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

Dirk Van Gucht

Released: Tuesday August 24, 2021
Due: Wednesday September 8, 2021 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 and constraints 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 problems in this assignment. The `assignment1.sql` file must be such

*This assignment covers lectures 1 through 4

[1]To solve this assignment, you will need to download and install PostgreSQL (version 12 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.

that the AI's can run it in their PostgreSQL environment. In addition, you will need to upload a separate `assignment1.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`.) Finally, you will need to upload an `assignment1.pdf` file that contains the solutions for problems related to TRC.[3]

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

```
Person(pid, pname, city)
Company(cname, headquarter)
Skill(skill)
worksFor(pid, cname, salary)
companyLocation(cname, city)
personSkill(pid, skill)
hasManager(eid, mid)
```

In this database we maintain a set of persons (`Person`), a set of companies (`Company`), and a set of (job) skills (`Skill`). The `pname` attribute in `Person` is the name of the person. The `city` attribute in `Person` specifies the city in which the person lives. The `cname` attribute in `Company` is the name of the company. The `headquarter` attribute in `Company` is the name of the city wherein the company has its headquarter. The `skill` attribute in `Skill` is the name of a (job) skill.

A person can work for at most one company. This information is maintained in the `worksFor` relation. (We permit that a person does not work for any company.) The `salary` attribute in `worksFor` specifies the salary made by the person.

The `city` attribute in `companyLocation` indicates a city in which the company is located. (Companies may be located in multiple cities.)

A person can have multiple job skills. This information is maintained in the `personSkill` relation. 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 (e, m) in `hasManager` indicates that person e has person m as one of his or her managers. We permit that an employee has multiple managers and that a manager may manage multiple employees. (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 for the same company.

The domain for the attributes `pid`, `salary`, `eid`, and `mid` is `integer`. The domain for all other attributes is `text`.

[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.

We assume the following foreign key constraints:

- `pid` is a foreign key in `worksFor` referencing the primary key `pid` in `Person`;
- `cname` is a foreign key in `worksFor` referencing the primary key `cname` in `Company`;
- `cname` is a foreign key in `companyLocation` referencing the primary key `cname` 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`;
- `eid` is a foreign key in `hasManager` referencing the primary key `pid` in `Person`; and
- `mid` is a foreign key in `hasManager` referencing the primary key `pid` in `Person`;

The file `data.sql` contains the data supplied for this assignment.

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. To solve this problem, you will need to experiment with the relation schemas and instances for this assignment. For example, you should consider altering primary keys and foreign key constraints and then consider various sequences of insert and delete operations. You may need to change some of the relation instances to observe the desired effects. Certain inserts and deletes should succeed but other should generate error conditions. (Consider the lecture notes about keys, foreign keys, and inserts and deletes as a guide to solve this problem.)

3 Formulating queries in SQL

For this assignment, you are required to use tuple variables in your SQL statements. For example, in formulating the query "Find the pid and pname of each person who lives in Bloomington" you should write the query

```
SELECT  p.pid, p.pname
FROM    Person p
WHERE   p.city = 'Bloomington'
```

rather than

```
SELECT  pid, pname
FROM    Person
WHERE   city = '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 the pid, pname of each person who (a) lives in Bloomington, (b) works for a company where he or she earn a salary that is higher than 30000, and (c) has at least one manager.

3. Find the pairs (c_1, c_2) of different company names who headquarters are located in the same city.

4. Find the pid and pname of each person who lives in a city that is different than each city in which his or her managers live. (Persons who have no manager should not be included in the answer.)

5. Find each skill that is the skill of at most 2 persons.

6. Find the pid, pname, and salary of each employee who has at least two managers such that these managers have a common job skill but provided that it is not the 'Networks' skill.

7. Find the cname of each company that not only employs persons who live in MountainView.

8. For each company, list its name along with the highest salary made by employees who work for it.

9. Find the pid and pname of each employee who has a salary that is higher than the salary of each of his or her managers. (Employees who have no manager should not be included.)

4 Translating TRC queries to SQL

Consider the following queries formulated in TRC. Translate each of these queries to an equivalent SQL query.[5]

You should note that this translating, modulo the handling of universal quantifiers, is almost a syntactic rewrite of the way in which the queries are formulated in TRC. This underscores the close correspondence between TRC and SQL.

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

10.

$$\{p.pid, p.pname, w.cname, w.salary \mid Person(p) \wedge worksFor(w) \wedge p.pid = w.pid \wedge p.city = \text{'Bloomington'} \wedge 40000 \leq w.salary \wedge w.cname \neq \text{'Apple'}\}.$$

11.

$$\{p.pid, p.pname \mid Person(p) \wedge \exists c \exists w(Company(c) \wedge worksFor(w) \wedge c.cname = w.cname \wedge p.pid = w.pid \wedge c.headquarter = \text{'LosGatos'} \wedge \exists hm \exists m(hasManager(hm) \wedge Person(m) \wedge hm.eid = p.pid \wedge hm.mid = m.pid \wedge m.city \neq \text{'LosGatos'}))\}.$$

12.

$$\{s.skill \mid Skill(s) \wedge \neg(\exists p \exists ps\, Person(p) \wedge personSkill(ps) \wedge p.pid = ps.pid \wedge ps.skill = s.skill \wedge p.city = \text{'Bloomington'})\}.$$

13.

$$\{m.pid, m.pname \mid Person(m) \wedge \forall hm((hasManager(hm) \wedge hm.mid = m.pid) \rightarrow \exists e(Person(e) \wedge hm.eid = e.pid \wedge e.city = m.city))\}$$

[5] You can not use SQL `JOIN` operations or aggregate functions.

5 Formulating queries in the Tuple Relational Calculus

Formulate each of the queries in the even-numbered problems (i.e., problems 2, 4, 6, and 8) in Section 3 as TRC queries.

The solutions of these problems should be included in the `assignment1.pdf` file.

14. (Problem 2) Find the pid, pname of each person who (a) lives in Bloomington, (b) works for a company where he or she earn a salary that is higher than 30000, and (c) has at least one manager.

$$\{p.pid, p.pname \mid Person(p) \wedge p.city = \texttt{Bloomington} \wedge \exists pw(worksFor(pw) \wedge p.pid = pw.pid \wedge pw.salary > 30000) \wedge \exists hm(hasManager(hm) \wedge hm.eid = p.pid)\}.$$

15. (Problem 4) Find the pid and pname of each person who lives in a city that is different than each city in which his or her managers live. (Persons who have no manager should not be included in the answer.)

$$\{p.pid, p.pname, p.city \mid Person(p) \wedge \exists hm(hasManager(hm) \wedge hm.eid = p.pid) \wedge \neg\exists m(Person(m) \wedge p.city = m.city \wedge hasManager(p.pid, m.pid))\}.$$

16. (Problem 6) Find the pid, pname, and salary of each employee who has at least two managers such that these managers have a common job skill but provided that it is not the 'Networks' skill.

$$\{p.pid \mid worksFor(p) \wedge \exists hm_1 \exists hm_2(hasManager(hm_1) \wedge hasManager(hm_2) \wedge hm_1.eid = p.pid \wedge hm_2.eid = p.pid \wedge hm_1.mid \neq hm_2.mid \wedge \exists ps_1 \exists ps_2(personSkill(ps_1) \wedge personSkill(ps_2) \wedge hm_1.mid = ps_1.pid \wedge hm_2.mid = ps_2.pid \wedge ps_1.skill = ps_2.skill \wedge ps_1.skill \neq \texttt{Networks}))\}.$$

17. (Problem 8) For each company, list its name along with the highest salary made by employees who work for it.

$$\{c.cname, w.salary \mid Company(c) \wedge worksFor(w) \wedge w.cname = c.cname \wedge \neg\exists w_1(worksFor(w_1) \wedge w_1.cname = c.cname \wedge w.salary < w_1.salary)\}.$$

6 Formulating constraints in the Tuple Relational Calculus and SQL

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 1 *Consider the constraint "*Each skill is the skill of a person.*" In TRC, this constraint can be formulated as follows:*

$$\forall s\ Skill(s) \rightarrow \exists ps\,(personSkill(ps) \wedge ps.skill = s.skill)$$

or, alternatively

$$\neg\exists s(Skill(s) \wedge \neg\exists ps(personSkill(ps) \wedge ps.skill = s.skill)).$$

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

```
select not exists (select 1
                   from   Skill s
                   where  not exists (select 1
                                      from   personSkill ps
                                      where  ps.skill = s.skill));
```

18. Each person works for a company and has at least two job skills.

$$\forall p\, Person(p) \rightarrow (\exists w(worksFor(w) \wedge w.pid = p.pid) \wedge \exists ps_1 \exists ps_2(personSkill(ps_1) \wedge personSkill(ps_2) \wedge ps_1.pid = p.pid \wedge ps_2.pid = p.pid \wedge ps_1.skill \neq ps_2.skill))$$

Equivalently,

$$\neg\exists p\, Person(p) \wedge (\neg\exists w(worksFor(w) \wedge w.pid = p.pid) \vee \neg\exists ps_1 \exists ps_2(personSkill(ps_1) \wedge personSkill(ps_2) \wedge ps_1.pid = p.pid \wedge ps_2.pid = p.pid \wedge ps_1.skill \neq ps_2.skill))$$

In SQL,

```
select not exists (select 1
                   from   Person p
                   where  not exists (select 1
                                      from   worksFor w
                                      where  w.pid = p.pid)
                          or
```

```
                        not exists (select 1
                                    from   personSkill ps1, personSkill ps2
                                    where  ps1.pid = p.pid and ps2.pid = p.pid and
                                           ps1.skill <> ps2.skill));
```

19. Some person has a salary that is strictly higher than the salary of each of his or her managers.

$$\exists p \exists w(Person(p) \wedge worksFor(pw) \wedge p.pid = pw.pid \wedge \\ \forall hm \forall mw(hasManager(hm) \wedge worksFor(mw) \wedge hm.eid = p.pid \wedge hm.mid = mw.pid) \rightarrow \\ pw.salary > mw.salary)$$

Equivalently,

$$\exists p \exists w(Person(p) \wedge worksFor(pw) \wedge p.pid = pw.pid \wedge \\ \neg \exists hm \exists mw(hasManager(hm) \wedge worksFor(mw) \wedge hm.eid = p.pid \wedge hm.mid = mw.pid \wedge \\ pw.salary \leq mw.salary))$$

```
select exists (select 1
               from   Person p,worksFor pw
               where  p.pid = pw.pid and
                      not exists (select 1
                                  from   hasManager hm,worksFor mw
                                  where  hm.eid = p.pid and hm.mid = mw.pid and
                                         pw.salary <= mw.salary));
```

20. Each employee and his or her managers work for the same company.

$$\forall em \, \forall e \, \forall m((hasManager(em) \wedge worksFor(e) \wedge worskFor(m) \wedge \\ em.eid = eid \wedge em.mid = m.pid) \rightarrow e.cname = m.cname).$$

Equivalently,

$$\neg \exists em \, \exists e \, \exists m(hasManager(em) \wedge worksFor(e) \wedge worskFor(m) \wedge \\ em.eid = eid \wedge em.mid = m.pid \wedge e.cname \neq m.cname).$$

In SQL

```
select not exists (select 1
                   from   hasManager em, worksFor e, worksFor m
                   where  em.eid = e.pid and
                          em.mid = m.pid and
                          e.cname <> m.cname);
```