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

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

Grading Info:

1. Any sql queries which cannot run in Postgres (because of syntax error for example) will get 0 score.
2. Contact corresponding TA according your *last name*, if request a regrading.
   - KANG : A - Gondi
   - LIN : Gupta - Lee
   - MENON : Li - Merigoux
   - ZHANG H. : Moesching - Vernet
   - ZHANG L. : Wang - Z

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 : 2016/10/13 10:14

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.

General Grading Policies

- -5 for not following instructions - examples include:
  - Submitting query code that is 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.
- Get 0 points 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):
  - 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: 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).

Solution:

```
select count(distinct(city))
from station
;
```

(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).

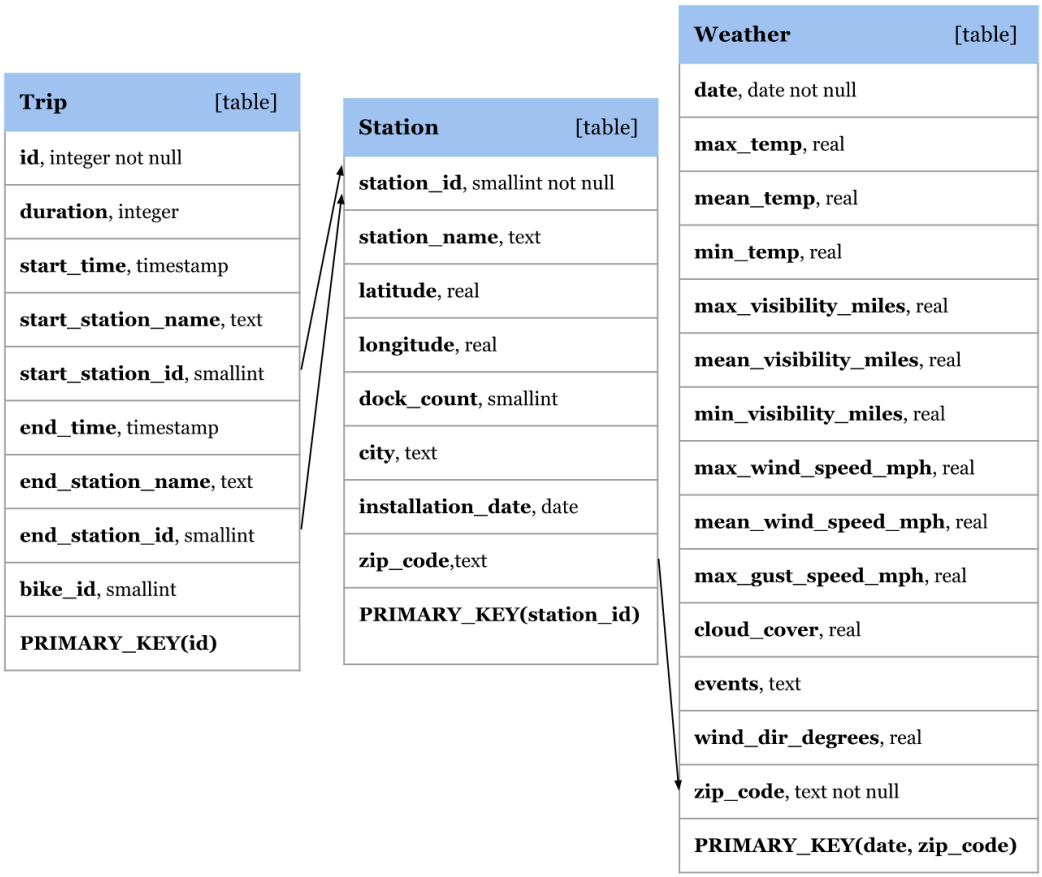

Figure 1: Tables for bike sharing

Solution:

```
select city, count(station_id) as cnt
from station
group by city
order by cnt desc, city asc
;
```

Grading info:

- ***For incorrect solutions (with diff output), check these:***
- *a). -1 for any extra columns.*
- *b). -2 for wrong order if results are right but order is wrong.*
- *c). -1 for each missing/extra/wrong tuple.*

(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).

Solution:

```
select station_name, city
from station
where station_name like '%' || city || '%'
order by city asc, station_name asc
;
```

Grading info:

- ***For incorrect solutions (with diff output), check these:***
- *a). -1 for any extra columns.*
- *b). -2 for wrong order if results are right but order is wrong.*
- *c). -1 for each missing/extra/wrong tuple.*

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

Solution:

```
select round(self_loop_cnt.cnt * 1.0 / trip_cnt.cnt, 4)
       as percentage
from (
  select count(*) as cnt
  from trip
  where start_station_id = end_station_id
```

```
) as self_loop_cnt,
(
  select count(*) as cnt
  from trip
) as trip_cnt
;
```

Grading info:

- ***For incorrect solutions (with diff output), check these:***
- *a). -2 for incorrect rouding.*
- *b). Get 0 for any other incorrectness.*

(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).

Solution:

```
with bike_and_station(bike_id, station_id) as
(select bike_id, start_station_id as station_id
from trip
union
select bike_id, end_station_id as station_id
from trip
)

select bike_id, count(station_id) as cnt
from (
  select distinct on(bike_id, station_id) bike_id,
         station_id
  from bike_and_station
) as tmp
group by bike_id
order by cnt desc, bike_id asc
limit 10
;
```

Grading info:

- ***For incorrect solutions (with diff output), check these:***
- *a). -1 for any extra columns.*
- *b). -2 for wrong order if results are right but order is wrong.*

- *c). -1 for each missing/extra/wrong tuple.*
- ***For seemingly correct solutions (no diff output), check d and e.***
- *d). -1 for only considering either start_station or end_station.*
- *e). -1 for using station_name instead of station_id to represent station (In this case, bike #27 "probabaly" has wrong count 62).*

(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.

Solution:

$$r \div s = \pi_{(R-S)}(r) - \pi_{(R-S)}[(\pi_{(R-S)}(r) \times s) - r]$$

For easy reading, let $\pi_{(R-S)}(r)$ be Pi_(R-S)(r),

```
-- solution one - Divison
with bike_in_sj(bike_id, station_id) as ( /* r */
  select distinct on(bike_id, station_id)
         bike_id, station_id
  from (
    select bike_id, start_station_id as station_id
    from trip, station
    where (trip.start_station_id = station.station_id)
          and
          (station.city = 'San Jose')
    union
    select bike_id, end_station_id as station_id
    from trip, station
    where (trip.end_station_id = station.station_id)
          and
          (station.city = 'San Jose')
    ) as bike_union
), station(station_id) as (  /* s */
  select station_id
  from station
  where city = 'San Jose'
), b(bike_id) as (          /* Pi_(R-S)(r) */
  select distinct(bike_id)
  from bike_in_sj
)
```

```
select count(*)
from (
  select bike_id            /* Pi_(R-S)(r) */
  from b
  except                    /* - */
  (
  select bike_id      /* (Pi_(R-S)(r) C_join s) - r */
  from(
    select bike_id, station_id
    from b, station         /* C_Join */
    except                  /* - */
    select *                /* r */
    from bike_in_sj
  ) as ruleout
  )
) as san_jose_lover
;

-- solution two - Count number
with bike_and_station(bike_id, station_id) as
(select bike_id, start_station_id as station_id
from trip
union
select bike_id, end_station_id as station_id
from trip
), station_in_sj (station_id) as (
select station_id
from station
where city = 'San_Jose'
)

select count(*)
from(
  select count(station_id) as cnt
  from bike_and_station
  where station_id in (select station_id
                       from station_in_sj)
  group by bike_id) as trip_in_sj
where trip_in_sj.cnt = (select count(*)
                        from station_in_sj);
```

Grading info:

- ***For incorrect solutions (with diff output), check these:***
- *a). -2 for using station_name instead of station_id to represent station (In this*

case, the wrong answer is probably 187).

- *b). -5 for outputing all valid bikes' id instead of the count.*
- *c). Get 0 for any other incorrectness.*
- ***For seemingly correct solutions (no diff output), check a and d.***
- *d). -2 for only considerting either start_station or end_station.*

(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.

Solution:

```
/*
Be careful with ''UNION"! ''UNION" will remove duplicate tuples autom
Try:
select 1 union select 1;
*/
-- solution one with ROW_NUMBER()
with visit(station_id, cnt) as (
  select start_cnt.station_id, start_cnt.cnt + end_cnt.cnt as cnt
  from
    (select start_station_id as station_id,
            count(*) as cnt
    from trip
    group by start_station_id) as start_cnt
    ,
    (select end_station_id as station_id,
            count(*) as cnt
    from trip
    group by end_station_id) as end_cnt
  where start_cnt.station_id = end_cnt.station_id
), visit_city(city, station_id, cnt, rank) as (
  select city,
         visit.station_id,
         visit.cnt,
        ROW_NUMBER() over
              (partition by city order by visit.cnt desc)
         as rank
```

```
  from visit, station
  where visit.station_id = station.station_id
)

select visit_city.city,
       station.station_name,
       visit_city.cnt
from visit_city, station
where  visit_city.rank = 1 and station.station_id = visit_city.station
order by city asc
;


-- solution two without ROW_NUMBER()
with visit(city, station_id, cnt) as (
  select start_cnt.city, start_cnt.station_id, start_cnt.cnt + end_cnt
  from (
    select city,
           start_station_id as station_id,
           count(*) as cnt
    from trip, station
    where station.station_id = trip.start_station_id
    group by city, start_station_id) as start_cnt
    , (
    select city,
           end_station_id as station_id,
           count(*) as cnt
    from trip, station
    where station.station_id = trip.end_station_id
    group by city, end_station_id) as end_cnt
  where start_cnt.station_id = end_cnt.station_id
), excluded_station(station) as (
  select distinct(v1.station_id)
  from visit as v1, visit as v2
  where v1.city = v2.city and v1.cnt < v2.cnt
)

select visit.city, station.station_name, visit.cnt
from visit, station
where station.station_id = visit.station_id and
      visit.station_id not in
         (select station from excluded_station)
order by city asc
```

;

Grading info:

- ***For incorrect solutions (with diff output), check these:***
- *a). -1 for any extra columns.*
- *b). -2 for wrong order if results are right but order is wrong.*
- *c). -1 for each missing/extra/wrong tuple.*
- ***For seemingly correct solutions (no diff output), check d.***
- *d). -1 for using station_name instead of station_id to represent station (outpus in this case may be the same as correct ones).*

(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.

Solution:

```
select end_time,
  (select sum(duration)
   from trip as i
   where i.bike_id = 697 and i.end_time <= o.end_time
   ) as ac
from trip as o
where o.bike_id = 697
order by ac asc
;
```

Grading info:

- ***For incorrect solutions (with diff output), check these:***

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.

- *a). -1 for any extra columns.*
- *b). -2 for wrong order if results are right but order is wrong.*
- *c). -1 for each missing/extra/wrong tuple.*

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

Solution:

```
with trips_with_id as (
  select row_number()
            over(order by start_time nulls last) as rownum,
         id,
         start_time,
         end_time,
         start_station_name,
         end_station_name,
         start_station_id,
         end_station_id
  from trip where bike_id = 697
  order by trip.id asc
)

select t1.id, t1.end_time, t1.end_station_name,
       t2.id, t2.start_time, t2.start_station_name
from trips_with_id as t1, trips_with_id as t2
where t1.rownum+1 = t2.rownum
      and
      t1.end_station_id != t2.start_station_id
order by t1.id asc, t2.id asc
;
```

Grading info:

- ***For incorrect solutions (with diff output), check these:***
- *a). -1 for any extra columns.*
- *b). -2 for wrong order if results are right but order is wrong.*

- *c). -1 for each missing/extra/wrong tuple.*
- ***For seemingly correct solutions (no diff output), check d.***
- *d). -1 for using station_name instead of station_id to represent station (outpus in this case may be the same as correct ones). An example is using "former_trip_end_station_name != latter_trip_start_station_name".*

(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:

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

Solution:

```
select t1.bike_id, t1.id, t1.start_time, t1.end_time,
       t2.id, t2.start_time, t2.end_time
from trip as t1, trip as t2
where t1.bike_id = t2.bike_id and
      t1.bike_id between 500 and 600 and
      t1.id < t2.id and
      t1.start_time < t2.end_time and
      t2.start_time < t1.end_time
order by t1.bike_id asc, t1.id asc, t2.id asc
;
```

Grading info:

- ***For incorrect solutions (with diff output), check these:***
- *a). -1 for any extra columns.*
- *b). -2 for wrong order if results are right but order is wrong.*
- *c). -1 for each missing/extra/wrong tuple.*
- ***For seemingly correct solutions (no diff output), check d.***
- *d).* ***If "where" statement differ from what we give in the hint*** *(hopefully they will take the hint), check the following boundary cases,* ***-2 for each case miss****:*
- *case a). t1.start_time = t2.start_time = t1.end_time = t2.end_time —> No conflict (multiple trips happening in one single minute)*

- *case b). t1.start_time = t2.start_time AND t1.end_time = t2.end_time AND t1.start_time != t1.end_time —> Conflict*
- *case c). t1.start_time = t2.start_time = t2.end_time AND t1.end_time > t1.start_time —> No conflict.*

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.

Solution:

```
select station.station_name, t.cnt
from station,
  (select trip.start_station_id, count(*) as cnt
  from trip, station, weather
  where station.zip_code = weather.zip_code and
        trip.start_station_id = station.station_id and
        date(trip.start_time) = weather.date and
        weather.events not like ''
  group by trip.start_station_id
  order by cnt desc, trip.start_station_id asc
  limit 1 ) as t
where station.station_id = t.start_station_id
;
```

Grading info:

- ***For incorrect solutions (with diff output), check these:***
- *a). -1 for any extra columns.*
- *b). Get 0 for wrong result.*
- ***For seemingly correct solutions (no diff output), check c.***
- *c). -1 for using station_name instead of station_id to represent station (outpus in this case may be the same as correct ones).*

(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()`.

Solution:

```
select round(short_trip_on_special_weather.cnt
```

```
      * 1.0 / short_trip.cnt, 4) as percentage
from (
  select count(*) as cnt
  from trip, station, weather
  where trip.start_station_id = station.station_id and
        station.zip_code = weather.zip_code and
        date(trip.start_time) = weather.date and
        duration <= 60 and
        weather.events not like ''
) as short_trip_on_special_weather, (
  select count(*) as cnt
  from trip
  where duration <= 60
) as short_trip
;
```

Grading info:

- *This question on the first version of released writeup is not clear stated, more specifically, the weather can include* ***"1. weather on start_station"****,* ***"2. weather on end_station"*** *or* ***"3. weather on start_station or end_station"****. In all these cases, the result is 0.1600. Students will not lose score when they consider any of the three cases.*
- ***For incorrect solutions (with diff output), check these:***
- *a). -1 for any extra columns.*
- *b). Get 0 for wrong result.*
- ***For seemingly correct solutions (no diff output), check c.***
- *c). -1 for using station_name instead of station_id to represent station (outputs in this case may be the same as correct ones).*

(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).

Solution:

```
select station.station_name,
       fewest.fewest_date,
       weather.events
```

```
from station, weather, (
  select o.station_id as station_id, (
    select date(start_time)
    from trip as i
    where o.station_id = i.start_station_id
    group by date(start_time)
    order by count(*) asc, date(start_time) asc
    limit 1
  ) as fewest_date
  from station as o
) as fewest
where fewest.station_id = station.station_id and
      station.zip_code = weather.zip_code and
      fewest.fewest_date = weather.date and
      weather.events not like ''
order by station.station_name asc
;
```

Grading info:

- *This question on the first version of released writeup is not clear stated, more specifically, all "least popular days" are considered.* ***In this case, the result contains 765 rows. Students will not lose score when considering all least popular days, but following rubrics still hold.***
- ***For incorrect solutions (with diff output), check these:***
- *a). -1 for any extra columns.*
- *b). -2 for wrong order if results are right but order is wrong.*
- *c). -5 for missing one column. Get 0 for missing more than one columns.*
- *d). -1 for each missing/extra/wrong tuple (choose a nearer value between 21 and 765 as base).*
- *e). Get half score for considering not only the earliest "least popular day".*
- ***For seemingly correct solutions (no diff output), check f.***
- *f). -1 for using station_name instead of station_id to represent station (outpus in this case may be the same as correct ones).*