Carnegie Mellon University
Department of Computer Science
15-415/615 - Database Applications
C. Faloutsos & A. Pavlo, Fall 2016

Homework 2 (by Lu Zhang)
Due: **hard copy, in class** at **3:00pm, on Monday, Sep. 26**
Due: **tarball, BlackBoard** at **3:00pm, on Monday, Sep. 26**

## Reminders:

- *Plagiarism*: Homework is to be completed *individually*.
- *Typeset* all of your answers. Handwritten answers will get zero points.
- *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. 4-6 hours for completing the questions*

*Revision* : 2016/09/21 01:18

| Question | Points | Score |
|---|---|---|
| Bike Sharing in Bay Area: Cities and Bikes | 75 | |
| Bike Sharing in Bay Area: Weather | 25 | |
| 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://15415.courses.cs.cmu.edu/fall2016/hws/HW2/`.

## 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. 26, 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 `gzip tar` file (`<your-andrew-id>.tar.gz`) with all your SQL code. See next paragraph on how to easily generate the tarball.
   - When: Sep. 26, 3:00pm
   - Where: on *Blackboard*, under 'Assignments'/'Homework #2'

**Details of the tar file.** Obtain `http://15415.courses.cs.cmu.edu/fall2016/hws/HW2/hw2-data.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-gzipped-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.

## Question 1: Bike Sharing in Bay Area: Cities and Bikes[75 points]

For this question you will look into bike sharing data collected from five cities in the Bay Area. Several of you will probably find jobs there, and you may like using these bikes.

Figure 1 gives the schema of the tables you will use. For example, the tuple in `trip` table

> (5088, 183, "2013-08-29 22:08:00", "Market at 4th", 76, "2013-08-29 22:12:00", "Post at Kearney", 47, 309)

means that the bike #309 had a trip #5088 from station #76 "Market at 4th" at 2013-08-29 22:08:00 to station #47 "Post at Kearney" at 2013-08-29 22:12:00.

The tuple in `station` table

> (2, "San Jose Diridon Caltrain Station", 37.3297, -121.902, 27, "San Jose", "2013-08-06", "95113")

means that the station #2 "San Jose Diridon Caltrain Station" in "San Jose" city with latitude as 37.3297, longitude as 121.902, has 27 docks and is installed in 2013-08-06. Its zip code is 95113.

The tuple in `weather` table

> ("2013-08-29", 74, 68, 61, 10, 10, 10, 23, 11, 28, 4, "", 286, "94107")

means that in 2013-08-29, the city with zip code "94107" has max temperature 74, mean temperature 68, minimal temperature 61, max visibility miles 10, mean visibility miles 10, maximum wind speed 23 mph, mean wind speed 11 mph, max gust speed 28 mph, cloud cover 4, no special weather events, and wind dir degree 286.

(a) **[0 points]** ("Sample"):
**Intention:** Count the number of cities. The purpose of this query is to make sure that the formatting of your output matches exactly the formatting of our auto-grading script.
**Details:** Print the `number` of cities (eliminating duplicates).

(b) **[5 points]** ("Warm-up"):
**Intention:** Count the number of stations in each city.
**Details:** Print `city name` and `number of stations`. Sort by number of stations (decreasing), and break ties by city name (increasing).

(c) **[5 points]** ("City-name stations"):
**Intention:** List all stations name of which contains its city's name, for example, station "Mountain View Caltrain Station" in city "Mountain View".
**Details:** Print `station name` and `city name`. Sort by city name (increasing), and break ties by station name (increasing).

(d) **[5 points]** ("Self-loops"):
**Intention:** Find the percentage of self-loop trips among all trips. A trip is self-loop

Question 1 continues...

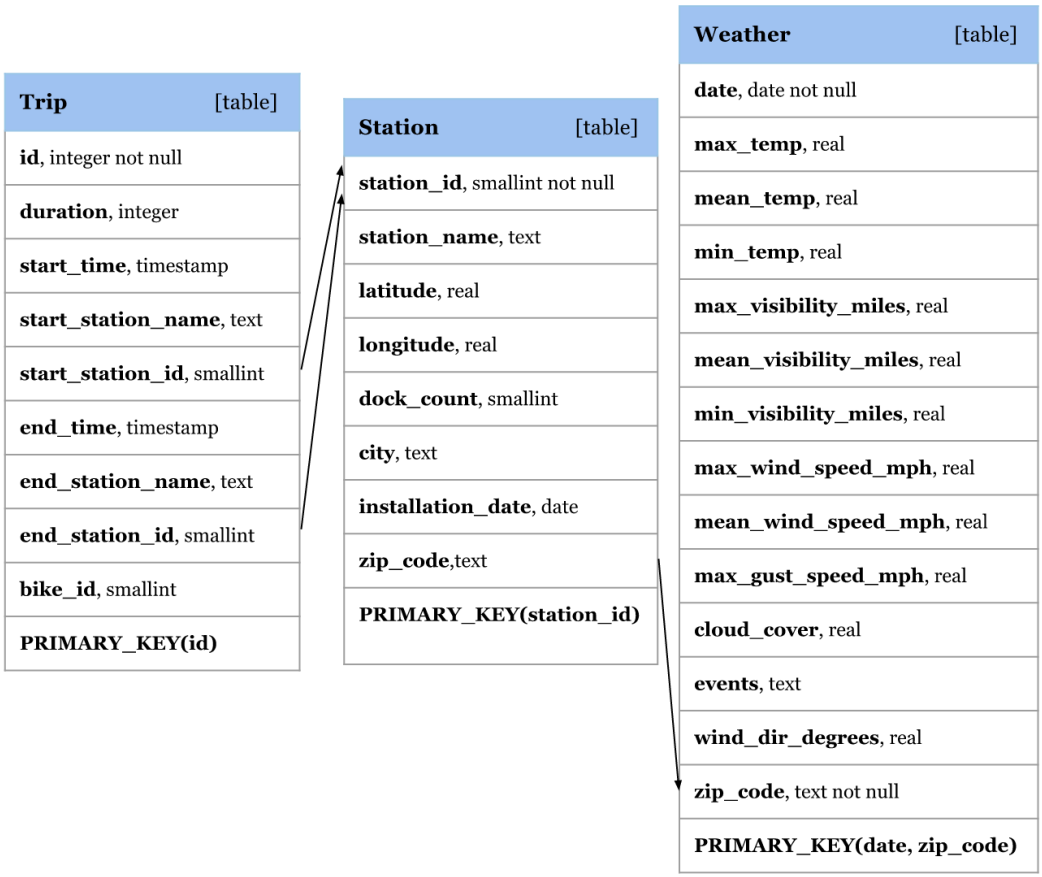


Figure 1: Tables for bike sharing

when its start station is the same as its end station.
**Details:** Print the percentage as a decimal (round to four decimal places using `ROUND()`).

(e) **[5 points]** ("Best travelled bikes"):
**Intention:** List the top 10 bikes which have been to most number of distinct stations. A bike has been to a station as long as this station is start station or end station in one of its trips. For example, if bike #1000 travelled from station A to station B in a certain trip, then bike #1000 has been to both station A and B.
**Details:** Print `bike id` and `number of distinct stations` that it has been to. Sort by number of distinct stations (decreasing), and break ties by bike id (increasing).

(f) **[15 points]** ("San Jose lover"):
**Intention:** Find all the bikes which have been to all stations in San Jose (and, zero or more, outside San Jose). As above, a bike has been to a station as long as this station is start station or end station in one of its trips. For example, if bike #1000 travelled from station A to station B in a certain trip, then bike #1000 has been to both station A and B.
**Details:** Print the `count` of such bikes.

(g) **[10 points]** ("Most popular station per city"):
**Intention:** For each city, find the most popular station in that city. "Popular" means that station has the highest count of visits. As above, *Either* starting a trip *or* finishing a trip at a station is counted as one "visit" to that station. For example, if a trip starts at station A, and also ends at station A, then station A has 2 counts of visits.
**Details:** For each city, print `city name`, most popular `station name` and its `visit count`. Sort by city name, ascending.
**Hint:** You *may* use Postgres's built-in function `ROW_NUMBER()` (faster execution, but less elegant). Either solution will get full points.

(h) **[10 points]** ("The cumulative traveling history of a bike #697"):
**Intention:** Find the accumulative traveling durations of bike #697.
**Details:** For bike #697's each trip, print `end time` and `accumulative duration so far`. Sort by cumulative duration (increasing). For example, if its traveling history is shown in Table 1, then the result should be as Table 2.

| End Time | Duration |
|---|---|
| 2016/08/01 00:05:00 | 2400 |
| 2016/09/01 00:06:00 | 3600 |
| 2016/09/02 00:07:10 | 2000 |

Table 1: Traveling example.

(i) **[10 points]** ("Unrecorded movement of bike #697"):
**Intention:** Usually, but not always, if a bike ends its trip at station "X", its next

| End Time | Cumulative Duration |
|---|---|
| 2016/08/01 00:05:00 | 2400 |
| 2016/09/01 00:06:00 | 6000 |
| 2016/09/02 00:07:10 | 8000 |

Table 2: Cumulative traveling duration.

trips starts at station "X". Occasionally, a truck collects some bikes from station "X", and moves them to station "Y", to satisfy the higher demand there. This is what we call "unrecorded movement". Find all unrecorded movements of bike #697.
**Details:** For each unrecorded movement, print `former trip id`, `former end time`, `former end station name`, `latter trip id`, `latter start time`, and `latter start station name`. Sort by former trip id (increasing), break ties with latter trip id (increasing).
**Hint:** You are *strongly recommended* to use the `ROW_NUMBER()` built-in function. It should be much faster; but again, we'll give full points for any correct solution.

(j) [**10 points**] ("Data entry errors - overlapping trips"):
**Intention:** One of the possible data-entry errors is to record a bike as being used in two different trips, at the same time. Thus, we want to spot pairs of overlapping intervals (start time, end time). To keep the output manageable, we ask you to do this check for bikes with id between 500 and 600 (both inclusive). *Note*: Assume that no trip has negative time, i.e., for all trips, `start time <= end time` .
**Details:** For each conflict (a pair of conflict trips), print the `bike id`, `former trip id`, `former start time`, `former end time`, `latter trip id`, `latter start time`, `latter end time`. Sort by bike id (increasing), break ties with former trip id (increasing) and then latter trip id (increasing).
**Hint:** 1. Report each conflict pair only once, so that `former trip id < latter trip id`. 2. We give you the (otherwise tricky) condition for conflicts:

$$start1 < end2 \;\; AND \;\; end1 > start2$$

## Question 2: Bike Sharing in Bay Area: Weather . . . . . [25 points]

Here we try to figure out how weather impacts biking in bay area.

(a) **[5 points]** ("The most popular station on special weather days"):
**Intention:** Find, on special weather days, which station was visited most as start station of trips? A "special weather day" is a day that has a non-empty string for `weather.events`, such as "Rain", "Fog" and so on.
**Details:** Print `the most popular station name` and its `times of being a start station`. If there are more than one most popular stations, print the alphabetically first station.

(b) **[5 points]** ("Bad weather - shorter trips?"):
**Intention:** Among all short trips, compute the percentage of short trips that are happening on special weather days. (As above, "special weather" day means that `weather.events` is not an empty string). A short trip is a trip whose duration is `<= 60` (units). **Note (update after release)**: Only consider weather of start stations.
**Details:** Print the `percentage` as a decimal, rounded to four decimal places using `ROUND()`.

(c) **[15 points]** ("Weather on quiet days"):
**Intention:** Are quiet (=low activity) days correlated with bad weather? Find the stations whose least popular day (as start station), had special weather events (non-empty weather events). Note: "popularity" in this question means "count of trips that *started* at that station", which is slightly different from definitions in previous questions. **Note (update after release):** 1. If there are more than one "least popular day" for a station, take the earliest day. 2. For simplicity, DO NOT take days with zero trips into account.
**Details:** Print `station name`, `date` (in Postgresql's date format, instead of timestamp), and `weather events`. Sort by station name (increasing).