1. Preliminaries

Before starting on this assignment, please be sure to read the PostgreSQL Info file (PSQL_Info.pdf) and the Docker Install Guide that are on Piazza.

2. Goal

The goal of the first assignment is to create a PostgreSQL data schema with a number of different tables. That is all that is required in this assignment. The other Lab Assignments will be more challenging. 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

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, unless you put them in quotation marks, e.g., "Lab1". But you don't have to bother using quotation marks for identifiers. We use capitals in assignments 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 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 cse180 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.) To see your current SEARCH_PATH, enter:

```
SHOW SEARCH_PATH;
```

If you forget to do the ALTER ROLE for Lab1 to modify your search path then your schema will be the default schema, PUBLIC.

3.2 Tables

You will be creating a (simplified) database for a pharmacy store chain. The data types and Referential Integrity for the attributes in these 7 tables are described in the next section. This schema might not provide everything that an actual database of such a company would include, but it's a decent start.

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. Do not do more than you're asked to do in this assignment.

Passenger(passengerID, passengerName, email, dateOfBirth, frequentFlyer)

Airport(airportCode, city, country, numRunways, avgDelayMinutes)

Flight(flightID, departureAirport, arrivalAirport, scheduledDeparture, scheduledArrival, actualDeparture, actualArrival, aircraftType)

CrewMember(crewID, crewName, crewRole, yearsExperience)

Reservation(reservationID, passengerID, flightID, bookingDate, seatClass, ticketPrice, paymentMethod)

CheckIn(reservationID, passengerID, checkInTime, bagCount, seatNumber)

FlightCrewAssignment (flightID, crewID, compensation)

The underlined attribute (or attributes) identifies the Primary Key of each table. A table can only have one Primary Key, but that Primary Key may involve multiple attributes.

- The relation **Passenger** specifies the passenger's ID, their name, email address, date of birth, and frequent flyer status.
- The relation **Airport** specifies the code of an airport, its city and country, the number of runways it has, and its average flight delay in minutes.
- The relation **Flight** specifies the ID of a flight, the departure and arrival airports, the scheduled departure and arrival times, the actual departure and arrival times, and the aircraft type.
    - Any departureAirport or arrivalAirport that appears in a Flight row must appear as an airportCode in the Airport table.
- The relation **CrewMember** specifies the crew member's ID, their name, their role (e.g., Pilot, Attendant), and their years of experience.
- The relation **Reservation** records an ID for every reservation made by a passenger, including the passenger ID, the flight ID, the booking date, the seat class, the ticket price, and the method of payment.
    - Any passengerID that's in a Reservation row must appear as a passengerID in the Passenger table.
    - Any flightID that's in a Reservation row must appear as a flightID in the Flight table.

- The relation **CheckIn** specifies the reservation ID, the passenger ID, the check-in time, the number of bags checked, and the seat number assigned.
    - Any reservationID that's in a CheckIn row must appear as a reservationID in the Reservation table.
    - Any passengerID that's in a CheckIn row must appear as a passengerID in the Passenger table.
- The relation **FlightCrewAssignment** records which crew members are assigned to which flights, including the flight ID, the crew ID, and the compensation amount for that assignment.
    - Any flightID that's in a FlightCrewAssignment row must appear as a flightID in the Flight table.
    - Any crewID that's in a FlightCrewAssignment row must appear as a crewID in the CrewMember 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 as" 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

- For passengerID, crewID, flightID, reservationID, numRunways, avgDelayMinutes, bagCount, and yearsExperience use integer.
- For passengerName, email, city, country, crewName, crewRole, aircraftType, paymentMethod, and seatNumber use character of variable length, with maximum length 40.
- For airportCode, departureAirport, and arrivalAirport use character with fixed length 3.
- For frequentFlyer and seatClass use character with fixed length 1. We'll say more about the values of these attributes later.
- For dateOfBirth and bookingDate use the date type.
- For scheduledDeparture, scheduledArrival, actualDeparture, actualArrival, and checkInTime use the timestamp type.
- For ticketPrice and compensation use numeric, with at most 7 digits to the left of the decimal point and 2 digits after it.

You must write a CREATE TABLE statement for each of the 7 tables. Write the statements in the same order that the tables are listed above. **Use the data types, Primary Keys and Referential Integrity described above.** Don't do anything beyond that, even if you think that it's sensible. Save your statements in the file create_lab1.sql.

PostgreSQL maps all SQL identifiers (e.g., table names and attributes) to lowercase. That's okay in your assignments. You won't lose points for Lab1 because Passenger appears in the database as passenger. It is possible to specify specific case choices for an identifier by putting that identifier inside double-quote symbols, e.g., as "Passenger". But then every time you refer to that identifier, you'll have to use the double-quotes. "PASSENGER" is not the same identifier as "Passenger", and neither of those is the same as Passenger (written without double-quotes), which PostgreSQL maps to passenger. We will use capitalization for readability, but we won't bother using double-quotes in our own Lectures, Lab Assignments and Exams.

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_lab1.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. (Why?) The following command, which you should put at the top of your create_lab1.sql script, 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;
```

The first statement will result in an error if the Lab1 Schema doesn't exist, but execution of your script will continue after that.

Before you submit your Lab1 solution, login to your database via psql and execute your create_lab1.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 When you execute \d, the tables may be displayed in any order, not necessarily in the order in which you created them.

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

We've supplied some load data that you can use to test your solution in the file load_lab1.sql. After you've created your tables, using the command : \i create_lab1.sql, you can load that data in psql by executing the command: \i load_lab1.sql. (Why will loading the data twice always result in errors?) If your solution fails on the load data, then it's likely that your solution has an error. But although testing can demonstrate that a program is buggy, testing cannot prove that a program is correct.

5. Submitting

1. Save your script as create_lab1.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_lab1.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_lab1.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. Registered students should automatically have access; students who are not registered will not be submit solutions. You can replace your original solution, if you like, up to the Lab1 deadline. (Canvas will give the new file a slightly different name, but that' okay.)

The TAs will help you set up PostgreSQL in a docker container and show you how to move and zip the necessary files in order to submit them 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, October 12. **Late submissions will not be accepted; Canvas won't take them, nor will we.** Always check to make sure that your submission is on Canvas, and that you've submitted the correct file.