SQL Exercise

Tom Kelliher, CS 317

1 Introduction

You'll be working with an example university database in the first part of this exercise. You'll be comparing the database schema against its E-R diagram, creating the database and populating it, and then creating SQL queries to answer questions about the university and gain an understanding of how the constraints work.

You'll create, by "hand," and work with a small movie database in the second part of this exercise.

2 Exercise

2.1 University Database

1. Login to phoenix and download the two SQL files (`univDDL.sql` and `univDat.sql`) for the University database from the course web site on phoenix.
2. Open the DDL (schema) file for viewing and note/think about the following:
   (a) The `CONSTRAINT` keyword is missing from check and key definitions. While the keyword is optional, using it allows the constraints to have names, allowing for them to be modified later on.
   (b) Note the column data types used. These are just a subset of all the possible data types. See Chapter 8 of the PostgreSQL documentation for the complete list.
3. Keeping the DDL file open, open the E-R diagram for the database. There are a few differences between the format used here and the textbook's format — an entity's attributes are part of the entity's rectangle and the "one" side of a one-to-many relationship is shown using an arrow tip. For example, `course_dept` is one-to-many from `department` to `course` and `takes` is many-to-many.
4. Why does the schema have more tables than the E-R diagram has entities?
5. Describe how the following relationships in the E-R diagram are handled in the schema:
   (a) One-to-many `course_dept`
   (b) Recursive `prereq`
   (c) Many-to-many `takes`
6. `section` is a weak entity, dependent upon `course` How is this handled in the schema?

7. Time to play detective — what's going on with `sec_time_slot`? Compare the E-R diagram to the DDL. Can you spot the discrepancy?

8. Open a shell window and use the `cd` command to navigate to the directory where you downloaded the two SQL files for the university. Run `psql`, PostgreSQL's command line access tool.

9. Create a new database, connect to it, and input the two sql files:

```
-- Database names have to be unique across the system; replace 'yyy'
-- with a unique number.
create database univyyy;
-- Connect to your university database.
\c univyyy
-- Input the SQL to create the schema and populate the database.
\i univDDL.sql
\i univDat.sql
```

10. Try running a few basic queries:

```
select * from instructor;

select name from instructor
where dept_name = 'Comp. Sci.' and salary > 70000;

select * from instructor, department
where instructor.dept_name = department.dept_name;

select * from instructor join department using (dept_name);

select student.name
from    (student join takes using (id))
     join
        (instructor join teaches using (id))
     using (course_id, sec_id, semester, year)
where instructor.name = 'Katz';

-- Use up-arrow or Ctrl-p to bring up the previous query, add the word
-- 'distinct' after 'select', and re-run the query.

select max(salary) from instructor;

select count(salary) from instructor where salary >= 90000;
```

11. Write and execute SQL queries to determine the following:

    (a) Find the names of all the instructors from the Biology department.

    (b) Find the names of courses in the computer science department which have three credits.

    (c) For the student with ID 12345, show course ids and titles of all courses for which the student has registered.

    (d) As above, but show the total number of credits for such courses taken by that student. You should use SQL aggregation on courses taken by the student.

    (e) As above, but display the total credits for *each* of the students, not just the student with ID 12345, along with the ID of the student; don't bother about the name of the student. (Don't bother about students who have not registered for any course, they can be omitted.)

    (f) Find the names of all students who have taken any computer science course ever (there should be no duplicate names).

    (g) Display the IDs of all instructors who have never taught a course. Interpret "taught" as "taught or is scheduled to teach."

    (h) As above, but display the names of the instructors also, not just the IDs.

12. Practice with insert, update, and delete:

    (a) Try to insert a record for yourself into the `instructor` table, using one of the existing id's. Why does this fail?

    (b) Try to insert a record for yourself into the `instructor` table, using a unique id, with a salary of $42.00. Why does this fail?

    (c) Insert a record for yourself into the `instructor`, giving yourself a salary of $42,000.

    (d) Update your salary to $45,000.

    (e) Give all instructors a raise of 10%. Use a single SQL statement.

    (f) Delete your record from `instructor`.

    (g) If you delete the instructor with id 10101 from `instructor`, what will happen in the `teaches` table? Why? Delete this instructor and check to see how the `instructor` and `teaches` tables have changed.

13. Connect back to your personal database, drop your university database, and exit `psql`.

```
-- Replace my username with your username.
\c kelliher
drop database univyyy;
\q
```

2.2 Movie Database

1. Create a movie database. (You may find it easiest to put your SQL commands into a file, and use `\i` in `psql` to run them.) Create three tables, one for actors (AID, name), one for movies (MID, title) and one for actor_role(MID, AID, rolename). Use appropriate data types for each of the attributes, and add appropriate primary/foreign key constraints.

2. Insert data into the above tables (approximately three to six rows in each table), including data for the actor "Charlie Chaplin", and for yourself.

3. Write a query to list all movies in which actor "Charlie Chaplin" has acted, along with the number of roles he had in that movie.

4. Write a query to list all actors who have not acted in any movie.

5. List names of actors, along with titles of movies they have acted in. If they have not acted in any movie, show the movie title as null. (An outer join will be useful here.)

6. Drop your movie database.