Homework Assignment #1

(SQL Refresher)

Congratulations are in order! You've just landed a development job at **ShopALot.com**, where you will be working on a website to promote safe and socially distant shopping as we continue to work towards ending the Covid19 pandemic. ShopALot.com was developed by a group of excellent students fed up with being cooped up at home 24x7. Their goal was to help out the stores, especially the smaller local stores, by providing a database-backed service that supports "parking lot shopping." To reduce the traffic inside busy stores such as grocery stores, users who know exactly what they want and verify that it's available will be able to submit their orders for goods in advance and then wait in the parking lot. From there, designated "shoppers" -- ShopALot.com employees -- will be assigned their orders and will shop for them in batches and then perform the handoffs to ShopALot customers who can remain in the comfort of their cars in store parking lots. An E-R diagram of the ShopAlot.com database is available **here** along with an informal English description of their business. Your assignment is to get ahold of the data, familiarize yourself with it, load it pretty much "as is" into a relational database system, explore it via SQL queries to refresh your SQL skills, and then report back to your boss about what you did and how. You should also comment on any changes that you would recommend to their database plans, were they to decide to "stay relational" in terms of their underlying database platform.

Get Ready...

You should start by adding PostgreSQL to the set of applications that you have installed on your favorite laptop or another computer if you don't already have it (refer to the setup document for helpful links). Once you manage to get Postgres installed, you should fire it up and use its **psql** command-line interface to create a toy table, insert a few toy rows, and then run a few toy queries to make sure that you're good to go. The Hoofers sailing club data from Lecture 2 would be a good toy dataset to play with if you don't want to create one of your own. Once you get this far, you may also want to download and install a copy of pgAdmin 4 if you'd like to have a nice GUI to use with PostgreSQL.

Get (Data) Set...

Now that you have PostgreSQL installed and working, your next step is to grab yourself a copy of the relevant ShopALot.com data. To facilitate that process, your boss has given you (a) a description of the data files, which themselves will be in a CSV (comma-separated value) format that is not unlike what you'd get if you were to export an Excel spreadsheet, and (b) a Google Drive location for the actual CSV files containing the records for ShopALot.com.

(a) Data Files' Schema:

```
User (user_id, email, first_name, last_name)

UserPhone(user_id, type, number)
//type can be one of (‘HOME’, ‘OFFICE’, ‘MOBILE’), but avoid creating an
“ENUM” type in PostgreSQL when possible

Customers (user_id)

Shoppers (user_id, capacity)

Vehicles (state, license_plate, year, model, make, color)

Stores (store_id, name, street, city, state, zip_code, phone, categories)

Products(product_id, category, name, description, list_price)
// category can be one of ('Baby Care', 'Beverages', 'Bread & Bakery',
'Breakfast & Cereal', 'Canned Goods & Soups', 'Condiments & Spice & Bake',
'Cookies & Snacks & Candy', 'Dairy & Eggs & Cheese', 'Deli', 'Frozen
Foods', 'Fruits & Vegetables', 'Grains & Pasta & Sides', 'Meat & Seafood',
'Paper & Cleaning & Home', 'Personal Care & Health', 'Pet Care'), similar
to UserPhone(type), it’s not recommended to use “ENUM” type here

Orders (order_id, total_price, time_placed, pickup_time, customer_id,
shopper_id, state, license_plate, store_id, time_fulfilled)

OrderItems (item_id, qty, selling_price, order_id, product_id)

StockedBy (product_id, store_id, qty)

Own (state, license_plate, user_id)

WorkFor (store_id, shopper_id)
```

(b) Data Set (All ShopALot.com Data):

https://drive.google.com/drive/folders/1u3sejWm6abX6J4WVUZpKoljrEJnsLe34?usp=sharing

Go...!

It’s time to get to work. Here’s what you are to do (using psql or whatever alternative PostgreSQL interface you’ve chosen to use):

1. Create an appropriate PostgreSQL table (using **CREATE TABLE**) for each of the twelve data files. Be sure to pick appropriate data types (**int**, **float**, **varchar**, **timestamp, etc...**) for each of the columns of your tables, and also specify an appropriate **PRIMARY KEY** for each table if there is one.

2. Use the PostgreSQL **\COPY** command to load your tables from the CSV data files. (You should use Google to locate PostgreSQL's online documentation and bookmark a link to those docs for your chosen PostgreSQL release.)

3. Explore the data using a **SELECT** statement in SQL to formulate and answer the following questions (queries) on it. You are to translate each of the following English queries into an appropriate SQL query (with just ***one*** query per problem!) and show both your queries and their results when reporting back to your boss.

    A. To start, you should get a general overview of the ShopALot.com site's stores, customers, and products. Write a query to print out how many stores, customers, and products there are in the given datasets - i.e., the total numbers of each.
    B. The stores in Seattle,WA recently received a big flurry of orders and they need shoppers who are able to handle at least 5 orders at a time. List all the shoppers with a capacity of at least 5 who work for one or more of the stores located in Seattle, WA.
    C. You would like to find out about stores that are offering discounts amid the pandemic. For each store that has offered discounts, print out the number of unique products that were sold at discount prices (i.e., at a price where the selling price was lower than the list price) along with their store IDs and store names. Rank the results going from the largest number of such products to the lowest. Print the first 10 rows.
    D. At the beginning of the pandemic, the number of hoarders hit an all-time high! Find all hoarding orders between 05/01/2020 (time placed) and 07/01/2020 (time placed). An order with more than 25 of any particular item (e.g. toilet paper, hand sanitizer, ...) can be considered a hoarding order. Print the order ID, total price, and time placed for these orders.
    E. To study the users' shopping habits, you want to explore trends in the average order's total price during the peak of the pandemic. Suppose the peak date was 04/01/2020 (time placed). What was the average total price of all orders over the prior seven days? (Hint: Take advantage of the timestamp type.)
    F. People like to shop for everything they need at one stop if possible, which means that a store with a larger variety of categories may be more attractive to customers. Read the "Hints on Multivalued Data" appendix at the end of this assignment. Use the PostgreSQL string_to_array() function to list the store ID, the name, the store's category list, and the length of the list for each store with a zip code of 44401. You may also find the **array_length()** function useful.

G. Continuing from the previous problem, use the **UNNEST** operator of PostgreSQL to create a *normalized* list with the store ID, the name, and the category for each store with a zip code of 44401. (Keep the list entries for each business together by using an **ORDER BY** clause.)
H. Your boss would like to know the top five store categories (i.e., the five categories that appear the most frequently as a store category) as well as the number of stores that include each such category and the minimum, maximum, and average list prices of the products in each such category.
I. Suppose your boss wants to identify orders totalling more than $650 in an effort to identify high-value customers. How many orders have a total price greater than $650? While you are working on this query, use the **EXPLAIN ANALYZE** command to explore PostgreSQL's view of the statistics for the query. How selective is this requirement? And how long does this query actually take to run?
J. We would like to do some clean-up in order to ensure the consistency of our data. In particular, we want to make sure that the total price on record for each order is accurate. (Your boss heard a rumor that one of the application's web developers may have skipped CS122B.) Some orders have inconsistencies between the recorded total price and the sum of the items' prices. Write a query to identify the offending orders and to report the number of such orders. Once you have done that, write and run an update query to fix the incorrect total prices.
K. With the correct total prices in place, create an index on the total price column and run query **I** and use the **EXPLAIN ANALYZE** command once again. Compare the query's running time with and without the index and briefly state what you think the main reason(s) for any running time difference is.
L. Your boss is developing a shopper hiring plan and would like you to provide a thorough aggregation of store locations. Use the **ROLLUP** operator of PostgreSQL to show the total number of stores for each (state, city, zip code) combination. Read the "Advanced Aggregation in SQL" handout on the course wiki for more information about how to formulate this kind of query. Sort the results in descending order of (state, city, zip code) and print the first 20 rows.
M. **[Extra Credit]** A new analytical request has been given to you from your boss: For each category, rank the products in that category in descending order of their list prices. Use SQL's window-based aggregation (i.e., the OVER clause) to find the ranks. Print out the product IDs, product names, categories, list prices, and ranks. Read the "Advanced Aggregation in SQL" handout on the course wiki for more information about window queries in SQL. Print the first 10 rows.

What To Turn In

When you have finished your assignment you should use Gradescope to turn in a *PDF file* that lists all of the PostgreSQL statements that you ran - from start to finish - to create the tables, load data into them, and run all the queries. Please follow the following steps in order to generate the file for submission.

1. Open the Google Doc Template file, Click [File] -> [Make a Copy] will create a copy of this template. You can edit it and once the editing is done, you can download it as a PDF file and submit it to Gradescope.
2. Write your table creation statements under the "Table Creation" section and **don't** forget to drop the tables before creating them. (Hint: Check out the IF EXISTS and IF NOT EXISTS options in the PostgreSQL DDL syntax.)
3. Write your COPY commands under the "Data Loading" section.
4. Write your queries for question #3 in their appropriate spaces and make sure that you include your notes as comments. (/* multi-line comment */).
    - Part J was placed at the end of the query set intentionally. Make sure that you drop your previously modified tables, create new tables, and load fresh data into them again before you run your query set the last time before turning it in.
5. Finally, after filling in the sample file with your SQL statements, run the file all at once for the last time using the command "\i filename;" and verify that your answers are all included in the output that's produced.
6. Submit the SQL file that was generated on Gradescope!

APPENDIX: Hints on Multivalued Data

Running and pondering the following set of PostgreSQL statements together with their results may be helpful to you as you tackle a couple of the queries in this first assignment:

```
CREATE TABLE Person (
    id int,
    name varchar(20),
    hobbies varchar(80)
);

INSERT INTO Person VALUES
   (1, 'Joe Cool', 'karate,skiing,skydiving'),
   (2, 'Susan Smith', 'scuba,piano'),
   (3, 'Hans Solo', 'flying');

SELECT * FROM Person p;

SELECT p.id, p.name, string_to_array(p.hobbies,',')
FROM Person p;

SELECT p.id, p.name, hobby
FROM Person p, UNNEST(string_to_array(p.hobbies,',')) AS hobby
ORDER BY p.id;
```