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

Dirk Van Gucht

Released: August 25, 2022
Due: September 8, 2022 by 11:45pm

1 Introduction

The goals for this assignment are to

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

To turn in your assignment, you will need to upload to Canvas a file with name `assignment1.sql` which contains the necessary SQL statements that solve the graded problems in this assignment. Analogously, you will need to upload to Canvas a file with name `assignment1.py` which contains the necessary Python statements that solve the graded problems in this assignment.

[*]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.

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 and `assignment1.py` must be so that the AI's can run it in their PostgreSQL and Python environments. You should use the `Assignment1Script.sql` and `Assignment1Script.py` files to construct the `assignment1.sql` file and the `assignment1.py`, respectively . (Note that the data to be used for this assignment is included in this file.) In addition, you will need to upload a separate `assignment1SQL.txt` file that contains the results of running your SQL code and a `assignment1Python.txt` file that contains the results of running your Python code. Finally, you need to upload a file `assignment1.pdf` that contains the solutions to the graded problems that require it.[2]

For the problems in this assignment we will use the following database schema:[3]

$$\begin{array}{l}
\texttt{Student}(\underline{\texttt{sid}}, \texttt{sname}, \texttt{homeCity}) \\
\texttt{Department}(\underline{\texttt{deptName}}, \texttt{mainOffice}) \\
\texttt{Major}(\underline{\texttt{major}}) \\
\texttt{employedBy}(\underline{\texttt{sid}}, \texttt{deptName}, \texttt{salary}) \\
\texttt{departmentLocation}(\underline{\texttt{deptName}, \texttt{building}}) \\
\texttt{studentMajor}(\underline{\texttt{sid}, \texttt{major}}) \\
\texttt{hasFriend}(\underline{\texttt{sid1}, \texttt{sid2}})
\end{array}$$

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.)

[2]It is strongly recommended that you use Latex to write safe 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).

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

A pair (s_1, s_2) in hasFriend indicates that student s_1 considers student s_2 as a friend. It is possible that 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.

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 and in the Assignment1Script.py.

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 `Assignment1Script.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.[4]

   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 this that discuss your examples. Nothing about this problem should go into the `assignment1.pdf` file.

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

3 Expressing queries in SQL and Python

Remark 2 *For this assignment, you are required to use tuple variables in your SQL statements.*[5] *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 and Python statements for the following queries.

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.

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.

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.

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.)

[5] 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.

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 safe TRC queries to SQL and Python

The problems in this section require expressing safe TRC queries as equivalent SQL and Python 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 safe 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 '∀' which, in SQL, is most directly translated as '`true` = `all`'. For the above safe 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);
```

And, we get the Python query

```
[{'deptName': w1['deptName'], 'sid': w1['sid']}
  for w1 in employedBy
  if all [w1['salary'] >= w2['salary']
          for w2 in employedBy
          if  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 safe 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);
```

And, in Python

```
[{'deptName': w1['deptName'], 'sid': w1['sid']}
  for w1 in employedBy
  if all([w1['deptName'] != w2['deptName'] or w1['salary'] >= w2['salary']
          for w2 in employedBy
          if  w1['deptName'] == w2['deptName']])]
```

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 safe 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);
```

And, in Python

```
[{'deptName': w1['deptName'], 'sid': w1['sid']}
  for w1 in employedBy
  if not any([w1['deptName'] == w2['deptName'] and w1['salary'] < w2['salary']
          for w2 in employedBy])]
```

Alternatively, to express a safe 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);
```

If we define the following Python 'exists' function, we can also express this query in Python in a similar way as we did above for SQL.

```
def exists(R): return R != []

[{'deptName': w1['deptName'], 'sid': w1['sid']}
  for w1 in employedBy
  if not exists ([1
                  for w2 in employedBy
                  if w1['deptName'] == w2['deptName'] and w1['salary'] < w2['salary']])]
```

Consider the following queries expressed in safe TRC. Translate each of these queries in three conceptually different ways as an equivalent SQL queries and as an equivalent Python queries. Consult Example 1 as a guide to solve these problems.

13. ●

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

14. ●

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

15. ●

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

16. ●

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

17. ●

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

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 safe Tuple Relational Calculus

20. Express each of the queries in problems 2, 3, and 4 in Section 3 as safe 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.

    (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.

    (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.

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

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

    (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.

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

6 Formulating constraints in the safe Tuple Relational Calculus and as boolean SQL and Python queries

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

The safe TRC solutions of these problems should be included in the `assignment1.pdf` file and the SQL solutions should be included in the `assignment1-SQL.sql` file. The Python solutions should be included in the `assignment1-Python.py` 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 safe 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);
```

And, in Python

```
all([any([sm['major'] == m['major']
          for m in studentMajor])
     for m in Major])
```

Alternatively, the constraint can be formulated in safe 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);
```

And, in Python

```
not any([not any([sm['major'] == m['major']
                  for sm in studentMajor])
         for m in Major])
```

Alternatively, in SQL

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

If we use the Python 'exists' defined in Example 1, we can express this boolean query in Python as follows

```
not exists ([1
            for m in Major m
            if not exists ([1
                          sm in studentMajor
                          if sm['major'] == m['major']])])
```

22. Consider the constraint "*Some major has fewer than 2 students with that major.*"
    (a) ● Formulate this constraint in safe TRC.
    (b) ● Formulate this constraint as a boolean SQL query.
    (c) ● Formulate this constraint as a boolean Python query.
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 safe TRC.
    (b) ● Formulate this constraint as a boolean SQL query.
    (c) ● Formulate this constraint as a boolean Python query.
24. Consider the constraint "*All students working in a same department share a major and earn the same salary.*"
    (a) ● Formulate this constraint in safe TRC.
    (b) ● Formulate this constraint as a boolean SQL query.
    (c) ● Formulate this constraint as a boolean Python query.
25. Consider the constraint "*Each student is employed by a department and has at least two majors.*"
    (a) ● Formulate this constraint in safe TRC.
    (b) ● Formulate this constraint as a boolean SQL query.
    (c) ● Formulate this constraint as a boolean Python query.
26. Consider the constraint "*Each student and his or her friends work for the same department.*"
    (a) ● Formulate this constraint in safe TRC.
    (b) ● Formulate this constraint as a boolean SQL query.
    (c) ● Formulate this constraint as a boolean Python 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 safe TRC.
    (b) ● Formulate this constraint as a boolean SQL query.
    (c) ● Formulate this constraint as a boolean Python query.

7 Theoretical Problem about Safe Tuple Relational Calculus

28. ● Prove that each query expressed in safe TRC can be computed in finite time.

29. ● Prove that each constraint expressed in safe TRC can be computed in finite time.