B561 Advanced Database Concepts
Assignment 2
Fall 2024

Chenghong Wang

This assignment relies on the lectures

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

Furthermore, the lecture on the correspondence between Tuple Relational Calculus and SQL should help you to solve problems in this assignment.

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]

```
Student(sid, sname, city)
Company(cname, headquarter)
Skill(skill)
worksFor(sid, cname, salary)
companyLocation(cname, city)
studentSkill(sid, skill)
hasManager(eid, mid)
Knows(sid1, sid2)
```

In this database we maintain a set of students (`Student`), a set of companies (`Company`), and a set of (job) skills (`Skill`). The `sname` attribute in `Student` is the name of the student. The `city` attribute in `Student` specifies the city in which the student 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 student can work for at most one company. This information is maintained in the `worksFor` relation. (We permit that a student does not work for any company.) The `salary` attribute in `worksFor` specifies the salary made by the student.

The `city` attribute in `companyLocation` indicates a city in which the company is located. (Companies may be located in multiple cities.)

A student can have multiple job skills. This information is maintained in the `studentSkill` relation. A job skill can be the job skill of multiple students. (A student may not have any job skills, and a job skill may have no students with that skill.)

A pair (e, m) in `hasManager` indicates that student e has student 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 (s_1, s_2) where s_1 and s_2 are sids of students. The pair (s_1, s_2) indicates that the student with sid s_1 knows the student with sid s_2. We do not assume that the relation `Knows` is symmetric: it is possible that (s_1, s_2) is in the relation but that (s_2, s_1) is not.

The domain for the attributes `sid`, `sid1`, `sid2`, `salary`, `eid`, and `mid` is `integer`. The domain for all other attributes is `text`.

We assume the following foreign key constraints:

- `sid` is a foreign key in `worksFor` referencing the primary key `sid` in `Student`;
- `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`;
- `sid` is a foreign key in `StudentSkill` referencing the primary key `sid` in `Student`;
- `skill` is a foreign key in `StudentSkill` referencing the primary key `skill` in `Skill`;
- `eid` is a foreign key in `hasManager` referencing the primary key `sid` in `Student`;
- `mid` is a foreign key in `hasManager` referencing the primary key `sid` in `Student`;
- `sid1` is a foreign key in `Knows` referencing the primary key `sid` in `Student`; and
- `sid2` is a foreign key in `Knows` referencing the primary key `sid` in `Student`

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 sid and name of each Student who (a) works for a company located in Bloomington and (b) knows a Student 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 s.sid, s.sname
from   Student s
where  exists (select 1
               from   worksFor w
               where  s.sid = w.sid and
                      exists (select 1
                              from   companyLocation c
                              where  w.cname = c.cname and c.city = 'Bloomington')) and
       exists (select 1
               from   Knows k
               where  s.sid = k.sid1 and
                      exists (select 1
                              from   Student s2
                              where  k.sid2 = s2.sid and
                                     s2.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 s.sid, s.sname
from   Student s
where  s.sid in (select w.sid
                 from   worksFor w
                 where  w.cname in (select c.cname
                                    from   companyLocation c
                                    where  c.city = 'Bloomington')) and
       s.sid in (select k.sid1
                 from   Knows k
                 where  k.sid2 in (select s2.sid
                                   from   Student s2
                                   where  s2.city = 'Chicago'));
```

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

```
select s.sid, s.sname
from   Student s
where  s.sid = some (select w.sid
                     from   worksFor w
                     where  w.cname = some (select c.cname
                                            from   companyLocation c
                                            where  c.city = 'Bloomington')) and
       s.sid in (select k.sid1
                 from   Knows k
                 where  k.sid2 in (select s2.sid
                                   from   Student s2
                                   where  s2.city = 'Chicago'));
```

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

```
select s.sid, s.sname
from   Student s, worksFor w, companyLocation c
where  s.sid = w.sid and
       w.cname = c.cname and
       c.city = 'Bloomington'
intersect
select s1.sid, s1.sname
from   Student s1, Knows k, Student s2
where  k.sid1 = s1.sid and
       k.sid2 = s2.sid and
       s2.city = 'Chicago';
```

We now turn to the problems for this section.

1. Consider the query "*Find each triple (c, s, sl) where c is the cname of a company, s is the sid of a student who earns the lowest salary at that company and knows at least someone who has Operating Systems skill, and sl is the salary of s.*"

   (a) • Formulate this query in Pure SQL by only using the `EXISTS` or `NOT EXISTS` set predicates. (4.5 points)

   (b) • Formulate this query in Pure SQL by only using the `IN`, `NOT IN`, `SOME`, or `ALL` set membership predicates. (4.5 points)

   (c) • Formulate this query in Pure SQL by only using the set operations `INTERSECT`, `UNION`, and `EXCEPT`. (4.5 points)

2. Consider the query "*Find the name, salary and city of each student who (a) lives in a city where no one has the Networks skill and (b) earns the highest salary in his/her company.*"

   (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`. (4.5 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`. (4.5 points)

   (c) • Formulate this query in Pure SQL by only using the set operations `INTERSECT`, `UNION`, and `EXCEPT`. (4.5 points)

3. Consider the query "*Find each pair* $(c1, c2)$ *of cnames of different companies such that no employee of c1 and no employee of c2 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`. (4.5 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`. (4.5 points)

(c) • Formulate this query in Pure SQL by only using the set operations `INTERSECT`, `UNION`, and `EXCEPT`. (4.5 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. (4.5 points)
- You can not use commas in any `FROM` clause. Rather, you should use `JOIN` operations. (4.5 points)

You can use the following letters, or indexed letters, to denote relation names in RA expressions:

$S, S_1, S_2, \cdots$	`Student`
$C, C_1, C_2, \cdots$	`Company`
$S, S_1, S_2, \cdots$	`Skill`
$W, W_1, W_2, \cdots$	`worksFor`
$cL, cL_1, cL_2, \cdots$	`companyLocation`
$sS, sS_1, sS_2, \cdots$	`StudentSkill`
$M, M_1, M_2, \cdots$	`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 sid and name of each student who (a) works for a company located in Bloomington and (b) knows a student who lives in Chicago."

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

A possible solution is

$$\pi_{sid,sname}(Student \bowtie worksFor \bowtie \pi_{cname}(\sigma_{city=\textbf{Bloomington}}(companyLocation)))\cap$$
$$\pi_{Student_1.sid,Student_1.sname}(Student_1 \bowtie_{Student_1.sid=sid1} Knows \bowtie_{sid2=Student_2.sid} \pi_{Student_2.sid}(\sigma_{city=\textbf{Chicago}}(Student_2))$$

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

$$\pi_{sid,sname}(S \bowtie W \bowtie \pi_{cname}(\sigma_{city=\textbf{Bloomington}}(cL))) \cap \\ \pi_{S_1.sid,S_1.sname}(S_1 \bowtie_{S_1.sid=sid1} K \bowtie_{sid2=S_2.sid} \pi_{S_2.sid}(\sigma_{city=\textbf{Chicago}}(S_2)))$$

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

$$\pi_{sid,sname}(S \bowtie W \bowtie \pi_{cname}(\sigma_{city=\textbf{Bloomington}}(cL)))$$

and

$$\pi_{S_1.sid,S_1.sname}(S_1 \bowtie_{S_1.sid=sid1} K \bowtie_{sid2=S_2.sid} \pi_{S_2.sid}(\sigma_{city=\textbf{Chicago}}(S_2))),$$

respectively. Then we can write the solution as follows:

$$E \cap F.$$

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

A possible solution is

```
select sid, sname
from   Student
       NATURAL JOIN worksFor
       NATURAL JOIN (select cname
                     from   companyLocation
                     where  city = 'Bloomington') C
INTERSECT
select S1.sid, S1.sname
from   Student S1
       JOIN Knows ON (S1.sid = sid1)
       JOIN (select sid
             from   Student
             where  city = 'Chicago') S2 ON (sid2 = S2.sid)
order by 1,2;
```

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

We now turn to the problems in this section.

4. Reconsider Problem 1. Find each triple (c, s, sal) where c is the cname of a company, s is the sid of a student who earns the lowest salary at that company and knows at least someone who has Operating Systems skill, and sal is the salary of s.

   (a) • Formulate this query in Relational Algebra in standard notation. (4.5 points)

   (b) • Formulate this query in RA SQL. (3 points)

5. Reconsider Problem 2. Find the name, salary and city of each student who (a) lives in a city where no one has the Networks skill and (b) earns the highest salary in his/her company.

   (a) • Formulate this query in Relational Algebra in standard notation. (4.5 points)

   (b) • Formulate this query in RA SQL. (3 points)

6. Reconsider Problem 3. Find each pair $(c1, c2)$ of cnames of different companies such that no employee of c1 and no employee of c2 live in Chicago.

   (a) • Formulate this query in Relational Algebra in standard notation. (4.5 points)

   (b) • Formulate this query in RA SQL. (3 points)

3 Formulating constraints using Relational Algebra

Recall that it is possible to express constraints in TRC and as boolean SQL queries. It is also possible to write constraints using RA expressions. Let E, F, and G denote RA expressions. Then we can write RA expression comparisons that express constraints:

$E \neq \emptyset$	which is true if E evaluates to an non-empty relation
$E = \emptyset$	which is true if E evaluates to the empty relation
$F \subseteq G$	which is true if F evaluates to a relation that is a subset of the relation obtained from G
$F \not\subseteq G$	which is true if F evaluates to a relation that is not a subset of the relation obtained from G

Here are some examples of writing constraints in this manner.

Example 3 "Some student works for Google." *This constraint can be written as follows:*

$$\pi_{sid}(\sigma_{cname=\mathbf{Google}}(worksFor)) \neq \emptyset.$$

Indeed, the RA expression

$$\pi_{sid}(\sigma_{cname=\mathbf{Google}}(worksFor))$$

computes the set of all student sids who work for Google. If this set is not empty then there are indeed students who work for Google.

Incidentally, the constraint "No one works for Google" *can be written as follows:*

$$\pi_{sid}(\sigma_{cname=\mathbf{Google}}(worksFor)) = \emptyset.$$

Example 4 Each student has at least two skills. *This constraint can be written as follows:*

$$\pi_{sid}(S) \subseteq \pi_{sS_1.sid}(\sigma_{sS_1.skill \neq sS_2.skill}(sS_1 \bowtie_{sS_1.sid=sS_2.sid} sS_2)).$$

Indeed,

$$\pi_{sid}(S)$$

computes the set of all student sids and

$$\pi_{sS_1.sid}(\sigma_{sS_1.skill \neq sS_2.skill}(sS_1 \bowtie_{sS_1.sid=sS_2.sid} sS_2))$$

computes the set of all sids of students who have at least two skills. When the first set is contained in the second we must have that each student has at least two skills.

Incidentally, the constraint "Some student has fewer than two skills" *can be written as follows:*

$$\pi_{sid}(S) \nsubseteq \pi_{sS_1.sid}(\sigma_{sS_1.skill \neq sS_2.skill}(sS_1 \bowtie_{sS_1.sid = sS_2.sid} sS_2)).$$

We now turn to the problems in this section.

Formulate each of the following constraints using RA expressions as illustrated in Example 3 and Example 4.

7. • Each manager knows all of his/her students.

8. • No student who works at Amazon knows at-most 2 people.

9. • Some student who works for a company headquartered at Cupertino has a salary less than student with no skills. (Assumption: Only 1 student with no skills)

4 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.

10. • Create a materialized view `CompanyKnownStudent` such that, for each company, the view returns the `sid` of Student who are known by at least two different student (other than pid) from the same company and the `sid` earns more salary than them. (6 points)

    Then test your view.

11. • Create a parameterized view `SkillOnlyOneStudent (skill1 text)` that returns pair of different Students sid1, sid2 such that `sid1` should have the skill identified by `skill1` and `sid2` should not have the skill identified by `skill1`. Note that `sid2` should have at least one skill. (6 points)

    Test your view for skill1 = 'WebDevelopment'.

12. • 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. (7 points)

    Test your view.