Fall 2022 Final
Preamble

1 Relational Database Schema

For some of the questions in this final, we will use the following database schema:[1]

```
Person(pid, pname, city)
Company(cname, headquarter)
Skill(skill)
worksFor(pid, cname, salary)
personSkill(pid, skill)
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. The `pid1` and `pid2` attributes are the pids of persons such that person with pid '`pid1`' knows the person with pid '`pid2`'.

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.

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 (p_1, p_2) in `Knows` indicates that the person with pid p_1 knows the person with pid p_2. (It is possible that a person knows no one and that a person is not known by any one.)

The domain for the attributes `pid` and `salary` 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`;
- `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`;
- `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`.

You can use the following name for relation names and for tuple variables over these relations:

[1] The primary key, which may consist of one or more attributes, of each of these relations is underlined.

Relation	Variable names
Person	p, p_1, p_2, etc
Company	c, c_1, c_2, etc
Skill	s, s_1, s_2, etc
worksFor	w, w_1, w_2, etc
personSkill	ps, ps_1, ps_2, etc
Knows	k, k_1, k_2, etc

2 Object-relational database schema

For the problem related to object relational databases, you can only use the relations `Person`, `Skill`, `Company`, and `worksFor`. Thus, you can not use the relations `personSkill` and `Knows`. Rather, for such problems, you need to use the following object-relational functions:

- The function `personHasSkills(pid)` which associates with each person, identified by a pid, his or her set of job skills.

```
create or replace function personHasSkills(p int)
returns text[] as
$$
select array(select skill
             from   personSkill
             where  pid = p order by 1);
$$ language sql;
```

- The function `skillOfPersons(skill)` which associates with each skill the set of pids of persons who have that skill.

```
create or replace function skillOfPersons(s text)
returns int[] as
$$
select array(select pid
             from   personSkill
             where  skill = s order by 1);
$$ language sql;
```

- The function `knowsPersons(pid)` which associates with each person, identified by a pid, the set of pids of persons who this person knows.

```
create or replace function knowsPersons (p int)
returns int[] as
$$
select array(select pid2
             from   Knows
             where  pid1 = p order by 1);
$$ language sql;
```

- The function `isKnownByPersons(pid)` which associates with each person, identified by a pid, the set of pids of persons who know this person.

```
create or replace function isKnownByPersons(p int) as
$$
select array(select pid1
             from   Knows
             where  pid2 = p order by 1);
$$ language sql;
```

3 Tuple Relational Calculus (TRC) textual notation

In order to type TRC formulas during the exam, if needed, you need to use the following symbol-to-text translation table:

TRC Symbol	TRC Text
$\forall\, t$	forall t
$\exists\, t$	exists t
$\vee$	or
$\wedge$	and
$\neg$	not
$\rightarrow$	->

For example, the formula

$$\exists t(R(t) \wedge (\forall s(U(s) \rightarrow (t.A = s.b \vee V(t)))$$

would be written as

$$\text{exists}\, t(R(t) \text{ and } (\text{forall}\, s(U(s) \texttt{->} (t.A = s.b \text{ or } V(t))).$$

4 Relational Algebra: standard and textual notations for RA expressions and RA conditions

In the first table, you will see in the first column the standard notation for RA expressions, and in the second column their corresponding textual notation to be used during the exam. In the second table, you will see in the first column the standard notation for RA conditions, and in the second column their corresponding textual notation to be used during the exam. In these tables

- R denotes a relation of the database schema under consideration
- **a** denotes a constant
- E and F denote RA expressions
- C, C_1, C_2 denote conditions
- L denotes an attribute list
- A and B denote attributes

RA standard notation	**RA textual notation**
R	R
$(A : \mathbf{a})$	$(A : \mathbf{a})$
$E \cup F$	$E\,\texttt{union}\,F$
$E \cap F$	$E\,\texttt{intersect}\,F$
$E - F$	$E - F$
$\sigma_C(E)$	$\texttt{sigma\_}\{C\}(E)$
$\pi_L(E)$	$\texttt{pi\_}\{L\}(E)$
$E \times F$	$E \times F$
$E \bowtie_C F$	$E\,\texttt{join\_}\{C\}\,F$
$E \bowtie F$	$E\,\texttt{join}\,F$
$E \ltimes F$	$E\,\texttt{semijoin}\,F$
$E \,\overline{\ltimes}\, F$	$E\,\texttt{antisemijoin}\,F$
(E)	(E)

RA standard notation	**RA textual notation**
$A = \mathbf{a}$	$A = \mathbf{a}$
$A \neq \mathbf{a}$	$A\,!{=}\,\mathbf{a}$
$A < \mathbf{a}$	$A < \mathbf{a}$
$A \leq \mathbf{a}$	$A <{=} \mathbf{a}$
$A > \mathbf{a}$	$A > \mathbf{a}$
$A \geq \mathbf{a}$	$A >{=} \mathbf{a}$
$A = B$	$A = B$
$A \neq B$	$A\,!{=}\,B$
$A < B$	$A < B$
$A \leq B$	$A <{=} B$
$A > B$	$A > B$
$A \geq B$	$A >{=} B$
$C_1 \wedge C_2$	$C_1\,\texttt{and}\,C_2$
$C_1 \vee C_2$	$C_1\,\texttt{or}\,C_2$
$\neg\, C$	not C
(C)	(C)

For example, consider the following query: "Find the sid of each student who is born after 1990 and who takes only courses that enroll at least two students."

RA standard notation: Consider the following RA expressions A and B in standard RA notation. (We use the notation E to denote the relation `Enroll`.)

$$\begin{aligned} A &= \pi_{sid}(\sigma_{byear>1990}(\texttt{Student})) \\ B &= \pi_{E_1.sid}(E_1 \bowtie_{E_1.sid=E_2.sid \wedge E_1.cno \neq E_2.cno} E_2) \end{aligned}$$

Then the RA expression for the query in standard RA notation is the following:

$$A - \pi_{sid}(E - E \ltimes B)$$

RA textual notation

In the table below, we show the above RA expressions along with their corresponding RA expressions in textual notation:

RA notation	RA textual notation
$A = \pi_{sid}(\sigma_{byear>1990}(\texttt{Student}))$	pi_{sid}(sigma_{byear > 1990}(Student)
$B = \pi_{E_1.sid}(E_1 \bowtie_{E_1.sid=E_2.sid \wedge E_1.cno \neq E_2.cno} E_2)$	pi_{E1.sid}(E1 join_{E1.sid = E2.sid and E1.cno != E2.cno} E2)
$A - \pi_{sid}(E - E \ltimes B)$	A - pi_sid(E - E semijoin B)

5 Polymorhically Defined Set Functions and Predicates

For the problem related to object relational databases, you need to use the following polymorphically defined set-returning functions and set predicates defined over sets that are represented by arrays. In addition, you can use the array cardinality function `cardinality`, the `array` and `array_agg` constructors, and the `UNNEST` operator, and analogous functions and constructor for the JSON type.

Set functions

set_union(A,B)	$A \cup B$
set_intersection(A,B)	$A \cap B$
set_difference(A,B)	$A - B$

Set predicates

isIn(x,A)	$x \in A$
isNotIn(x,A)	$x \in A$
isEmpty(A)	$A = \emptyset$
isNotEmpty(A)	$A = \emptyset$
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)	$A \cap B = \emptyset$

Here are the codes for these functions and predicates:

```
-- Set Operations

-- Set union:
create or replace function set_union(A anyarray, B anyarray)
returns anyarray as
$$
   select array(select unnest(A) union select unnest(B) order by 1);
$$ language sql;

-- Set intersection:
create or replace function set_intersection(A anyarray, B anyarray)
returns anyarray as
$$
   select array(select unnest(A) intersect select unnest(B) order by 1);
$$ language sql;

-- Set difference:
create or replace function set_difference(A anyarray, B anyarray)
returns anyarray as
$$
   select array(select unnest(A) except select unnest(B) order by 1);
$$ language sql;

-- Set Predicates:

-- Set membership
create or replace function isIin(x anyelement, A anyarray)
returns boolean as
$$
   select x = SOME(A);
$$ language sql;

-- Set non membership:
create or replace function isNotIn(x anyelement, A anyarray)
returns boolean as
$$
   select not(x = SOME(A));
$$ language sql;

-- emptyset test:
create or replace function is_empty(A anyarray)
returns boolean as
$$
   select A <@ '{}';
$$ language sql;
```

```
-- not emptyset test:
create or replace function is_empty(A anyarray)
returns boolean as
$$
   not select A <@ '{}';
$$ language sql;

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

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

-- Equality test:
create or replace function equal(A anyarray, B anyarray)
returns boolean as
$$
   select A <@ B and A @> B;
$$ language sql;

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

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