HW2: SQL

Total points: 5*1=5, plus 2 add'l points, but with a CAP of 5 points [if you score 6, or 7, your score would 'only' be a 5].

In this assignment, you will code solutions to the **five** SQL problems described below. Guess what - the questions relate back to HW1 - here, you're asked to create tables and data to get the STEM tutoring business going, and perform queries on the data (like the business owner would).

ALL the SQL knowledge/commands you need to answer the questions have been covered in class! You **do NOT** need to learn more commands or techniques (eg. use of 'triggers') etc. on your own in order to do this HW set.

To run SQL code, you can use one of the three ways mentioned in the lecture notes [a locally installed DB, or a remote server-based DB via an online shell running in a browser page, or a cloud DB] to do the problems.

One more cloud-y way, not mentioned in class, to do your homework (and continue learning and practicing SQL) is to use https://livesql.oracle.com/ (https://livesql.oracle.com/); after you sign up for a free account, you can create tables, insert rows and do queries - you can save all your work in separate sessions, reload [any/all of] them after logging in again in order to recreate your tables+data and rerun queries, and also add more tables/data/code to the mix - it's very convenient and cool, try it! I **strongly** encourage you to do your homework using livesql - it is convenient, powerful, safe (in terms of storing your work, long term).

What you need to submit are text files with the SQL commands that you come up with, one file for each question (Q1.sql, Q2.sql.. Q5.sql). PLEASE MENTION AT THE TOP OF EACH FILE, which database (eg. livesql, Oracle XE, SQLite..) you used for that question! Your grader(s) will execute the SQL commands from the text files you submit, using the same software you used, to see if they produce the expected results.

You can talk to your friends/classmates to informally discuss approaches to the problems, but DO NOT {'share'/look up/ask others for} actual code - that would be plagiarism, and will get you an 'F' in the course.

Please see your TAs/graders if you need one-on-one help (or see me). You can also post (and reply!) in the Piazza 'hw2' forum. Don't wait, start early. Good luck, have fun!

Q1 (1 point). To work on their projects, each student group (who sit in numbered tables, as described in HW1) can also (in addition to working at those numbered tables) reserve one of ten available rooms to work on their project. The rooms can be reserved for a block of a few hours (eg. 3 hours), with a start time and end time, eg. 3pm-6pm, 9am-2pm etc. At the end of the day, everyone goes home, so there's no possibility of rooms being booked for multiple days. The table structure you are asked to use, is this [you might need to change the syntax slightly, to make it work on your specific platform - same for other questions that follow]:

```
CREATE TABLE ProjectRoomBookings
(roomNum INTEGER NOT NULL,
startTime INTEGER NOT NULL,
endTime INTEGER NOT NULL,
groupName CHAR(10) NOT NULL,
PRIMARY KEY (roomNum, startTime));
```

There are two issues with the above. First, the start time could be incorrectly entered to be later than the end time. Second, a new entry (for a new group) could be accidentally put in to occupy a room, even before the existing group in that room is done using that room. For simplicity, you can express times in the 24h military-style format, eg. 9 for 9AM, 17 for 5PM, etc. For further simplicity, all bookings start and end 'on the hour', so, ints between 7 (7AM) and 18 (6PM) should be sufficient.

How would you **redesign the table** to fix both these issues? For your answer, you can either provide a textual explanation, and/or provide SQL statements. Hint - "do not be concerned with efficiency" - ANY working solution is acceptable :) Another hint - no need to learn new techniques/syntax.

Q2 (1 point). Given the following portion of the enrollment table, **write a query** to create a listing that includes class name and the number of students enrolled in the class, sorted in reverse order of enrollment (eg. to tell which were the most popular classes, at the end of the term).

```
SID  ClassName Grade

123  Processing A
123  Python     B
123  Scratch    B
662  Java       B
662  Python     A
662  JavaScript A
662  Scratch    B
345  Scratch    A
345  JavaScript B
345  Python     A
555  Python     B
555  JavaScript B
213  JavaScript A
```

Given something like the above, the output could be:

```
ClassName  Total

JavaScript 4
Python     4
Scratch    3
Processing 1
Java       1
```

Q3 (1 point). Below is a small table that tracks work being done on the students' projects. We have a project ID column on the left, a 'step' column in the middle (0,1,2.. denote steps of the project), and a status column on the right (where 'W' denotes 'waiting', 'C' denotes 'completed'). Such a table lets project instructors get a quick status on the various aspects (steps) of their projects ("where they're at", in colloquial, ungrammatical language). Specifically, 'W' would mean that the students are stalled for whatever reason (don't understand what

to do, a part broke, they have bugs they can't fix, etc) and need additional help from the instructors - students would have a way to input these (step number, status), and instructors can review the table periodically and run queries like the one you will be creating.

```
PID         Step  Status

P100        0     C
P100        1     W
P100        2     W
P201        0     C
P201        1     C
P333        0     W
P333        1     W
P333        2     W
P333        3     W
```

Write a query to output the project(s) where only step 0 has been completed, ie. the project gotten started but the rest of the steps are in waiting mode. In the above table, such a query would output just 'P100'. You can assume that steps get completed in order, ie. P333 will never have C,W,C,W for example [all Cs will occur before all Ws].

Q4 (1 point). The owners decide to reward instructors with a bonus, at the end of the term. The bonus (a one-time payout) is calculated, like so:

```
bonus = hourly_rate * sum_of_class_counts * 0.1
```

'sum_of_class_counts' is simply a total, of student count from each class an instructor taught - eg. if instructor Dat taught JavaScript, Python and Scratch, with enrollments of 20, 20 and 15 students respectively, the sum_of_class_counts for Dat will be 55 (if a student takes multiple courses - we count that student as many times, not count them as 1). Also, this bonus is just for teaching, not for supervising projects.

Write a query that will output the highest bonus amount paid. Do feel free to create whatever table(s) you need, and populate it/them with your own data.

Q5 (1 point). We post job descriptions, to build a list of qualified instructors that we can pick from. Here's an abbreviated table of possible candidates, with just their name and the subject(s) they can teach:

```
Instructor   Subject

Aleph        Scratch
Aleph        Java
Aleph        Processing
Bit          Python
Bit          JavaScript
Bit          Java
CRC          Python
CRC          JavaScript
Dat          Scratch
Dat          Python
Dat          JavaScript
Emscr        Scratch
Emscr        Processing
Emscr        JavaScript
Emscr        Python
```

Write a query that will pick out just the instructors who can teach every subject in the table below (this is so we can hire a small number of instructors who would be easy to manage, compared to a larger group) - we're deciding to offer just the classes listed below:

```
JavaScript
Scratch
Python
```

With the above data, the query would output

```
Instructor

Dat
Emscr
```

You can hardcode the subjects just for submission purposes, but your query should work for ANY such table! Using comments in the code, **YOU NEED TO EXPLAIN IN YOUR OWN WORDS, WHAT THE QUERY DOES** (how it works); don't just say things like "Now I'm using a WHERE condition", instead explain what it's for (why you're using it).

Bonus(1(+1) point(s)). You will get 1 extra point if you can reformulate the query in a **very different way**! If you do this, submit a separate text file with the code, eg. Q5_v2.sql. If you come up with **YET ANOTHER very different way**, you can get 1 more bonus point (submit yet another file, eg. Q5_v3.sql). 'Very different' means just that - the approaches do have to be totally distinct, eg. you can't use NOT to invert an existing solution, or use IN() instead of OR, etc. Sooo... is this actually possible [to do it in two, or three, different ways]? **Yes!** Or, as they say in Minne-so-ttta - "yooo betcha!"

To reiterate, everything you need is in the slides (the material we went through in class) - you do not need to read ahead or look up more commands online! Look at each relational operator, each command, each function, each keyword that we covered, and ask yourself how it could be of use in constructing your query.

You'll **lose points** if you:

- don't name your files properly
- submit a .zip (you need to submit individual files instead)
- neglect to mention what DB software you used
- don't explain properly, in Q5, what the query does
- submit/resubmit after the deadline (be sure to do it ahead of time, and make sure you did)

Good luck, **have fun** working on the problems!!
