)

$$= \pi_{Q1}\,(\sigma_{C1}(R1) \bowtie (\pi_{r1,\ldots,rn}(S1 \times \ldots \times Sm)) \\ \neg\sigma_{C2\neg C3}\,(s1 \times \ldots \times sn)$$

To get practice, first consider the following special case where $n = 1$, $m = 1$, and $k = 1$. I.e., the following case: [2]

```
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(s)
                               from   S s
                               where  C3(s,r))
```

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

2 Translating Pure SQL to RA SQL

In this section, you are asked to *translate* Pure SQL queries into RA SQL queries as well as standard RA expressions using the *translation algorithm*.

You are required to show the intermediate steps that you took during the translation.

You can use the following notation 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`

Note: Please make note of the following example, and use it as a template to construct your answers for this section. You should write all the RA expressions in Latex or a word editor. Images of handwritten notes will NOT be accepted.

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);
```

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;
```

Translation of '`in`' and '`<= ALL`'.

```
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;
```

Move 'constant' condition.

```
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;
```

Introduction of natural join and join.

```
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

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

2. *"*Find the pid of each person that knows at least 2 people, such that at least 1 of them works at Apple or Netflix *."*
   A possible way to write this query in Pure SQL is

```
select distinct p.pid from Person p, Knows k1, Knows k2
where k1.pid1 = p.pid
and k2.pid1 = p.pid
and k1.pid2 <> k2.pid2
and exists (
            select 1 from worksFor w
            where (w.pid = k1.pid2 or w.pid = k2.pid2)
            and   (w.cname = 'Apple' or w.cname = 'Netflix')
);
```

   (a) • *Translate the Pure SQL query to RA SQL using the translation algorithm step-by-step. [*`12 pts`*]*
   (b) • *Write the RA Expression of the translated RA SQL query in standard notation. [*`5 pts`*]*

   where

$$\mathbf{Wx} = \pi_{W.pid}(\sigma_{cname='Apple'}(W) \cup \sigma_{cname='Netflix}(W))$$

   and

$$\textbf{Answer} = \pi_{p.pid}(P \bowtie_{P.pid=K1.pid1} K_1 \bowtie_{P.pid=K2.pid1, \wedge K1.pid2 \neq K2.pid2} K_2 \bowtie_{Wx.pid=K1.pid2, \vee Wx.pid=K2.pid2} Wx)$$

3. *"*Return the the pair (p, c) where p is the pid of a person, and c is the cname of the company where p works, such that (1) p is managed by someone who has at-least 2 skills and (2) p does not know anyone that lives in Seattle.*"*
   A possible way to write this query in Pure SQL is

```
select p.pid, c.cname
from person p , company c, worksfor w
where p.pid = w.pid and c.cname = w.cname
and exists (
     select 1 from hasManager m, personSkill ps1, personSkill ps2
     where m.eid = p.pid and ps1.pid = m.mid and ps2.pid = m.mid
     and ps1.skill <> ps2.skill
)
and p.pid not in (
     select k1.pid1 from knows k1
     where k1.pid2 in (select p1.pid from person p1 where p1.city='Seattle')
);
```

(a) • *Translate the Pure SQL query to RA SQL using the translation algorithm step-by-step. [`12 pts`]*

(b) • *Write the RA Expression of the translated RA SQL query in standard notation. [`5 pts`]*

where

$$\mathbf{Px} = \pi_{P.pid}(\sigma_{city='Seattle'}(P))$$

and

$$\mathbf{FIRST} = \pi_{W.pid,C.cname}(C \bowtie_{C.cname=W.cname} W \bowtie_{W.pid=M.eid} M \bowtie_{M.mid=PS1.pid} PS1 \bowtie_{M.mid=PS2.pid, \wedge PS1.skill \neq PS2.skill} PS2)$$

and

$$\mathbf{SECOND} = \pi_{W.pid,C.cname}(C \bowtie_{C.cname=W.cname} W \bowtie_{W.pid=M.eid}) M \bowtie_{M.mid=PS1.pid} PS1 \bowtie_{M.mid=PS2.pid, \wedge PS1.skill \neq PS2.skill} PS2 \bowtie_{W.pid=K1.pid1} K1 \bowtie_{K1.pid2=Px.pid} Px)$$

and

$$\mathbf{ANSWER} = \pi_{pid,cname}(FIRST - SECOND)$$

4. "Return each skill that is the skill of at least 2 persons, such that at least 1 of them lives in Bloomington"
A possible way to write this query in Pure SQL is

```
select s.skill from skill s
where exists (
    select 1 from personskill ps1, personskill ps2
    where ps1.pid <> ps2.pid
    and ps1.skill = ps2.skill and s.skill = ps1.skill
    and exists (
        select 1 from person p where p.city = 'Bloomington'
        and (ps1.pid = p.pid or ps2.pid = p.pid)
    )
);
```

(a) • *Translate the Pure SQL query to RA SQL using the translation algorithm step-by-step. [`12 pts`]*

(b) • *Write the RA Expression of the translated RA SQL query in standard notation. [`5 pts`]* *where*

$$\mathbf{Py} = \pi_{pid}(\sigma_{city='Bloomington'}(P))$$

and

$$\mathbf{Answer} = \pi_{S.skill}(S \bowtie_{S.skill=PS1.skill} PS1 \bowtie_{PS1.skill=PS2.skill} PS2 \bowtie_{PS1.pid=Py.pid, \vee\, PS2.pid=Py.pid} Py)$$

5. "Return the pair (p, s) where p is the pid of a person that works at a company headquartered in MountainView and s is the minimum salary among all people that know p."

   A possible way to write this query in Pure SQL is

```
    with mv
     as (select P.pid
         from   person P,worksfor W,company C
         where  P.pid = W.pid and W.cname = C.cname
                and C.headquarter = 'MountainView'),
     all_that_know_p
     as (select mv.pid,W.salary
         from   mv MV,knows K,worksfor W
         where  K.pid2 = mv.pid and W.pid = K.pid1
         order  by 1),
     min_salary
     as (select distinct ATP.pid,ATP.salary
         from   all_that_know_p ATP
         where  not exists (select 1 from   all_that_know_p ATP1 where   ATP.salary >

         ATP1.salary and ATP1.pid = ATP.pid))
select * from   min_salary;
```

   (a) • *Translate the Pure SQL query to RA SQL using the translation algorithm step-by-step. [`12 pts`]*

   (b) • *Write the RA Expression of the translated RA SQL query in standard notation. [`5 pts`]*

where

$$\mathbf{mv} = \pi_{p.pid}(P \bowtie_{P.pid=W.pid} W \bowtie_{W.cname=Cx.cname} Cx)$$

and

$$\mathbf{Cx} = \pi_{cname}(\sigma_{headquarter='MountainView'}C)$$

and

$$\textbf{all\_that\_know\_p} = \pi_{MV.pid,W.salary}(MV \bowtie_{MV.pid=K.pid2} K \bowtie_{K.pid1=W.pid} W)$$

and

$$\textbf{q} = \pi_{ATP.pid,ATP.salary}(ATP) - \pi_{ATP.pid,ATP.salary}(ATP \bowtie_{ATP.salary>ATP1.salary,\wedge ATP1.pid=ATP.pid} ATP1)$$

and

$$\textbf{min\_salary} = \pi_{pid,salary}(q)$$

and

$$\textbf{Answer} = \pi_{*}(min\_salary)$$

IDK if the project all is neccessary, or I could just put min_salary

6. "Return each cname such that
(1) at least 1 person working there has the OperatingSystems skill
(2) at least 2 persons working there live in different cities"

A possible way to write this query in Pure SQL is

```
select C.cname from company C
where  exists (select 1
               from   worksfor W,personskill PS
               where  W.cname = C.cname
                      and W.pid = PS.pid
                      and PS.skill = 'OperatingSystems')
       and exists (select 1
                   from   worksfor W1,worksfor W2,person P1,person P2
                   where  W1.cname = C.cname
                          and W2.cname = C.cname and W1.pid = P1.pid
                          and W2.pid = P2.pid and W1.pid <> W2.pid
                          and P1.city <> P2.city);
```

(a) • *Translate the Pure SQL query to RA SQL using the translation algorithm step-by-step. [`12 pts`]*

(b) • *Write the RA Expression of the translated RA SQL query in standard notation. [`5 pts`]*

where

$$\textbf{PSx} = \pi_{pid}(\sigma_{skill='OperatingSystems'} PS)$$

and

$$\textbf{Answer} = \pi_{C.cname}(C \bowtie_{C.cname=W.cname} W \bowtie_{W.pid=PSx.pid})$$
$$PSx \bowtie_{W1.cname=C.cname} W1 \bowtie_{W2.cname=C.cname, \wedge W1.pid \neq W2.pid}$$
$$W2 \bowtie_{P1.pid=W1.pid} P1 \bowtie_{W2.pid=P2.pid, \wedge P1.city \neq P2.city} P2$$