## 1. Preliminaries

Before starting on this assignment, please be sure to read the General Instructions that are on Piazza (under Resources->General Resources), and also make sure you are able to log in to the class PostgreSQL server. You'll get help on this in your Lab Section, not the Lectures, so *be sure to attend Lab Sections*.

## 2. Goal

The goal of the first assignment is to create a PostgreSQL data schema with 6 tables. That is all that is required in this assignment; don't do anything that's not required. The other Lab Assignments are much more difficult. In your Lab Sections, you may be given information about how to load data into a table and issue simple SQL queries, because that's fun, but loading data and issuing queries are **not** required in this assignment. (That will show up in the Lab2 assignment.)

## 3. Lab1 Description

### 3.1 Create PostgreSQL Schema Lab1

As we noted in the general instructions, you will create a Lab1 schema to set apart the database tables created in this lab from tables you will create in future labs, as well as from tables (and other objects) in the default (public) schema. Note that the meaning of schema here is specific to PostgreSQL, and distinct from the general meaning of schema. See here for more details on PostgreSQL schemas. You create the Lab1 schema using the following command:

```
CREATE SCHEMA Lab1;
```

[PostgreSQL makes all identifiers lowercase; that's okay unless you put them in quotation marks, e.g.,, "Lab1". That's okay.; you don't have to bother using quotation marks for identifiers. We use capitals for readability, but it's okay (and equivalent) if you use lab1 as the schema name.]

Now that you have created the schema, you want to make Lab1 be the default schema when you use psql. If you do not set Lab1 as the default schema, then you will have to qualify your table names with the schema name (e.g., by writing Lab1.customer, rather than just customer). To set the default schema, you modify your search path as follows. (For more details, see here.)

```
ALTER ROLE username SET SEARCH_PATH to Lab1;
```

You will need to log out and log back in to the server for this default schema change to take effect. (Students **often forget** to do this, and then are surprised that their tables aren't in the expected schema.)

### 3.2 Tables

You'll be creating tables for a very simplified version of a Movie Theater database schema, with tables for Movies, Theaters, TheaterSeats, Showings, Customers and Tickets. Data types and referential integrity for the attributes in these 6 Movie Theater tables are described in the next section. No, this schema doesn't provide everything that Fandango has, but it's a decent start. We'll call this the Gavotte database schema; like the fandango, a gavotte is a kind of dance.

**Important**: To receive full credit, you must use the attribute names as given, and the attributes must be in the order given. Also, the data types and referential integrity must match the specifications given in the next section. Follow directions; do not do more than you're asked to do in this assignment.

Movies(movieID, name, year, rating, length, totalEarned)

Theaters(theaterID, address, numSeats)

TheaterSeats(theaterID, seatNum, brokenSeat)

Showings(theaterID, showingDate, startTime, movieID, priceCode)

Customers(customerID, name, address, joinDate, status)

Tickets(theaterID, seatNum, showingDate, startTime, customerID, ticketPrice)

The underlined attribute (or attributes) identifies the Primary Key of each table.

- A movie has an ID, a name, the year in which it was made, a rating, a length (in minutes), and the total that the movie has earned.
- A theater has an ID, an address, and the number of seats that it holds.
- A theater seat is a particular seat (seatNum) inside a particular theater. brokenSeat indicates whether that seat is broken. The theaterID for that theater seat must appear in the Theaters table. If the theater hold 12 seats, that seatNum will be a value from 1 to 12. (But we won't explain how to check for valid seat numbers until later in the term.)
- A showing describes the movie that is being shown at a theater starting at a particular time on a particular date. A showing has a priceCode that will be explained later.
    - That movie must appear in the Movies table.
    - That theater must appear in the Theaters table.
- A customer has an ID, a name, an address, a date when they joined as a customer, and a status, which will be explained later.
- A ticket indicates that a particular customer purchased a particular seat at a particular showing.
    - That customer must appear in the Customers table.
    - That theater and start time must appear in a tuple in the Showings table..
    - That theater and seatNum must appear in a tuple in the TheaterSeats table.

In this assignment, you'll just have to create tables with the correct table names, attributes, data types, primary keys and referential integrity. "Must appear in" means that there's a referential integrity requirement. **Be sure not to forget primary keys and referential integrity when you do Lab1!**

### 3.2.1 Data types

Sometimes an attribute (such as movieID, theaterID, address and name) appears in more than one table. Attributes that have the same attribute name might not have the same data type in all tables, but in our schema, they do.

- For movieID, theaterID, customerID, year, length, numSeats and seatNum, use *integer*.
- For status, which classifies types of customers, priceCode, which classifies showings, and rating, which classifies movies, use *character* with fixed length 1. (We'll explain values of these attributes later.)
- For name, use *character* of variable length, with maximum length 30.
- For address, use *character* of variable length, with maximum length 40.

- ticketPrice should be *numeric,* with at most 2 digits to the left of the decimal point and 2 decimal digits after it.
- totalEarned should be *numeric,* with at most 5 digits to the left of the decimal point and 2 decimal digits after it.
- showingDate and joinDate should be of type *date*.
- startTime should be of type *time*.
- The brokenSeat attribute, which indicates whether a theater seat is broken, should be of type *boolean*.

You must write a CREATE TABLE statement for each of the six tables in Section 3.2. Write the statements in the same order that the tables are listed above. Use the data types, primary keys and referential integrity described above. Save your statements in the file create.sql

**4. Testing**

While you're working on your solution, it is a good idea to drop all objects from the schema every time you run the create.sql script, so you can start fresh. Dropping each object in a schema may be tedious, and sometimes there may be a particular order in which objects must be dropped. The following command, which you should put at the top of your create.sql, will drop your Lab1 schema (and all the objects within it), and then create the (empty) schema again:

```
DROP SCHEMA Lab1 CASCADE;
CREATE SCHEMA Lab1;
```

Before you submit your Lab1 solution, login to your database via psql and execute your create.sql script. As you'll learn in Lab Sections, the command to execute a script is: \i <filename>. Verify that every table has been created by using the command: \d

Also, verify that the attributes of each table are in the correct order, and that each attribute is assigned its correct data type using the following command: \d <table>.

**5. Submitting**

1. Save your script as create.sql You may add informative comments to your scripts if you want. Put any other information for the Graders in a separate README file that you may submit.

2. Zip the file(s) to a single file with name Lab1_XXXXXXX.zip where XXXXXXX is your 7-digit student ID. For example, if a student's ID is 1234567, then the file that this student submits for Lab1 should be named Lab1_1234567.zip

   If you have a README file (which is not required), you can use the Unix command:

   ```
   zip Lab1_1234567 create.sql README
   ```

   If you don't have a README file, to create the zip file you can use the Unix command:

   ```
   zip Lab1_1234567 create.sql
   ```

(Of course, you should use **your own student ID**, not 1234567.) Submit a zip file, even if you only have one file.

Submit the zip file on Canvas under Assignment Lab1. Please be sure that you have access to Canvas for CSE 180. Registered students should automatically have access; students who are not registered in CSE 180 will not have access. No students will be admitted to CSE 180 after the Lab1 due date.

If you are working on the UNIX timeshare and your zip file is located there, you will need to copy your file to your computer so that you can upload it to Canvas through your browser. For that, you will need an FTP (File Transfer Protocol) client to securely transfer files from the UNIX timeshare. A widely used secure FTP client is Filezilla. Installation instructions are found in the site of FileZilla (make sure you install the distribution suitable for your operating system). After opening the Filezilla client, you will need to set the host field to unix.ucsc.edu, the username to your CruzId and the password to your Blue password, while the port number should be set to 22 (the default port for remote login). By clicking the Quickconnect button, if your credentials are correct, you will connect and be able to see the contents of your remote Unix folder at the right pane (under the title "Remote site"), while the left pane (under the title "Local site") will display the contents of your local file system. With the mouse, you can drag the file from the Unix folder and drop it to the desired location at your computer. This will transfer the file to your local machine, without erasing it from its original remote location. Filezilla is only one of several options for an FTP client. If you are finding it difficult to install the necessary tools and successfully do file transfers, you should promptly ask for help in the Lab Sections; **do not** postpone this until the deadline date. The computers at the Lab also have pre-installed SSH and FTP clients (PuTTY and PSFTP).

Other approaches to copy files includes using SCP (Secure Copy) and using Cut-and-Paste, where you copy the contents of the file from the unix system, and then paste contents into a file on your computer. Cut-and-Paste may work with for small files, but that's a hack that does not work well for large files.

The CSE 180 Teaching Assistants, Golam Md Muktadir and Natasha Mittal, will discuss approaches to access unix remotely (SSH for Mac/Linux and PuTTY for Windows) and to move files to your computer (SCP for Mac/Linux and Filezilla for Windows/Mac/Linux) with you during Lab Sections. Attend your Lab Section to ensure that you know how to handle this correctly!

Lab1 is due by 11:59pm on Sunday, Jan 19. **Late submissions will not be accepted (Canvas won't take them), and there will be no make-up Lab assignments.** Check to make sure that your submission is on Canvas, and that you've submitted the correct file. You will receive **no credit** if you accidentally submit the wrong file, even if you attempt to "prove" that you completed the correct file on time.