# B561 Advanced Database Concepts
# Assignment 3
# Spring 2022

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 e2
   where  exists (select distinct row() from F);
   ```

   **Remark 1** *The subquery 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 the following `if-then-else` query:[4] Display all the records of $E_1$ if there are no common records between $E_1$ and F else display all the records of $E_2$.

[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 referred to as the *empty tuple*, i.e., the tuple without components. It is akin to the empty string $\epsilon$ in the theory of formal languages. I.e., the string without 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 Sample boolean SQL query:

```
select not exists(select 1
                 from   R r1, R r2
                 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 and the sample boolean query, Create a similar boolean SQL query (add it in the pdf file itself) for the relation - worksFor, such that it returns the constant "**`true`**" if there are at least 2 employees working for the company 'Apple' and returns the constant "**`false`**" otherwise. Then write an RA expression in standard notation that expresses the above boolean SQL query.[5]

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.

```
select L1(r1,...,rn)
from   R1 r1, ..., Rn rn
where  C1(r1,...,rn) and
              [not] exists (select L2(s1,...,sm)
                            from   S1 s1,..., S1 sm
                            where  C2(s1,...,sm,r1,...,rn)
                            [union | intersect | except]
                            select L3(t1,...,tk)
```

[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.

```
                                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 in standard notation. 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 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))
```

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[7], the correctness of the following rewrite rule:

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

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.

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

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

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.

## 2 Joins in RA expressions

In this section, you will be making the use of various types of joins to form the asked queries in RA standard notation.

6. • Give one example of each type of join - Cartesian join, Inner join, Natural join, Semijoin and antijoin using the given schema in RA Standard notation. Also include one liner description of how the join works in your examples.

For the following questions give the correct RA notation **using only the given type/s of join/s** and also mention clearly the letters used to denote relation names in the RA expressions:

7. • Formulate the constraint - "No person who works for Microsoft has the AI skill" in RA expression **using only natural joins**.

8. • Formulate the query - "Find the name of each company that only employs persons such that no two employees know the same person" in RA expression **using only inner joins**.

9. • Formulate the query - "Find the name and city of each person that earns more than 50000, works a company which has a headquarter in 'Cupertino' and has some skill" in RA expression **using only Semijoins**.

10. • Formulate the query - "Find all the details of each person that is not employed by any company, doesn't have a skill and does not know anyone" in RA expression **using at least 1 Antijoin**.

## 3 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*[8]

```
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
```

[8]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*[9]

```
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*[10]

```
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*[11]

```
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.*

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

[10] Move 'constant' condition.

[11] 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.

11. Consider the query "*Find the cname and headquarter of each company that employs persons who earn less than 60000 and has Networks skill.*"

    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 < 60000 and
                          w.pid = SOME (select ps.pid
                                        from   personSkill ps
                                        where  ps.skill = 'Networks'));
```

    (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 the above obtained RA SQL query.

    (c) • Provide the optimized expression in standard RA notation. Specify at least three conceptually different rewrite rules that you used during the optimization.

12. Consider the query "*Find the manager id whose salary is greater than 55000 and his/her employee earns less than 55000 and the employee doesn't stay in the same city as the company's headquarter.*"

    A possible way to write this query in Pure SQL is

```
select hm.mid
from hasManager hm, worksfor w, Company c
where w.cname = c.cname and
      exists(select 1
                 from person p
                 where p.pid = hm.mid and w.pid = hm.mid and w.salary > 55000)
                 and exists(select 1
                               from Person e
                               where (e.pid, w.cname) in (select ws.pid, ws.cname
                                                            from worksFor ws
                                                            where ws.pid = hm.eid
                                                            and ws.salary < 55000
                                                            and e.city <> c.headquarter)));
```

(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 the above obtained RA SQL query.

(c) • Provide the optimized expression in standard RA notation. Specify at least three conceptually different rewrite rules that you used during the optimization.

13. Consider the query "*Find the pid and name of each person who works for a company headquartered at Seattle and earns strictly greater than 50000 but his/her manager does not live in Seattle.*"

A possible way to write this query in Pure SQL is

```
select p.pid, p.pname
from   person p
where not exists (select 1
               from   worksFor w,
               where  p.pid = w.pid and
       w.cname not in (select c.cname
                     from company c
                     where w.salary > 50000 and c.headquarter = 'Seattle'))
                     and exists (select 1
                                 from hasManager hm
                                 where p.pid in (select hm.eid
                                                from person p1
                                                where hm.mid = p1.pid
                                                and  p1.city <> 'Seattle'));
```

(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 the above obtained RA SQL query.

(c) • Provide the optimized expression in standard RA notation. Specify at least three conceptually different rewrite rules that you used during the optimization.

14. Consider the query "*Find the skill of each employee who lives in Cupertino and earns not greater than 50000.*"

A possible way to write this query in Pure SQL is

```
select s.skill
from skill s
where s.skill in (select ps.skill
                  from personSkill ps
                  where ps.pid not in (select p.pid
                                       from person p
                                       where p.city <> 'Cupertino') and
                  ps.pid  not in (select w.pid
                                  from worksFor w
                                  where w.salary > 50000));
```

(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 the above obtained RA SQL query.

(c) • Provide the optimized expression in standard RA notation. Specify at least three conceptually different rewrite rules that you used during the optimization.

15. Consider the query "*Find the pids of persons who work for a company that has two managers..*"

A possible way to write this query in Pure SQL is

```
select p.pid
from person p
where exists (select 1
              from worksfor w
              where p.pid = w.pid)
          and p.pid in (select hm1.eid
                        from hasManager hm1
                        where exists(select 1
                                     from hasManager hm2
                                     where hm1.eid = hm2.eid and
                                           hm1.mid <> hm2.mid));
```

(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 the above obtained RA SQL query.

(c) • Provide the optimized expression in standard RA notation. Specify at least three conceptually different rewrite rules that you used during the optimization.