EXERCISE 8

Andrea Crotti (#299466), Can Liu (#286105), Sebastian Roidl (#281941)

16 Dicembre 2009

1. Query Languages (5 pt.)

1. Assume that no two columns of relations have the same name.

Show that a query in the normal form $\pi_c(\sigma_F$ (R1 x R2 x … x Rn)) can be expressed in Domain Relational Calculus (DRC), where c is a vector of columns, F is a boolean formula built from conjunctions of atoms in the form ci = cj or ci = constant (ci and ci are columns). To show that we must simply demonstrate that the operators involved here are also expressible in DRC. So we have that:

- Projection is expressible in DRC
- Cross is translated to conjunction of all the occurring predicates using all distinct *ci* variables
- Selection is done by merging variables or replacing variables with constants according to the selection predicate.

2. What does "relational completeness" mean?

Show that SQL is relational complete by enumerating SQL constructs corresponding to selection, projection, cartesian product, union, and difference. Give two examples of SQL constructs/semantics not expressible in relational algebra (RA).

A language is *relational complete* if it can express all possible queries expressible by RA. If we can map the RA operations to SQL constructs than we have shown that SQL is in fact relational complete.

RA	SQL
selection	where
projection	select
Join	from
union	union
difference	except

SQL is more expressive than relational algebra, for example these SQL operators are not found in relational algebra:

- ORDER BY
- GROUP BY
- UPDATE

  $\vdots$

3. Suppose we have three tables VIP(id), Employee(id), and Male(id). Translate the following SQL query into relational calculus and relational algebra.

select Male.id from VIP, Employee, Male where VIP.id=Male.id or Employee.id=Male.id

Relational calculus

- Domain

  {m.id | Male(m) $\wedge((\exists e Employee(e) \wedge e.id = m.id) \vee (\exists v VIP(v) \wedge v.id = m.id))$}

- Tuple

m.id OF EACH m in Male: SOME v in VIP (Some e in Employee (m.id=v.id $\vee m.id = e.id$)

Relational algebra

$\pi_{\text{Male.id}}(\sigma_{\text{Vip.id=Male.id} \vee \text{Employee.id=Mal.id}}(\text{VIP} \times \text{Employee} \times \text{Male})$

4. For the following database (VIP is empty), what is the result of the query above? What is the result of π_ϕ(Employee) (Employee)?

The result of the above query gives nothing as result, because the VIP table doesn't contain any tuple and the join becomes also empty.

```
CREATE table VIP(id INTEGER, PRIMARY KEY(id));
CREATE table EMPLOYEE(id INTEGER, PRIMARY KEY(id));
CREATE table MALE(id INTEGER, PRIMARY KEY(id));

insert into MALE VALUES (1);
insert into EMPLOYEE VALUES (1);
insert into EMPLOYEE VALUES (2);

select MALE.id
from EMPLOYEE, MALE, VIP
where VIP.id=MALE.id or EMPLOYEE.id=Male.id;
```

π_ϕ**(Employee) (Employee)**

5. Figure 1 shows the flow of a query through a DBMS, in which different forms are used to represent a query at different stages. Fill in the three blanks with the corresponding query languages (i.e., SQL, RC, RA).

STEP	LANG
1	SQL
2	RC
3	RA

2. Query Formulation (15 pt.)

Formulate the following queries as expressions in relational algebra, tuple relational calculus, domain relational calculus and SQL: I used this schema to try the queries with sqlite:

```
CREATE TABLE lives(person_name PRIMARY KEY, city, street);
CREATE TABLE works(person_name PRIMARY KEY, company_name, salary);
CREATE TABLE located(company_name PRIMARY KEY, city);
CREATE TABLE boss(person_name PRIMARY KEY, manager_name);

insert into lives values(1, 'mantova', 'pippo');
insert into lives values(2, 'brescia', 'pippo');
insert into lives values(3, 'brescia', 'abc');
insert into lives values(4, 'Hamburg', 'abc');
insert into lives values(5, 'nonwork', 'df');

insert into works values(1, 'MyComp', 100);
insert into works values(2, 'BigComp', 200);
insert into works values(3, 'MyComp', 100000);
insert into works values(4, 'Ham', 200);

insert into located values('MyComp', 'brescia');
insert into located values('BigComp', 'verona');
insert into located values('Ham', 'Hamburg');

insert into boss values(1, 'firstboss');
insert into boss values(2, 'secondboss');
insert into boss values(4, 'third');
```

a) Find name and city of all persons who work for the company 'MyComp' and earn less than 10000.

Relational Algebra:

$\pi_{\text{person\_name,city}}\ (\sigma_{\text{company\_name='MyComp'}\wedge\text{salary}<10000}\ (\text{ lives } \bowtie works))$

Tuple Calculus:

$\{\text{n.person\_name,n.city} \mid \text{lives(n)} \wedge \exists y(works(y) \wedge n.person\_name = y.person\_name \wedge y.salary < 10000 \wedge y.company\_name =' MyComp')\}$

Domain Calculuis:

$\{\text{p, c} \mid \exists st(lives(p, c, st) \wedge \exists f, g(works(p, f, g) \wedge g < 10000))\}$

SQL

Using an explicit join on the person_name attribute

```
select p.person_name, city from lives p join works w
on p.person_name=w.person_name
where (w.salary < 10000 and w.company_name='MyComp')
```

b) Find the names of all persons, who don't work for 'MyComp' (or do not work at all).

Relational Algebra:

$\pi_{\text{person\_name}}(\text{lives})$ - $\pi_{\text{person\_name}}(\sigma_{\text{company\_name}='\text{MyComp}'(\text{works}})$

Tuple Calculus:

{ p.person_name | lives(p) $\wedge\exists y \wedge works(y)(p.person\_name = y.person\_name \wedge$ $y.company\_name! = `MyComp')\}$

Domain Calculus:

{name | lives(name, _ , $_)$ $\wedge works(name, comp,_)$ $\wedge comp! = `MyComp'\}$

SQL

First I take all the elements that don't work for any company and put them together with persons that don't work for 'MyComp'.

```
select person_name from lives
except
select person_name from works
union
select lives.person_name
from lives join works on lives.person_name=works.person_name
where (works.company_name <> 'MyComp')
```

c) Find the names of all persons, who live in a city that the company they are working for is not located in.

Relational Algebra:

$\pi_{\text{person\_name}}$ (lives $\bowtie$ $works$ $\bowtie$ (π_{city} (lives) X $\pi_{\text{company\_name}}$(works) - located))

Tuple Calculus:

{ p.person_name | lives(p) $\wedge \exists x \wedge works(x) \wedge \exists y \wedge located(y)(p.person\_name = x.person\_name \wedge x.company\_name = y.company\_name \wedge y.city! = p.city)\}$

Domain Calculus:

{ p | lives (p,c,st) $\wedge \exists works(p, f, g) \wedge \exists located(f, c_1) \wedge c! = c_1$

SQL

Here we concatenate 2 joins on the person and on the company$_{\text{name}}$.

```
select p.person_name,p.city
from lives p join works w on p.person_name=w.person_name
join located l on w.company_name=l.company_name
where l.city <> p.city;
```

d) Find the names of all managers, whose company is not placed in Munich or Hamburg.

Relational Algebra:

$\pi_{\text{manager\_name}}(\text{boss})$-$(\pi_{\text{manager\_name}}(\text{boss} \bowtie \pi_{\text{manager\_name=person\_name}}(\text{works}) \bowtie (\sigma_{\text{city='Munich'}}$ (located) U $\sigma_{\text{city='Hamburg'}}(\text{located}))))$

Tuple Calculus:

{m.manager_name | boss(m) $\wedge \exists x \wedge located(x) \exists y \wedge works(y)(m.manager\_name = y.person\_name \wedge x.city = y.city \wedge y.city! = `Hamburg' \wedge y.city! = `Munich')\}$

Domain Calculus:

{ m | $\exists p(boss(p, m) \backslash exists f, s located(f, s) \exists g(works(m, f, g) \wedge s! = `Munich' \wedge s! = `Hamburg'))\}$

SQL

Three join and one condition.

```
select m.manager_name
from boss m join works w
on m.person_name=w.person_name
```

```
    join located l on l.company_name=w.company_name
    where l.city <> 'Hamburg'
```

e) Find the names of all companies that are located in exactly the same cities as 'MyComp', assuming each company is located in some city.

Relational Algebra:

$\pi_{\text{company\_name}}\,(\sigma_{\text{C2='MyComp'}\wedge\text{city=city2}}(\text{located X p}_{\text{C2}\leftarrow\text{company\_name}}(\text{p}_{\{}city2 \leftarrow company\_name(located))))$

Tuple Calculus:

$\{\text{t(company\_name)} \mid \text{located (t)} \wedge \neq (\exists l2)(located(l2)\wedge(l2company\_name = tcompany\_name)\wedge \neq ((\exists myCL)(located(myCL)\wedge(myCLcompany\_name = `MyComp')\wedge \neq (\exists l_2)(\text{located}(\text{l}_2) \wedge(l_2company\_name = tcompany\_name)\wedge (l_2city = myCLcity)))\}$

Domain Calculus:

$\{ \text{ f } \mid \exists s(located(f, s)\wedge located(myCompany, s)\wedge myCompany = `MyComp')\}$

SQL

Innested selection used to get the location of 'MyComp', we also remove the trivial case where company is 'MyComp' in the end.

```
    select l.company_name
    from located l
    where l.city = (
          select city from located
          where company_name='MyComp')
          and l.company_name <> 'MyComp'
```