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

Homework 5 (by Lu Zhang)
Due: **hard copy, in class** at **3:00pm, on Wednesday, Oct. 26**
Due: **tarball, BlackBoard** at **3:00pm, on Wednesday, Oct. 26**

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 5)`
  - and the count of slip-days you are using.

For your information:

- Graded out of **100** points; **4** questions total
- Rough time estimate: *30min for setting up postgres; approx. 1 hour for each question*

Revision : 2016/10/11 14:51

Question	Points	Score
EXPLAIN and ANALYZE	34	
Using indexes	23	
Joins	18	
More complicated join with order by	25	
Total:	100	

Preliminaries

Database set-up

In this homework, we will use Postgres and the bay-area-bike-sharing dataset used in Homework 2. Please use the machine and port assigned to you for Homework 2. Please follow Homework 2's Postgres setup instructions, available at `http://15415.courses.cs.cmu.edu/fall2016/hws/HW2/index.html` for setting up Postgres and loading data.

What to deliver: Check-list

Both hard copy, and soft copy:

1. **Hard copy**:
   - What: hard copy of your answers to all questions.
   - When: Oct. 26, 3:00pm
   - Where: in class

   Keep all your answers in one document, but still provide (`course#`, `Homework#`, `Andrew ID`, `name`).
2. **Soft copy: tar-file**:
   - What: A `tar.gz` file (`<your-andrew-id>.tar.gz`) with all your SQL code. Please see the next paragraph for creating the tarball for submission.
   - When: Oct. 26, 3:00pm
   - Where: on *Blackboard*, under 'Assignments'/'Homework #5'

Create the tarball for submission. Obtain the HW5 template folder from `http://15415.courses.cs.cmu.edu/fall2016/hws/HW5/hw5.tar.gz`. After `tar xvzf`, check the directory **`./hw5`** and replace the content of each place-holder `hw5/queries/*.sql` file with your SQL code. Once all your SQL code is in place, run `make submission` inside `./hw5` to create the tarball for submission, which is named as `$USER`.tar.gz, where `$USER` is your andrew ID.

Introduction

The purpose of this homework is to make you familiar with the query execution engine of PostgreSQL. In particular, you will have to analyze a few queries, and answer questions regarding their performance when turning different knobs of the execution engine.

In order to answer the questions, you might find the following documentation links useful:

- Documentation of `EXPLAIN ANALYZE`:
  http://www.postgresql.org/docs/9.2/static/sql-explain.html.
- Making sense of the `EXPLAIN ANALYZE` output:
  http://www.postgresql.org/docs/9.2/static/performance-tips.html.
- PostgreSQL query planner documentation:
  http://www.postgresql.org/docs/9.2/static/runtime-config-query.html.

- How to create an index:
  http://www.postgresql.org/docs/9.2/static/sql-createindex.html.
- The system table `pg_class`:
  http://www.postgresql.org/docs/9.2/static/catalog-pg-class.html.

FAQs

- *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: EXPLAIN and ANALYZE . . . . . [34 points]

In this question, you'll learn how to use `EXPLAIN` and `ANALYZE` to understand the impact of indexes on simple queries.

Answer the questions (a) - (e) based on the query below:

```
SELECT * FROM trip
WHERE bike_id = 10;
```

(a) [**5 points**] Provide the **execution plan** of the query and **the SQL statement** you use to generate the result.

(b) Based on the execution plan:

i. [**1 point**] What was the estimated cost of the query? (in arbitrary units)

ii. [**1 point**] What was the total runtime? (in ms)

(c) [**3 points**] Create an index on the attribute `bike_id` on the table `trip`.[1] **Provide the SQL statement**.

(d) [**3 points**] **Provide the new execution plan of the query**, with the index in place.

(e) Based on the new execution plan:

i. [**1 point**] What was the estimated cost of the query? (in arbitrary units)

ii. [**1 point**] What was the total runtime? (in ms)

iii. [**1 point**] What was the estimated number of tuples to be output?

iv. [**1 point**] What was the actual number of tuples to be output?

(f) Use the table `pg_class` to answer the following questions about table `trip`:

i. [**4 points**] How many pages are used to store the index you created? **Provide the answer and the query** you use to generate the answer.

ii. [**4 points**] How many tuples are in the index you created on column bike_id? **Provide the answer and the query** you use to generate the answer.

(g) Use the table `pg_class` to answer the following questions about table `weather`:

i. [**2 points**] How many tuples are in the table `weather`, according to `pg_class`?

ii. [**4 points**] In Table `weather`, delete all records of which date is earlier than `'2013-10-01'`. **Provide the SQL statement you use**.

iii. [**1 point**] After deletion, rerun your query in (g).i. Is the new result equal to the result of running `SELECT COUNT(*) FROM weather`?

iv. [**2 points**] `ANALYZE` is a Postgres function used to collect statistics about a database. You want to use it especially after considerable number of modificaitons happen to that database. Run `ANALYZE`, and then rerun your query in (g).i again. Is the new result equal to the result of running `SELECT COUNT(*) FROM weather`?

[1] Using the default PostgreSQL options.

Question 2: Using indexes . . . . . [23 points]

In this question, you'll learn the conditions under which indexes may or may not be used by the query optimizer.

(a) **[1 point]** Create an index on the column `start_station_name` on the table `trip`.[2] **Provide the SQL command you use**.

(b) For each of those queries, answer (yes) if the index you created on trip.start_station_name was used or (not) if it wasn't:

i. **[1 point]**
```
SELECT * FROM trip
WHERE start_station_name like 'San';
```

ii. **[1 point]**
```
SELECT * FROM trip
WHERE start_station_name like '%San';
```

iii. **[1 point]**
```
SELECT * FROM trip
WHERE start_station_name != 'Mountain View Caltrain Station';
```

iv. **[1 point]**
```
SELECT * FROM trip
WHERE start_station_name > 'San';
```

v. **[1 point]**
```
SELECT * FROM trip
WHERE start_station_name BETWEEN 'San Francisco' AND 'San Jose'
AND end_station_name > 'San';
```

vi. **[1 point]**
```
SELECT * FROM trip
WHERE start_station_name BETWEEN 'San Francisco' AND 'San Jose'
OR end_station_name > 'San';
```

(c) **Make sure you still have an index on the column** `trip.bike_id`. For each of those queries, answer (1) if only the index on `start_station_name` was used, (2) if only the index on `bike_id` was used, (3) if both were used, or (4) if neither one of the indexes were used:

i. **[1 point]**
```
SELECT * FROM trip
WHERE start_station_name BETWEEN 'San Francisco' AND 'San Jose'
AND bike_id < 10;
```

ii. **[1 point]**
```
SELECT * FROM trip
WHERE start_station_name BETWEEN 'San Francisco' AND 'San Jose'
AND bike_id < 500;
```

[2] Using the default PostgreSQL options.

iii. **[1 point]**
```
SELECT * FROM trip
WHERE start_station_name BETWEEN 'San Francisco' AND 'San Jose'
AND bike_id BETWEEN 500 AND 510;
```

iv. **[1 point]**
```
SELECT * FROM trip
WHERE start_station_name > 'San Francisco'
AND bike_id < 500;
```

(d) For the query
```
SELECT * FROM trip
WHERE bike_id BETWEEN 10 AND 20;
```
, answer the following questions:

i. **[1 point]** Was the index on `bike_id` used?

ii. **[1 point]** What percentage of the total records in the table `trip` was returned? Provide a percent and retain two significant figures.

(e) For the query
```
SELECT * FROM trip
WHERE bike_id > 10 ,
```
answer the following questions:

i. **[1 point]** Was the index on `bike_id` used?

ii. **[1 point]** What percentage of the total records in the table `trip` was returned? Provide a percent and retain two significant figures.

(f) For the query
```
SELECT * FROM trip
WHERE bike_id > 10 ORDER BY start_time;
```
, answer the following questions:

i. **[1 point]** Which method was used for sorting?

ii. **[1 point]** Where did the sorting happen – memory or disk?

iii. **[1 point]** How much space was used for sorting?

iv. **[1 point]** What was the total runtime? (in ms)

(g) Increase PostgreSQL working memory with the command `SET work_mem = '128MB';`. For the same query as (f), answer the following questions:

i. **[1 point]** Which method was used for sorting?

ii. **[1 point]** Where did the sorting happen – memory or disk?

iii. **[1 point]** How much space was used for sorting?

iv. **[1 point]** What was the total runtime? (in ms)

(h) **[0 points]** Execute the command `RESET work_mem;` to get PostgreSQL working memory back to the default value (or your answers for the next questions will turn out wrong).

Question 3: Joins . . . . . [18 points]

In this question, you'll learn more about the different methods used by PostgreSQL for executing joins.

Make sure you reset `work_mem` to its default value, as per Q2-(h).

Answer the questions based on the query below:

```
SELECT trip.*, station.city
FROM trip, station
WHERE trip.start_station_id = station.station_id AND bike_id < 200;
```

(a) Answer the following questions according to the query above:

i. [**3 points**] **Provide the query plan for the query above**.

ii. [**2 points**] Which join method was used – nested loop, merge, or hash?

iii. [**1 point**] What was the estimated cost of the query? (in arbitrary units)

iv. [**1 point**] What was the total runtime? (in ms)

(b) Execute the command `SET enable_hashjoin = false;` to disable hash joins. **Provide the new query plan**, and answer the following questions:

i. [**2 points**] Which join method was used – nested loop, merge, or hash?

ii. [**1 point**] What was the estimated cost of the query? (in arbitrary units)

iii. [**1 point**] What was the total runtime? (in ms)

(c) Execute the command `SET enable_mergejoin = false;` to disable merge joins. Provide the new query plan, and answer the following questions:

i. [**2 points**] Which join method was used – nested loop, merge, or hash?

ii. [**1 point**] What was the estimated cost of the query? (in arbitrary units)

iii. [**1 point**] What was the total runtime? (in ms)

(d) Execute the command `SET enable_indexscan = false; SET enable_bitmapscan = false;` to disable index scans. Provide the new query plan, and answer the following questions:

i. [**2 points**] Which join method was used – nested loop, merge, or hash?

ii. [**1 point**] What was the total runtime? (in ms)

(e) [**0 points**] Execute these commands to re-enable the different joins (or your answers for the next questions will turn out wrong):

```
RESET enable_mergejoin;
RESET enable_hashjoin;
RESET enable_indexscan;
RESET enable_bitmapscan;
```

Question 4: More complicated join with order by . . . . [25 points]

Answer the following questions based on the query below:

```
SELECT o.bike_id, end_time,
(SELECT SUM(duration)
FROM trip AS i
WHERE i.bike_id = o.bike_id and i.end_time <= o.end_time
) AS ac
FROM trip AS o
WHERE o.bike_id < 20
ORDER BY bike_id ASC, ac ASC
;
```

(a) **[0 points]** Destroy any indexes created on the previous questions.

(b) **[5 points]** **Provide the query plan** for the query above.

(c) i. **[1 point]** What was the estimated cost of the query? (in arbitrary units)

ii. **[1 point]** What was the total runtime? (in ms)

(d) i. **[2 points]** What sorting algorithm was used for ordering by bike_id?

ii. **[2 points]** Where did the sort happen (disk or memory)?

(e) **[5 points]** An index on end_time or bike_id, which do you think will be more helpful? Create an index on the one you choose and **provide the SQL statement you use**.

(f) i. **[4 points]** **Provide the new query plan** after the index is created.

ii. **[5 points]** According to the new query plan and the old query plan, did the index help reduce the estimated cost? If no, why? If yes, which part (of the plan) was improved mostly by the new index?