B461 Database Concepts–Jonathan Wells
Assignment 2
Fall 2023

This assignment relies on the lectures

- SQL Part 1 and SQL Part 2 (Pure SQL);
- Views;
- Relational Algebra (RA); and
- Joins and semijoins.

To turn in your assignment, you will need to upload to Canvas a file with name `assignment2.sql` which contains the necessary SQL statements that solve the problems in this assignment. The `assignment2.sql` file must be so that the AI's can run it in their PostgreSQL environment. You should use the `Assignment2Script.sql` file to construct the `assignment2.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 `assignment2.txt` file that contains the results of running your queries. Finally, you need to upload a file `assignment2.pdf` that contains the solutions to the problems that require it. In short, 3 files `assignment2.sql`, `assignment2.txt`, and `assignment2.pdf` should be submitted in canvas.

The problems that need to be included in the `assignment2.sql` are marked with a blue bullet •. The problems that need to be included in the `assignment2.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`

The file `Assignment2Script.sql` contains the data supplied for this assignment.

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 Formulating queries in Pure SQL with and without set predicates

An important consideration in formulating queries is to contemplate how they can be written in different, but equivalent, ways. In fact, this is an aspect of programming in general and, as can expected, is also true for SQL. A learning outcome of this course is to acquire skills for writing queries in different ways. One of the main motivations for this is to learn that different formulations of the same query can dramatically impact performance, sometimes by orders of magnitude.

For the problems in this section, you will need to formulate queries in Pure SQL with and without set predicates. You can use the SQL operators `INTERSECT`, `UNION`, and `EXCEPT`, unless otherwise specified. You are however allowed and encouraged to use views including temporary and user-defined views.

To illustrate what you need to do, consider the following example.

Example 1 *Consider the query* "Find the pid and name of each person who (a) works for a company located in Bloomington and (b) knows a person who lives in Chicago."

(a) *Formulate this query in Pure SQL by only using the* `EXISTS` *or* `NOT EXISTS` *set predicates. You can not use the set operations* `INTERSECT`*,* `UNION`*, and* `EXCEPT`*.*

A possible solution is

```
select p.pid, p.pname
from   Person p
where  exists (select 1
               from   worksFor w
               where  p.pid = w.pid and
                      exists (select 1
                              from   companyLocation c
                              where  w.cname = c.cname and c.city = 'Bloomington')) and
       exists (select 1
               from   Knows k
               where  p.pid = k.pid1 and
                      exists (select 1
                              from   Person p2
                              where  k.pid2 = p2.pid and
                                     p2.city = 'Chicago'));
```

(b) *Formulate this query in Pure SQL by only using the* IN, NOT IN, SOME, *or* ALL *set membership predicates. You can not use the set operations* INTERSECT, UNION, *and* EXCEPT.

A possible solution is

```
select p.pid, p.pname
from   Person p
where  p.pid in (select w.pid
                 from   worksFor w
                 where  w.cname in (select c.cname
                                    from   companyLocation c
                                    where  c.city = 'Bloomington')) and
       p.pid in (select k.pid1
                 from   Knows k
                 where  k.pid2 in (select p2.pid
                                   from   Person p2
                                   where  p2.city = 'Chicago'));
```

Another possible solution using the SOME *and* IN *predicates*

```
select p.pid, p.pname
from   Person p
where  p.pid = some (select w.pid
                     from   worksFor w
                     where  w.cname = some (select c.cname
                                            from   companyLocation c
                                            where  c.city = 'Bloomington')) and
       p.pid in (select k.pid1
                 from   Knows k
                 where  k.pid2 in (select p2.pid
                                   from   Person p2
                                   where  p2.city = 'Chicago'));
```

(c) *Formulate this query in Pure SQL by only using the set operations* INTERSECT, UNION, *and* EXCEPT. *A possible solution is*

```
select p.pid, p.pname
from   Person p, worksFor w, companyLocation c
where  p.pid = w.pid and
       w.cname = c.cname and
       c.city = 'Bloomington'
intersect
select p1.pid, p1.pname
from   Person p1, Knows k, Person p2
where  k.pid1 = p1.pid and
       k.pid2 = p2.pid and
       p2.city = 'Chicago';
```

We now turn to the problems for this section.

1. Consider the query "*Find each triple* (c, p, s) *where:*

   - *c is the* **`cname`** *of a company.*
   - *p is the* **`pid`** *of a person who earns the lowest salary at that company c and knows at least someone who works at Apple.*
   - *s is the* **`salary`** *of p".*

   (a) • Formulate this query in Pure SQL by only using the `EXISTS` or `NOT EXISTS` set predicates. 7.5 points

   (b) • Formulate this query in Pure SQL by only using the `IN`, `NOT IN`, `SOME`, or `ALL` set membership predicates. 7.5 points

   (c) • Formulate this query in Pure SQL by only using the set operations `INTERSECT`, `UNION`, and `EXCEPT`. 7.5 points

2. Consider the query "*Find each pair* (c_1, c_2) *such that:*

   - *c_1 and c_2 are* **`cnames`** *of different companies and*
   - *no employee of c_1 and no employee of c_2 live in Chicago".*

   (a) • Formulate this query in Pure SQL by only using the `EXISTS` or `NOT EXISTS` set predicates. You can not use the set operations `INTERSECT`, `UNION`, and `EXCEPT`. 7.5 points

   (b) • Formulate this query in Pure SQL by only using the `IN`, `NOT IN`, `SOME`, or `ALL` set membership predicates. You can not use the set operations `INTERSECT`, `UNION`, and `EXCEPT`.
   . 7.5 points

   (c) • Formulate this query in Pure SQL by only using the set operations `INTERSECT`, `UNION`, and `EXCEPT`. 7.5 points

2 Formulating queries in Relational Algebra and RA SQL

Reconsider the queries in Section 1. The goal of the problems in this section is to formulate these queries in Relational Algebra in standard notation and in RA SQL.

There are some further restrictions:

- You can only use `WHERE` clauses that use conditions involving constants. For example conditions of the form "$t.A\,\theta\,$'a'" are allowed, but conditions of the form '$t.A\,\theta\,s.B$' are not allowed. The latter conditions can be absorbed in `JOIN` operations in the `FROM` clause.
- You can not use commas in any `FROM` clause. Rather, you should use `JOIN` operations.

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`

To illustrate what you need to do reconsider the query in Example 1 in Section 1.

Example 2 *Consider the query* "Find the pid and name of each person who (a) works for a company located in Bloomington and (b) knows a person who lives in Chicago."

(a) *Formulate this query in Relational Algebra in standard notation.*

A possible solution is

$$\pi_{pid,pname}(Person \bowtie worksFor \bowtie \pi_{cname}(\sigma_{city=\textbf{Bloomington}}(companyLocation)))\cap$$
$$\pi_{Person_1.pid,Person_1.pname}(Person_1 \bowtie_{Person_1.pid=pid1} Knows \bowtie_{pid2=Person_2.pid} \pi_{Person_2.pid}(\sigma_{city=\textbf{Chicago}}(Person_2)))$$

If we use the letters in the above box, this expression becomes more succinct:

$$\pi_{pid,pname}(P \bowtie W \bowtie \pi_{cname}(\sigma_{city=\mathbf{Bloomington}}(cL)))\cap \\ \pi_{P_1.pid,P_1.pname}(P_1 \bowtie_{P_1.pid=pid1} K \bowtie_{pid2=P_2.pid} \pi_{P_2.pid}(\sigma_{city=\mathbf{Chicago}}(P_2)))$$

You are also allowed to introduce letters that denote expressions. For example, let E and F denote the expression

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

and

$$\pi_{P_1.pid,P_1.pname}(P_1 \bowtie_{P_1.pid=pid1} K \bowtie_{pid2=P_2.pid} \pi_{P_2.pid}(\sigma_{city=\mathbf{Chicago}}(P_2))),$$

respectively. Then we can write the solution as follows:

$$E \cap F.$$

(b) *Formulate this query in RA SQL.*

A possible solution is

```
select pid, pname
from   Person
       NATURAL JOIN worksFor
       NATURAL JOIN (select cname
                     from   companyLocation
                     where  city = 'Bloomington') C
INTERSECT
select P1.pid, P1.pname
from   Person P1
       JOIN Knows ON (P1.pid = pid1)
       JOIN (select pid
             from   Person
             where  city = 'Chicago') P2 ON (pid2 = P2.pid)
order by 1,2;
```

Observe that the `WHERE` *clauses only use conditions involving constants.*

We now turn to the problems in this section.

3. Reconsider Problem 1. "*Find each triple* (c, p, s) *where:*
   - c *is the* **`cname`** *of a company.*
   - p *is the* **`pid`** *of a person who earns the lowest salary at that company* c *and knows at least someone who works at Apple.*
   - s *is the* **`salary`** *of* p*".*

   (a) • Formulate this query in Relational Algebra in standard notation. 7.5 points

   c, p, s = w.cname, p.pid, W1.salary

   $Q1 \leftarrow \pi_{c,p,s}(W1 \bowtie_{w1.pid = p1.pid} (P1))$

   $Q2 \leftarrow \pi_{c,p,s}(\sigma\, W2 \bowtie_{w2.pid <> w3.pid \wedge w3.cname = 'Apple'} (W3) \wedge$
   $W2 \bowtie_{w2.pid = k.pid1 \wedge w3.pid = k.pid2} (K))$

   $Q3 \leftarrow \pi_{c,p,s}(W3 \bowtie_{w3.cname = w4.cname \wedge w3.salary > w4.salary} (W4))$

   $result \leftarrow Q1 \cap Q2 - Q3$

   (b) • Formulate this query in RA SQL. 7.5 points

4. Reconsider Problem 2. "*Find each pair* (c_1, c_2) *such that:*
   - c_1 *and* c_2 *are* **`cnames`** *of different companies and*
   - *no employee of* c_1 *and no employee of* c_2 *live in Chicago".*

   (a) • Formulate this query in Relational Algebra in standard notation. 6.5 points

   (b) • Formulate this query in RA SQL. 6.5 points

$fChicago \leftarrow \pi_{w.cname}(P \bowtie_{p.pid = w.pid \wedge p.city = 'Chicago'} (W))$

$a1 \leftarrow \pi_{c1.cname}(C1) - \pi_{fc.cname}(fChicago)$

$a2 \leftarrow \pi_{c2.cname}(C2) - \pi_{fc.cname}(fChicago)$

$result \leftarrow \pi_{a1.cname\ as\ c1,\ a2.cname\ as\ c2}(a1 \bowtie_{a1.cname <> a2.cname} (a2))$

3 Formulating queries in SQL using views

Formulate the following views and queries in SQL. You are allowed to combine the features of both Pure SQL and RA SQL.

5. • Create a materialized view `CompanyKnownPerson` such that, for each company, the view returns the `pid` of Persons who are known by atleast one different person (other than pid) from the same company and earn the same salary.

   Then test your view. 7.5 points

6. • Create a parameterized view `SkillOnlyOnePerson (skill1 text)` that returns pair of different persons pid1, pid2 such that `pid1` should have the skill identified by `skill1` and `pid2` should not have the skill identified by `skill1`.

   Test your view for skill1 = 'Networks'. 7.5 points

7. • Let $PC(parent : integer, child : integer)$ be a rooted parent-child tree. So a pair (n, m) in PC indicates that node n is a parent of node m. The `sameGeneration(n1, n2)` binary relation is inductively defined using the following two rules:

   - If n is a node in PC, then the pair (n, n) is in the `sameGeneration` relation. (**Base rule**)
   - If n_1 is the parent of m_1 in PC and n_2 is the parent of m_2 in $Tree$ and (n_1, n_2) is a pair in the `sameGeneration` relation then (m_1, m_2) is a pair in the `sameGeneration` relation. (**Inductive Rule**)

   Write a recursive view for the `sameGeneration` relation.

   Test your view. 12 points