CIS 550: Database and Information Systems

Exercise 2: Advanced SQL Queries

In this exercise you will set up a "Family and Friends" database using MySQL, and write advanced SQL queries. This exercise should be completed using DataGrip and AWS RDS just like in Exercise 1 Part 2.

Download the exercise2.py template file in Readings: Module 2 Recitation Exercise and add your solutions in the appropriate places as instructed. Please use the following conventions:

- Put each clause (SELECT...FROM...WHERE, etc) on a new line, e.g.
  SELECT name FROM Student s JOIN Takes t ON s.sid=t.tid WHERE name LIKE '%a'
  should be
  SELECT name
  FROM Student s JOIN Takes T ON s.sid=t.tid
  WHERE name LIKE '%a'

- Don't use smartquotes e.g. “%a” and ‘%a’ or else your query won't run.

The database has the following schema:

Person(login, name, sex, relationshipStatus, birthyear)
Family(login, member, role)
Friends(login, friend)

Keys are underlined; attributes login and member in Family are foreign keys to Person; and attributes login and friend in Friends are foreign keys to Person. For example, a tuple in Family {login: “abc”, member: “xyz”, role: “daughter”} represents the fact that the person with login “abc” is the daughter of the person with login “xyz.”

Constraints (in addition to the key and foreign key constraints mentioned above):

- Attribute sex should only take on values 'male' or 'female'.
- Attribute relationshipStatus should only take on values 'single', 'married', 'divorced', or 'relationship'.
- Attribute role should only take on values 'mother', 'father', 'son', 'daughter','aunt', 'uncle', 'cousin', 'brother', or 'sister'.

1. Connect to your RDS MySQL database via DataGrip as instructed in the DataGrip and RDS handouts. Fill in the connection details to the RDS database you set up in the template file within db_config.

Write SQL DDL statements in SQL Developer to create the "Friends and Family" database and enforce the constraints above. Use varchar(255) as the type of login and name, varchar(7) for sex, and varchar(12) for relationshipStatus and role.

Execute your DDL statements in DataGrip and include your DDL statements in the template submission file we provided as answers 1a through 1c.

ANSWER:

```
CREATE TABLE Person (
        login varchar(255),
        name varchar(255),
        sex varchar (7),
        relationshipStatus varchar(12),
        birthyear int,
        PRIMARY KEY (login),
        CHECK (sex IN ('male','female')),
        CHECK (relationshipStatus IN ('single','married','divorced','relationship'))
);

CREATE TABLE Family (
        login varchar(255),
        member varchar(255),
        role varchar(10),
        PRIMARY KEY (login, member),
        FOREIGN KEY (login) REFERENCES Person (login),
        FOREIGN KEY (member) REFERENCES Person (login),
        CHECK (role IN ('mother','father','son','daughter','aunt', 'uncle','cousin','brother','sister'))
);

CREATE TABLE Friends (
        login varchar(255),
        friend varchar(255),
        PRIMARY KEY (login, friend),
        FOREIGN KEY (login) REFERENCES Person (login ),
        FOREIGN KEY (friend) REFERENCES Person (login)
);
```

Once you have created the Person, Family and Friends relations, download the dataset zip file in Readings: Module 2 Recitation Exercise and extract the files Person.sql, Family.sql, and Friends.sql.

Populate the relations you just created:
In DataGrip, make sure that you have switched to the database where you had the Person, Family and Friends relations created by running the script:
`use <your_database_name>`.
Then, paste the contents of the script in the DataGrip console for all three .sql source files in the correct order and run them. For example, you would run the contents of Person.sql to populate the Person relation. This script imports entries for the relations into the database.

2. In what order should you execute the three import scripts and why? In the template submission file, fill in your response, formatted as a string, as answer2.

ANSWER: The order must obey the foreign key constraints, so Person must come before Family and Friends.
One possible order is:
@<path_to_Person.sql>
@<path_to_Family.sql>
@<path_to_Friends.sql>

Another is:
@<path_to_Person.sql>
@<path_to_Friends.sql>
@<path_to_Family.sql>

3. Print the login of all people who list someone as their sibling (brother or sister) but the sibling does not list them as such (i.e. the relationship is not symmetric). Your answer should contain mbeck@nova.edu and jen.westad@gmail.com.
Schema: (login)

ANSWER:
```
SELECT DISTINCT login
FROM Family f
WHERE (role='brother'  OR role='sister') AND login NOT IN

        (SELECT member
         FROM Family f1
         WHERE f1.login=f.member AND (f1.role='brother'  OR f1.role='sister'));
```

equivalently…

```
SELECT DISTINCT login
FROM Family f
WHERE (role='brother' OR role='sister') AND member NOT IN
        (SELECT login
         FROM Family f1
         WHERE f1.member=f.login AND (f1.role='brother' OR f1.role='sister'));
```

4. For each person with two or more family members, print their login, name and number of family members.
Schema: (login, name, num)

You should get the following as the result:
```
+-----------------------------+-----------------------+-----+
| login                       | name                  | num|
+-----------------------------+-----------------------+-----+
| awest@gmail.com             | Ashton Westad         |   2 |
| jen.westad@gmail.com | Jenny Westad       |   2 |
| lyd.jasp@gmail.com   | Lydia Jasper       |   2 |
| mbeck@nova.edu       | Meredith Beckner   |   5 |
+----------------------------+----------------------+-----+
```

ANSWER:

```
SELECT P.login, P.name, count(F.member) AS num
FROM Person P JOIN Family F ON P.login=F.login
GROUP BY P.login, P.name
HAVING COUNT(F.member) >= 2;
```

5. Print the login, name and number of family members that each person has, *including people with no family members with num set to 0*. The result should have 23 rows.

```
SELECT P.login, P.name, count(F.member) AS num
FROM Person P LEFT JOIN Family F ON P.login=F.login
GROUP BY P.login, P.name;
```

or

```
(SELECT P.login, P.name, count(F.member) AS num
FROM Person P JOIN Family F ON P.login=F.login
GROUP BY P.login, P.name)
UNION
(SELECT P.login, P.name, 0 AS num
FROM Person P
WHERE login NOT IN
        (SELECT login
          FROM Family)
)
```