B561 Advanced Database Concepts
Assignment 3

This assignment is designed to test your knowledge of the following lectures:

- Lecture 6: Aggregate functions and data partitioning
- Lecture 7: Queries with quantifiers (Part 1)
- Lecture 8: Queries with quantifiers (Part 2)
- Lecture 9: Triggers

To turn in your assignment, you will need to upload to Canvas a single file with name `assignment3.sql` which contains the necessary SQL statements that solve the problems in this assignment. The `assignment3.sql` file must be such that the AI's can run it in their PostgreSQL environment. In addition, you will need to upload a separate `assignment3.txt` file that contains the results of running your queries. We have posted the exact requirements and an example for uploading your solution files. (See the module `Instructions for turning in assignments`.)

For most of the problems in the assignment, we will use a database that maintains a set of persons `Person`, a relation `Knows`, a set of companies `Company`, a relation `WorksFor`, a set of job skills (`JobSkill`), and a relation `PersonSkills`. (The data for this database are given along with this assignment in the `data.sql` file.) The schemas for these sets and relations are as follows (primary keys are underlined):

```
Person(pid : integer, name:text, city:  text, birthYear:  integer)
Knows(pid1 : integer, pid2 : integer)
Company(cname : text, city : text)
WorksFor(pid : integer, cname:text, salary:integer)
JobSkill(skill : text)
PersonSkill(pid : text, skill : text),
```

- The `city` and `birthYear` in `Person` specify the city in which the person lives and his or her birth year.
- The relation `Knows` maintains a set of pairs (p_1, p_2) where p_1 and p_2 are pids of persons. The pair (p_1, p_2) indicates that the person with pid p_1 knows the person with pid p_2. We do not assume that the relation `Knows` is symmetric: it is possible that (p_1, p_2) is in the relation but that (p_2, p_1) is not.
- The `city` attribute in `Company` indicates a city in which the company is located. (Companies may be located in multiple cities.)

- The relation `WorksFor` stores the unique company (identified by `cname`) for which a person works along with the salary he or she makes at that company. (Incidentally, it is possible that a person in the `Person` relation does not work for any company.)
- The relation `JobSkill` only has the attribute `skill` which is the name of a possible job skill.
- The relation `PersonSkill` provides for each person his or her job skills. A person may have multiple job skills. It is also possible that a person does not have any job skills.

We assume the following primary key and foreign key constraints:

- `pid` is the primary key of `Person`
- (`pid1`, `pid2`) is the primary key of `Knows`
- (`cname`, `city`) is the primary key of `Company`
- `pid` is the primary key of `WorksFor`
- `skill` is the primary key of `JobSkill`
- (`pid`, `skill`) is the primary key of `PersonSkill`
- `pid1` is a foreign key in `Knows` referencing the primary key `pid` in `Person`
- `pid2` is a foreign key in `Knows` referencing the primary key `pid` in `Person`
- `pid` is a foreign key in `WorksFor` referencing the primary key `pid` in `Person`
- `cname` in `WorksFor` references a `cname` that appears 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`

1 Operations on polynomials and matrices using SQL aggregate functions

In the problems in this section, you will practice working with aggregate functions.

A useful other aspect of solving these problems is that you will learn how relations can be used to represent polynomials and matrices and how SQL can be used to define operations on such objects.

1. Let $P(x)$ be a polynomial with integer coefficients. For example, $P(x)$ could be the polynomial $3x^3 - 2x^2 + 5$.

   We can represent a polynomial $P(x)$ with a binary relation

   P(coefficient: integer, degree: integer)

   wherein each pair (c, d) represents the term cx^d in $P(x)$. For example, $P(x) = 3x^3 - 2x^2 + 5$ is represented in **P** as follows:

   P

coefficient	degree
3	3
-2	2
0	1
5	0

   Write a SQL function

   ```
   create or replace function evaluatePolynomial(x numeric)
          returns numeric as
   $$
   ...
   $$ language sql
   ```

   such that, for an input number x_0,

   `select evaluatePolynomial(`x_0`);`

   returns the value $P(x_0)$. For example, for the polynomial $P(x) = 3x^3 - 2x^2 + 5$ and $x_0 = 7$, $P(7) = 3 \times 7^3 - 2 \times 7^2 + 5 = 936$ and so

   ```
   select evaluatePolynomial(7);
   ```

should return the value 936.

Your solution should work for any polynomial $P(x)$ of degree n, for $n \geq 0$. Observe that if $P(x) = c_0 + c_1x + c_2x^2 + \cdots + c_nx^n$ is such a polynomial, then $P(x_0)$ is an aggregated sum, i.e., $P(x_0) = \sum_{k=0}^{n} c_d(x_0)^k$. (You can use the PostgreSQL function `power`(x, k) to computes the value x^k.)

2. Let $p(x)$ and $q(x)$ be 2 polynomials with integer coefficients.

   Let `P(coefficient, degree)` and `Q(coefficient, degree)` be two binary relations representing $p(x)$ and $q(x)$, respectively. E.g., if $p(x) = 2x^2 - 5x + 5$ and $q(x) = 4x^4 + 3x^3 + x^2 - x$ then their representations in the relations **P** and **Q** are as follows:

   P

coefficient	degree
2	2
−5	1
5	0

   Q

coefficient	degree
4	4
3	3
1	2
−1	1
0	0

   Write a SQL function

```
create or replace function multiplicationPandQ()
      returns table(coefficient integer, degree integer) as
$$
...
$$ language sql;
```

   that computes a binary relation representing the multiplication of $p(x)$ and $q(x)$, i.e., the polynomial $p(x) * q(x)$.

   For example, consider $p(x) = 2x^2 - 5x + 5$ and $q(x) = 4x^4 + 3x^3 + x^2 - x$. Then $p(x) * q(x) = (8)x^6 + (6-20)x^5 + (2-15+20)x^4 + (-2-5+15)x^3 + (5+5)x^2 + (-5)x = 8x^6 - 14x^5 + 7x^4 + 8x^3 + 10x^2 - 5x$. So, for these polynomials, your SQL query should return the relation

coefficient	degree
8	6
−14	5
7	4
8	3
10	2
−5	1
0	0

   Your solution should work for arbitrary polynomials $p(x)$ and $q(x)$.

3. 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 *dot product* $X \cdot Y$ of X and Y is defined as the aggregated sum

$$x_1 \times y_1 + x_2 \times y_2 + \cdots + x_n \times y_n = \sum_{i=1}^{n} x_i \times y_i$$

   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 or replace function dotProductXandY() returns numeric as
$$
...
$$ language sql;
```

   such that, `select dotProductXandY()` returns the value $X \cdot Y$. For example, for the vectors $X = (7, -1, 2)$ and $Y = (1, 1, -10)$, $X \cdot Y = 7 \times 1 + (-1) \times 1 + 2 \times (-10) = 7 - 1 - 20 = -14$, and therefore `select dotProductXandY()` should return the value -14.

   Your solution should work for any pair of n-dimensional vectors X and Y, with $n \geq 1$.

4. Let M be an $n \times n$ matrix of integers with $n \geq 1$.

   For $i, j \in [1, n]$, we will denote by $M[i, j]$ the element in matrix M at row i and column j. We will assume that $n \geq 1$.

   Given two $n \times n$ matrix M and N, denote by $M * N$ the matrix multiplication of M and N. By definition, $M * N$ is again a $n \times n$ matrix where, for $i, j \in [1, n]$, $M * N[i, j]$ is defined by

$$M * N[i,j] = \sum_{k=1}^{n} M[i,k] \times N[k,j].$$

   The matrix M can be represented using a relation `M` with schema

   (`row: integer`, `colmn: integer`, `value: integer`)

   and similarly for the matrix N.[1]

   For example if M is the 3×3 matrix

$$M = \begin{array}{ccc} 1 & 2 & 3 \\ 1 & -3 & 5 \\ 4 & 0 & -2 \end{array}$$

   then `M` is the following relation of 9 tuples:

   `M`

`row`	`colmn`	`value`
1	1	1
1	2	2
1	3	3
2	1	1
2	2	−3
2	3	5
3	1	4
3	2	0
3	3	−2

   Let M and N be two $n \times n$ matrices represented by the two relations `M` and `N`.

   Write a SQL query that computes a relation over schema (`row`, `column`, `value`) that represents the matrix $M * N$. Your solution should work for any $n \geq 1$.

   For example if M and N are given by the following 3×3 matrices stored as the relations

[1]Notice that we use the attribute name '`colmn`' since the word '`column`' is a reserved word in PostgreSQL.

M

row	colmn	value
1	1	1
1	2	2
1	3	3
2	1	1
2	2	−3
2	3	5
3	1	4
3	2	0
3	3	−2

N

row	colmn	value
1	1	−1
1	2	2
1	3	−1
2	1	2
2	2	−3
2	3	4
3	1	0
3	2	0
3	3	3

then your query should produce the relational representation of $M * N$, i.e., the relation

M ∗ N

row	colmn	value
1	1	3
1	2	−4
1	3	16
2	1	−7
2	2	11
2	3	2
3	1	−4
3	2	8
3	3	−10

2 Simulating Set Predicates with the COUNT Function

The problems in this section aim to illustrate that the set predicates of SQL are redundant provided one can use the COUNT aggregate function. This underscores the importance and naturalness of counting to express quantifiers.

Rewrite each of the following SQL queries into an equivalent SQL query wherein the set predicates are simulated using the `COUNT` aggregate function. In other words, you solution should use the COUNT aggregate function and can not use any of the [NOT] EXISTS, [NOT] IN, SOME, and ALL set predicates. You can also not use the set operations UNION, INTERSECT, and EXCEPT.

5.
```
select p.pid, p.name
from   Person p
where  p.city = 'Bloomington' and
       exists(select 1
              from   personSkill ps
              where  ps.pid = p.pid and ps.skill  <> 'Programming' and
                     ps.skill in (select ps1.skill
                                  from   worksFor w1, personSkill ps1
                                  where  w1.cname = 'Netflix' and
                                         ps1.skill <> 'AI' and
                                         w1.pid = p.pid));
```

6.
```
select w.pid, w.cname, w.salary
from   worksFor w
where  not(w.salary <= all (select w1.salary
                            from   worksFor w1, Company c
                            where  w.pid <> w1.pid and w.cname = w1.cname and
                                   c.city not in (select p.city
                                                  from   Person p
                                                  where  p.birthyear <> 1990)));
```

3 Solving queries using aggregate functions

Formulate the following queries in SQL. You should use aggregate functions to solve these queries. You can use views, including temporary views as well as parameterized views defined by user-defined functions that return relations (i.e., tables).

7. Find the pid and name of each person who lives in 'Bloomington' and who knows at most one person who has at least 3 job skills.
8. Find the pid and name of each person who has all but three job skills. I.e., such a person lacks precisely three job skills from the possible job skills that are stored in the relation `jobSkill`.
9. Find the cname of each company along with the number of persons who work for that company and who know at least two other persons who also work for that company. (Make sure to also consider companies that have no such employees and to include for each such company a tuple of the form (cname,0) in the answer.)
10. Find the pid and name of each person who works for 'Netflix' and who knows the most persons who work for 'IBM'.
11. Find the cname of each company who spends the highest aggregated amount on the salaries of persons who work for that company and who have the fewest skills.

4 Queries with quantifiers using Venn diagrams with conditions

Using the method of Venn diagrams with conditions and without using the `COUNT` function, write SQL queries for the following queries with quantifiers.

In these problems, you must write appropriate views and parameterized views for the sets A and B that occur in the Venn diagram with conditions for these queries. (See the lecture on Queries with Quantifiers.)

12. Find the pid and name of each person who does not know any person who has a salary strictly above 55000.

13. Find the pid and name of each person who knows all the persons who (a) work at Netflix, (b) make at least 55000, and (c) are born after 1985.

    Changed 'are born before 1990' by 'are born after 1985'.

14. Find the cname of each company who only employs persons who make less than 55000.

15. Find the pairs of different skills (s_1, s_2) such that not every person who works at 'IBM' and has skill s_1 knows a person who works at 'Netflix' and has skill s_2.

16. The concept of Venn diagrams with conditions can be generalized to more than 2 sets. Here is a query that requires a condition on a Venn diagram with 3 sets. To solve this problem, you should consider definining 3 parameterized views.

    Find each triple (p, c, s) where p is the pid of a person, c is the cname of a company, and s is a skill, such that each person who is known by the person with pid p and who works for company with cname c is a person who has the jobskill s.

5 Queries with quantifiers using Venn diagrams with counting conditions

Using the method of Venn diagram with counting conditions, write SQL queries for the following queries with quantifiers.

In these problems, you should write appropriate views and parameterized views for the sets A and B that occur in the Venn diagrams for these queries. (See the lecture on Queries with Quantifiers Using the COUNT function.)

17. Find the pid and name of each person who has more than 2 of the combined set of job skills of persons who work for 'IBM'.

    Changed 'more than 3' to more than 2' and 'Netflix' to 'IBM'.

18. Find the cname of each company that employs an odd number of persons whose salary is at least 50000.

19. Find the pid and name of each person who knows at least 2 persons who each have exactly 3 job skills.

20. Find the pairs (p_1, p_2) of different person pids such that the person with pid p_1 and the person with pid p_2 knows the same set of persons.

21. Find the pairs (p_1, p_2) of different person pids such that the person with pid p_1 and the person with pid p_2 knows the same number of persons.