HW2: SQL

Total points: 1+1+1+1+2=6 [and 1+1 bonus points!]

In this assignment, you will code solutions to the five SQL problems given below.

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 remote DB via an online shell, a locally installed DB, or a cloud DB] to do the problems. One more cloud-y way 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!

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. Oracle, 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' actual code - that would be plagiarism.

Please see your TAs/graders if you need one-on-one help (or see me). You can also post (and reply!) in the D2L forum. Don't wait, start early. Good luck, have fun!

Q1 (1 point). Here is a table for recording guests' stays at a hotel (arrDate denotes arrival date, depDate is departure date):

```
CREATE TABLE HotelStays
(roomNum INTEGER NOT NULL,
arrDate DATE NOT NULL,
depDate DATE NOT NULL,
guestName CHAR(30) NOT NULL,
PRIMARY KEY (roomNum, arrDate));
```

There are two problems (issues) with the above. First, the arrival date could be incorrectly entered to be later than the departure date. Second, a new entry (for a new guest) could be accidentally put in for a room number, even before the existing guest in that room has checked out:

```
INSERT INTO HotelStays(roomNum, arrDate, depDate, guestName)
VALUES
(123, to_date('20160202', 'YYYYMMDD'), to_date('20160206','YYYYMM
(123, to_date('20160204', 'YYYYMMDD'), to_date('20160208','YYYYMM
(201, to_date('20160210', 'YYYYMMDD'), to_date('20160206','YYYYMM
;
```

Note that the to_date() function in the above INSERT statement is specific to PostgreSQL (I used v.9.3). So if you want to use the above line as-is, you need to run it under Postgres; alternately, you can use any other RDB (Oracle, MySQL..) - just be sure to use the appropriate date representation. In other words, the above statement is there just as an example, you can modify it however you want.

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 :)

Q2 (1 point). Given the following enrollment table, **write a query** to create a listing that includes course name and the number of students enrolled in the course.

```
SID  ClassName Grade

123  ART123    A
123  BUS456    B
666  REL100    D
666  ECO966    A
666  BUS456    B
345  BUS456    A
345  ECO966    F
```

Given the above, the output needs to be:

```
ClassName  Total

ART123     1
REL100     1
ECO966     2
BUS456     3
```

Q3 (1 point). Below is a small table that tracks work orders for 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 managers get a quick status on the various aspects (steps) of their projects ("where they're at", in colloquial, ungrammatical language).

```
ProjectID  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). Below is a small sample of a junkmail database we own, ie. people we want to 'spam' via postal mail [lol].

```
Name      Address  ID  SameFam

'Alice'   'A'      10  NULL
'Bob'     'B'      15  NULL
'Carmen'  'C'      22  NULL
'Diego'   'A'      9   10
'Ella'    'B'      3   15
'Farkhad' 'D'      11  NULL
```

Each entry consists of a name, address, ID, and whether there is a prior family member already in the db; in the last column, NULL means that entry is the first family member in our db, and a non-null value is the ID of the first family member (eg. Diego points to Alice, and Ella points to Bob).

Write a query to delete from the table, names that have 'NULL' for SameFam **and** another family member in the db. So in our example above, the query would delete Alice and Bob, but not Carmen and Farkhad. In other words, we want to

save postage by only sending one mail per household if we have two people from a house in our db (we assume that our db contains either one person per household, or two).

Q5 (2 points). Below is a table of chefs and the YUMMY desserts they know to make [for fun, Google (image search) ones you aren't familiar with!]:

```
Chef         Dish

'A'          'Mint chocolate brownie'
'B'          'Upside down pineapple cake'
'B'          'Creme brulee'
'B'          'Mint chocolate brownie'
'C'          'Upside down pineapple cake'
'C'          'Creme brulee'
'D'          'Apple pie'
'D'          'Upside down pineapple cake'
'D'          'Creme brulee'
'E'          'Apple pie'
'E'          'Upside down pineapple cake'
'E'          'Creme brulee'
'E'          'Bananas Foster'
```

Write a query that will pick out all those chefs who can make every item in the table below (maybe someone wants us to cater a weekend party and requests these dishes, and we'd like to pay just one chef from our query result, overtime wages to make the dishes):

```
'Apple pie'
'Upside down pineapple cake'
'Creme brulee'
```

With the above data, the query would output

```
Chef

'D'
'E'
```

You can hardcode the dishes 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!

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** figuring it all out!!
