Homework Assignment #1

(SQL Refresher)

Congratulations are in order! You have just landed a development position at **SWOOSH.com**, where you will be helping to create an encompassing new online course management system called ***SWOOSH*** (Superior Web-Optimized Online Student Habitat). SWOOSH.com was started by a group of UC students who've lived through the past two years' worth of online learning and decided to create a new web application for use across various learning settings — including universities, teach-for-dollars companies, corporate tech online training programs, and more. Students and instructors alike have struggled to use online services, not just to hold and record remote lectures, but also to manage follow-up discussions and clarify points of confusion. SWOOSH aims to help out by providing a (database-backed, of course) service that neatly integrates teleconferencing with a forum-type, discussion-centric platform. To centralize the management of courses, instructors can schedule, hold, and record their courses' meetings on SWOOSH, after which the recordings are uploaded and published automatically. Students can choose to attend the course meetings or to watch the recordings of the meetings, after which they can create posts about the meetings in order to start follow-up discussions or to ask questions, and they can also reply to already created posts.

An E-R diagram of the SWOOSH.com database is available: (https://drive.google.com/file/d/1oQK66p72ygeTkVeYC-wK3y7QWZ1ETfKK/view?usp=sharing) along with an informal English description of their business: https://drive.google.com/file/d/1_WxIOZJivazXMGpTM0H9PX4rhah3PcEZ/view?usp=sharing. Your first assignment is to get your hands on 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 PostgreSQL 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 might 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 pgAdmin4 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 SWOOSH.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 current data records for SWOOSH.com.

(a) Data Files' Schema:

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

Instructors ( user_id,title)

InstuctorEducation(instructor_id, education_id, degree, major , school,
graduation_year)

Students( user_id, occupation)

EnrolledIn ( user_id, course_id, enroll_date)

Courses(course_id, course_name, description)

Teaches(user_id, course_id)

Attended(user_id, meeting_id)

Meetings(meeting_id, meeting_name, passcode, start_at, duration,
mute_participants, course_id, instructor_id, recurr_id)

Recurrences(recurr_id, repeat_on, end_date)

Recordings(recording_id, start_time, end_time, meeting_id)

Posts( post_id, post_type, body, created_at, user_id, meeting_id,
replied_to_post_id, topics)

ThumbsUp(user_id, post_id)

Watched( recording_id, user_id)

WatchedSegment( recording_id, user_id, segment_id, watched_from,
watched_to)
```

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

https://drive.google.com/drive/folders/1Tou22u2iW_0JQIFqeEfbtNbWjGPTi4Bl?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 15 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 courses and users on SWOOSH. Write a query to print out how many students, instructors, and courses there are in the given tables - i.e., the total numbers of each.
    B. You want to know which are the experienced instructors on the platform. Find instructors who have taken more than 5 courses to date on the SWOOSH platform and print their instructor_id and name.
    C. You would like to identify the popular courses on the platform and would like to do that based on the number of students enrolled. For each course that has been offered in the past, print out the number of students that were enrolled along with the instructor's name and course id for each course offering. Rank the results going from the largest number of students to the lowest. Print only the first 10 rows.
    D. There was a new online course-related product launched by a competitor in the month of April 2020, and the number of students enrolled in SWOOSH courses seemed to drop. To explore this trend, consider the periods between 03/01/2020 and 04/01/2020 (time range 1) and between 04/01/2020 and 05/01/2020 (time range 2). Print the number of students who enrolled in one or more courses in time range 1, the number of students who enrolled in one or more courses in time

range 2, and the difference between these counts. *Note*: Be careful not to double-count students who have enrolled in multiple classes in a given time range.

E. You want to study the recording duration trends for courses taught by instructor Sabrina (first name) Lawrence (last name). What was the average recording duration for meetings during the time period from 1/1/2020 (mm/dd/yy) to 2/1/2020 (mm/dd/yy) for the courses taught by Sabrina?

F. Students on the platform want to search for information in posts based on particular topics that they're interested in. As a first step towards addressing such searches against the given data, read the "Hints on Multivalued Data" appendix at the end of this assignment. Use the PostgreSQL string_to_array() function to list the post_id, the post_type, the posts topics list, and the length of the topics list for the posts created between '2020-06-21 23:00' and '2020-06-21 24:00'. (You may also find the **array_length()** function useful.) An example output would be:

	post_id [PK] text	post_type text	topics text[]	num_topics integer
1	6	note	{project2,module1,logistics}	3

G. Continuing from the previous problem ,as the next step use the **UNNEST** operator of PostgreSQL to create a *normalized* list with the post_id, the post_type, and the posts topics for the posts that were between '2020-06-21 23:00' and '2020-06-21 24:00' .(Keep the list entries for each post together by using an **ORDER BY** clause.)

H. The students on the platform would like to explore the top post topics, ones that were given lots of "thumbs up" ratings (i.e., the post topics seen to be most liked by users). Print the top 5 such post topics along with their associated thumbs up counts.

I. Suppose you want to identify recordings that have been "well-watched" by students. Find the number of recordings that have had at least one student watch them for a duration of over 30 minutes. While working on this query, to brush up on query performance debugging, use the **Explain** feature in PgAdmin (or EXPLAIN command) to analyze the query plan and statistics. And how long does this query actually take to run against the current database?

J. In the data you have been given, the E-R model's derived duration attribute was not actually stored. Alter the database design to store this information by writing an ALTER TABLE command to add this attribute to the previous table used (watched_segment) and then an UPDATE command to add the appropriate values for each row into the newly added field.

K. With the new duration field added, rewrite the query from I using the new schema from J, record the time taken by the modified query to execute, and then look at the query plan for the modified query using the **Explain** feature in PgAdmin. Now create an index on the duration column, run your modified query again, record the new time, and look again at the query plan. Compare your query's running

times with and without the index and briefly explain what you think the main reason(s) for any observed difference is.

L. We are interested in seeing the educational backgrounds of the instructors who are using our platform. Use the CUBE operation in SQL to show the instructors' backgrounds broken down by university and also by degree. In addition to printing the number of instructors for each school/degree pair, your query should also print the number of instructors with degrees from each school (aggregating all degree types) and the number of instructors with each type of degree (aggregating the school names) - and to top it all off (literally), the overall number of instructors' degrees aggregating of school or degree type. After using the CUBE operation to answer this question, rerun your query with ROLLUP in place of CUBE and briefly explain the difference (in 1-2 sentences) between CUBE and ROLLUP based on the results that you see. (Read the “Advanced Aggregation in SQL” handout on the course wiki to get more information about CUBE and ROLLUP queries in SQL.). Print only the first 10 rows in both cases sorted by the instructor counts in descending order.

M. **[Extra Credit]** We also want to see the courses with high student enrollments. To do so, for each course, rank the courses in descending order of their enrolled student count. Since there is a potential for ties, and we don’t want them broken arbitrarily, use SQL’s window-based aggregation (i.e., the OVER clause) rather than the simpler LIMIT clause to find the ranks. Print out the course ids, number of students enrolled, and ranks for such heavily-enrolled courses. (Read the “Advanced Aggregation in SQL" handout on the course wiki to get more information about window queries in SQL). Print the 10 most heavily-enrolled courses, being sure to think about how you want to deal with possible ties and then choosing the ranking function variant that you use based on how you feel ties should be handled in cases like this.

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 - in order to create the tables, load data into them, and run all of the queries - along with the results that they produce when run. Please follow the following steps in order to generate the file for submission:

1. Open the Google Doc Template file and Click [File] -> [Make a Copy] to create a copy of this template. You can edit your copy, and once the editing is done, you can download it as a PDF file and submit it to Gradescope.
2. Place your table creation statements in the “Table Creation” section and **do not** forget to drop any existing tables before creating them for the final turn-in step. (Hint: Check out the IF EXISTS and IF NOT EXISTS options in the PostgreSQL DDL syntax.)
3. Place your COPY commands in the “Data Loading” section.
4. Place your queries for question #3 in their appropriate spaces and make sure that you include your notes as comments. (/* multi-line comment */).

- Problem J was placed at the end of the query set intentionally. Be sure that you alter the queries and add the latest schema according to the changes mentioned!

5. Finally, after filling in your copy of the template 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 have all been included in the output that is produced by doing so.
6. Once you have a hw1.sql file ready with all of your commands, open your command-line tool or terminal window and navigate to the folder where your .sql file is located.
7. Type psql RoleName -h 127.0.0.1 -d DBName -a -f hw1.sql >> results.txt . (The placeholders RoleName and DBName should be replaced based on your own DB settings.) Note that hw1.sql is your input file, and results.txt is the output file that will result from running your work - i.e., it is where the PostgreSQL SQL statements interleaved with the results that they produce will go.
8. Open the results.txt file and verify that you see the proper results after each query.
9. Save a copy of this file as a PDF (results.pdf), e.g., by "PDF printing" it.
10. Submit the resulting PDF file to 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;
```