1 Preliminaries

Before starting on this assignment, make sure you are able to log in to the class PostgreSQL server.

2 Goal

The second assignment has multiple goals:

1. Create a PostgreSQL schema
2. Create tables using SQL
3. Load data into the tables
4. Merge data where necessary
5. Write queries over the data

3 Lab2 Description

3.1 Create PostgreSQL Schema

We will create a lab2 schema to set apart the database objects created in this lab from ones we will create in the future, as well as from objects in the default (public) schema. Note that the meaning of schema here is specific to PostgreSQL and distinct from the general meaning used in class and in the previous lab. See here for more details on PostgreSQL schemas. We create the schema like this:

```
CREATE SCHEMA lab2;
```

Now that we have created the schema, we want to set lab2 to be our default schema when we use psql. If we do not, then we have to qualify our table names with the schema name (e.g. lab2.mg_patient). To set the default schema, we modify our search path. For more details, see here. ALTER ROLE <username> SET SEARCH_PATH TO lab2;

3.2 Create tables

You will create tables in schema lab2 for the tables in *HospitalDB and NewMedicines*.

Important: To receive full credit, you must use the attribute names as given, and the attributes must be in the order given.

```
h_patients (patient_id, name, address, email, doctor_id,admitted)
h_medicines (medicine_id, name, price)
h_doctors (doctor_id, name, location, speciality)
h_prescriptions (prescription_id, doctor_id, medicine_id, patient_id, prescription_date)
```

Listing 1: HospitalDB

```
new_medicines (name, price)
```

Listing 2: NewMedicines

```
mg_medicines (medicine_id, name, price)
```

Listing 3: Merged Medicines

3.2.1 Data Types

For id attributes, price and date use integer. For name, email, dates, and all address/location-related fields, use text. For the admitted attribute, which indicates whether the patient is in the hospital, use boolean.

For specialty of doctors, create the following enumerated type. Be sure to put this command *immediately before* the command to create the h_doctors table.

```
CREATE TYPE specialty AS ENUM (
        'D',
        'OBG',
        'OPH’,
        'PD',
        'OS’
);
```

You will write a CREATE TABLE command for each of the six tables in the above listings. Write the commands in the order the tables are listed above. Save the commands in the file create.sql.

3.3 Load Data

Download the data from the Resources page on Piazza. There are five files, one for each table except the mg_medicines. As discussed in lab sections, you can copy data from each file into its corresponding table using the following psql command. It is equivalent to the SQL COPY command, but does not require superuser privileges for the database.

```
\copy <table> FROM  '/path/to/filename.csv' delimiter ',' csv;
```

Save the commands in a logical order in the file load.sql.

3.4 Merge Data

The idea of merging is that a hospital has received a price list for new medicines (name, price). It will retain existing prices for medicines that it already had, but add new medicines (with prices, adding IDs) if it didn’t already have those medicines.

3.4.1 Copy Data

You need to populate the merged table (mg_medicines) with data from both the *h_medicines* and *new_medicines* tables. Note that h_medicines tuples have medicine_id values, but new_medicines tuples do not. For this reason, you'll first copy data from this table into the merged table.

To copy, use the below syntax. Specify the destination table and attributes, and the source table and attributes.

```
INSERT INTO <destination table> (attribute 1, attribute 2, ...)
SELECT <source table>.<attribute a>, <source table>.<attribute b>, ...
FROM <source table>
```

Ensure that there is the same number of source and destination attributes, and that they are in the correct order. Here's an example: Suppose table dst has attributes X, Y, Z, and table src has attributes P, Q, R, S. The following will copy values of P and S from src into Y and Z in new rows of dst:

```
INSERT INTO dst (Y, Z)
SELECT src.P, src.S FROM
src;
```

3.4.2 Create Sequences

Create a sequence for the merged table mg_medicines. For simplicity, we hard-code the start of the sequence to start just after the highest respective values already in the merged table:

```
CREATE SEQUENCE mg_medicines_seq START 80;
```

3.4.3 Associate Sequences with Attributes

Then ensure that the id attribute in the merged table uses the sequence to generate values:

```
ALTER TABLE mg_medicines
ALTER COLUMN medicine_id
SET DEFAULT NEXTVAL ('mg_medicines_seq');
```

Now when you insert a new record into one of these merged tables, you should omit the id attribute (and corresponding value). The id will be automatically generated by the sequence.

3.4.4 Copy Additional Data

You've copied the *h_medicines* data into the merged table *mg_medicines*. Now copy the *new_medicines* data into *mg_medicines*, including price. But there may be overlaps because some new_medicines may already be *h_medicines*. Existing medicines from *h_medicines* are retained as is in *mg_medicines*, with existing medicines_id and price. Only new_medicines that didn't already exist before should be copied into *mg_medicines*; you don't want to copy the same medicine twice.

That is, select only those *new_medicines* whose name *does not* match any *h_medicines* name. One approach is to select the names of all *new_medicines* medicines. Separately, select the *h_medicines* and *new_medicines* whose names do match. Then use the EXCEPT keyword to find the difference of the two sets. That will give you the names of the new_medicines names that aren't already in h_medicines.

Insert these medicines with their prices into the merged table *mg_medicines*. The sequence you created previously will automatically generate new id values.

Save the commands that you used in a logical order in a file merge.sql that you create, which will allow us to test your work.

3.5 Write queries

1. What are the names of the medicines that are in both *h_medicines* and *new_medicines*? Order the results in alphabetical order.

2. What is the name and price difference for every medicine that is in both *h_medicines* and *new_medicines*? Order the results decreasing by price difference. (Price difference is obtained by taking the price from *h_medicines* and subtracting the price in *new_medicines*.)

3. List name of patient and name of doctor whenever that doctor has prescribed a medication for that patient. Order the result alphabetically by patient name, and then for each patient by doctor name. Even if a doctor has prescribed more than one medication for a particular patient, that patient and doctor should only appear once in the result.

4. What speciality is the least common in the *Hospital database*, and how many doctors have that speciality? Return both the speciality and the number of doctors in one result.

Save each query in its own file, named query1.sql through query4.sql.

4 Testing

While your solution is still a work in progress, it is a good idea to drop all objects from the database every time you run the script, so you start fresh. Of course, dropping each object may be tedious, and there may be a particular order in which objects must be dropped. The following command, put at the top of create.sql, will drop your lab2 schema (and all objects within it), and then create the (empty) schema again:

```
DROP SCHEMA lab2 CASCADE;
CREATE SCHEMA lab2;
```

Before you submit, login to your database via psql and execute all four scripts in order. As discussed in lab sections, the command to execute a script is: \i <filename>. Verify that every table has been created and that the attributes are in the correct order. Verify that each attribute is assigned its correct data type using the following command: \d <table>. This command will also show you the sequence associated with a particular attribute. Verify that all data is loaded into your tables, and that the correct data is copied into the merged tables. Finally, verify the results of the queries.

5 Submitting

1. Save your scripts as create.sql, load.sql, merge.sql, and your queries as query1.sql through query4.sql. You may add informative comments to your scripts. Put any other information for the grader in a separate README file.

2. Copy the scripts to your home directory on unix.ic.ucsc.edu.
3. Login to unix.ic.ucsc.edu. At the shell prompt, submit your work: submit cmps182-sf.s15 lab2 create.sql load.sql merge.sql query [1-4].sql .You can submit more than once. Only your latest submission will be graded.
4. Lab2 is due by 11:59pm on Tuesday Feb 2. Lab2 will be accepted until 11:59pm on Wednesday, Feb 3, but assignments submitted on Wednesday will receive half credit. Only the last submission for each student will be graded.