Name:

UMD College Park Department of Computer-Science
CMSC 424: Database Design Spring 2018
First Midterm Exam

There are 14 questions and 10 pages in this exam (including this page). To receive credit for a question, answer it according to the instructions given. You can receive partial credit on questions that ask you to write SQL. You have 70 minutes to answer the questions (we will not start until 5 mins into the class period).

Write your name on this cover sheet (see above) AND at the top of each page of this exam. Some questions may be harder than others. Attack them in the order that allows you to make the most progress. If you find a question ambiguous, be sure to write down any assumptions you make. Be neat. If I can't understand your answer, I can't give you credit!

Don't worry if you don't complete all the questions. You receive 25% credit for any question that you leave **COMPLETELY** blank. This credit goes away once you try to start to answer it. In addition, every student will receive a bonus amount of points based on a complicated formula based on the average number of questions (weighted by point value) left empty across the class.

Grades (leave blank --- this will be filled in by graders):

Question 1:	/1
Question 2:	/2
Question 3:	/3
Question 4:	/2
Question 5:	/3
Question 6:	/3
Question 7	/4
Question 8	/4
Question 9	/4.5
Question 10	/4
Question 11	/2
Question 12	/7
Question 13	/4
Question 14	/6.5
Total:	/50

Name:

This entire midterm will use the following schema for a simplified Piazza/Reddit-like discussion board. The database contains the following tables:

1. Users: user_id, name, email, password
2. Topics: topic_id, name, description
3. Moderators: user_id, topic_id
4. Posts: post_id, author_id, topic_id, parent_id, title, content

Created (correctly and without errors) according using the following SQL commands:

```
create table users(user_id int primary key, name varchar(10),
                   email varchar(20), password varchar(10));

create table topics(topic_id int primary key, name varchar(10),
                    description varchar(20));

create table moderators(user_id int references users,
                        topic_id int references topics);

create table posts(post_id int primary key,
                   author_id int references users,
                   topic_id int references topics, parent_id int,
                   title varchar(15), content varchar(100));
```

The references keyword creates a foreign-key/primary-key constraint. Thus, for example, user_id in the moderators table is a foreign key that references the users table.

The users table contains basic information like name, email, etc., about the users of the system.

The topics table contains a list of topics, e.g., Politics, Weather, Science, etc. Each topic has a unique id, name (e.g., "Politics"), and a short description about it.

Every topic has one or more moderators, which we store in the moderators table.

Users of the system can create new notes about any topic and other users can comment on notes. The posts table contains all the notes and the comments associated with it. For example, user "1" may create note "10" on topic "3". Since it is a note, and not a comment on an existing note, we will set the parent_id to NULL. Thus, the complete row for this note is:

```
(10, 1, 3, NULL, 'This is n10', 'content of note').
```

If another user "2", comments on note "10", this comment gets a new id (say "11"), and we add it to the posts table, with "10" as the parent_id. That is, the complete row for this comment is:

```
(11, 2, 3, 10, 'First comment', 'content of comment')
```

For simplicity, assume the topic_id for all comments of a note is the same as the note itself.

Name:

Question 1 [1 point]: Were the create table commands given above DDL statements or DML statements? DDL

Question 2 [2 points]: Write SQL to insert a new user into the users table, with user_id of **1**, whose name is **joe**, email is **joe@umd.edu**, and password is **4321**.

insert into users values (1, 'joe', 'joe@umd.edu', '4321');

Question 3 [3 points]: Assume other users have been inserted into the users table subsequently to question 2, but thus far, only insert statements have been sent to the database. Write SQL to update the value of joe's password (the tuple that you inserted as part of question 1, and not any other tuple in the users table, no matter what values they may have for any of their attributes) to **1234**.

update users set password = '1234' where user_id = 1;

Question 4 [2 points]:
Write SQL to delete all users that have an email address of **joe@umd.edu**

delete from users where email = 'joe@umd.edu';

Question 5 [3 points]:
Assume (for this question only) that only the users table exists in the database (the other tables haven't been created yet) and that users have already been inserted into it. One way to remove the password attribute from the users table is run the following two SQL statements:

```
drop table users;
create table users(user_id int primary key, name varchar(10),
                   email varchar(20));
```

which drops the users table, and creates a new one without the password attribute.

True or false: (Assume the users table is not empty.) This solution (shown above) for removing the password attribute causes more data loss than just the password values. In particular, a new table will be created, but all the tuples that had been inserted into the old table will gone.
True

True or false: There's no way to remove the password attribute from the users table (with or without data loss) using a single SQL statement.
False

True or false: There's no way to remove the password attribute from the users table using a single SQL statement without losing more data than just the password values.
False

Name:

Question 6 [3 points]: Write a simple query that returns the user_id of all users whose email address ends in **edu**

select user_id from users where email like ‘%edu’;

Question 7 [4 points]:
We want to compute the set of users who have contributed a lot of posts (not counting comments --- just original notes) --- more than 100 --- and who are not moderators (for any topic). Modify the WHERE clause in the query below to compute the answer.

```
SELECT p.author_id
FROM Posts p
WHERE <your-solution-will-be-inserted-here>
GROUP BY p.author_id
HAVING COUNT(*) > 100
```

p.author_id not in (select user_id from moderators) and p.parent_id is NULL

Question 8 [4 points]:

Name:

Define the **mode** user_id (there could be more than one) in the posts table as the user(s) with the most notes or comments in the posts table (in other words, the user_id(s) that appear(s) the most frequently posts table). Circle all of the following queries that correctly return an ordered list of post_ids of all the posts by the **mode** user_id(s) in the posts table.

```
with user_posts as
        (select author_id, post_id, count(*) as ct
         from posts group by author_id)
select post_id from user_posts
where ct = (select max(ct) from user_posts) order by post_id;
```

```
CORRECT with user_posts as
        (select author_id, count(*) as ct
         from posts group by author_id)
select post_id from user_posts natural join posts
where ct = (select max(ct) from user_posts) order by post_id;
```

```
CORRECT with user_posts as
        (select author_id, count(*) as ct
         from posts group by author_id),
     mode_users as
        (select author_id from user_posts
         where ct = (select max(ct)
                     from user_posts))
select post_id from posts
where author_id in (select * from mode_users) order by post_id;
```

```
CORRECT with user_posts as
        (select author_id, count(*) as ct
         from posts group by author_id),
     mode_users as
        (select author_id from user_posts
         where ct = (select max(ct) from user_posts))
     select post_id
     from posts inner join mode_users
                   on (posts.author_id =  mode_users.author_id)
     order by post_id;
```

Question 9 [4.5 points]: Circle each of the following SQL queries on the next page that will correctly return, for every user in the database, the number of comments (not including original notes) that the user has authored. (Hint: at least one of the queries is correct).

Please note that the 9 queries below are in groups of three, where each group of three is the same exact query, except the type of join is changed (left outer join, right outer join, and full outer join).

Name:

```
SELECT u.user_id, COUNT(*) AS number_of_comments
FROM users AS u
     LEFT OUTER JOIN posts AS p ON u.user_id = p.author_id
WHERE p.parent_id IS NOT NULL
GROUP BY u.user_id

SELECT u.user_id, COUNT(*) AS number_of_comments
FROM users AS u
     RIGHT OUTER JOIN posts AS p ON u.user_id = p.author_id
WHERE p.parent_id IS NOT NULL
GROUP BY u.user_id

SELECT u.user_id, COUNT(*) AS number_of_comments
FROM users AS u
     FULL OUTER JOIN posts AS p ON u.user_id = p.author_id
WHERE p.parent_id IS NOT NULL
GROUP BY u.user_id
```

CORRECT

```
SELECT u.user_id, count(p.parent_id)
FROM users as u
     LEFT OUTER JOIN posts as p on u.user_id = p.author_id
GROUP BY u.user_id;

SELECT u.user_id, count(p.parent_id)
FROM users as u
     RIGHT OUTER JOIN posts as p on u.user_id = p.author_id
GROUP BY u.user_id;
```

CORRECT

```
SELECT u.user_id, count(p.parent_id)
FROM users as u
     FULL OUTER JOIN posts as p on u.user_id = p.author_id
GROUP BY u.user_id;

SELECT u.user_id, count(distinct p.parent_id)
FROM users as u
     LEFT OUTER JOIN posts as p on u.user_id = p.author_id
GROUP BY u.user_id;

SELECT u.user_id, count(distinct p.parent_id)
FROM users as u
     RIGHT OUTER JOIN posts as p on u.user_id = p.author_id
GROUP BY u.user_id;

SELECT u.user_id, count(distinct p.parent_id)
FROM users as u
     FULL OUTER JOIN posts as p on u.user_id = p.author_id
GROUP BY u.user_id;
```

Name:

Question 10 [4 points]: It is possible that some topics have exactly one moderator (every topic will have at least one moderator). It is also possible that a user has authored notes and/or comments on the same topic for which that user is the only moderator. We want to identify such notes and comments, so that we can add more moderators to that topic. Specifically, we wish to find the set of all notes/comments where the author of the note/comment is the sole moderator of the post's topic. Complete the WHERE clause in the query below. (Note: the use of the sum aggregator below is not a typo.)

```
WITH topics_with_one_mod AS (
  SELECT topic_id, sum(user_id) as uid
  FROM moderators
  GROUP BY topic_id
  HAVING COUNT(*) = 1)
SELECT p.post_id
FROM posts AS p
WHERE EXISTS <write solution here>;
```

```
(select * from topics_with_one_mod as t1
 where p.topic_id = t1.topic_id and p.author_id = t1.uid)
```

Question 11 [2 points]: True or false: No matter what the contents are of the moderators and users tables, it is **impossible** for the following query to return more than 0 rows:

```
SELECT * FROM users NATURAL FULL OUTER JOIN moderators
       EXCEPT
SELECT * FROM users NATURAL LEFT OUTER JOIN moderators
```

false

Name:

Question 12: After loading data into the database, the user ran the following query:

```
select * from moderators;
```

and got the response:

```
user_id | topic_id
---------+----------
       1 |        1
       2 |        2
       2 |        1
       2 |        3
```

How many **rows** are returned by the following queries? (If any of them will return an error, write "error".)

[1 point]

```
SELECT max(user_id) FROM moderators;   1
```

[1 point]

```
SELECT DISTINCT user_id FROM moderators;    2
```

[1 point]

```
SELECT user_id, topic_id, count(*)         4
FROM moderators
GROUP BY user_id, topic_id;
```

[2 points]

```
SELECT * FROM moderators as m1 WHERE m1.user_id =
   (SELECT m2.user_id
    FROM moderators as m2
    WHERE m2.user_id  > ALL
        (SELECT m3.user_id FROM moderators as m3));    0
```

[1 point]

$\sigma_{(topic\_id = user\_id)}$(moderators) 2

[1 point]

$\Pi_{(topic\_id)}$(moderators) 3

Name:

Question 13 [4 points]: We decide to have a rule that all users' names must begin with the letter 'j'. Otherwise they are not allowed to exist in our database. To enforce this rule, we decide to check every new tuple that is inserted to make sure it meets our standards (that the name must begin with 'j'), and also check every tuple that is updated to ensure that the name wasn't changed to a new value that doesn't meet our standards. I wrote the below trigger in Postgres called check_name() that checks the new value of a record (that was either inserted or modified) to ensure that the name attribute starts with 'j'. If it does, then the new value is returned and the insert or modification can proceed. If it does not start with 'j', then null is returned, which causes the insert or modification to fail. You can assume that the code in the function works correctly.

```
CREATE OR REPLACE FUNCTION check_name() RETURNS trigger AS $$
BEGIN
  IF NEW.name LIKE 'j%' THEN --if name starts with j
    RETURN NEW;  --return the new tuple to be inserted
  ELSE
    RETURN NULL; --reject new tuple because didn't start with j
  END IF;
END;
$$ LANGUAGE plpgsql;
```

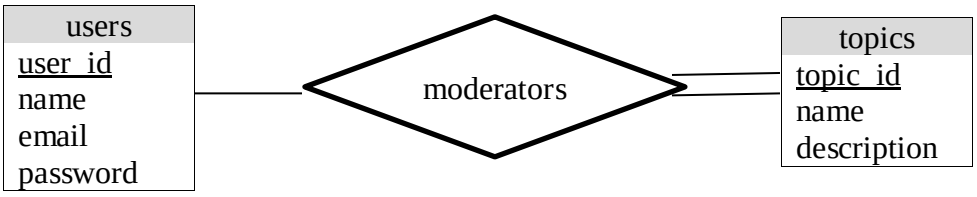
I then ran a CREATE TRIGGER command in Postgres to ensure that trigger is fired on the users table. However, I removed the part of the statement that says when the trigger should be fired. Please complete the missing part of the CREATE TRIGGER statement to tell Postgres when the trigger should be fired.

```
CREATE TRIGGER check_name <your-answer-will-go-here> ON users
  FOR EACH ROW EXECUTE PROCEDURE check_name();
```

BEFORE UPDATE OR INSERT

Name:

Question 14: If we ignore the posts table, the ER diagram for the schema for our database will look something like:



Part 1 [4 points]: Assume the only constraints that exist in the application are those mentioned in the description page on page 2 of this midterm (including the create table statements which define the foreign key constraints). Remember we said that that every topic has at least one moderator. Draw the lines that connect the users and topics entity sets to the moderators relationship set in a way that shows mapping cardinality constraints using the standard notation (from the textbook and lecture notes) of using directed lines that point to the "one" side of a many-to-one, one-to-many, or a one-to-one relationship, and undirected lines for many-to-many relationships. Distinguish between partial and total participation constraints using the standard notation (from the textbook and lecture notes) of single vs. double lines.

Part 2 [1.5 point]: Add any attributes to the moderators relationship set to the figure above (if appropriate).

Part 3 [1 point]: Underline all attributes (either the ones that existed already or the ones you added in Part 2) in the ER diagram above that compose part of a primary key.