Object-Relational Databases

You will need to submit 2 files:

1. One such file is a .sql file that contains the SQL code relating to problems.
2. A second file with .txt extension that contains the answer of running your SQL code for the problems.

1 Set operations and predicates

Before beginning with the problems, we define some useful functions and operators on sets. Notice that these functions are defined polymorphically. You are encouraged/expected to use these function where appropriate.

In these functions let A and B be sets.

- Set union $A \cup B$:

```
create or replace function setunion(A anyarray, B anyarray)
returns anyarray as
$$
select array( select unnest(A) union select unnest(B) order by 1);
$$ language sql;
```

- Set intersection $A \cap B$:

```
create or replace function setintersection(A anyarray, B anyarray)
returns anyarray as
$$
select array( select unnest(A) intersect select unnest(B) order by 1);
$$ language sql;
```

- Set difference $A - B$:

```
create or replace function setdifference(A anyarray, B anyarray)
returns anyarray as
$$
select array( select unnest(A) except select unnest(B) order by 1);
$$ language sql;
```

- Set membership $x \in A$:

```
create or replace function isIn(x anyelement, A anyarray)
   returns boolean as
$$
select x = SOME(A);
$$ language sql;
```

- Subset test $A \supseteq B$:

```
create or replace function subset(A anyarray, B anyarray)
   returns boolean as
$$
select A <@ B;
$$ language sql;
```

- Superset test $A \supseteq B$:

```
create or replace function superset(A anyarray, B anyarray)
   returns boolean as
$$
select A @> B;
$$ language sql;
```

- Overlap test $A \cap B \neq \emptyset$:

```
create or replace function overlap(A anyarray, B anyarray)
   returns boolean as
$$
select A && B;
$$ language sql;
```

- Disjointness test $A \cap B = \emptyset$:

```
create or replace function disjoint(A anyarray, B anyarray)
   returns boolean as
$$
select not A && B;
$$ language sql;
```

2 Formulating Queries in the Object-Relational Model

In the following problems, use the data provided for the Person, Company, jobSkill, worksFor, companyLocation, Knows, and personSkill relations.

Example 1 *Consider the view* `companyEmployees(cname,employees)` *which associates with each company, identified by a cname, the set of pids of persons who work for that company. This view can be defined as follows:*

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

An alternative definition is as follows:[1]

```
create or replace view companyEmployees as
  (select cname, array_agg(pid order by pid) as employees
   from   worksfor
   group by cname
   union
   select cname, array[]::int[]
   from   company
   where  cname not in (select cname from worksfor)
   order by cname)
```

[1] Observe that the second component of the `union` in the definition of this view is required since there may be companies that have no employees. On large data, the second view runs faster than the first.

1. In this question you are asked to define various views. Each of these views is defined in a way that is similar to the way in which the view `companyEmployees` in Example 1 is defined.

   (a) Define a view `cityHasCompanies(city,companies)` which associates with each city the set of cnames of companies who are located in that city. (Test your view.)

   (b) Define a view `companyLocations(cname,locations)` which associates with each company, identified by a cname, the set of cities in which that company is located. (Test your view.)

   (c) Define a view `knowsPersons(pid,persons)` which associates with each person, identified by a pid, the set of pids of persons he or she knows. Observe that a person may know no one. (Test your view.)

   (d) Define a 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. (Test your view.)

   (e) Define a view `personHasSkills(pid,skills)` which associates with each person, identified by a pid, his or her set of job skills. Observe that a person may not have any job skills. (Test your view.)

   (f) Define a view `skillOfPersons(skills,persons)` which associates with each job skill the set of pids of persons who have that job skill. Observe that there may be job skills that are not job skills of any person. (Test your view.)

2. In this question, you are asked to formulate queries in object-relational SQL. You should use the set operations and set predicates defined in Section 1, the view `companyEmployees`, the views defined in Problem 1, and the relations `person`, `company`, `jobSkill`, and `worksFor`. So, in particular, 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 the views defined in Problem 1.

   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.

   Example 2 *Consider the query* "Find the pid of each person who knows a person who has a salary greater than 55000."
   The formulation for this query

```
select distinct k.pid1
from   knows k, worksfor w
where  k.pid2 = w.pid and w.salary > 55000
order by 1;
```

is **not** *permitted since it uses the relation* `Knows`*. Rather, you should use an object-relational SQL formulation:*

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

Or, alternatively, the equivalent, but less readable, object-relational SQL query

```
select distinct unnest(k.persons)
from   worksfor w, knownByPersons k
where  w.pid = k.pid and w.salary > 55000;
```

Example 3 *Consider the query* "Find the pid and name of each person who has both the 'Databases' and 'Programming' skills."

```
select p.pid, p.name
from   person p
where  p.pid in (select ps.pid
                 from   personHasSkills ps
                 where  subset(array['Databases', 'Programming'], ps.skills))
order by 1,2;
```

This query can also be specified using the `isIn` *predicate:*

```
select p.pid, p.name
from   person p
where  isIn(p.pid, array(select ps.pid
                         from   personHasSkills ps
                         where  subset(array['Databases', 'Programming'], ps.skills)))
order by 1,2;
```

In fact, the SQL set predicates `in` *and* `not in` *are redundant since they can be simulated using the* `isIn` *predicate along with set (i.e., array) construction.*

Example 4 *Consider the query* "Find the pid and name of each person who knows at least 5 person."

```
select p.pid, p.name
from person p
where cardinality((select k.persons
                   from   knowsPersons k
                   where  k.pid = p.pid)) >= 5;
```

Example 5 *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 'Amazon'".

```
select p.pid, p.name, setdifference((select ps.skills
                                     from   personHasSkills ps
                                     where  ps.pid = p.pid),
                                    (select array(select unnest(ps.skills)
                                                  from   personHasSkills ps
                                                  where  isIn(ps.pid, (select employees
                                                                       from   companyEmployees
                                                                       where  cname = 'Amazon')))))
from   person p;
```

Example 6 *Consider the query "Find each (p, c) pair where p is the pid of a person and c is the cname of a company located in 'Bloomington' and such that each person with birthyear at most 1990 and who is known by p is an employee of company c."*

```
select p.pid, c.cname
from   person p, company c
where  isIn(c.cname, (select companies
                      from   cityHasCompanies
                      where  city = 'Bloomington')) and
       subset(array(select p1.pid
                    from   person p1
                    where  p1.birthyear <= 1990 and
                           isIn(p1.pid, (select k.persons
                                         from   knowspersons k
                                         where  k.pid = p.pid))),
              array(select p1.pid
                    from   person p1
                    where  isIn(p1.pid,(select c1.employees
                                        from   companyEmployees c1
                                        where  c1.cname = c.cname))));
```

Now formulate the following queries in object-relational SQL.

(a) Find the pid and name of each person who knows at least 2 persons who work for 'Amazon'.

(b) Find the pid and name of each person who knows all persons who work for 'Amazon' and who make at most 40000.

(c) Find each skill that is not among the skills of employees of companies located in 'Bloomington'

(d) Find each skill that is not among the skills of employees of companies that have more than 5 employees.

(e) Find the pid of each person who only has skills that are skills of persons who make less than 50000 and who work at 'Amazon'.

(f) Find the pid of each person who has all but 4 job skills.

(g) Find the pid and name of each person along with the set of persons he or she knows and who work for 'Amazon'.

(h) Find the pairs (p_1, p_2) such that not all persons who know person p_1 are persons that are known by person p_2.

(i) Find the pid of each person along with the set of his of her skills that are among the skills of employees who work for 'Amazon' or for 'Google'.