Write your answers in the box below only. Do not write on the back or outside the box.

Database Systems — CSci 4380
Midterm Exam #2
March 25, 2018

RCS ID: ______________ @rpi.edu **Name:** ____________________

RIN # : ______________

Rules. The exam is 110 minutes for a total of 100 points. Open book and notes. Do not use any electronic tools including your computer, phone or tablet. Work alone. You **cannot** talk to anyone in class, or share notes or thoughts.

Question 1. Write the following queries using SQL using the data model below. The model is described in detail in the back of the exam.

In all queries, use `DISTINCT` only if you have to. Do not use `ORDER BY` unless a specific ordering is asked. Write your queries in a readable format.

```
Users(username, name, email, password, address)
FriendsWith(username1, username2, sincewhen)
Landmarks(id, landmarkname, landmarktype, state, city, zip, country)
Segments(id, startx, starty, endx, endy, landmark_id)
Events(id, username, eventtype, eventdate, starttime, endtime)
DataPoints(id, event_id, seqno, starttime, endtime, segment_id)
Comments(id, username, event_id, comment_text, whenposted)
```

(a) **(12 points)** Return the id, name of all landmarks in `Troy` (city) `NY` (state) that users pass by in `cycling` or `skating` events (eventtype).

(b) **(10 points)** For each user, return their username, the number of events they participated in and the duration of their longest duration event (duration is `events.endtime-events.startime`). Note that the number of events can be zero for a user, in which case you will return null for the duration value of this user.

(c) **(10 points)** Return the `username` and `email` of all users who have commented on 4 or more different events (`event_id`) that were created on the same date (`eventdate`).

(d) **(12 points)** We are going to sell some user data to a third party for political targeting. Return the email of all users who either have a `powerwalking` event (eventtype) themselves or are friends with a person who has a `powerwalking` event (eventtype) in the database.

(e) **(10 points)** Delete all landmarks from the database with no associated segment.

(f) **(14 points)** Return pairs of segment id and username for each segment in the database such that the username belongs to a user with the fastest running time (smallest `endtime-starttime`) for that segment. Remember, multiple users may have the same running time; we will return them all in different tuples.

Question 2 (16 points). For this question only, you can use a single expression, or you can piece together multiple queries, inserts and auxiliary tables for this question. You do not have to put them inside a procedure block and you do not need to drop your auxiliary tables.

We are running a promotion to send energy drinks with drones to a select set of users in the database just before their next event.

To facilitate this, find and return the `username` of highly predictable users in the database and the `segment_id` of their most popular starting location. The starting location is the `segment_id` for the `datapoint` with `seqno=1` for that event. Returned users must have had an different event in at least 300 days within 2017. The returned `segment_id` for this user must be their starting location in 90% or more of all his/her events in the database.

Question 3 (16 points). You are given the following table definitions and instances. For each operation, show the changes to the tables by directly drawing on the tables. Provide a short sentence of why these tuples were changed or not changed right below the query.

```
CREATE TABLE abc (
    id INT PRIMARY KEY, name CHAR(2) );

CREATE TABLE def (
    id     INT PRIMARY KEY, key INT NOT NULL) ;

CREATE TABLE ghi (
    id1    INT, id2 INT, val INT
    , PRIMARY KEY (id1, id2)
    , FOREIGN KEY (id1) REFERENCES abc(id)
      ON UPDATE CASCADE
    , FOREIGN KEY (id2) REFERENCES def(id)
      ON UPDATE CASCADE ) ;

CREATE TABLE jkl (
     id INT PRIMARY KEY, id2 INT, val INT
     , FOREIGN KEY (id2) REFERENCES def(id)
       ON DELETE CASCADE ON UPDATE CASCADE);
```

```
create function e2f(x int) returns void AS $$
DECLARE
    c INT ;
BEGIN
    BEGIN; -- start a transaction block
       SELECT count(*) INTO c FROM abc WHERE id=x;
       DELETE FROM abc WHERE id = x;
       IF c>0 THEN
          INSERT INTO def(id) VALUES(x);
       END IF;
    COMMIT;
END ; $$ LANGUAGE plpgsql ;

CREATE TRIGGER fixit BEFORE DELETE ON def
FOR EACH ROW REFERENCING OLD ROW AS OLD
BEGIN
    UPDATE ghi SET id2 =
       (SELECT min(id) FROM def WHERE key IS NOT NULL)
    WHERE id2 = OLD.id ;
END ;
```

(a) `DELETE FROM abc WHERE name = 'joy';`

id	name
1	joy
2	nya
3	sky

(abc)

id	key
6	5
7	4
8	1

(def)

id1	id2	val
1	6	4
1	7	5
3	8	3

(ghi)

id	id2	val
11	6	3
12	6	2
13	8	

(jkl)

(b) `DELETE FROM def WHERE key>4;`

id	name
1	joy
2	nya
3	sky

(abc)

id	key
6	5
7	4
8	1

(def)

id1	id2	val
1	6	4
1	7	5
3	8	3

(ghi)

id	id2	val
11	6	3
12	6	2
13	8	

(jkl)

(c) `UPDATE def SET id=id*10 WHERE key=1;`

id	name
1	joy
2	nya
3	sky

(abc)

id	key
6	5
7	4
8	1

(def)

id1	id2	val
1	6	4
1	7	5
3	8	3

(ghi)

id	id2	val
11	6	3
12	6	2
13	8	

(jkl)

(d) `SELECT e2f(3);`

id	name
1	joy
2	nya
3	sky

(abc)

id	key
6	5
7	4
8	1

(def)

id1	id2	val
1	6	4
1	7	5
3	8	3

(ghi)

id	id2	val
11	6	3
12	6	2
13	8	

(jkl)

Data model to be used in Exam #2

This is a slightly modified version of the data model from Exam #1. The main change is the primary key for landmarks now is an id, allowing multiple landmarks of the same name. We have also added comments.

```
create table users ( -- all users in the system
       username       varchar(12) primary key
       , name         varchar(100) not null
       , email        varchar(100) not null
       , password     varchar(100) not null
       , address      varchar(100)
) ;


create table friendswith (
       -- friendship is mutual, but stored in one direction only,
       -- username1 is the person initiated the friendship
       username1      varchar(12)
       , username2    varchar(12)
       , sincewhen    date  -- when friendship was confirmed
       , primary key (username1, username2)
       , foreign key (username1) references users(username)
       , foreign key (username2) references users(username)
) ;


create table landmarks (
       id               int  primary key
       , landmarkname   varchar(100)  not null
       , landmarktype   varchar(100)  --e.g. building, monument, etc.
       , state          varchar(100)
       , city           varchar(100)
       , zip            varchar(20)
       , country        varchar(100)
) ;

create table segments (
       id               int primary key
       , startx         numeric(8,4)  not null
       , starty         numeric(8,4)  not null
       , endx           numeric(8,4)  not null
       , endy           numeric(8,4)  not null
       , landmark_id    int
       , foreign key (landmark_id) references landmarks(id)
) ;

create table events (
       id               int primary key
       , username       varchar(12) not null
       , eventtype      varchar(100)  not null --cycling, running, etc.
       , eventdate      date  not null
       , starttime      time  not null
       , endtime        time
       , foreign key (username) references users(username)
) ;
```

```
create table datapoints (
       id                 int primary key
       , event_id         int not null
       , seqno            int not null
       , starttime        time
       , endtime          time
       , segment_id       int
       , foreign key (event_id) references events(id)
       , foreign key (segment_id) references segments(id)
) ;
-- a data point is a segment on someone's running or cycling event,
-- the first data point has seqno=1, followed by seqno: 2,3,4 ...
-- describing the segments that one has passed during the event.
-- The end point of a segment for a data point with seqno x is the starting
-- point of the segment for the data point with seqno x+1.

create table comments (
       -- each comment is made by the user with given username
       -- for a specific event
       id                int primary key
       , username        varchar(12) not null
       , event_id        int not null
       , comment_text    text not null
       , whenposted      date
       , foreign key (username) references users(username)
       , foreign key (event_id) references events(id)
) ;
```