B461: Database Concepts
Assignment 6
Fall 2022

Due: 12/02/2022, Friday

To turn in your assignment, you will need to upload the following files:

- `assignment6.sql`
- `assignment6.txt`

The `assignment6.sql` contains the necessary SQL statements that solve the problems in this assignment. The `assignment6.sql` file must be such that the AI's can run it in their PostgreSQL environment.

The `assignment6.txt` file contains the results of running your queries.

Grading Rubric (100 pts total)

1. `10 pts` if the query returns expected results.
2. `0 - 9 pts` for incorrect results, the deduction of points will be gauged on how logically sound the query is.

Note :

1. We will be using a new schema for this assignment.
2. The assignment will be in two sections and will total 7 questions out of which 2 will be for extra credit.

You will need to use the data provided for the below schema

```
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 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 `SetOperationsAndPredicates.sql`

Functions

set union(A,B)	A$\cup B$
set intersection(A,B)	A$\cap B$
set difference(A,B)	AB
add element(x,A)	$x \cup A$
remove element(x,A)	A-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)	A $\in B$
is_not_in(x,A)	A$\notin B$
is_empty(A)	A ϕ
is_not_emptyset(A)	A $\neq \phi$
subset(A,B)	A$\subseteq B$
superset(A,B)	A$\supseteq B$
equal(A,B))	A = B
overlap(A,B)	A$\cap B \neq \phi$
disjoint(A,B)	A$\cap B = \phi$

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 associates 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 associates 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 `SetOperationsAndPredicates.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 SetOperationsAndPredicates.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 of 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;
```

1. Formulate the following queries in object-relational SQL.

   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 `Programming` skill and `Networks` skill.

2. Formulate the following queries in object-relational SQL.

   Find the pid and name of each person who has all the skills of the the combined set of job skills of the lowest paid persons who work for Google.
   Example: *Let A and B be the lowest paid persons at Google and let S = X, Y, Z be the combined set of their job skills. We need the pid and name of each person who has all the skills in S.*

3. Find the following set of sets

$$\{S \mid S \subseteq \text{Skill} \wedge |S| \leq 2\}.$$

   I.e., this is the set consisting of each set of job skills whose size (cardinality) is at most 2.

   Example: *Let Skill = {A, B, C } be the set of skills, then S = {{},{A},{B},{C},{A, B}, {B, C}, {A, C}}, i.e. set of all subsets of 0 (empty), size 1 and size 2 of skills.*

4. (This is an extra credit question.)
   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.

Example: *Let $A = \{p1, p2, p3, p4\}$ be set of pids who have skill $s1$ and $B = \{p2, p3\}$ be set of pids who have skill $s2$. The unordered pair of skills $\{s1, s2\}$ will feature in the result of the function $JaccardSimilar(0.5)$ because the jaccard similarity of set of persons having skills $s1$ and $s2$ is calculated as $\frac{|A \cap B|}{|A \cup B|}$ = 0.5*

2 Nested Relations and Semi-structured databases

Consider the lecture on Nested and Semi-structured Data models. In that lecture, we considered the `studentGrades` nested relation and we constructed it using a PostgreSQL query starting from the `Enroll` relation.

5. (For a relatable example for this question, look into slide 14 from Week 10 Nested and Semi-Structured DataModel.)

   Write a PostgreSQL view `courseGrades` that creates the nested relation of type

   $$(\texttt{cno}, \texttt{gradeInfo}\{(\texttt{grade}, \texttt{students}\{(\texttt{sid})\})\})$$

   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.

6. Starting from the `courseGrades` view in Problem (5) solve the following queries:
   Find each cno c where c is a course in which all students received the same grade.

7. (For a relatable example for this question, look into slide 30 from Week 10 Nested and Semi-Structured DataModel.)
   (This is an extra credit question.)
   Write a PostgreSQL view jcourseGrades that creates a semi-structured database which stores jsonb objects whose structure conforms with the structure of tuples as described for the courseGrades in Problem (5). Test your view.