B561 Assignment 5
Fall 2021

Object-relational databases
Nested relational and semi-structured databases
(Draft)

Dirk Van Gucht

For this assignment, you will need the material covered in the lectures

- Lecture 13: Object-relational databases and queries
- Lecture 14: Nested relational and semi-structured databases

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 `Assignment-Script-2021-Fall-assignment5.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. You will also see several problems that are listed as practice problems. You should not include your solutions for practice problems in the materials you submit for this assignment.

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)	$A - B$
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)	$x \in A$
is_not_in(x,A)	$x \notin A$
is_empty(A)	$A = \emptyset$
is_not_emptyset(A)	$A \neq \emptyset$
subset(A,B)	$A \subseteq B$
superset(A,B)	$A \supseteq B$
equal(A,B)	$A = B$
overlap(A,B)	$A \cap B \neq \emptyset$
disjoint(A,B)	$A \cap B = \emptyset$

We now turn to the problems in this section. You will need use the data provided for the `Person`, `Company`, `companyLocation`, `worksFor`, `jobSkill`, `personSkill`, and `Knows` relations. But before turning to the problems, we will introduce various object-relational views defined over these relations:[1]

- The view `companyHasEmployees(cname,employees)` which associates with each company, identified 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;
```

[1] The various `order by` clauses in these views are not essential: they simply aid to read the data more easily.

- 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;
```

- 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;
```

- The view `knowsPersons(pid,persons)` which associates with each person, 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;
```

- The view `isKnownByPersons(pid,persons)` which associates with each person, identified 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;
```

- The view `personHasSkills(pid,skills)` which associates with each person, 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;
```

- 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`.

Example 1 *Consider the query* "Find the pid of each person who knows a person who has a salary greater than 55000."[2]

```
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.*

[2] In this example, focus on the `is_in` predicate.

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

Example 2 *Consider the query* "Find the pid and name of each person p who (1) has both the AI and Programming and (2) knows at least 5 persons, and report the number of persons who know p."[3]

```
select p.pid, p.pname, (select cardinality(kp.persons)
                        from   isKnownByPersons kp
                        where  kp.pid = p.pid) as ct_knownByPersons
from   person p
where  p.pid in (select ps.pid
                 from   personHasSkills ps
                 where  subset(’"AI", "Programming"’, ps.skills)) and
       cardinality((select kp.persons
                    from   knowsPersons kp
                    where  kp.pid = p.pid)) >= 5;
```

Example 3 *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'".[4]

```
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.

    (a) Find the cname and headquarter of each company that employs at least two persons who each have both the AI and the Programming job skills.

    (b) Find each skill that is not a job skill of any person who works for Yahoo or for Netflix.

    (c) 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.)

    (d) 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.

[3] In this example, focus on the set (array) construction `’{"AI", "Programming"}’` and the `subset` predicate. Also focus on the use of `cardinality` function.

[4] In this example, focus on (1) the `set_difference` operation and (2) the `unnest` operation followed by a set (`array`) construction.

(e) Find each unordered pairs $\{s_1, s_2\}$ of different skills s_1 and s_2 such that the number of employees who have skill s_1 and who make strictly more than 50000 is less than the number of employees who have skill s_2 and who make at most 50000.

(f) (**Practice Problem: not-graded**).

Find each (c, p) pair where c is the cname of a company and p is the pid of a person who works for that company and who is known by all other persons who work for that company.

(g) (**Practice Problem: not-graded**).

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 Yahoo.

2. Find the following set of sets

$$\{S \mid S \subseteq \mathsf{Skill} \wedge |S| \leq 3\}.$$

I.e., this is the set consisting of each set of job skills whose size (cardinality) is at most 3.

3. (**Practice Problem: not-graded**).

Reconsider Problem 2. Let

$$\mathcal{S} = \{S \mid S \subseteq \mathsf{Skill} \wedge |S| \leq 3\}.$$

Find the following set of sets

$$\{X \mid X \subseteq \mathcal{S} \wedge |X| \leq 2\}.$$

4. Let t be a number called a *threshold*. We say that a (unordered) pair of different person pids $\{p_1, p_2\}$ *co-occur* with frequency at least t if there are at least t skills that are skills of both the person with pid p_1 and the person with pid p_2.

Write a function `coOccur(t integer)` that returns the (unordered) pairs $\{p_1, p_2\}$ of person pid that co-occur with frequency at least `t`.

Test your `coOccur` function for t in the range $[0, 3]$.

5. 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.

2 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.

6. 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 sids students who obtained that grade in that course.

   Test your view.

7. Starting from the `courseGrades` view in Problem 6 solve the following queries:

   (a) Find each pair (c, S) where c is the cno of a course and S is the set of sids of students who received an 'A' or a 'B' in course c. The type of your answer relation should be $(\text{cno} : \text{text}, \text{Students} : \{(\text{sid} : \text{text})\})$.

   (b) Find each (s, C) pairs where s is the sid of a students and C is the set of cnos of courses in which the student received an 'A'. The type of your answer relation should be $(\text{sid} : \text{text}, \text{Courses} : \{(\text{cno} : \text{text})\})$.

   (c) (**Practice Problem: not-graded**).

   Find each cno c where c is a course in which all students received the same grade.

8. 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 6.

   Test your view.

9. Starting from the `jcourseGrades` view in Problem 8 solve the following queries. Note that the output of each of these queries is a nested relation.

   (a) Find each pair (c, s) where c is the cno of a course and s is the sid of a student who did not received an 'A' in course c. The type of your answer relation should be (cno:text, sid:text).

   (b) Find each pair $(\{c_1, c_2\}, S)$ where c_1 and c_2 are the course numbers of two different courses and S is the set of sids of students who received a 'B' in both courses c_1 and c_2. The type of your answer relation should be $(\text{coursePair} : \{(\text{cno} : \text{text})\}, \text{Students} : \{(\text{sid} : \text{text}))\}$.