Database Systems, CSCI 4380-01
Homework # 3
Due Friday March 4, 2016 at 11:59:59 PM

Introduction.

This homework is on SQL. It attempts to teach you how to write simple SQL queries. All queries in this homework can be solved using SQL equivalent of bag relational algebra (i.e. using SELECT/FROM/WHERE/GROUP BY/HAVING/ORDER BY/LIMIT and UNION/INTERSECT/EXCEPT). There is no need to use complex nested queries (no need for: IN, NOT IN, EXISTS, NOT EXISTS, no subqueries in SELECT, FROM, WHERE OR GROUP BY statements, etc.). There is also no need for OUTER JOINS. So, concentrate on simple structure as much as possible.
As a special treat, we will also provide the correct answers to each query to test against. If we find an error, the correct answers may be updated. So, please remember to check against the latest copy on Piazza.

Database Server Use Rules

If you want to install and create the database on your own computer, you can use the data scripts I used. Please do not share the data outside of this classroom as I had to sign an agreement to get this date.
You do not have to have a database server installed to do this homework. This database is already created on the shared database server at:

`http://csci4380rpi.cloudapp.net/phppgadmin`

Feel free to use it for testing your queries, but please do considerate of others when using the server. Here are a few ground rules:

- We have no idea how the server will respond to load. This is our first experience with this cloud service. So, be patient if you see some slow down. This is a medium sized database for a 100+ class, so you can expect a serious slow down near the homework deadline. Please do not wait till the last minute to submit your homeworks.
- Test your queries one at a time. Best set up is using a browser and a text editor. Write queries elsewhere and test in the server with cut and paste.
- Make every effort to read your queries before submitting. A forgotten join condition may mean disaster. Check join conditions first, then run your queries.
- Remember if you have an unresponsive query, it will continue to use system resources even if you quit your browser. So, opening a new browser window will solve your problem but may slow things down for everyone.

  If you are experiencing problems with a query, read it carefully before running it again in a separate window. Remember: missing join conditions is the difference between: 2,094,266,610,000 tuples and 3335 tuples.
- If the server is not responsive, let us know. We will see if some jobs need to be killed. We will see how this works and adjust. Please be patient.
- Please do not include a query that does not run in your homework submission. We will run all your queries in a batch job and an incomplete query will cause us a great deal of problems.

Homework Description

In this homework, you will use a real database, a snapshot of data from Yelp taken in early 2012. So, the data is kind of old but real. The database contains businesses near Rensselaer. Note that the database is provided as is, so I apologize if there is any content that offends anyone.

The database is divided into `businesses`, `reviews` for these businesses and `users` who provide the reviews. Additionally, businesses have user defined categories stored in a separate table called `business_categories`.. The schema is given at the end of this document.

Note that I have also created a function to be used in one query. The language used in this function, pl/pgsql is a Postgres specific language that we will learn later in the semester. You are not required to understand the details of this function, but I am providing the code for completeness.

Given this database, write the following queries using SQL:

1. Find all users who have written a review for `DeFazio's Pizzeria` for `5 stars` on or after `January 1st, 2012`. Return the name the reviewers, the name of the business and the review date. Order results by review date.

2. Return the name of all categories that are associated with a business in `'Troy'` that is also given the category `'Active Life'`. Do not return `'Active Life'` but the other categories, ordered by category name.

3. Find the maximum funny, useful and cool votes ever received by a review for a business in `'Troy'` (note that the funniest, useful and coolest review may not be the same one!).

4. Now that you know how many funny votes the funniest post for a `'Troy'` business has gotten (17), return the date and text of reviews with this many funny votes and the business name.

   You can hard code the value 17 into your query for this question. We'll get fancy in the next homework.

5. Find users with no reviews in the database. Return their id and name.

6. Find all users with at least `20` reviews in the review table and have at least `100` reviews overall in their profile (user table). Note that we have only a subset of the Yelp reviews, only for the region close to Rensselaer. So, the user may have more reviews than what we have in the database.

7. Find all restaurants that are `open`, have an average rating of `4 stars or higher`, have at least `10 reviews` (`review_count` attribute) and have not received a rating of `2 stars or lower` since `January 1st 2009` in a review with `at least 1 vote` (any or all of funny, cool or useful). Return the id, name, stars, review count and the full address of restaurant, sorted by the name.

8. You are given a function

   `geodistance(latitude1, longitude1, latitude2, longitude2)`

   that computes the distance between two coordinates in miles using the Haversine formula (the correctness is not 100% guaranteed). The function is already part of the database. For example, the query:

```
select
    b.name
    , geodistance(42.726768, -73.678405, b.latitude, b.longitude) dist
from
    businesses b
order by
    dist asc ;
```

returns businesses ordered by their distance to `Moe's Southwest Grill` (you know for your Moe's tuesdays planning).

Use this function to find the top `10 closest` businesses in the `Restaurants'` or `'Food'` category (or both) to `'Rensselaer Polytechnic Institute'`. Do not hard code the latitude and longitude for RPI into your query, but use the values stored in the database by querying the businesses relation. Return the name and distance of businesses. The distance should be formatted with 2 digits after comma. Do not return any business with distance exactly 0.

9. Find all users who have written at least 2 reviews for a business in Albany and 2 reviews for a business in Troy. Return their name and URL.

Submission Instructions.

Submit a single ASCII text file named `hw3ans.sql` that contains all your queries. Your script should be formatted as shown below:

```
-- Print your answer and RCS id first
SELECT 'Student: Sibel Adali (adalis@rpi.edu)';

-- Print the name of each question before the question output
-- Pay close attention to the columns requested as well as the
-- requirements for ordering of results for each comparison

SELECT 'Question 1';

-- Replace this with your query for Question 1.
SELECT count(*) FROM businesses ;

--- Repeat this pattern for each question

SELECT 'Question 2';

-- Replace this with your answer for Question 2.
SELECT count(*) FROM users ;


SELECT 'Question 3';

-- Replace this with your answer for Question 3.
SELECT count(*) FROM reviews ;
```

The course website for homework submissions will become available by this tuesday. We will announce it on Piazza.

Database Schema

```
create table businesses (
    business_id   varchar(40) primary key
    , name        varchar(100)
    , full_address varchar(255)
    , isopen      boolean
    , photo_url   varchar(255)
    , city        varchar(255)
    , state       varchar(100)
    , review_count int
    , stars       numeric(4,2)
    , url         varchar(255)
    , latitude    float
    , longitude   float
) ;

create table business_categories (
    business_id   varchar(40)
    , category    varchar(40)
    , primary key (business_id, category)
) ;

create table users (
    user_id       varchar(40) primary key
    , funny_votes  int
    , useful_votes int
    , cool_votes   int
    , name         varchar(100)
    , url          varchar(255)
    , average_stars numeric(4,2)
    , review_count int
) ;

create table reviews (
    review_id     varchar(40) primary key
    , funny_votes  int
    , useful_votes int
    , cool_votes   int
    , user_id     varchar(40)
    , stars       int
    , review_date  date
    , review_text  text
    , business_id  varchar(40)
    , constraint reviews_fk foreign key (business_id)
      references businesses(business_id)
    , constraint reviews_fk2 foreign key (user_id)
      references users(user_id)
      ) ;

create function
   geodistance(latitude1 float
   ,longitude1 float
   ,latitude2 float
   , longitude2 float) returns float as $$
declare
   lat1 float;
   long1 float;
```

```
   lat2 float;
   long2 float;
   a float ;
begin
    lat1 = latitude1 * pi() / 180.0;
    long1 = longitude1 * pi() / 180.0;
    lat2 = latitude2 * pi() / 180.0;
    long2 = longitude2 * pi() / 180.0;
    a = sin((lat1-lat2)/2)^2 + cos(lat1) * cos(lat2) * sin((long1-long2)/2)^2;
    a = 2*tan( sqrt(a)/sqrt(1-a) );
    return (6371 / 1.609)*a ;
end;
$$ language plpgsql;
```