B561 Advanced Database Concepts
Assignment 5
Fall 2023

Muazzam Siddiqui, Pranav Mahajan, Pavan Kalyan Thota

This assignment tests the following concepts:

- Object Relational Queries
- Nested Relations and Semi-Structured Databases
- PL/pgSQL

To turn in your assignment, you will need to upload to Canvas a single file with name assignment5.sql which contains the necessary SQL statements that solve the problems in this assignment. The assignment5.sql file must be so that the AI's can run it in their PostgreSQL environment. You should use the assignment5-Script-Fall2023.sql file to construct the assignment5.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 assignment5.txt file that contains the results of running your queries.

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`

1 Formulating Query in Object-Relational SQL

For the problems in the section, you will need to use the polymorphically defined functions and predicates that are defined in the document assignment5-Script-Fall2023.sql

Functions

- `set_union(A,B)` : $A \cup B$
- `set_intersection(A,B)` : $A \cap B$
- `set_difference(A,B)` : $A \setminus B$
- `add_element(x,A)` : $x \cup A$
- `remove_element(x,A)` : $A \setminus \{x\}$
- `make_singleton(x)` : $\{x\}$
- `choose_element(A)` : choose some element from A
- `bag_union(A,B)` : the bag union of A and B
- `bag_to_set(A)` : coerce the bag A to the corresponding set

Predicates

- `is_in(x,A)` : $x \in A$
- `is_not_in(x,A)` : $x \notin A$
- `is_empty(A)` : $A = \varnothing$
- `is_not_empty(A)` : $A \neq \varnothing$
- `subset(A,B)` : $A \subseteq B$
- `superset(A,B)` : $A \supseteq B$
- `equal(A,B)` : $A = B$
- `overlap(A,B)` : $A \cap B \neq \varnothing$
- `disjoint(A,B)` : $A \cap B = \varnothing$

But before turning to the problems, we will introduce various object-relational views defined over these relations in the schema:

1. The view companyHasEmployees(cname,employees) which as- sociates with each company, identfied by a cname, the set of pids of persons who work for that company.

```
create or replace view companyHasEmployees as
select cname, array(select pid
from worksfor w
where w.cname = c.cname order by 1) as employees
from company c order by 1;
```

2. The view cityHasCompanies(city,companies) which associates with each city the set of cnames of companies that are located in that city.

```
create or replace view cityHasCompanies as
select city, array_agg(cname order by 1) as companies
from companyLocation
group by city order by 1;
```

3. The view companyHasLocations(cname,locations) which asso- ciates with each company, identified by a cname, the set of cities in which that company is located.

```
create or replace view companyHasLocations as
select cname, array(select city
from companyLocation cc
where c.cname = cc.cname order by 1) as locations
from company c order by 1;
```

4. The view knowsPersons(pid,persons) which associates with each per- son, identified by a pid, the set of pids of persons he or she knows.

```
create or replace view knowsPersons as
select p.pid, array(select k.pid2
from knows k
where k.pid1 = p.pid order by pid2) as persons
from person p order by 1;
```

5. The view isKnownByPersons(pid,persons) which associates with each person, identifed by a pid, the set of pids of persons who know that person. Observe that there may be persons who are not known by any one.

```
create or replace view isKnownByPersons as
select distinct p.pid, array(select k.pid1
from knows k
where k.pid2 = p.pid) as persons
from person p order by 1;
```

6. The view personHasSkills(pid,skills) which associates with each per- son, identified by a pid, his or her set of job skills.

```
create or replace view personHasSkills as
select distinct p.pid, array(select s.skill
from personSkill s
where s.pid = p.pid order by 1) as skills
from person p order by 1;
```

7. The view skillOfPersons(skills,persons) which associates with each job skill the set of pids of persons who have that job skill.

```
create or replace view skillOfPersons as
select js.skill, array(select ps.pid
from personSkill ps
where ps.skill = js.skill order by pid) as persons
from jobSkill js order by skill;
```

In the problems in this section, you are asked to formulate queries in object- relational SQL. You should use the set operations and set predicates defined in the document assignment5-Script-Fall2023.sql,
THE RELATIONS:

```
Person
Company
Skill
worksFor
```

AND THE VIEWS:

```
companyHasEmployees
cityHasCompanies
companyHasLocations
knowsPersons
isKnownByPersons
personHasSkills
skillOfPersons
```

However, you are not permitted to use the **Knows**, **companyLocation**, and **personSkill** relations in the object-relation SQL formulation of the queries. Observe that you actually don't need these relations since they are encapsulated in these views.

Before listing the queries that you are asked to formulate, we present some examples of queries that are formulated in object-relational SQL using the assumptions stated in the previous paragraph. Your solutions need to be in the style of these examples. The goals is to maximize the utilization of the functions and predicates defined in document `assignment5-Script-Fall2023.sql`.

1. **Example 1:** Consider the query "Find the pid of each person who knows a person who has a salary greater than 55000."

   ```
   select distinct pk.pid
   from knowsPersons pk, worksfor w
   where is_in(w.pid, pk.persons) and w.salary > 55000
   order by 1;
   ```

   Note that the following formulation for this query is not allowed since it uses the relation `Knows` which is not permitted.

```
select distinct k.pid1
from knows k, worksfor w
where k.pid2 = w.pid and w.salary > 55000;
```

2. **Example 2:** Consider the query "Find the pid and name of each person along with the set of his or her skills that are not among the skills of persons who work for 'Netflix'."

```
select p.pid, p.pname,
    set_difference((select ps.skills
       from personHasSkills ps
       where ps.pid = p.pid),
            array(select unnest(ps.skills)
            from personHasSkills ps
            where is_in(ps.pid, (select employees
                                 from companyHasEmployees
                                 where cname = 'Netflix'))))
    from person p;
```

2 Object Relational Queries

1. Find the cname and headquarter of each company that employs at least two persons who have a common job skill.

2. Find the pid and name of each person p along with the set of pids of persons who (1) know p and (2) who have the AI skill but not the Networks skill.

3. Find the pid and name of each person who has all the skills of the combined set of job skills of the highest paid persons who work for Google.

4. Find the set of companies that employ at least 3 persons who each know at least five persons. (So this query returns only one object, i.e., the set of companies specified in the query.)

5. Find the following set of sets

$$SS = \{S \mid S \subseteq \text{Skill} \wedge |S| \leq 4\}$$

   i.e., this is the set consisting of each set of job skills whose size (cardinality) is at most 4.

6. Let

$$SS = \{S \mid S \subseteq \text{Skill} \wedge |S| \leq 4\}$$

   Find the following set of sets

$$\{X \mid X \subseteq SS \wedge |X| \leq 2\}$$

7. Let A and B be sets such that $A \cup B \neq \emptyset$. The Jaccard index $J(A, B)$ is defined as the quantity

$$\frac{|A \cap B|}{|A \cup B|}$$

   The Jaccard index is a frequently used measure to determine the similarity between two sets. Note that if $A \cap B = \emptyset$ then $J(A, B) = 0$, and if $A = B$ then $J(A, B) = 1$.

   Let t be a number called a threshold. We assume that t is a float in the range $[0, 1]$.

Write a function `JaccardSimilar(t float)` that returns the set of unordered pairs $\{s_1, s_2\}$ of different skills such that the set of persons who have skill s_1 and the set of persons who have skill s_2 have a Jaccard index of at least t.

Test your function `JaccardSimilar` for the following values for t: 0, 0.25, 0.5, 0.75, and 1.

3 Nested Relations and Semi-structured databases

Consider the lecture on Nested relational and semi-structured databases. In that lecture, we considered the `studentGrades` nested relation and the `jstudentGrades` semi-structured database, and we constructed these using a PostgreSQL query starting from the Enroll relation.

8. Write a PostgreSQL view `courseGrades` that creates the nested relation of type $(\text{cno}, \text{gradeInfo}\{(\text{grade}, \text{students}\{(\text{sid})\})\})$ This view should compute for each course, the grade information of the students enrolled in this course. In particular, for each course and for each grade, this relation stores in a set the students who obtained that grade in that course.

9. Starting from the `courseGrades` view in Problem 8, solve the following query: Find each (s, C) pair where s is the sid of a student and C is the set of cnos of courses in which the student received an 'A' or a 'B' but not a 'C'. The type of your answer relation should be $(\text{sid} : \text{text}, \text{Courses} : \{(\text{cno} : \text{text})\})$.

10. Write a PostgreSQL view `jcourseGrades` that creates a semi-structured relation which stores jsonb objects whose structure conforms with the structure of tuples as described for the `courseGrades` in Problem 8. Test your view.

11. Starting from the `jcourseGrades` view in Problem 10, solve the following queries. Note that the output of each of these queries is a nested relation. Find each triple $(c1, c2, S)$ where $c1$ and $c2$ are the course numbers of two different courses and S is the set of sids of students who received an 'A' in both courses $c1$ and $c2$. The type of your answer relation should be $(\text{coursePair} : \{(\text{cno} : \text{text})\}, \text{Students} : \{(\text{sid} : \text{text})\})$.

4 Object Relational Programming

The following problems require you to write object-relational programs. Many of these require programs written in Postgres' PL/pgSQL database programming language.

12. Write a PL/pgSQL function that takes an array of integers as input and returns a new array where each element is the sum of all the elements in the input array up to and including that element. For example, if the input array is $\{1, 2, 3, 4\}$, the output array should be $\{1, 3, 6, 10\}$.

13. Write a function that takes a positive integer as input and returns an array of all the prime numbers up to and including that integer.

14. Consider a parent-child relation PC(parent, child). (You can assume that PC is a rooted tree and the domain of the attributes parent and child is int.) An edge (p, c) in PC indicates that node p is a parent of node c. Now consider a pair of nodes (m, n) in PC (m and n may be the same nodes.) We say that m and n are in the same generation when the distance from m to the root of PC is the same as the distance from n to the root of PC. Consider the following recursive query that computes the `sameGeneration` relation:

```
WITH RECURSIVE sameGeneration(m, n) AS
((SELECT parent, parent FROM PC)
UNION
(select child, child from PC)
UNION
SELECT t1.child, t2.child
FROM sameGeneration pair, PC t1, PC t2
WHERE pair.m = t1.parent and pair.n = t2.parent)
select distinct pair.m, pair.n from sameGeneration pair order by m, n;
```

Write a non-recursive function `sameGeneration()` in the language PL/pgSQL that computes the `sameGeneration` relation.