Homework 6: More SQL (Hands-on) [100 points]

Due Date: Monday, May 20 (5:00 PM)

Submission
All HW assignments should contain both your student ID and your name and must be submitted online (e.g., 12345678_John_Doe.pdf) via the HW6 dropbox on Gradescope. Also, you will need to submit a zipped file (12345678_John_Doe.zip) of the SQL queries/statements and outputs for certain questions via EEE. Read the instruction about exporting the query outputs to CSV files. The required queries/statements and outputs and their names will be specified following the question descriptions. See the table below for the HW6 submission window. Note that after 7 PM on Tuesday May the 21st no further HW6 submissions will be accepted. (We will be releasing the solution at that time.) Please strive to get all your work in on time! If possible, try to save your one dropped assignment for the end of the term when you are most likely to want/need it.

<table>
<thead>
<tr>
<th>Date / Time</th>
<th>Grade Implications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Monday, May 20 (5:00 PM)</td>
<td>Full credit will be available</td>
</tr>
<tr>
<td>Tuesday, May 21 (7:00 PM)</td>
<td>20 points will be deducted</td>
</tr>
</tbody>
</table>

More Structured Query Language (SQL) [30 pts]

We will continue to use the dataset from HW5, and by now you should have already that dataset loaded into your local MySQL instance. If you have modified the provided data, remember to reload it to make sure your query outputs are consistent with the expected data. You are going to use MySQL for all of the queries in the assignment and turn in the queries and results according to the provided template.

Write the following queries in SQL against the PHLog.com test database. Show the result of each query where requested to do so. Please note that you will not get points for providing the result of a query on this assignment if your SQL query is syntactically incorrect (i.e., if it doesn’t execute). Since you have a “live” system at your disposal, this should not be an issue – you can easily run all of your queries that way. Also make sure that the result of your queries do not contain duplicate records as you will lose points for that.

NOTE: For questions requiring you to write DDLs (e.g., views, triggers, stored procedures, alter table), you must write the required DDL statements by hand. You will not get the points for a problem if you let the GUI tool generate the DDL for you!
1. [10pts] For PHLogger users who are interested in the “alcoholism” topic and have “wasted” thoughts about that topic, list their ids, their names, and the start time of their most recent event. A thought is defined as being “wasted” if its text contains the term “wasted”, and an event is considered to be “recent” based on its start time. Note that publishing thoughts into an interest group doesn’t mean a user is interested in this group. A user has to become a member of a certain interest group to be able to receive the information he/she is interested in. Order your results by user id. Expected result row(s): 8

a) [7pts] SQL Query (**Q1_query.sql**):

```sql
SELECT m.phlid, p.name, max(e.start)
FROM Member m, Event e, PHLogger p
WHERE p.phlid = m.phlid AND e.phlid = m.phlid
AND m.phlid IN
    (SELECT m.phlid
     FROM Member m, Interest i, Thought t
     WHERE i.topic = "alcoholism" AND t.text LIKE "%wasted%" AND t.phlid = m.phlid AND m.iname = i.iname)
GROUP BY m.phlid, p.name
ORDER BY m.phlid;
```

b) [3pts] Result (**Q1_output.csv**):

<table>
<thead>
<tr>
<th>phlid</th>
<th>name</th>
<th>max(e.start)</th>
</tr>
</thead>
<tbody>
<tr>
<td>32</td>
<td>Michell Sipes</td>
<td>2019-03-05 12:25:15</td>
</tr>
<tr>
<td>37</td>
<td>Connie Kirlin</td>
<td>2019-04-27 17:39:26</td>
</tr>
<tr>
<td>44</td>
<td>Camellia Hoeger</td>
<td>2019-03-26 15:20:39</td>
</tr>
<tr>
<td>54</td>
<td>Randall Lubowitz</td>
<td>2019-04-09 03:44:34</td>
</tr>
<tr>
<td>58</td>
<td>Dallas Boehm</td>
<td>2019-04-16 08:41:19</td>
</tr>
<tr>
<td>76</td>
<td>Vivian Schumm</td>
<td>2019-04-25 05:03:12</td>
</tr>
<tr>
<td>91</td>
<td>Robt Donnelly</td>
<td>2019-04-24 20:13:01</td>
</tr>
</tbody>
</table>
2. [10pts] Find the ids of users who own at least one of every kind of observer. Note that a user can have multiple observers of the same type, and note also that new observers can be added at any time (so don’t use any constants in your query, e.g., do not assume static knowledge of the set of all possible observer kinds). Expected result row(s): 3

a) [7pts] SQL Query (Q2_query.sql):

```sql
SELECT o.phlid FROM Observer o GROUP BY o.phlid
HAVING COUNT(DISTINCT o.kind) = (SELECT count(DISTINCT o.kind) FROM Observer o);
```

b) [3pts] Result (Q2_output.csv):

<table>
<thead>
<tr>
<th>phlid</th>
</tr>
</thead>
<tbody>
<tr>
<td>61</td>
</tr>
<tr>
<td>62</td>
</tr>
<tr>
<td>63</td>
</tr>
</tbody>
</table>
3. [10pts] Find the number of users who are members of a certain interest group but have never posted any thoughts into that group. Expected result row(s): 1

a) [7pts] SQL Query (**Q3_query.sql**):

```sql
SELECT count(distinct p.phlid) as cnt
FROM PHLogger p, Member m
WHERE p.phlid = m.phlid AND NOT EXISTS (SELECT * FROM About a WHERE a.phlid = p.phlid AND a.iname = m.iname);
```

b) [3pts] Result (**Q3_output.csv**):

<table>
<thead>
<tr>
<th>cnt</th>
</tr>
</thead>
<tbody>
<tr>
<td>85</td>
</tr>
</tbody>
</table>
4. **Views** [20 pts]

A recent CNN news report reveals that there is a severe design flaw in some heart rate observers. To verify whether your valued customers are affected by this, the company’s CTO has formed a data science team to analyze collected heart rates and their associated observers. As the lead database designer, you have been asked to create a SQL view to consolidate the heart rate observables with observers. The consolidated view (Heart_rate_view) needs to have the following fields:

- **observation_id**
  - The original observation id of this heart rate record.
- **phlid**
  - The id of the PHLogger whose heart rate device produced this heart rate record.
- **name**
  - The name of the associated PHLogger.
- **heart_rate**
  - Just use the observed rate.
- **manufacturer**
  - The manufacturer of the observer that recorded this heart rate record.
- **model**
  - The model of the observer that recorded this heart rate record.
- **kind**
  - The kind of the observer that recorded this heart rate record.
- **software_version**
  - The software version of the observer that recorded this heart rate record.

a) [10 pts] Create the desired view (Heart_rate_view) by writing an appropriate CREATE VIEW statement (Q4a_stmt.sql).

```sql
DROP VIEW IF EXISTS Heart_rate_view;
CREATE VIEW Heart_rate_view AS
SELECT obe.observation_id obid, obr.phlid, p.name, obe.rate as heart_rate,
     obr.manufacturer, obr.model, obr.kind, obr.software_version
FROM Observable obe, Observer obr, PHLogger p
WHERE obe.kind="heartrate" AND obe.observer_id = obr.observer_id AND p.phlid = obr.phlid;
```

b) [10 pts] As you may know, malfunctioned observers could report wrong heart rate readings, cause wrong diagnoses, and lead to disastrous result later. You need to write a query to find out those possible buggy observers.

Given that a PHLogger’s heart rate can be reported by different observers from different manufacturers with different kinds, models, and software versions, we want to know whether the average heart rate of a PHLogger recorded by a certain series of observers is severely lower than the average heart rate of this PHLogger recorded by all observers. We consider the observers of the same kind, from the same manufacturer, having the same model and software version to be a series. We define it is “severely
lower” if the average heart rate of a PHLogger reported by a series of observers is smaller than 80% of the average heart rate of that PHLogger (e.g., avg(all of Joy’s heart rates) * 0.8) reported by all observers. For the buggy observer series and the corresponding affected PHLogger, output the PHLogger’s id; PHLogger’s name; the kind, manufacturer, model, and software version of this observer series; the average heart rate record of this PHLogger recorded by this series of observers; the average heart rate of this PHLogger recorded by all observers. Expected result row(s): 3. (Q4b_query.sql, Q4b_output.csv)

```sql
SELECT h1.phlid, h1.name, h1.kind, h1.manufacturer, h1.model, h1.software_version, AVG(h1.heart_rate) avg_reading, avg_hr.all_avg all_avg
FROM Heart_rate_view h1,
     (SELECT h2.phlid, avg(heart_rate) all_avg FROM Heart_rate_view h2 GROUP BY h2.phlid) avg_hr
WHERE avg_hr.phlid = h1.phlid
GROUP BY h1.phlid, h1.kind, h1.manufacturer, h1.model, h1.software_version
HAVING avg_reading < 0.8 * all_avg
ORDER BY h1.phlid;
```

<table>
<thead>
<tr>
<th>phlid</th>
<th>name</th>
<th>kind</th>
<th>manufacturer</th>
<th>model</th>
<th>software_version</th>
<th>avg_reading</th>
<th>all_avg</th>
</tr>
</thead>
<tbody>
<tr>
<td>35</td>
<td>Robby Crist</td>
<td>smartphone</td>
<td>Microsoft</td>
<td>Model 4</td>
<td>10.3</td>
<td>61.0000</td>
<td>76.4286</td>
</tr>
<tr>
<td>7</td>
<td>Arianna Volkman</td>
<td>camera</td>
<td>Google</td>
<td>Model 2</td>
<td>8.8</td>
<td>53.0000</td>
<td>82.1000</td>
</tr>
<tr>
<td>99</td>
<td>Alonac Greenfelder</td>
<td>smartwatch</td>
<td>Sony</td>
<td>Model 7</td>
<td>4.8</td>
<td>61.0000</td>
<td>81.3000</td>
</tr>
</tbody>
</table>
5. **Stored Procedures** [20 pts]

Your PHLogger APP has been released, and it’s getting more popular than ever! Lots of PHLoggers have started to share their thoughts and send them to different interest groups. The CTO thinks it’s a hassle for the APP to have to insert data into the Thought and About tables separately. He wants you to create a stored procedure to simplify this process for the APP team.

a) [15 pts] Given the following template, complete the create procedure statement, which adds a thought to the Thought table and inserts a corresponding record into the About table. You can assume that the passed-in input Interest Group name (iname) always exists. (Hint: A new thought number needs to be assigned to the new thought of the corresponding PHLogger, and this number should increase sequentially. It’s possible that this user hasn’t posted thoughts before. Look up info about the DECLARE keyword if you wish to use variables. You can use IFNULL(NULL, 0) to avoid returning a NULL value.)

(Q5_stmt.sql)

```sql
DROP PROCEDURE IF EXISTS AddThought;
DELIMITER //
CREATE PROCEDURE AddThought (
    phlid varchar(8),
    iname VARCHAR(20),
    text VARCHAR(300)
) 
BEGIN
    DECLARE next_tnum INT DEFAULT 0;
    SELECT IFNULL(MAX(tnum) + 1, 0) INTO next_tnum FROM Thought t WHERE t.phlid = phlid;
    INSERT INTO Thought(phlid, tnum, text, date) VALUES (phlid, next_tnum, text, current_timestamp());
    INSERT INTO About(phlid, tnum, iname) VALUES (phlid, next_tnum, iname);
END//
DELIMITER ;
```

b) [5pts] Verify that your new stored procedure works properly by calling it as follows to add a thought to the “alcoholism 0” group for PHLogger 1. Once you have done this, use a SELECT query to verify the stored procedure’s after-effects: (Q5_output.csv)

```sql
call AddThought(1, "alcoholism 0", "PHLogger APP IS THE BEST!");
SELECT * FROM About a, Thought t WHERE a.phlid = 1 AND t.phlid = a.phlid AND a.tnum = t.tnum AND t.text LIKE "PHLogger%";
```
<table>
<thead>
<tr>
<th>phid</th>
<th>tnum</th>
<th>lname</th>
<th>phid</th>
<th>tnum</th>
<th>text</th>
<th>date</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>8</td>
<td>alcoholism</td>
<td>0</td>
<td>1</td>
<td>8 PHLogger APP IS THE BE... 2019-05-10 17:30:49</td>
<td></td>
</tr>
</tbody>
</table>
6. **Alter Table** [10 pts]

With great power comes great responsibility. As many more PHLoggers have started to share their thoughts, the APP has drawn the attention of the authorities. In accordance with a new requirement from them, the thoughts from all users, even for those who have been removed, need to be retained for future reference. To achieve that, we need to adjust our database so that: a) when a user is removed, their thoughts are retained; b) a user is able to be removed even if his or her thoughts are not empty.

a) [5 pts] Write and execute an ALTER TABLE statement to modify the Thought table to follow the new requirements. (Note: The name of the existing foreign key constraint for user_id is Thought_ibfk_1).

```
ALTER TABLE Thought DROP FOREIGN KEY Thought_ibfk_1;
```

b) [5 pts] Execute the following DELETE and SELECT statements to show the effect of your change. Record the result returned by the SELECT statement.

```
DELETE FROM PHLogger WHERE (`phlid` = '4');
SELECT count(*) from Thought where phlid = 4;
```
7. **Triggers** [20 pts]

a) [5 pts] As a result of Question 6, now our database has now lost the protection of preventing thoughts from being added without being attached to a PHLogger at that point in time. This is error-prone and needs to be fixed. Without going back to the original solution, you can fix this by using the Triggers that you just learned about. Create a trigger on the Thought table to make sure that a new thought being added belongs to an existing PHLogger user. If it does not, throw an exception with a “02000” error code and a message says “ERROR: PHLogger does not exist!”.

```sql
DELIMITER //
CREATE TRIGGER addThought
BEFORE INSERT ON Thought
FOR EACH ROW
BEGIN
    IF NOT exists(SELECT * FROM PHLogger p WHERE p.phlid = NEW.phlid) THEN
        SIGNAL SQLSTATE '02000'
        SET MESSAGE_TEXT = 'ERROR: PHLogger does not exist!';
    END IF;
END;
DELIMITER ;
```

b) [10 pts] To help improve user engagement, the APP’s Product Manager wants every new PHLogger to automatically become a member of a random interest group when he/she joins, and a new thought with the text “Hello!” will be sent to this interest group on behalf of this user. (You can use either INSERT statements or the Stored Procedure that you created in Q5 to do this) Create a Trigger on the PHLogger table to meet this new requirement. (Hint: You can use “SELECT * FROM Interest ORDER BY RAND();” to get a randomly ordered version of the Interest table.)

```sql
DROP TRIGGER IF EXISTS addIntForNewPHLogger;
DELIMITER //
CREATE TRIGGER addIntForNewPHLogger
AFTER INSERT ON PHLogger
FOR EACH ROW
BEGIN
    DECLARE random_active_group VARCHAR(20);

    SET random_active_group =
    (SELECT * FROM Interest ORDER BY RAND());

    DECLARE new_thought_text VARCHAR(20) = 'Hello!';

    INSERT INTO Thought (text) VALUES ('Hello!');
END;
DELIMITER ;
```
SELECT iname INTO random_active_group FROM Interest ORDER BY RAND() LIMIT 1;
INSERT INTO Member(phlid, iname) VALUES(NEW.phlid, random_active_group);
call AddThought(NEW.phlid, random_active_group, "Hello!");
END;;/
DELIMITER ;

c) [5 pts] Execute the following INSERT and SELECT statements to show the effect of your trigger. Record the result returned by each SELECT statement. (Q7c_output.csv)

INSERT INTO User (phlid, email, pswd) VALUES (10005, "george@uci.edu", "simple password");

INSERT INTO PHLogger (phlid, name, address_street, address_city, address_state, address_pcode) VALUES (10005, "George", "this street", "that city", "CA", "92617");

SELECT * FROM Member WHERE phlid = 10005;

<table>
<thead>
<tr>
<th>phlid</th>
<th>iname</th>
</tr>
</thead>
<tbody>
<tr>
<td>10005</td>
<td>exercise 1</td>
</tr>
<tr>
<td>NULL</td>
<td>NULL</td>
</tr>
</tbody>
</table>