Fall 2022 B561 Assignment 1 Relational Databases, Expressing Queries and Constraints in SQL and in Tuple Relational Calculus (TRC).*

Dirk Van Gucht

Released: August 25, 2022
Due: September 12, 2022 by 11:45pm

1 Introduction

The goals for this assignment are to

1. become familiar with the PostgreSQL system[1];
2. create a relational database and populate it with data;
3. examine the side-effects on the state of the database caused by inserts and deletes in the presence or absence of primary and foreign key constraints;
4. formulate some queries in SQL and evaluate them in PostgreSQL; and
5. translate TRC queries to SQL and formulate queries and constraints in TRC.[2]

To turn in your assignment, you will need to upload to Canvas a single file with name `assignment1.sql` which contains the necessary SQL statements that solve the graded problems in this assignment. The graded problems are indicated with a blue bullet point •. The non-graded problems are indicated with a red bullet point •. The `assignment1.sql` file must be so that the AI's can run it in their PostgreSQL environment. You should use the `Assignment1Script.sql`

*This assignment covers lectures 1 through 4

[1]To solve this assignment, you will need to download and install PostgreSQL (version 13 or higher) on your computer.

[2]To solve problems related to TRC, follow the syntax and semantics described in the `TRC_SQL.pdf` document in the module *Tuple Relational Calculus and SQL (lecture 4)*. That document contains multiple examples of TRC queries and constraints and how they can be translated to SQL.

file to construct the assignment1.sql file. (Note that the data to be used for this assignment is included in this file.) In addition, you will need to upload a separate assignment1.txt file that contains the results of running your SQL queries. Finally, you need to upload a file assignment1.pdf that contains the solutions to the graded problems that require it.[3]

For the problems in this assignment we will use the following database schema:[4]

```
Student(sid, sname, homeCity)
Department(deptName, mainOffice)
Major(major)
employedBy(sid, deptName, salary)
departmentLocation(deptName, building)
studentMajor(sid, major)
hasFriend(sid1, sid2)
```

In this database we maintain a set of students (`Student`), a set of departments (`Department`), and a set of (major) majors (`Major`). The `sname` attribute in `Student` is the name of the student. The `homeCity` attribute in `Student` specifies the home city of the student. The `deptName` attribute in `Department` is the name of the department. The `mainOffice` attribute in `Department` is the name of the building where the department has its main office. The `major` attribute in `Major` is the name of a (major) major.

A student can be employed by at most one department. This information is maintained in the `employedBy` relation. (We permit that a student is not employed by any department and that a department may not employ any students.) The `salary` attribute in `employedBy` specifies the salary made by the student.

The `building` attribute in `departmentLocation` indicates a building in which the department is located. (A department may be located in multiple buildings.)

A student can have multiple majors. This information is maintained in the `studentMajor` relation. A major can be the major of multiple students. (A student may not have any majors, and a major may have no students with that major.)

A pair (s_1, s_2) in `hasFriend` indicates that student s_1 considers student s_2 as a friend. It is possible that a s_1 considers s_2 as a friend, but not necessarily the other way around. In other words, the `hasFriend` relation can not be assumed to be symmetric.

[3] It is strongly recommended that you use Latex to write TRC formulas and queries. For a good way to learn about Latex, look at https://www.overleaf.com/learn/latex/Free_online_introduction_to_LaTeX_(part_1). You can also inspect the Latex source code for this assignment as well as the document TRC_SQL.tex provided in Module 4.

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

The domain for the attributes `sid`, `salary`, `sid1`, and `sid2` is `integer`. The domain for all other attributes is `text`.

We assume the following foreign key constraints:

- `sid` is a foreign key in `employedBy` referencing the primary key `sid` in `Student`;
- `deptName` is a foreign key in `employedBy` referencing the primary key `deptName` in `Department`;
- `deptName` is a foreign key in `departmentLocation` referencing the primary key `deptName` in `Department`;
- `sid` is a foreign key in `studentMajor` referencing the primary key `sid` in `Student`;
- `major` is a foreign key in `studentMajor` referencing the primary key `major` in `Major`;
- `sid1` is a foreign key in `hasFriend` referencing the primary key `sid` in `Student`; and
- `sid2` is a foreign key in `hasFriend` referencing the primary key `sid` in `Student`;

The data for the assignment is included in the Assignment1Script.sql.

Remark 1 *We will typically use the primary key of an object to specify that object. This should not cause any confusion since an object can be referenced uniquely with its primary key value. E.g., we will write '...* student *s ... ' rather than '...* the student with sid *s ...'. When posing a query, we may write '*Find each student who *...' instead of '*Find the sid of each student who *...'. The expected answer for such a query should be the set of sids of students who meet the criteria of the query.*

2 Database creation and impact of constraints on insert and delete statements.

Create a database in PostgreSQL that stores the data provided in the `data.sql` file. Make sure to specify primary and foreign keys.

1. • Provide 4 conceptually different examples that illustrate how the presence or absence of primary and foreign keys affect insert and deletes in these relations.[5]

   To solve this problem, you will need to experiment with the relation schemas and instances for this assignment. For example

   - you may consider altering primary keys and foreign key constraints and then consider various sequences of insert and delete operations; or
   - you may consider changing the given relation instances to observe the desired effects.

   The SQL code that corresponds to each of these four examples should be included in the `assignment1.sql` file (even though it may generate error conditions.) You can also put comments in the `assignment1.sql` that discuss your examples. Nothing about this problem should go into the `assignment1.pdf` file.

[5] Consider the lecture notes about keys, foreign keys, and inserts and deletes.

3 Formulating queries in SQL

Remark 2 *For this assignment, you are required to use tuple variables in your SQL statements.*[6] *For example, in formulating the query "Find the sid and sname of each student who lives in Bloomington" you should write the query*

```
SELECT   s.sid, s.sname
FROM     Student s
WHERE    s.homeCity = 'Bloomington'
```

rather than

```
SELECT   sid, sname
FROM     Student
WHERE    homeCity = 'Bloomington'
```

Write SQL statements for the following queries. Make sure that each of your queries returns a set but not a bag. In other words, make appropriate use of the `DISTINCT` clause where necessary.

You can **not** use the SQL JOIN operations or SQL aggregate functions such as `COUNT`, `SUM`, `MAX`, `MIN`, etc in your solutions.

2. • Find each pair (d, m) where d is the name of a department and m is a major of a student who is employed by that department and who earns a salary of at least 20000.

   In TRC:

   $$\{(d.deptName, m.major) \mid Department(d) \wedge Major(m) \wedge \\ \exists s \in Student(studentMajor(s.sid, m.major) \wedge \\ \exists w \in employedBy(w.sid = s.sid \wedge w.deptName = d.deptName \wedge w.salary \geq 20000))\}.$$

   Alternatively,

```
select d.deptName, m.major
from   department d, Major m
where  true = some (select (s.sid, m.major) in (select * from studentMajor) and
                           true = some (select w.sid = s.sid and
                                               w.deptName = d.deptName and w.salary >= 20000
                                       from   employedBy w)
                    from   Student s);

select d.deptName, m.major
from   department d, Major m
where  exists (select 1
               from   Student s
               where  (s.sid, m.major) in (select * from studentMajor) and
                      exists (select 1
```

[6] In later assignments, we will drop that requirement. Still it is important to realize the distinction between a relation name and a tuple variable name in a query.

```
                                from   employedBy w
                                where  w.sid = s.sid and
                                       w.deptName = d.deptName and
                                       w.salary >= 20000));
```

Alternatively,

```
select d.deptName, m.major
from   department d, Major m
where  m.major in (select sm.major
                   from   studentMajor sm
                   where  sm.major = m.major and
                          sm.sid in (select w.sid
                                     from   employedBy w
                                     where  w.deptName = d.deptName and w.salary >= 20000));
```

3. • Find each pair (s_1, s_2) of sids of different students who have the same (set of) friends who work for the CS department.

   In TRC:

$$\{(s_1.sid, s_2.sid) \mid Student(s_1) \wedge Student(s_2) \wedge s_1.sid \neq s_2.sid \wedge$$
$$\forall w \in employedBy$$
$$((w.deptName = \texttt{CS} \wedge hasFriend(s_1.sid, w.sid)) \rightarrow hasFriend(s_2.sid, w.sid)) \wedge$$
$$((w.deptName = \texttt{CS} \wedge hasFriend(s_2.sid, w.sid)) \rightarrow hasFriend(s_1.sid, w.sid)))\}$$

```
select s1.sid as "sid1", s2.sid as "sid2"
from   Student s1, Student s2
where  true = all (select (s2.sid,w.sid) in (select * from hasFriend)
                   from   employedBy w
                   where  w.deptName = 'CS' and
                          (s1.sid,w.sid) in (select * from hasFriend)) and
       true = all (select (s1.sid,w.sid) in (select * from hasFriend)
                   from   employedBy w
                   where  w.deptName = 'CS' and
                          (s2.sid,w.sid) in (select * from hasFriend)) and
       s1.sid <> s2.sid;
```

$$\{(s_1.sid, s_2.sid) \mid Student(s_1) \wedge Student(s_2) \wedge s_1.sid \neq s_2.sid \wedge$$
$$\forall w \in employedBy(w.deptName = \texttt{CS} \rightarrow$$
$$(hasFriend(s_1.sid, w.sid) \leftrightarrow hasFriend(s_2.sid, w.sid)))\}$$

   Observing that '$P \leftrightarrow Q$' is logically equivalent with '$(P \wedge Q) \vee (\neg P and \neg Q)$', we can express the query as

```
select s1.sid as "sid1", s2.sid as "sid2"
from   Student s1, Student s2
where  true = all (select ((s1.sid,w.sid) in (select * from hasFriend) and
                           (s2.sid,w.sid) in (select * from hasFriend)) OR
```

```
                        ((s1.sid,w.sid) not in (select * from hasFriend) and
                         (s2.sid,w.sid) not in (select * from hasFriend))
                from    employedBy w
                where   w.deptName = 'CS') and
      s1.sid <> s2.sid;
```

Or, alternatively

$$\{(s_1.sid, s_2.sid) \mid Student(s_1) \wedge Student(s_2) \wedge s_1.sid \neq s_2.sid \wedge \\ \neg\exists w \in employedBy(w.deptName = \texttt{CS} \wedge \\ ((hasFriend(s_1.sid, w.sid) \wedge \neg hasFriend(s_2.sid, w.sid) \vee \\ (\neg hasFriend(s_1.sid, w.sid) \wedge hasFriend(s_2.sid, w.sid)))))\}.$$

```
select s1.sid as "sid1", s2.sid as "sid2"
from   Student s1, Student s2
where  not true = some (select ((s1.sid,w.sid) in (select * from hasFriend) and
                                (s2.sid,w.sid) not in (select * from hasFriend)) OR
                                ((s1.sid,w.sid) not in (select * from hasFriend) and
                                (s2.sid,w.sid) in (select * from hasFriend))
                        from    employedBy w
                        where   w.deptName = 'CS') and
       s1.sid <> s2.sid;
```

Or, alternatively,

```
select s1.sid as "sid1", s2.sid as "sid2"
from   Student s1, Student s2
where  not exists (select 1
                   from   employedBy w
                   where  w.deptName = 'CS' and
                          (((s1.sid,w.sid) in (select * from hasFriend) and
                            (s2.sid,w.sid) not in (select * from hasFriend)) or
                           ((s2.sid,w.sid) in (select * from hasFriend) and
                            (s1.sid,w.sid) not in (select * from hasFriend)))) and
       s1.sid <> s2.sid;
```

4. • Find each major for which there exists a student with that major and who does not only have friends who also have that major.

   In TRC:

   $$\{m.major \mid Major(m) \wedge \exists s \in Student(studentMajor(s.sid, m.major) \wedge \\ \neg(\forall s_1 \in Student(hasFriend(s.sid, s_1.sid) \rightarrow studentMajor(s_1, m.major)))\}$$

   Alternatively,

   $$\{m.major \mid Major(m) \wedge \exists s \in Student(studentMajor(s.sid, m.major) \wedge \\ \exists s_1 \in Student(hasFriend(s.sid, s_1.sid) \wedge \neg studentMajor(s_1, m.major)))\}.$$

Alternatively,

$$\{m.major \mid Major(m) \wedge \exists s \in Student \exists s_1 \in Student(studentMajor(s.sid, m.major) \wedge hasFriend(s.sid, s_1.sid) \wedge \neg studentMajor(s_1, m.major)))\}.$$

```
select m.major
from   Major m
where  exists (select 1
               from   Student s1, Student s2
               where  (s1.sid,m.major) in (select * from studentMajor) and
                      (s1.sid,s2.sid) in (select * from hasFriend) and
                      (s2.sid,m.major) not in (select * from studentMajor));
```

For solutions to problems 5 through 12 see the file '`Solutions-SQL-2022-Fall-Assignment1.sql`'.

5. • Find the sid, sname of each student who (a) has home city Bloomington, (b) works for a department where he or she earns a salary that is higher than 20000, and (c) has at least one friend.

6. • Find the pairs (d_1, d_2) of names of different departments whose main offices are located in the same building.

7. • Find the sid and sname of each student whose home city is different than those of his or her friends.

8. • Find each major that is the major of at most 2 students.

9. • Find the sid, sname, and salary of each student who has at least two friends such that these friends have a common major but provided that it is not the 'Mathematics' major.

10. • Find the deptName of each department that not only employs students whose home city is Indianapolis. (In other words, there exists at least one student who is employed by such a department whose home city is not Indianapolis.)

11. • For each department, list its name along with the highest salary made by students who are employed by it.

12. • Find the sid and sname of each student s who is employed by a department d and who has a salary that is strictly higher than the salary of each of his or her friends who is employed by that department d. (Student s should only be considered if indeed he or she has a friend who is employed by department d.)

4 Translating TRC queries to SQL

The problems in this section require expressing TRC queries as equivalent SQL queries.

Example 1 *Consider the query which lists for each department, those students who earn the highest salary in that department. This query can be expressed in TRC as*

$$\{(w_1.deptName, w_1.sid) \mid employedBy(w_1) \wedge \\ \forall w_2 \in employeedBy(w_1.deptName = w_2.deptName \rightarrow w_1.salary \geq w_2.salary)\}.$$

Observe that this query contains a universal quantifier '$\forall$' which, in SQL, is most directly translated as '`true = all`*'. For the above TRC query, we therefore get in SQL the query*

```
select w1.deptName, w1.sid
from   employedBy w1
where  true = all (select w1.salary >= w2.salary
                   from   employedBy w2
                   where  w1.deptName = w2.deptName);
```

Because a conditional '$F_1 \rightarrow F_2$' is logically equivalent with the disjunction '$\neg F_1 \vee F_2$', the above query can also be expressed in TRC as

$$\{(w_1.deptName, w_1.sid) \mid employedBy(w_1) \wedge \\ \forall w_2 \in employeedBy(\neg(w_1.deptName = w_2.deptName) \vee w_1.salary \geq w_2.salary)\}.$$

or

$$\{(w_1.deptName, w_1.sid, w_1.salary) \mid employedBy(w_1) \wedge \\ \forall w_2 \in employeedBy(w_1.deptName \neq w_2.deptName \vee w_1.salary \geq w_2.salary)\}.$$

Therefore, in SQL, this query can be also expressed as

```
select w1.deptName, w1.sid
from   employedBy w1
where  true = all (select w1.deptName <> w2.deptName or w1.salary >= w2.salary
                   from   employedBy w2);
```

Using the logically equivalency of '$\forall x F_1 \rightarrow F_2$' and '$\neg\exists x F_1 \wedge \neg F_2$', the above query can also be expressed in TRC as

$$\{(w_1.deptName, w_1.sid) \mid employedBy(w_1) \wedge \\ \neg\exists w_2 \in employeedBy(w_1.deptName = w_2.deptName \wedge w_1.salary < w_2.salary)\}.$$

Observe that this query contains a existential quantifier '$\exists$' which, in SQL, is most directly translated as '`true = some`*'. We therefore get in SQL the query*

```
select w1.deptName, w1.sid
from   employedBy w1
where  not true = some (select w1.deptName = w2.deptName and w1.salary < w2.salary
                        from   employedBy w2);
```

Alternatively, to express a TRC existential quantifier, we can use the SQL '`exists`*' set predicate and get the SQL query*

```
select w1.deptName, w1.sid
from   employedBy w1
where  not exists (select 1
                   from   employedBy w2
                   where  w1.deptName = w2.deptName and w1.salary < w2.salary);
```

Consider the following queries formulated in TRC. Translate each of these queries as an equivalent SQL query.

The SQL queries should be included in the `assignment1.sql` file and their outputs should be reported in the `assignment.txt` file.

13. •

$$\{(s_1.sid, s_1.sname) \mid Student(s_1) \wedge \\ \exists d \in Department\, \exists w \in employedBy(d.deptName = w.deptName \wedge s_1.sid = w.sid \wedge d.mainOffice = \texttt{LuddyHall} \wedge \\ \exists s_2 \in Student(hasFriend(s_1.sid, s_2.sid) \wedge s_2.homeCity \neq \texttt{Bloomington}))\}.$$

```
select s1.sid, s1.sname
from   Student s1
where  exists (select 1
               from   Department d, employedBy w
               where  d.deptName = w.deptName and
                      s1.sid = w.sid and
                      d.mainOffice = 'LuddyHall' and
                      exists (select 1
                              from   Student s2
                              where  (s1.sid, s2.sid) in (select * from hasFriend) and
                                     s2.homeCity <> 'Bloomington'));
```

14. •

$$\{s_1.sid \mid Student(s_1) \wedge \forall s_2 \in Student(s_2)\ (hasFriend(s_1.sid, s_2.sid) \rightarrow \exists sm_1 \in studentMajor\, \exists sm_2 \in studentMajor(sm_1.sid = s_1.sid \wedge sm_2.sid = s_2.sid \wedge sm_1.major = sm_2.major \wedge sm_1.sid \neq sm_2.sid))\}.$$

```
select s1.sid
from   Student s1
where  true = all (select true = some (select sm1.sid = s1.sid and
                                              sm2.sid = s2.sid and
                                              sm1.major = sm2.major and
                                              sm1.sid <> sm2.sid
                                       from   studentMajor sm1, studentMajor sm2)
                   from   Student s2
                   where  (s1.sid, s2.sid) in (select * from hasFriend));
```

15. •

$$\{(s_1.sid, s_2.sid) \mid Student(s_1) \wedge Student(s_2) \wedge s_1.sid \leq s_2.sid \wedge \forall f_1 \in hasFriend(s_1.sid1 = f_1.sid1 \rightarrow \exists f_2 \in hasFriend(f_2.sid1 = s_2.sid \wedge f_1.sid2 = f_2.sid2))\}.$$

```
select s1.sid as "sid1", s2.sid as "sid2"
from   Student s1, Student s2
where  s1.sid <> s2.sid and
       true = all (select true = some (select f2.sid1 = s2.sid and f1.sid2 = f2.sid2
                                       from   hasFriend f2)
                   from   hasFriend f1
                   where  s1.sid = f1.sid1);
```

For solutions to problems 16 through 19 see the file '`Solutions-SQL-2022-Fall-Assignment1.sql`'.

16. •

$$\{s.sid, s.sname, w.deptName, w.salary \mid Student(s) \wedge employedBy(w) \wedge s.sid = w.sid \\ s.homeCity = \text{`}\texttt{Bloomington}\text{'} \wedge 10000 \leq w.salary \wedge w.deptName \neq \text{`}\texttt{Mathematics}\text{'}\}.$$

17. •

$$\{s.sid, s.sname \mid Student(s) \wedge \\ \exists d \in Department\, \exists w \in employedBy(d.deptName = w.deptName \wedge s.sid = w.sid \wedge d.mainOffice = \text{`}\texttt{LuddyHall}\text{'} \wedge \\ \exists f \in hasFriend\, \exists s_1 \in Student(f.sid1 = s.sid \wedge f.sid2 = s_1.sid \wedge s_1.homeCity \neq \text{`}\texttt{Bloomington}\text{'}))\}.$$

18. •

$$\{m.major \mid Major(m) \wedge \neg(\exists s \in Student\, \exists sm \in studentMajor(s.sid = sm.sid \wedge \\ sm.major = m.major \wedge s.homeCity = \text{`Bloomington'})\}.$$

19. •

$$\{s.sid, s.sname \mid Student(s) \wedge \\ \forall f \in hasFriend(f.sid2 = s.sid \rightarrow \exists s_1 \in Student(f.sid1 = s_1.sid \wedge s.homeCity = s_1.homeCity))\}$$

5 Formulating queries in the Tuple Relational Calculus

20. Formulate each of the queries in problems 2, 3, and 4 in Section 3 as TRC queries. The solutions of these problems should be included in the `assignment1.pdf` file.

    (a) •(Problem 2) Find each pair (d, m) where d is the name of a department and m is a major of a student who is employed by that department and who earns a salary of at least 20000.

    $$\{(d.deptName, m.major) \mid Department(d) \wedge Majorm \wedge \\ \exists s \in Student(studentMajor(s.sid, m.major) \wedge \\ \exists w \in employedBy(w.sid = s.sid \wedge w.deptName = d.deptName \wedge w.salary \geq 20000))\}.$$

    (b) •(Problem 3) Find each pair (s_1, s_2) of sids of different students who have the same (set of) friends who work for the CS department.

    $$\{(s_1.sid, s_2.sid) \mid Student(s_1) \wedge Student(s_2) \wedge s_1.sid \neq s_2.sid \wedge \\ \forall w \in employedBy(w.deptName = \texttt{CS} \rightarrow \\ ((hasFriend(s_1.sid, w.sid) \rightarrow hasFriend(s_2.sid, w.sid)) \wedge \\ (hasFriend(s_2.sid, w.sid) \rightarrow hasFriend(s_1.sid, w.sid))))\}$$

    (c) •(Problem 4) Find each major for which there exists a student with that major and who does not only have friends who also have that major.

    $$\{m.major \mid Major(m) \wedge \exists s \in Student(studentMajor(s.sid, m.major) \wedge \\ \neg(\forall s_1 \in Student(hasFriend(s.sid, s_1.sid) \rightarrow studentMajor(s_1, m.major)))\}$$

    Alternatively,

    $$\{m.major \mid Major(m) \wedge \exists s \in Student(studentMajor(s.sid, m.major) \wedge \\ \exists s_1 \in Student(hasFriend(s.sid, s_1.sid) \wedge \neg studentMajor(s_1, m.major)))\}.$$

    Alternatively,

    $$\{m.major \mid Major(m) \wedge \exists s \in Student \exists s_1 \in Student(studentMajor(s.sid, m.major) \wedge \\ hasFriend(s.sid, s_1.sid) \wedge \neg studentMajor(s_1, m.major)))\}.$$

21. Formulate each of the queries in problems 5, 7, 9, and 11 in Section 3 as TRC queries.

(d) • (Problem 5) Find the sid, sname of each student who (a) has home city Bloomington, (b) works for a department where he or she earns a salary that is higher than 20000, and (c) has at least one friend.

$$\{s.sid, s.sname \mid Student(s) \wedge s.city = \texttt{Bloomington} \wedge \\ \exists w(employedBy(w) \wedge s.sid = w.sid \wedge w.salary > 20000) \wedge \\ \exists f(hasFriend(f) \wedge f.sid1 = s.sid)\}.$$

(e) • (Problem 7) Find the sid and sname of each student whose home city is different than those of his or her friends.

$$\{s.sid, s.sname, s.city \mid Student(s) \wedge \exists f(hasFriend(f) \wedge f.sid1 = s.sid) \wedge \\ \neg\exists s_1(Person(s_1) \wedge s.city = s_1.city \wedge hasFriend(s.sid, s_1.sid))\}.$$

(f) • (Problem 9) Find the sid, sname, and salary of each student who has at least two friends such that these friends have a common major but provided that it is not the 'Mathematics' major.

$$\{s.sid \mid Student(s) \wedge \exists f_1 \exists f_2(hasFriend(f_1) \wedge hasFriend(f_2) \wedge \\ f_1.sid1 = s.sid \wedge f_2.sid1 = s.sid \wedge f_1.sid2 \neq f_2.sid2 \wedge \\ \exists sm_1 \exists sm_2(studentMajor(sm_1) \wedge studentMajor(sm_2) \wedge \\ f_1.mid = sm_1.sid \wedge f_2.mid = sm_2.sid \wedge sm_1.major = sm_2.major \wedge \\ sm_1.major \neq \texttt{Mathematics}))\}.$$

(g) • (Problem 11) For each department, list its name along with the highest salary made by students who are employed by it.

$$\{d.dname, w.salary \mid Department(d) \wedge employedBy(w) \wedge w.dname = d.dname \wedge \\ \neg\exists w_1(employedBy(w_1) \wedge w_1.dname = d.dname \wedge w.salary < w_1.salary)\}.$$

6 Formulating constraints in the Tuple Relational Calculus an as boolean SQL queries

Formulate the following constraints in TRC and as boolean SQL queries.

The TRC solutions of these problems should be included in the `assignment1.pdf` file and the SQL solutions should be included in the `assignment1.sql` file.

Here is an example of what is expected for your answers.

Example 2 *Consider the constraint "*Each major is the major of a student.*" In TRC, this constraint can be formulated as follows:*

$$\forall m\ Major(n) \rightarrow \exists sm\,(studentMajor(sm) \wedge sm.major = m.major)$$

This constraint can be specified using the following boolean SQL query.

```
select true = all (select true = some (select sm.major = m.major
                                       from   studentMajor sm)
                   from   Major m);
```

Alternatively, the constraint can be formulated in TRC as

$$\neg\exists m(Major(m) \wedge \neg\exists sm(studentMajor(sm) \wedge sm.major = m.major)).$$

This constraint can be specified using the following boolean SQL query:

```
select not true = some (select not true = some (select sm.major = m.major
                                                from   studentMajor sm)
                        from   Major m);
```

or, alternatively

```
select not exists (select 1
                   from   Major m
                   where  not exists (select 1
                                      from   studentMajor sm
                                      where  sm.major = m.major));
```

22. Consider the constraint "*Some major has fewer than 2 students with that major.*"

    (a) • Formulate this constraint in TRC.

    $$\exists m \in Major \wedge \neg\exists s_1 \in Student \exists s_2 \in Student(s_1.sid \neq s_2.sid \wedge studentMajor(s_1.sid, m.major) \wedge studentMajor(s_2.sid, m.major)).$$

    Alternatively,

    $$\exists m \in Major \wedge \forall s_1 \in Student \forall s_2 \in Student((studentMajor(s_1.sid, m.major) \wedge studentMajor(s_2.sid, m.major)) \rightarrow s_1.sid = s_2.sid)$$

(b) • Formulate this constraint as a boolean SQL query.

```
select true = some (select true = all (select s1.sid = s2.sid
                                       from  Student s1, Student s2
                                       where (s1.sid, m.major) in (select * from studentMajor) and
                                             (s2.sid, m.major) in (select * from studentMajor))
                    from Major m);
```

23. Consider the constraint "*Each student who works for a department has a friend who also works for that department and who earns the same salary.*"

(a) • Formulate this constraint in TRC.

$$\forall s \in Student \forall d \in Department \forall w \in employedBy((w.sid, w.deptName) = (s.sid, d.deptName) \rightarrow \\ \exists w_1 \in employedBy(hasFriend(w.deptname, w_1.sid) \wedge (w_1.deptname, w_1.salary) = (w.deptname, w.salary)))$$

(b) • Formulate this constraint as a boolean SQL query.

```
select true = all (select true = some (select (w.sid,w1.sid) in (select * from hasFriend) and
                                              (w1.deptname, w1.salary) = (w.deptname, w.salary)
                                       from   employedBy w1)
                   from  Student s, Department d, employedBy w
                   where (w.sid,w.deptName) = (s.sid,d.deptName));
```

24. Consider the constraint "*All students working in a same department share a major and earn the same salary.*"

(a) • Formulate this constraint in TRC.

$$\forall s_1 \in Student \forall s_2 \in Student \\ \forall w_1 \in employedBy \forall w_2 \in employedBy((w_1.sid, w_1.deptName, w_2.sid) = (s_1.sid, w_2.deptName, s_2.sid) \rightarrow \\ (w_1.salary = w_2.salary \wedge \exists m \in Major(studentMajor(s_1.sid, m.major) \wedge studentMajor(s_2.sid, m.major))))$$

(b) • Formulate this constraint as a boolean SQL query.

```
select true = all (select w1.salary = w2.salary and
                          true = some (select (s1.sid,m.major) in (select * from studentMajor) and
                                              (s2.sid,m.major) in (select * from studentMajor)
                                       from   Major m)
                   from   Student s1, Student s2, employedBy w1, employedBy w2
                   where  (w1.sid,w1.deptName,w2.sid) = (s1.sid,w2.deptName,s2.sid));
```

For solutions to problems 25.b, 26.b, and 27.b see the file '`Solutions-SQL-2022-Fall-Assignment1.sql`'.

25. Consider the constraint "*Each student is employed by a department and has at least two majors.*"

    (a) • Formulate this constraint in TRC.

    $$\forall s\, Student(s) \rightarrow (\exists w(employedBy(w) \wedge w.sid = s.sid) \wedge \exists sm_1 \exists sm_2(studentMajor(sm_1) \wedge studentMajor(sm_2) \wedge sm_1.sid = s.sid \wedge sm_2.sid = s.sid \wedge sm_1.major \neq sm_2.major))$$

    Equivalently,

    $$\neg\exists p\, Student(s) \wedge (\neg\exists w(employedBy(w) \wedge w.sid = s.sid) \vee \neg\exists ps_1 \exists ps_2(studentMajor(ps_1) \wedge studentMajor(ps_2) \wedge ps_1.sid = s.sid \wedge ps_2.sid = s.sid \wedge ps_1.major \neq ps_2.major))$$

    (b) • Formulate this constraint as a boolean SQL query.

26. Consider the constraint "*Each student and his or her friends work for the same department.*"

    (a) • Formulate this constraint in TRC.

    $$\forall f\, \forall w_1\, \forall w_2((hasFriend(f) \wedge employedBy(w_1) \wedge employedBy(w_2) \wedge f.sid1 = w_1.sid \wedge f.sid2 = w_2.sid) \rightarrow w_1.dname = w_2.dname).$$

    Equivalently,

    $$\nexists f \exists w_1\, \exists w_2(hasFriend(f) \wedge employedBy(w_1) \wedge employedBy(w_2) \wedge f.sid1 = w_1.sid \wedge f.sid2 = w_2.sid \wedge w_1.dname \neq w_2.dname).$$

    (b) • Formulate this constraint as a boolean SQL query.

27. Consider the constraint "*Some employed student has a salary that is strictly higher than the salary of each of his or her employed friends.*"

    (a) • Formulate this constraint in TRC.

    $$\exists s \exists w(Student(s) \wedge employedBy(w) \wedge s.sid = w.sid \wedge \forall f \forall w_1(hasFriend(f) \wedge employedBy(w_1) \wedge f.sid1 = s.sid \wedge f.sid2 = w_2.sid) \rightarrow w.salary > w_1.salary)$$

    Equivalently,

    $$\exists s \exists w(Student(s) \wedge employedBy(w) \wedge s.sid = w.sid \wedge \neg\, \exists f \exists w_1(hasFriend(f) \wedge employedBy(w_1) \wedge f.sid1 = s.sid \wedge f.mid = w_1.sid \wedge w.salary \leq w_1.salary))$$

    (b) • Formulate this constraint as a boolean SQL query.