Exam 1 Practice Problems with Solutions

1 Pure SQL and General SQL

1.1 Pure SQL and General SQL

For this exam, we distinguish between Pure SQL and general SQL. Below we list the features that are allowed in Pure SQL and general SQL, respectively. **Comment**: the SQL JOIN operations are **not** allowed in Pure SQL.

Features allowed in Pure SQL
`SELECT ... FROM ... WHERE`
`UNION, INTERSECT, EXCEPT`
`IN` and `NOT IN` predicates
`ALL` and `SOME` predicates
`EXISTS` and `NOT EXISTS` predicates
`VIEW`s and user-defined `FUNCTION`s that can only use the above SQL features

Features allowed in general SQL
all the features of Pure SQL
aggregate functions `COUNT`, `SUM`, `AVERAGE`, `MIN` and `MAX`
`GROUP BY` and `HAVING` clauses
`VIEW`s and user-defined `FUNCTION`s that can use all of the above SQL features

2 Problems about Pure SQL and General SQL

For the problems in this section, we will use the following database schema:

```
employee(eid, ename, city, cname, salary)
company(cname, city)
jobSkill(eid, skill)
manages(mid, eid)
```

In this database, we maintain a set of employees (`employee`), a set of companies (`company`), and a set of employee job skills (`jobSkill`). The `city`, `cname`, and `salary` attributes in `employee` specify the city in which the employee lives, the (unique) company the employee works for, and the employee's salary at that company. The `city` attribute in `company` indicates a city in which the company is located. (Companies may be located in multiple cities.) An employee can have multiple job skills (`jobSkill`). It is permitted that an employee has no job skills. A pair (m, e) in `manages` indicates that m denotes an employee who is a manager of another employee denoted by e. We permit that a manager manages multiple employees and that an employee can have multiple managers. (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 work for the same company. In the relation `manages`, the attributes `mid` and `eid` refer to the `eid` of the manager and the employee, respectively.

1. **Variant 1** Consider the following SQL query:

```
select not exists (select 1
                   from  employee e, company c
                   where  e.cname = c.cname and
                          c.city <> some (select c.city
                                          from   company c
                                          where  c.city = 'Bloomington'));
```

   Translate this query into an equivalent SQL query wherein the set predicates "`not exists`" and "`<> some`" are simulated using the SQL `COUNT` aggregate function.
   Solution:

```
select (select count(1)
        from   employee e, company c
        where  e.cname = c.cname and
               (select count(1)
                from   company c1
                where  c.city <> c1.city and c1.city = 'Bloomington') >= 1) = 0;
```

   Variant 2 Consider the following SQL query:

```
select exists (select 1
               from  employee e, company c
               where e.cname <> c.cname and
                     c.city = some (select c.city
                                    from   company c
                                    where  c.city <> 'Bloomington'));
```

   Translate this query into an equivalent SQL query wherein the set predicates "`exists`" and "`= some`" are simulated using the SQL `COUNT` aggregate function.
   Solution:

```
select (select count(1)
from   employee e, company c
where  e.cname <> c.cname and
               (select count(1)
                from   company c1
                where  c.city = c1.city and c1.city <> 'Bloomington') >= 1) >= 1;
```

2. **Variant 1** Consider the following constraint: "*Some employee has a skill that is not among the skills of his or her managers.*"

   Write a Pure SQL boolean query that returns `true` if this constraint is satisfied in the database, and returns `false` if this is not the case.

   Solution:

```
select exists(select 1
              from   jobskill je
              where  je.skill not in(select jm.skill
                                     from   jobskill jm
                                     where  jm.eid in (select m.mid
                                                       from   manages m
                                                       where  m.eid = je.eid)));
```

   or, if we also want to insist that the employee has a manager to qualify then

```
select exists(select 1
              from   jobskill je, manages m
              where  je.eid = m.eid and
                     je.skill not in(select jm.skill
                                     from   jobskill jm
                                     where  jm.eid in (select m.mid
                                                       from   manages m
                                                       where  m.eid = je.eid)));
```

Variant 2 Consider the following constraint: "*Not each manager has a higher salary than the salaries of the employees he or her manages.*"

   Write a Pure SQL boolean query that returns `true` if this constraint is satisfied in the database, and returns `false` if this is not the case.

   Solution

   We can rephrase this constraints as "*Some manager has a which is less than or equal to a salary some employee he or her manages.*"

```
select exists(select 1
              from   employee m
              where  m.salary <= some (select e.salary
                                       from  employee e
                                       where  (m.eid,e.eid) in
                                          (select mid, eid from manages)));
```

3. Formulate the following query in Pure SQL. In particular, you can not use SQL aggregate functions in your solution:

   Variant 1 Find the cname of each company that only employs persons who live in ‘Bloomington’.

   Solution:

```
select distinct c.cname
from   company c
where  not exists(select e.eid
                  from   employee e
                  where  e.cname = c.cname and
                         e.eid not in (select e1.eid
                                       from   employee e1
                                       where  e1.city = 'Bloomington'));
```

   Variant 2 Find the cname of each company that does not employs all persons who live in ‘Bloomington’.

   Solution:

```
select distinct c.cname
from   company c
where  exists(select e.eid
              from   employee e
              where  e.city = 'Bloomington' and
                     e.eid not in (select e1.eid
                                   from   employee e1
                                   where  e1.cname = c.cname));
```

4. Formulate the following query in Pure SQL. In particular, you can not use SQL aggregate functions in your solution:

   Variant 1 Find each job skill that is shared by all employees that are managed by the manager with mid 1000.
   Solution

```
create or replace function employeeManagedbyManager(manager int)
returns table (eid int) as
$$
select m.eid
from   manages m
where  m.mid = manager;
$$ language sql;

select distinct j.skill
from   jobskill j
where  not exists(select 1
                  from   employeeManagedbyManager(1000) e
                  where  j.skill not in (select je.skill
                                         from   jobskill je
                                         where  je.eid = e.eid));
```

   Variant 2 Find each job skill that is shared by all managers of the employee with eid 2000. **Solution**

```
create or replace function managerofEmployee(e int)
returns table (mid int) as
$$
select m.mid
from   manages m
where  m.eid = e;
$$ language sql;

select distinct j.skill
from   jobskill j
where  not exists(select 1
                  from   managerofEmployee(1) m
                  where  j.skill not in (select jm.skill
                                         from   jobskill jm
                                         where  jm.eid = m.mid));
```

5. Formulate the following query in general SQL. So now you can use aggregate functions.

   Variant 1 Find, for each company, the cname and the number or employees who have a salary that is equal to the salaries of at least two of their managers.

   Solution:

```
create or replace function SalaryofManagerofEmployee(e int)
returns table(salary int) as
$$
select m.salary
from   employee m
where  m.eid in (select m.mid from manages m where m.eid = e);
$$ language sql;

select  distinct c.cname,
   (select count(1)
    from   employee e
    where  e.cname = c.cname and
                (select count(1)
                 from   SalaryofManagerofEmployee(e.eid) s
                 where  e.salary = s.salary) = 2)
from    company c;
```

Variant 2 Find, for each company, the cname and the number or employees who have a salary that is lower than the salaries of at least three of their managers.

Solution:

```
create or replace function SalaryofManagerofEmployee(e int)
returns table(salary int) as
$$
select m.salary
from   employee m
where  m.eid in (select m.mid from manages m where m.eid = e);
$$ language sql;

select  distinct c.cname,
   (select count(1)
    from   employee e
    where  e.cname = c.cname and
                (select count(1)
                 from   SalaryofManagerofEmployee(e.eid) s
                 where  e.salary < s.salary) < 3)
from    company c;
```

6. Formulate the following query in general SQL. So now you can use aggregate functions.

   Variant 1 Find the cname of each company that is located in 'Indianapolis' and that employs the largest number of persons who do not live in 'Indianapolis'.
   Solution:

   ```
   create or replace function employeeNotLivingInBloomington(c text)
   returns bigint as
   $$
   select count(1)
   from   employee e
   where  e.cname = c and e.city <> 'Indianapolis';
   $$ language sql;

   select distinct c.cname
   from   company c
   where c.city = 'Indianapolis' and
                employeeNotLivingInBloomington(c.cname) >= ALL
                        (select employeeNotLivingInBloomington(c.cname)
                         from   company c);
   ```

   Variant 2 Find the cname of each company that is not located in 'Bloomington' and that employs the smallest number of persons who live in 'Bloomington'.

   Solution:

   ```
   create or replace function employeeLivingInBloomington(c text)
   returns bigint as
   $$
   select count(1)
   from   employee e
   where  e.cname = c and e.city = 'Bloomington';
   $$ language sql;

   select distinct c.cname
   from   company c
   where c.city <> 'Bloomington' and
                employeeLivingInBloomington(c.cname) <= ALL
                        (select employeeLivingInBloomington(c.cname)
                         from   company c);
   ```

7. In the following query with quantifiers you have to use the method of **Venn diagrams with conditions**. In particular, you need to use views and parameterized views to specify the relevant sets that are involved in these queries.

   Using this method, formulate the following query in Pure SQL. In particular, you can not use aggregate functions.

   Variant 1 Find the cname of each company located in 'Bloomington' that not only employ people who live in 'Bloomington'.
   Solution:

```
create or replace view employeeInBloomington as
(select e.eid
 from employee e
 where city = 'Bloomington');

 create or replace function employeeWorkingForCompany(c text)
 returns table (eid int) as
 $$
 select e.eid
 from  employee e
 where e.cname = c;
 $$ language sql;

 select  distinct c.cname
 from   company c
 where c.city = 'Bloomington' and
           exists (select e.eid
                   from employeeWorkingForCompany(c.cname) e
                   except
                   select e.eid
                   from employeeInBloomington e);
```

Variant 2 Find the cname of each company located in ‘Chicago’ that does not employ any person who lives in ’Bloomington’ or in ‘Indianapolis’

Solution:

```
create or replace view employeeInBloomingtonorIndianapolis as
(select e.eid
 from employee e
 where city = 'Bloomington' or city = 'Indianapolis');

 create or replace function employeeWorkingForCompany(c text)
 returns table (eid int) as
 $$
 select e.eid
 from  employee e
 where e.cname = c;
 $$ language sql;

 select  distinct c.cname
 from   company c
 where c.city = 'Chicago' and
 not exists (select e.eid
             from employeeWorkingForCompany(c.cname) e
             intersect
             select e.eid
             from employeeInBloomingtonorIndianapolis e);
```

8. In the following query with quantifiers you have to use the method of **Venn diagrams with conditions**. In particular, you need to use views and parameterized views to specify the relevant sets that are involved in these queries.

   Using this method, formulate the following query in Pure SQL. In particular, you can not use aggregate functions.

   Variant 1 Find the pairs (e_1, e_2) of different eids of employees who satisfy the following conditions:

   - employees e_1 and e_2 each have a salary that is lower than \$50,000; and
   - some managers of employee e_1 are not managers of employee e_2.

   Solution:

```
create or replace function managerOfEmployee(e int)
returns table (mid int) as
$$
select m.mid
from   manages m
where m.eid = e;
$$ language sql;

select  e1.eid, e2.eid
from    employee e1, employee e2
where  e1.eid <> e2.eid and
           e1.salary < 50000 and e2.salary < 50000 and
       exists( select m1.mid
               from   managerOfEmployee(e1.eid) m1
               except
               select m2.mid
               from   managerOfEmployee(e2.eid) m2);
```

Variant 2 Find the pairs (e_1, e_2) of different eids of employees who satisfy the following conditions:

- employees e_1 and e_2 live in the same city;
- each manager of employee e_1 is also a manager of employee e_2.

Solution:

```
create or replace function managerOfEmployee(e int)
returns table (mid int) as
$$
select m.mid
from   manages m
where m.eid = e;
$$ language sql;

select  e1.eid, e2.eid
from    employee e1, employee e2
where  e1.eid <> e2.eid and
           e1.city = e2.city and
   not exists( select m1.mid
               from   managerOfEmployee(e1.eid) m1
               except
               select m2.mid
               from   managerOfEmployee(e2.eid) m2);
```

9. In the following query with quantifiers you have to use the method of **Venn diagrams with counting conditions**. In particular, you need to use views and parameterized views to specify the relevant sets that are involved in this queries and make appropriate use of the SQL `COUNT` aggregate function.

   Formulate the following query in SQL:

   Variant 1 Find each pair (m_1, m_2) of different mids of managers who work at the same company and such that m_1 and m_2 do not manage the same set of employees who live in Bloomington.

   Solution:

```
create or replace function BloomingtonEmployeeManagedby(manager int)
returns table (eid int) as
$$
select e.eid
from   employee e
where e.city = 'Bloomington' and
        e.eid in (select m.eid
                         from manages m
                         where m.mid = manager);
$$ language sql;

select  distinct m1.mid, m2.mid
from    employee e1, manages m1, employee e2, manages m2
where e1.city = e2.city and m1.mid <> m2.mid and
        (select count(1)
         from  (select e1.eid
                from   BloomingtonEmployeeManagedby(m1.mid) e1
                except
                select e2.eid
                from   BloomingtonEmployeeManagedby(m2.mid) e2) q) = 0 and
        (select count(1)
          from  (select e2.eid
                 from   BloomingtonEmployeeManagedby(m2.mid) e2
                 except
                 select e1.eid
                 from   BloomingtonEmployeeManagedby(m1.mid) e1) q) = 0;
```

Variant 2 Find each pair (e_1, e_2) of different eids of employees whose salary is more \$50000 and such that e_1 and e_2 have the same set of managers.

Solution:

```
create or replace function ManagerofEmployee(e int)
returns table (mid int) as
$$
select distinct m.mid
from   manages m
where m.eid = e;
$$ language sql;

select  distinct e1.eid, e2.eid
from    employee e1, employee e2
where e1.eid <> e2.eid and e1.salary > 50000 and e2.salary > 50000 and
        (select count(1)
         from  (select m1.mid
                from   ManagerofEmployee(e1.eid) m1
                except
                select m2.mid
                from   ManagerofEmployee(e2.eid) m2) q) = 0 and
        (select count(1)
          from  (select m2.mid
                 from   ManagerofEmployee(e2.eid) m2
                 except
                 select m1.mid
                 from   ManagerofEmployee(e1.eid) m1) q) = 0;
```

10. **Variant 1** Let $X = (x_1, \ldots, x_n)$ and $Y = (y_1, \ldots, y_n)$ be two n-dimensional vectors of numbers. (We will assume that $n \geq 1$). For example, for $n = 3$, X could be the vector $(7, -1, 2)$ and Y the vector $(1, 1, -10)$.

    The euclidean distance between X and Y is defined as

$$\sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}.$$

We will represent the vector X with a binary relation **X**(index,value) such that (i, v) is in **X** if $x_i = v$. Analogously, we will represent the vector Y with a binary relation **Y**(index,value). For example, for the vectors $X = (7, -1, 2)$ and $Y = (1, 1, -10)$, **X** and **Y** are the following binary relations:

X

index	value
1	7
2	−1
3	2

Y

index	value
1	1
2	1
3	−10

Write a SQL function

```
CREATE FUNCTION dist() RETURNS numeric AS
$$
...
$$ LANGUAGE SQL;
```

such that, `SELECT dist()` returns the distance between vectors X and Y as represented by the relation **X** and **Y**.

You can use the function `sqrt`(z) which computes the square root of a nonnegative number z.

Solution:

```
create or replace function dist()
returns float as
$$
select sqrt(sum(power(x.value-y.value,2)))
from  X x, Y y
where x.index = y.index;
$$ language sql;
```

Variant 2 Let $X = (x_1, \ldots, x_n)$ and $Y = (y_1, \ldots, y_n)$ be two n-dimensional vectors of numbers. (We will assume that $n \geq 1$). For example, for $n = 3$, X could be the vector $(7, -1, 2)$ and Y the vector $(1, 1, -10)$.

The l_1 distance between X and Y is defined as

$$\sum_{i=1}^{n} |x_i - y_i|.$$

Here, $|x_i - y_i|$ denotes the absolute value of $x_i - y_i$.

We will represent the vector X with a binary relation **X**(index,value) such that (i, v) is in **X** if $x_i = v$. Analogously, we will represent the vector Y with a binary relation **Y**(index,value). For example, for the vectors $X = (7, -1, 2)$ and $Y = (1, 1, -10)$, **X** and **Y** are the following binary relations:

X

index	value
1	7
2	-1
3	2

Y

index	value
1	1
2	1
3	-10

Write a SQL function

```
CREATE FUNCTION dist() RETURNS numeric AS
$$
...
$$ LANGUAGE SQL;
```

such that, `SELECT dist()` returns the l_1 distance between vectors X and Y as represented by the relation **X** and **Y**.

You can use the function `abs`(z) which computes the absolute value of a number z.

Solution:

```
create or replace function dist()
returns bigint as
$$
select sum(abs(x.value-y.value))
from  X x, Y y
where x.index = y.index;
$$ language sql;
```

3 Relational Algebra

3.1 Database schema used for problems on RA

For the problems in this section, we will use the following database schema[1]:

$\texttt{Person(}\underline{\texttt{pid}}\texttt{, pname, city)}$
$\texttt{Company(}\underline{\texttt{cname,city}}\texttt{)}$
$\texttt{jobSkill(}\underline{\texttt{skill}}\texttt{)}$
$\texttt{Works(}\underline{\texttt{pid}}\texttt{, cname, salary)}$
$\texttt{personSkill(}\underline{\texttt{pid,skill}}\texttt{)}$
$\texttt{Manages(}\underline{\texttt{mid,eid}}\texttt{)}$

In this database, we maintain a set of persons (`Person`), a set of companies (`Company`), and a set of job skills (`Job_Skill`). The `city` attribute in `Person` specifies the city in which the person lives. The `city` attribute in `Company` indicates a city in which the company is located. (Companies may be located in multiple cities.) A person can be employed by at most one company. (We permit that a person is not employed.) A person can have multiple job skills (`Person_Skill`). 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 (m, e) in `Manages` indicates that m is the pid of a person who is a manager of an employee who is a person with pid e. We permit that a manager manages multiple employees and that an employee can have multiple managers. (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 in the same company. The attributes `mid` and `eid` refer to the `pid` of the employee and the manager, respectively.

The primary keys in the relations are underlined. The foreign key constraints are as follows:

Foreign key constraints
pid in Works is a foreign key referencing pid in Person
cname in Works is a foreign key referencing cname in Company
pid in PersonSkill is a foreign key referencing pid in Person
skill in PersonSkill is a foreign key referencing skill in jobSkill
mid in Manages is a foreign key referencing pid in Person
eid in Manages is a foreign key referencing pid in Person

[1]Notice that the relations are somewhat different than those in the previous Scetion.

In the problems in this section, you will be asked to write expressions in the standard notation of the Relational Algebra and into SQL with relational operators.

Recall that the standard notation for RA require writing expressions that operators σ, π, $\times$, $\bowtie_C$, $\bowtie$, $\ltimes$, $\overline{\ltimes}$, $\cup$, $\cap$, and $-$. To write your expressions, you can use abbreviates to denote subexpressions. Furthermore, you can use the following notations to denote relations:

Relation	Notation
Person	P
Company	C
jobSkill	jS
Works	W
personSkill	pS
Manages	M

3.2 Relational Algebra standard and textual notation

In the first table, you will see in the first column the standard notation for each RA operation, and in the second column its corresponding exam notation. In the second table, you will see in the first column the standard notation for each condition, and in the second column its corresponding textual notation.

In these tables,

- R denotes a relation
- **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

Standard notation	**Textual notation**
R	R
$(A : \mathbf{a})$	$(A : \mathbf{a})$
$E \cup F$	E `union` F
$E \cap F$	E `intersect` F
$E - F$	$E - F$
$\sigma_C(E)$	`sigma_`$\{C\}(E)$
$\pi_L(E)$	`pi_`$\{L\}(E)$
$E \times F$	$E \times F$
$E \bowtie_C F$	E `join_`$\{C\}$ F
$E \bowtie F$	E `join` F
$E \ltimes F$	E `semijoin` F
$E \overline{\ltimes} F$	E `antijoin` F
(E)	(E)

Standard notation	**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 `and` C_2
$C_1 \vee C_2$	C_1 `or` C_2
$\neg C$	`not` C
(C)	(C)

3.3 Formulating queries in Relational Algebra

Formulate the following queries in the Relational Algebra. You should be able to do this using the standard notation of Relational Algebra as well as in SQL with RA operations. (Do not use set predicates, user-defined functions, nor aggregate functions.) Your queries do not need to optimized.

1. Find the cname of each company located in ‘Bloomington’ and that employs only persons who have at least two job skills.

   Solution:

   Standard RA notation: Consider the following RA expressions E and C in standard RA notation

   $$\begin{array}{rcl} C & = & \pi_{cname}(\sigma_{city=\text{‘Bloomington’}}(Company)) \\ E & = & \pi_{jS_1.pid}(jS_1 \bowtie_{jS_1.pid=jS_2.pid \wedge jS_1.skill \neq jS_2.pid} jS_2)) \end{array}$$

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

   $$C - \pi_{cname}(W - W \ltimes E)$$

   RA notation in textual notation

   Since, during the exam, it is not possible to type RA symbols, we here give the textual representation of these expressions:

RA notation	RA textual notation
$C = \pi_{cname}(\sigma_{city=\text{‘Bloomington’}}(Company))$	pi_{cname}(sigma_{city = ‘Bloomington’}(Company))
$E = \pi_{jS_1.pid}(jS_1 \bowtie_{jS_1.pid=jS_2.pid \wedge jS_1.skill \neq jS_2.pid} jS_2))$	pi_{jS1.pid}(jS1 join_{jS1.pid = jS2.pid and jS1.skill <> jS2.pid} jS2))
$C - \pi_{cname}(W - W \ltimes E)$	C - pi_{cname}(W - W semijoin E)

   Formulation of the query in SQL with RA operations

```
with C as (select DISTINCT cname
           from   Company
           where  city = ‘Bloomington’),
     E as (select DISTINCT js1.pid
           from   jobSkill js1 JOIN jobskill js2
                       ON (js1.pid = js2.pid and js1.skill <> js2.skill))
select cname
from   C
EXCEPT
```

```
select cname
from   (select w.*
        from   Works w
        EXCEPT
        select w.*
        from   W NATURAL JOIN E) q
```

2. Find the name of each company that is located in ‘Bloomington’ or in ‘Indianapolis’ (or both) but not in ‘Chicago’.

$$(\pi_{cname}(\sigma_{city=Bloomington}(C)) \cup \pi_{cname}(\sigma_{city=Indianapolis}(C))) - \pi_{cname}(\sigma_{cname=Chicago}(C))$$

```
(select  c.cname
from    company c
where  c.city = 'Bloomington'
union
select
select  c.cname
from    company c
where  c.city = 'Indianapolis')
except
(select  c.cname
from    company c
where  c.city = 'Chicago'
```

3. Find the pid and name of each person who lives in a city wherein a company is located for which he or she works.

$$\pi_{pid,pname}((P \bowtie W) \bowtie_{W.cname=C.cname \wedge P.city=C.city} C))$$

```
select  distinct p.pid, p.pname
from    Person p
        natural join Works w
        join Company c ON (w.cname = c.cname and p.city = c.city)
```

4. Find the eid of each employee who has a salary that is the same as that of at least one of his or her managers.

$$\pi_{W_1.pid}(\sigma_{W_1.salary=W_2.salary}(W_1 \bowtie_{W_1.pid=M.eid} M \bowtie_{M.mid=W_2.pid} W_2))$$

5. Find the eid of each employee who is not a manager.

$$\pi_{pid}(W) - \pi_{mid}(M)$$

6. Find the pid and pname of each person who has a job skill that is not a job skill of any other person.

$$\pi_{pid,pname}(P \bowtie pS) - \pi_{pid,pname}((P \bowtie pS) \bowtie_{P.pid \neq pS_1.pid \wedge pS.skill = pS_1.skill} pS_1)$$

7. Find the pairs (m_1, m_2) of different mids of managers who work at the same company and such that m_1 manages non of the employees managed by m_2.

$$\begin{array}{lcl} E & = & \pi_{M_1.mid,M_2.mid}((M_1 \bowtie_{M_1.mid=W_1.pid} W_1) \bowtie_{W_1.cname=W_2.cname \wedge M_1.mid \neq M_2.mid} (M_2 \bowtie_{M_2.mid=W_2.pid} W_2)) \\ F & = & \pi_{M_1.mid,M_2.mid}(M_1 \bowtie_{M_1.eid=M_2.eid} M_2) \end{array}$$

E is the set of different manager pairs that work at the same company. F is the set of manager that manages at least one employee jointly. Therefore, the result is

$$E - F$$

8. Find the pid of each person who has a job skill that is not among the job skills of any of his or her managers.

$$\pi_{PS.pid}(PS - \pi_{PS_1.pid,PS_1.skill}((PS_1 \bowtie_{PS_1.skill=PS_2.skill} PS_2) \bowtie_{M.eid=PS_1.pid \wedge\ M.mid=PS_2.pid} M))$$

9. Find each pair (m, s) where m is a manager mid and s is a skill that is not among any of the skills of the employees he or she manages.

$$\pi_{M.mid,JS.skill}((M \times JS) - \pi_{M.eid,M.pid,JS.skill}((M \times JS) \bowtie_{JS.skill=PS.skill \wedge M.eid=PS.pid} PS))$$

10. Find the pid of each person who has fewer than 2 job skills.

$$\pi_{pid}(Person) - \pi_{PS_1.pid}(PS_1 \bowtie_{PS_1.pid=PS_2.pid \wedge PS_1.skill \neq PS_2.skill} PS_2)$$

11. Find the pid of each person who has exactly 1 job skill.

$$\pi_{pid}(PersonSkill) - \pi_{PS_1.pid}(PS_1 \bowtie_{PS_1.pid=PS_2.pid \wedge PS_1.skill \neq PS_2.skill} PS_2)$$

12. Find the name of each company that does not employ all persons who live in ‘Bloomington’.

$$\pi_{cname}(\pi_{pid}(\sigma_{city=Bloomington}(P)) \times \pi_{cname}(C) - \pi_{pid,cname}(W))$$

13. Find the name of each company that does not employ any persons who live in ‘Bloomington’.

$$\pi_{cname}(C) - \pi_{cname}(Works \bowtie \sigma_{city=\text{‘}Bloomington'}(Person))$$

14. Find the pairs (p_1, p_2) such that p_1 and p_2 are the pids of persons such that if p_1 works for a company located in ’Bloomington’ then p_2 also works for a company that is located in ’Bloomington’. (Notice that these companies need not be the same.)

    The following expression WB gives the pids of persons who work for a company located in Bloomington.

$$WB \quad = \quad \pi_{W.pid}(W \bowtie \sigma_{city=\text{‘Bloomington’}}(C))$$

    Or, equivalently,

$$WB \quad = \quad \pi_{W.pid}(W \ltimes \sigma_{city=\text{‘Bloomington’}}(C))$$

    Then the result is

$$((\pi_{pid}(P_1) - \pi_{pid}(WB)) \times \pi_{pid}(P_2)) \cup WB_1 \times WB_2$$

    Notice that $(\pi_{pid}(P_1) - \pi_{pid}(WB))$ consists of the pids of persons who do not work for a company located in Bloomington.