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**):

```
SELECT p.phlid, p.name, MAX(e.start) AS most_recent_event
FROM PHLogger p, Member m, Interest i, About a, Thought t, Event e
WHERE p.phlid = m.phlid
AND m.iname = i.iname
AND p.phlid = a.phlid
AND a.phlid = t.phlid
AND a.tnum = t.tnum
AND p.phlid = e.phlid
AND i.topic = 'alcoholism'
AND t.text LIKE '%wasted%'
GROUP BY p.phlid, p.name
ORDER BY p.phlid;
```

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

Q1_output

phlid	name	most_recent_event
32	Michell Sipes	2019-03-05 12:25:15
37	Connie Kirlin	2019-04-27 17:39:26
38	Clark Nader	2019-04-15 12:49:36
44	Camellia Hoeger	2019-03-26 15:20:39
54	Randall Lubowitz	2019-04-09 03:44:34
58	Dallas Boehm	2019-04-16 08:41:19
76	Vivian Schumm	2019-04-25 05:03:12
91	Robt Donnelly	2019-04-24 20:13:01

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**):

```
SELECT o.phlid
FROM Observer o, PHLogger p
WHERE o.phlid = p.phlid
GROUP BY o.phlid
HAVING COUNT(DISTINCT o.kind) = (SELECT COUNT(DISTINCT kind) FROM Observer);
```

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

Q2_o

phlid
61
62
63

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**):

```
SELECT COUNT(DISTINCT m.phlid) AS num_users
FROM Member m, PHLogger p
WHERE NOT EXISTS (
   SELECT a.phlid
   FROM About a
   WHERE m.phlid = a.phlid
   AND m.iname = a.iname
);
```

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

Q3_output

num_users
85

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**).

```
DROP VIEW IF EXISTS Heart_rate_view;

CREATE VIEW Heart_rate_view AS
SELECT
   o.observation_id,
   ph.phlid,
   ph.name,
   ob.rate AS heart_rate,
   obs.manufacturer,
   obs.model,
   obs.kind,
   obs.software_version
FROM Observable ob, Observation o, Observer obs, PHLogger ph
```

```
WHERE ob.observation_id = o.observation_id
AND ob.observer_id = obs.observer_id
AND obs.phlid = ph.phlid
AND ob.kind = 'heartrate';
```

b) [10 pts] As you may know, malfunctioned observers could report wrong heart rate readings, cause wrong diagnoses, and eventually lead to disastrous consequences. 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 resultrow(s): 3. (**Q4b_query.sql, Q4b_output.csv**)

```
WITH PHLogger_Avg AS (

    SELECT
        phlid,
        AVG(heart_rate) AS overall_avg_hr,
        name
    FROM Heart_rate_view
    GROUP BY phlid
),
Series_Avg AS (

    SELECT
        phlid,
        kind,
        manufacturer,
```

```
        model,
        software_version,
        AVG(heart_rate) AS series_avg_hr
    FROM Heart_rate_view
    GROUP BY phlid, kind, manufacturer, model, software_version
)

SELECT
    pa.phlid,
    pa.name,
    sa.kind,
    sa.manufacturer,
    sa.model,
    sa.software_version,
    sa.series_avg_hr,
    pa.overall_avg_hr
FROM Series_Avg sa, PHLogger_Avg pa
WHERE sa.phlid = pa.phlid
AND sa.series_avg_hr < (pa.overall_avg_hr * 0.8);
```

Q4b_output

phlid	name	kind	manufacturer	model	software_version	series_avg_hr	overall_avg_hr
98	Alonso Greenfelder	smartwatch	Sony	Model 7	4.8	61.0000	81.8000
7	Arianne Volkman	camera	Google	Model 2	8.6	63.0000	82.1000
35	Robby Crist	smartphone	Microsoft	Model 4	10.3	61.0000	76.4286

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 andAbout 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**)

```
DROP PROCEDURE IF EXISTS AddThought;
DELIMITER //
CREATE PROCEDURE AddThought (
   IN phlid VARCHAR(8),
   IN iname VARCHAR(20),
   IN text VARCHAR(300)
)
BEGIN
   DECLARE new_tnum INT;

   SELECT IFNULL(MAX(tnum), 0) + 1 INTO new_tnum
   FROM Thought
   WHERE phlid = phlid;

   INSERT INTO Thought (phlid, tnum, text, date)
   VALUES (phlid, new_tnum, text, NOW());

   INSERT INTO About (phlid, tnum, iname)
   VALUES (phlid, new_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 forPHLogger 1. Once you have done this, use a SELECT query to verify the stored procedure's after-effects: (**Q5_output.csv**)

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%";

Q5_output

phlid	tnum	iname	phlid	tnum	text	date
1	9	alcoholism 0	1	9	PHLogger APP IS THE BEST!	2025-02-26 16:58:24

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(*) fromThought where phlid = 4;
```

Q6_out

count(*)
5

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!". (**Q7a_stmt.sql**)

```
DROP TRIGGER IF EXISTS check_phlid_exists_before_insert;

DELIMITER //

CREATE TRIGGER check_phlid_exists_before_insert
BEFORE INSERT ON Thought
FOR EACH ROW
BEGIN

   IF NOT EXISTS (SELECT 1 FROM PHLogger WHERE 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.) (**Q7b_stmt.sql**)

```
DROP TRIGGER IF EXISTS auto_register_and_add_thought;

DELIMITER //

CREATE TRIGGER auto_register_and_add_thought
AFTER INSERT ON PHLogger
FOR EACH ROW
BEGIN
   DECLARE random_iname VARCHAR(20);

   SELECT iname INTO random_iname
   FROM Interest
   ORDER BY RAND()
   LIMIT 1;

   INSERT INTO Member (phlid, iname) VALUES (NEW.phlid, random_iname);

   CALL AddThought(NEW.phlid, random_iname, '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@scu.edu", "simple password");

INSERT INTO PHLogger (phlid, name, address_street, address_city, address_state, address_pcode) VALUES (10005, "George", "this street", "that city", "CA", "95053");

SELECT * FROM Member WHERE phlid = 10005;

Q7c_output

phlid	iname
10005	alcoholism 1