CARNEGIE MELLON UNIVERSITY
DEPARTMENT OF COMPUTER SCIENCE
15-415/615 - DATABASE APPLICATIONS
C. FALOUTSOS & A. PAVLO, FALL 2015

Homework 2 (by Jinliang Wei) - Solutions
Due: **hard copy, in class** at **3:00pm, on Monday, Sep. 28**
Due: **tarball, BlackBoard** at **3:00pm, on Monday, Sep. 28**

Reminders:

- *Plagiarism*: Homework is to be completed *individually*.
- *Typeset* all of your answers whenever possible. Illegible handwriting may get zero points, at the discretion of the graders.
- *Late homeworks*: in that case, please email it
  - to all TAs
  - with subject line: `15-415 Homework Submission (HW 2)`
  - and the count of slip-days you are using.

For your information:

- Graded out of **100** points; **2** questions total
- Rough time estimate: *1-2 hours for setting up postgres; approx. 4-8 hours for completing the questions*

Revision : 2015/10/07 01:19

Question	Points	Score
Yelp Reviews: Part 1 - Statistics, etc.	55	
Yelp Reviews - Part 2 - Fraud Detection	45	
Total:	100	

Preliminaries

Cluster machine assignments

Each student is assigned with an andrew cluster machine, and a specific port, for this homework (you might be sharing the machine with other 415/615 students). Your machine and port assignment is on Blackboard, under "grade center". You use some other machines in the range `ghc25..86` (say, if your machine is down) but you **MUST** use your assigned port on any machine that you try, to avoid collisions with other students.

Postgres set-up

Before starting the homework, follow the instructions for setting up PostgreSQL and importing the data we'll be working with, available at `http://www.cs.cmu.edu/~christos/courses/dbms.F15/hws/HW2/postgresql-setup.html`.

What to deliver: Check-list

Both hard copy, and soft copy:

1. **Hard copy**:
   - What: hard copy of your **SQL queries**, plus their **output**.
   - When: Sep. 28, 3:00pm
   - Where: in class

   Contrary to HW1, keep all your answers in one document, but still provide (`course#`, `Homework#`, `Andrew ID`, `name`).
2. **Soft copy: tar-file**:
   - What: A `tar` file (`<your-andrew-id>.tar`) with all your SQL code. See next paragraph on how to generate the tarball.
   - When: Sep. 28, 3:00pm
   - Where: on *Blackboard*, under 'Assignments'/'Homework #2'

Details of the tar file. Obtain `http://www.cs.cmu.edu/~christos/courses/dbms.F15/415F15HW2/hw2.tar.gz` - after `tar xvfz`, check the directory `./`**hw2**: replace the content of each place-holder `hw2/queries/*.sql` file, with your SQL code.

- Your SQL queries will be auto-graded by comparing their outputs to the correct outputs. When comparing each query's output, the grading script prints "`checking` *qnn*`; correct if nothing below ----`", where *qnn* is the number of the question. Only when your answer is wrong, the grading script prints the difference (computed by `diff`) between your query output and the correct output. The goal is that, with your answers in the directory `queries`, when the grader loads the correct outputs in the directory `outputs` and types `make`, he/she should see no difference.
- For your queries, the **order** of the output columns is important; their names, are not. (our grading script turns off the column headers.)

Naming Convention. The place-holder `queries/.sql` files have the obvious naming convention: For example, the place-holder file for Question 1(a) is named as `q1a.sql`. Each

place-holder file contains a trivial query `SELECT 'hi';`. Except for `outputs/q1a.txt`, all other outputs just contain `hi`. The only exception is `outputs/q1a.txt`, which contains the output for the ("Sample") question.

Sanity Check. Before doing any other question, check your answers to the ("Sample") question (Question 1(a)) to ensure your output matches the formatting (in addition to being correct): Enter your answer into `queries/q1a.sql` and run `make`; the `diff` script should show no difference.

Easy Packaging. For your convenience, we automated the packaging of the submission: When you are done, type `make submission`, and this should automatically generate the tar-file you need to submit. However, it still is **your responsibility** to make sure that all the sql query files are included.

FAQs

- *Q: May we use* `views`*?*
- A: Yes - you may use any feature of SQL that is supported by `postgres` on the andrew/ghc linux machines. But, make sure that no extra output is generated - remember, we'll do a `diff`, against our answers.
- *Q: What if a question is unclear?*
- A: Our apologies - please post on blackboard; or write down your assumptions, and solve *your* interpretation of the query. We will accept all reasonable interpretations.
- *Q: What if my assigned machine is not responding?*
- A: Our apologies again - as we said earlier, please use another machine, in the range `ghc25..86` but with **your assigned port number**, YYYYY.

General Grading Policies

- -5 for not following instructions, examples include:
  - Submitting query code thats not prepared by make submission and it wont work with the autograding script;
  - Hard copy does not contain the outputs;
  - Tarballs are not submitted on Blackboard (except the ones that use slip days);
  - No hard copy submitted nor email submission;
- No penalty for creating views but not dropping them. Each students code is graded against a clean database.
- -2 per query for leftover template code (SELECT hi;).
- -2 per query if query code is incorrect but hard copy is correct.
- Get 50% of the points if the result is wrong, but the sql code is close to correct. "Close to correct" means there's one minor mistake; get 0 points for more than one mistakes. Examples of minor mistake include (also see each question):
  - Forgetting to filter by one or both of city/state;
  - Missing limit;
  - Incorrect column order or missing a column (0 points for missing more than one column);
  - Tuples are not ordered as required.

Question 1: Yelp Reviews: Part 1 - Statistics, etc. . . . [55 points]

For this question you will look into Yelp reviews collected from multiple cities across the world, including Pittsburgh. Figure 1 gives the schema of the tables you will use. For example, the tuple in the `review` table

("1202", "1001", 3.0, "2011-08-10", 2, 1, 0)

means that the #1202 business was reviewed by #1001 user on August 10th, 2010 and got 3 stars. 2 other users think this review is useful, 1 user thinks this review is funny and 0 users think this review is cool.

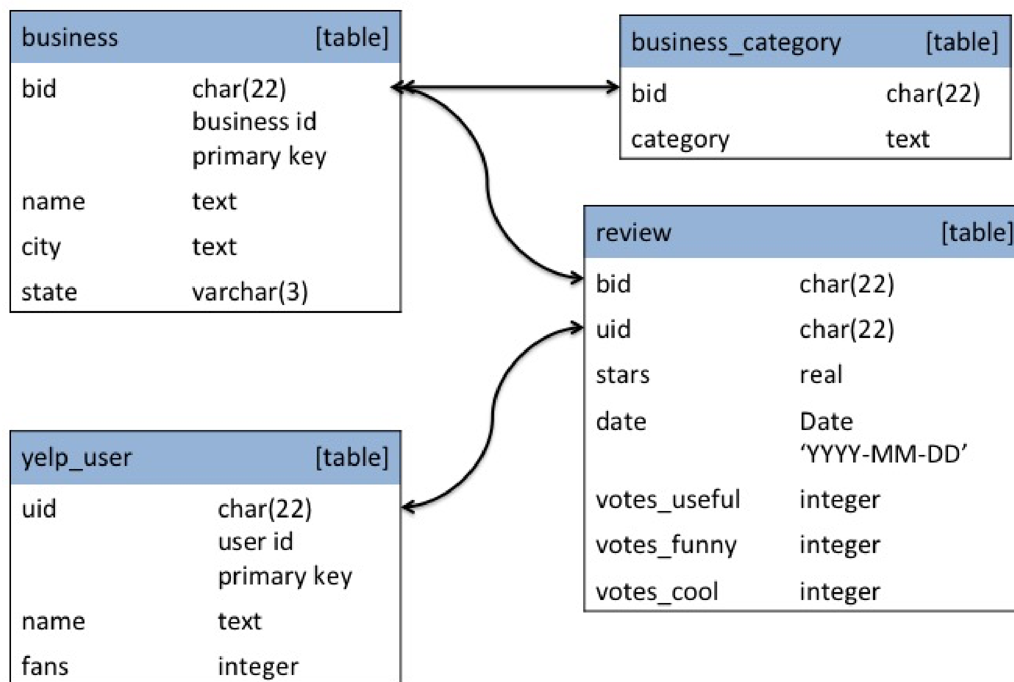

Figure 1: Tables for Yelp Reviews

(a) **[0 points]** ("Sample"): Count the distinct categories of businesses. The purpose of this query is to make sure that the formatting of your output matches exactly the formatting of our auto-grading script.

Solution:

```
SELECT count(distinct(category)) FROM business_category;
```

(b) **[5 points]** ("Warm-up"): Count the number of businesses in Pennsylvania ("PA").

Solution:

```
SELECT count(*) FROM business
WHERE state = 'PA';
```

(c) **[5 points]** ("Misclassified"): Find every business in Pennsylvania that has the word "Coffee" (case-sensitive) in its `name` but is not classified as a coffee place (i.e. has no word "Coffee" in any of its categories). List the business' `bid` (i.e. business id) and `name` in ascending `bid` order.

Solution:

```
SELECT bid, name
FROM business
WHERE name LIKE '%Coffee%'
AND state = 'PA'
AND bid NOT IN (
    SELECT bid
    FROM business_category
    WHERE category LIKE '%Coffee%'
)
ORDER BY bid ASC;
```

(d) **[10 points]** ("Most talked-about bar, per state"): For each state, give the most popular bar in that state. "Popular" means that it has the highest *count* of reviews, among all establishments that have one category as "Bars", in said state. Sort by state name (increasing). In the remote case of a tie in first place, list all bars, in ascending `bid` order. More specifically, for each bar, print its `bid` (i.e. business id), `name`, number of reviews and `state`.

Solution:

```
SELECT bid, name, number_reviews, state
FROM (
     SELECT bid, name, number_reviews, state, rank()
     OVER (PARTITION BY state
     ORDER BY number_reviews DESC)
     FROM (
          SELECT b.bid, b.name, count(*), b.state
          FROM business AS b, review AS r
          WHERE b.bid = r.bid
          AND b.bid IN (
               SELECT bid
               FROM business_category
               WHERE category = 'Bars'
          )
          GROUP BY b.bid
    ) AS review_counts(bid, name, number_reviews, state)
) AS ranked_review_counts(
  bid, name, number_reviews, state, rank)
WHERE rank = 1;
```

```
ORDER BY state ASC, bid ASC;
```

Grading info: Get 50% of the points for one minor mistake; 0 points for more than one minor mistakes. Exemplar minor mistakes include:

- *'%Bars%' instead of 'Bars'*

(e) [**10 points**] ("Best breakfast in Pittsburgh, PA"): We define *score* as average number of stars. List the top 10 businesses in Pittsburgh, Pennsylvania [1] for breakfast or brunch. "Top" means that it has the highest score among all businesses that have one category as "Breakfast & Brunch". Sort by their scores and break ties with number of reviews received (decreasing), then `bid` (increasing). For each business, print its `bid` (business id), `name`, average # of stars, and number of reviews.

Solution:

```
SELECT b.bid, b.name, avg(r.stars) AS avg_stars,
       count(*) AS number_reviews
FROM business AS b, review AS r
WHERE b.bid = r.bid
AND b.city = 'Pittsburgh'
AND b.state = 'PA'
AND b.bid IN (
    SELECT bid
    FROM business_category
    WHERE category = 'Breakfast & Brunch'
)
GROUP BY b.bid
ORDER BY avg(r.stars) DESC, count(*) DESC, b.bid ASC
LIMIT 10;
```

(f) [**5 points**] ("Frequent travelers"): List users that have been to more than 5 `distinct` states. Order by number of states traveled to (decreasing), break ties with `uid` (user id, increasing). For each user, list his/her `uid`, `name` and number of states traveled to.

Solution:

```
SELECT uid, name, count(*) AS number_states
FROM (
      SELECT u.uid, u.name, b.state, count(*)
      FROM yelp_user AS u, review AS r, business AS b
      WHERE u.uid = r.uid
      AND b.bid = r.bid
      GROUP BY b.state, u.uid
```

[1] There is Pittsburgh, California, too - sigh!

```
) AS user_state(uid, name, state, number_reviews)
GROUP BY uid, name
HAVING count(*) > 5
ORDER BY count(*) DESC, uid ASC;
```

(g) **[10 points]** ("Burgers for travelers"): Consider only reviews by travelers (users who have been to more than 2 states). We define *score* as the average number of stars received, considering only reviews by travelers (ie., *ignore* reviews by everybody else). List the top 5 burger restaurants (i.e. have the highest score) in Pittsburgh, Pennsylvania. "Top" means that it has the highest score among all businesses in Pittsburgh, Pennsylvania that have one category as "Burgers". Sort by score (highest, first) and break ties with number of reviews by travelers (decreasing), then `bid` (increasing). For each restaurant, print its `bid`, `name`, score, and number of traveler reviews received.

Solution:

```
SELECT b.bid, b.name, avg(r.stars) AS avg_stars,
       count(*) AS number_reviews
FROM (
     SELECT uid, name, count(*)
     FROM (
          SELECT u.uid, u.name, b.state, count(*)
          FROM yelp_user AS u, review AS r, business AS b
          WHERE u.uid = r.uid
          AND b.bid = r.bid
          GROUP BY b.state, u.uid
      ) AS user_state(uid, name, state, number_reviews)
      GROUP BY uid, name
      HAVING count(*) > 2
) AS travelers(uid, name, states_been_to),
  review AS r, business AS b
WHERE travelers.uid = r.uid
AND r.bid = b.bid
AND b.state = 'PA'
AND b.city = 'Pittsburgh'
AND b.bid IN (
    SELECT bid
    FROM business_category
    WHERE category = 'Burgers'
)
GROUP BY b.bid, b.name
ORDER BY avg(r.stars) DESC, count(*) DESC, bid ASC LIMIT 5;
```

Grading info: *Get 50% of the points for one minor mistake; 0 points for more than one minor mistakes. Exemplar minor mistakes include:*

- *'%Burgers%' instead of 'Burgers'*

(h) **[10 points]** ("The most helpful user"): We define as "usefulness count" of a user, say "Smith", as the sum of the 'useful' votes that his/her reviews have attracted. Find the user with the highest such count. If there is a tie in first place, list all such users, ordered by `uid` (i.e. user id). For each user, print his/her `uid` (i.e. user id), `name`, and usefulness count.

Solution:

```
WITH user_useful_votes(uid, name, cnt) AS (
      SELECT u.uid, u.name, sum(r.votes_useful)
      FROM yelp_user AS u
      INNER JOIN review AS r
      ON u.uid = r.uid
      GROUP BY u.uid
)
SELECT uid, name, cnt AS usefulness
FROM user_useful_votes
WHERE cnt = (
       SELECT MAX(cnt)
       FROM user_useful_votes
)
ORDER BY uid ASC;
```

Question 2: Yelp Reviews - Part 2 - Fraud Detection [45 points]

Here we look for a few suspicious patterns in the reviews, which may signal fraud.

(a) **[15 points]** ("Too good to be true"): List Pennsylvania businesses that have more than 10 reviews and all of them are "5 stars". (This is probably a sign of fraud). Order by the number of reviews received (decreasing), and `bid` (i.e. business id, increasing). For each business, print `bid`, `name` and number of reviews.

Solution:

```
SELECT b.bid, b.name, count(*) AS number_reviews
FROM business as b
INNER JOIN review as r
ON b.bid = r.bid
WHERE b.bid NOT IN (
        SELECT distinct(b.bid)
        FROM business AS b
        INNER JOIN review AS r
        ON b.bid = r.bid
        WHERE r.stars < 5
        AND b.state = 'PA'
)
AND b.state = 'PA'
GROUP BY b.bid
HAVING count(*) > 10
ORDER BY count(*) DESC, bid ASC;
```

Grading info: *Get 50% of the points for one minor mistake; 0 points for more than one minor mistakes. Exemplar minor mistakes include:*

- *'>= 10' instead of '>' 10*

(b) **[15 points]** ("Too many one-hit wonders"): We refer to users that have only voted once as one-hit wonders and such users are likely to be fake accounts. In this query, find the businesses that got more than 150 5-star reviews from such one-hit wonders. For each business, print its `bid` (i.e. business id), `name` and the number of reviews from one-hit wonders. Order by the number of reviews received from one-hit wonders (decreasing), then business name (increasing) then bid (increasing).

Solution:

```
SELECT b.bid, b.name, count(*) AS number_reviews
FROM business AS b
INNER JOIN review AS r
ON b.bid = r.bid
WHERE r.stars = 5
AND r.uid IN (
     SELECT u.uid
```

```
    FROM yelp_user AS u
    INNER JOIN review AS r
    ON u.uid = r.uid
    GROUP BY u.uid
    HAVING count(*) = 1
)
GROUP BY b.bid
HAVING count(*) > 150
ORDER BY count(*) DESC, b.name ASC, b.bid ASC;
```

(c) **[15 points]** ("Big jump"): Find the businesses whose average rating was raised by more than 1 stars from May 2011 to June 2011. (Such sudden jumps are also suspicious, often due to fabricated reviews). Order your results by the magnitude of the jump (largest, first), then business names (increasing) and id (increasing). Print the first 10 if there are more than 10. For each business, print its `bid` (i.e. business id), `name` and jump magnitude.

Solution:

```
WITH m1_stars(bid, avg_stars, nreviews) AS (
     SELECT bid, avg(stars), count(*)
     FROM review
     WHERE extract (month from date) = 5
     AND extract (year from date) = 2011
     GROUP BY bid
     ORDER BY avg(stars) DESC
), m2_stars(bid, avg_stars, nreviews) AS (
     SELECT bid, avg(stars), count(*)
     FROM review
     WHERE extract (month from date) = 6
     AND extract (year from date) = 2011
     GROUP BY bid
     ORDER BY avg(stars) DESC
)
SELECT b.bid, b.name,
       (m2.avg_stars - m1.avg_stars) AS jump_magnitude
FROM business AS b
INNER JOIN m1_stars AS m1
ON b.bid = m1.bid
INNER JOIN m2_stars AS m2
ON b.bid = m2.bid
WHERE m2.avg_stars - m1.avg_stars > 1
ORDER BY (m2.avg_stars - m1.avg_stars) DESC,
      b.name ASC, b.bid ASC
```

Question 2 continues...

LIMIT 10;

Grading info: *Get 50% of the points for one minor mistake; 0 points for more than one minor mistakes. Exemplar minor mistakes include:*

- *query and date format is correct but dates used in query are incorrect (e.g. May avg. includes all reviews before May instead of within month of May); no penalty if explicitly stated the assumption.*
- *reviews are not limited to year 2011*
- *abs(June.avg - May.avg) > 1*
- *May.avg - June.avg > 1*