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

Homework 5 (by Lu Zhang) - Solutions
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 : 2016/10/30 13:38

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.

Solution:

Grading info:

- *-3 if SQL statement is incorrect.*
- *-2 if query plan is not given.*

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

QUERY PLAN

Seq Scan on trip (cost=0.00..17997.49 rows=793 width=80)
(actual time=1.333..120.170 rows=248 loops=1)
Filter: (bike_id = 10)
Rows Removed by Filter: 669711
Total runtime: 120.247 ms

(b) Based on the execution plan:

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

Solution: 17997.49

Grading info:

- *Get 0 if answer does not match the number in the given query plan.*

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

Solution: 120.247

Grading info:

- *Get 0 if total runtime is not provided in the query plan.*
- *Get 0 if answer does not match the number in the given query plan.*

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

Solution:

```
CREATE INDEX idx_bike_id ON trip(bike_id);
```

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

[1] Using the default PostgreSQL options.

Question 1 continues. . .

Solution:

QUERY PLAN
Bitmap Heap Scan on trip (cost=18.47..2433.10 rows=793 width=80)
(actual time=0.558..2.090 rows=248 loops=1)
Recheck Cond: (bike_id = 10)
-> Bitmap Index Scan on idx_bike_id
(cost=0.00..18.27 rows=793 width=0)
(actual time=0.453..0.453 rows=248 loops=1)
Index Cond: (bike_id = 10)
Total runtime: 2.168 ms

(e) Based on the new execution plan:

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

Solution: 2433.10
Grading info:
- *Get 0 if answer does not match the number in the given query plan.*

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

Solution: 2.168
Grading info:
- *Get 0 if total runtime is not provided in the query plan.*
- *Get 0 if answer does not match the number in the given query plan.*

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

Solution: 793
Grading info:
- *Get 0 if answer does not match the number in the given query plan.*

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

Solution: 248
Grading info:
- *Get 0 if answer does not match the number in the given query plan.*

(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.

Solution:
Grading info:
- *-2 if query is not given or incorrect. Relname should match the index name created in c (if index name is not specified, it gets a default name $table_$column_idx)).*
- *-2 if answer is not given or incorrect.*

```
SELECT relpages FROM pg_class
WHERE relname = 'idx_bike_id';
```

relpages
1840

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.

Solution:
Grading info:
- *-2 if query is not given or incorrect. Relname should match the index name created in c (if index name is not specified, it gets a default name $table_$column_idx)).*
- *-2 if answer is not given or incorrect.*

```
SELECT reltuples FROM pg_class
WHERE relname = 'idx_bike_id';
```

reltuples
669959

(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`?

Solution: (No need to provide the query.)
```
SELECT reltuples FROM pg_class
WHERE relname = 'weather';
```

reltuples
3665

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

Solution:
```
DELETE FROM weather
WHERE date < '2013-10-01';
```

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`?

Solution: No.

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`?

Solution: YES.

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**.

Solution:

```
CREATE INDEX idx_start_sta_name ON trip(start_station_name);
```

(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';
```

Solution: yes

ii. **[1 point]**

```
SELECT * FROM trip
WHERE start_station_name like '%San';
```

Solution: no

iii. **[1 point]**

```
SELECT * FROM trip
WHERE start_station_name != 'Mountain View Caltrain Station';
```

Solution: no

iv. **[1 point]**

```
SELECT * FROM trip
WHERE start_station_name > 'San';
```

Solution: no

v. **[1 point]**

```
SELECT * FROM trip
WHERE start_station_name BETWEEN 'San Francisco' AND 'San Jose'
AND end_station_name > 'San';
```

Solution: yes

vi. **[1 point]**

```
SELECT * FROM trip
WHERE start_station_name BETWEEN 'San Francisco' AND 'San Jose'
OR end_station_name > 'San';
```

[2] Using the default PostgreSQL options.

Solution: no

(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;
```

Solution: (2) only bike_id

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

Solution: (1) only start_station_name

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

Solution: (3) both

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

Solution: (4) neither

(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?

Solution: yes

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

Solution: ∼0.54% (3594 out of 669959 tuples)
Grading info:
- *Get 0 score if answer is wrong or has different number of 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?

Solution: no

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

Solution: 100% or 99% (669460 out of 669959 tuples)
Grading info:
- *Get 0 score if answer is wrong or has different number of 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?

Solution: external merge

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

Solution: disk

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

Solution: 62256kB
Grading info:
- *Get 0 if answer is different from above (including 62256kb).*

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

Solution: 1643.905 ms (any number)

(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?

Solution: quicksort (or the same thing but different names)

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

Solution: memory

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

Solution: 122344kB
Grading info:
- *Get 0 if answer is different from above (including 122344kb).*

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

Solution: 374.410 ms (any number)

(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**.

Solution:

Grading info:

- *iii) and iv) get 0 score if the query plan is not provided for that question.*
- *iv) gets 0 if total runtime is not provided in the query plan.*
- *Answers should match the numbers in the given query plan.*

QUERY PLAN

Hash Join (cost=1038.19..12108.49 rows=55135 width=92)
(actual time=21.440..104.826 rows=58161 loops=1)
Hash Cond: (trip.start_station_id = station.station_id)
-> Bitmap Heap Scan on trip
(cost=1035.62..11347.81 rows=55135 width=80)
(actual time=21.341..73.717 rows=58161 loops=1)
Recheck Cond: (bike_id < 200)
-> Bitmap Index Scan on idx_bike_id
(cost=0.00..1021.84 rows=55135 width=0)
(actual time=17.274..17.274 rows=58161 loops=1)
Index Cond: (bike_id < 200)
-> Hash (cost=1.70..1.70 rows=70 width=14)
(actual time=0.080..0.080 rows=70 loops=1)
Buckets: 1024 Batches: 1 Memory Usage: 4kB
-> Seq Scan on station (cost=0.00..1.70 rows=70 width=14)
(actual time=0.009..0.041 rows=70 loops=1)
Total runtime: 109.201 ms

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

Solution: Hash join

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

Solution: 12108.49

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

Solution: 109.201 ms

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

Solution:

Grading info:

- *ii) and iii) get 0 score if the query plan is not provided for that question.*
- *iii) gets 0 if total runtime is not provided in the query plan.*
- *Answers should match the numbers in the given query plan.*

QUERY PLAN

Merge Join (cost=18143.72..19108.93 rows=55135 width=92)
(actual time=113.293..140.804 rows=58161 loops=1)
Merge Cond: (station.station_id = trip.start_station_id)
-> Sort (cost=3.85..4.02 rows=70 width=14)
(actual time=0.080..0.084 rows=70 loops=1)
Sort Key: station.station_id
Sort Method: quicksort Memory: 28kB
-> Seq Scan on station (cost=0.00..1.70 rows=70 width=14)
(actual time=0.008..0.032 rows=70 loops=1)
-> Materialize (cost=18139.88..18415.55 rows=55135 width=80)
(actual time=113.206..130.260 rows=58161 loops=1)
-> Sort (cost=18139.88..18277.71 rows=55135 width=80)
(actual time=113.203..124.024 rows=58161 loops=1)
Sort Key: trip.start_station_id
Sort Method: external merge Disk: 5360kB
-> Bitmap Heap Scan on trip (cost=1035.62..11347.81 rows=55135 width=80)
(actual time=18.447..60.556 rows=58161 loops=1)
Recheck Cond: (bike_id < 200)
-> Bitmap Index Scan on idx_bike_id
(cost=0.00..1021.84 rows=55135 width=0)
(actual time=14.971..14.971 rows=58161 loops=1)
Index Cond: (bike_id < 200)
Total runtime: 222.155 ms

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

Solution: Merge Join

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

Solution: 19108.93

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

Solution: 222.155 ms

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

Solution:

Grading info:

- *ii) and iii) get 0 score if the query plan is not provided for that question.*
- *iii) gets 0 if total runtime is not provided in the query plan.*
- *Answers should match the numbers in the given query plan.*

QUERY PLAN

Nested Loop (cost=1035.62..26802.02 rows=55135 width=92)
(actual time=21.351..159.626 rows=58161 loops=1)
-> Bitmap Heap Scan on trip
(cost=1035.62..11347.81 rows=55135 width=80)
(actual time=21.327..64.955 rows=58161 loops=1)
Recheck Cond: (bike_id < 200)
-> Bitmap Index Scan on idx_bike_id
(cost=0.00..1021.84 rows=55135 width=0)
(actual time=17.263..17.263 rows=58161 loops=1)
Index Cond: (bike_id < 200)
-> Index Scan using station_pkey on station
(cost=0.00..0.27 rows=1 width=14)
(actual time=0.001..0.001 rows=1 loops=58161)
Index Cond: (station_id = trip.start_station_id)
Total runtime: 164.439 ms

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

Solution: nested loop

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

Solution: 26802.02

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

Solution: 164.439 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:

Solution:

Grading info:

- *ii) gets 0 score if the query plan is not provided for that question.*
- *ii) gets 0 if total runtime is not provided in the query plan.*

- *Answers should match the numbers in the given query plan.*

QUERY PLAN

```
Nested Loop (cost=0.00..75891.11 rows=55135 width=92)
(actual time=0.088..639.783 rows=58161 loops=1)
Join Filter: (trip.start_station_id = station.station_id)
Rows Removed by Join Filter: 4013109
-> Seq Scan on trip (cost=0.00..17997.49 rows=55135 width=80)
(actual time=0.047..97.649 rows=58161 loops=1)
Filter: (bike_id < 200)
Rows Removed by Filter: 611798
-> Materialize (cost=0.00..2.05 rows=70 width=14)
(actual time=0.000..0.003 rows=70 loops=58161)
-> Seq Scan on station (cost=0.00..1.70 rows=70 width=14)
(actual time=0.007..0.039 rows=70 loops=1)
Total runtime: 642.132 ms
```

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

Solution: nested loop

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

Solution: 642.132 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.

Solution:

QUERY PLAN

Sort (cost=51148785.20..51148791.70 rows=2599 width=10)
(actual time=301126.910..301127.045 rows=3583 loops=1)
Sort Key: o.bike_id, ((SubPlan 1))
Sort Method: quicksort Memory: 376kB
-> Seq Scan on trip o (cost=0.00..51148637.79 rows=2599 width=10)
(actual time=87.905..301122.164 rows=3583 loops=1)
Filter: (bike_id < 20)
Rows Removed by Filter: 666376
SubPlan 1
-> Aggregate (cost=19673.19..19673.20 rows=1 width=4)
(actual time=84.016..84.016 rows=1 loops=3583)
-> Seq Scan on trip i (cost=0.00..19672.39 rows=321 width=4)
(actual time=0.232..83.984 rows=299 loops=3583)
Filter: ((end_time <= o.end_time) AND (bike_id = o.bike_id))
Rows Removed by Filter: 669660
Total runtime: 301127.309 ms

Grading info:

- *c-i) and c-ii) get 0 score if the query plan is not provided for that question.*
- *c-ii) gets 0 if total runtime is not provided in the query plan.*
- *Answers should match the numbers in the given query plan.*

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

Solution: 51148791.70

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

Solution: 301127.309 ms

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

Solution: quick sort

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

Solution: 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**.

Solution:

```
CREATE INDEX idx_bike_id on trip(bike_id);
```

Grading info:

- *Get 0 scores if a different column is chosen to build index on.*

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

Solution:

QUERY PLAN

```
Sort (cost=7429499.05..7429505.55 rows=2599 width=10)
(actual time=1368.986..1369.121 rows=3583 loops=1)
Sort Key: o.bike_id, ((SubPlan 1))
Sort Method: quicksort Memory: 376kB
-> Bitmap Heap Scan on trip o
(cost=52.47..7429351.64 rows=2599 width=10)
(actual time=3.437..1367.394 rows=3583 loops=1)
Recheck Cond: (bike_id < 20)
-> Bitmap Index Scan on idx_bike_id
(cost=0.00..51.82 rows=2599 width=0)
(actual time=1.563..1.563 rows=3583 loops=1)
Index Cond: (bike_id < 20)
SubPlan 1
-> Aggregate (cost=2856.26..2856.27 rows=1 width=4)
(actual time=0.380..0.380 rows=1 loops=3583)
-> Bitmap Heap Scan on trip i
(cost=19.63..2855.46 rows=321 width=4)
(actual time=0.110..0.360 rows=299 loops=3583)
Recheck Cond: (bike_id = o.bike_id)
Filter: (end_time <= o.end_time)
Rows Removed by Filter: 298
-> Bitmap Index Scan on idx_bike_id (cost=0.00..19.55 rows=963 width=0)
(actual time=0.058..0.058 rows=597 loops=3583)
Index Cond: (bike_id = o.bike_id)
Total runtime: 1369.389 ms
```

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?

Solution: yes. Sequential scan on outer table was replaced with Bitmap Heap Scan on column bike_id, therefore filtering on o.bike_id was much faster. The same happened to inner table, which is less significant.

Grading info:

- *if index is created on bike_id in (e), get 2 scores for "YES".*
- *if index is created on bike_id in (e), get 3 scores for mentioning* ***outer*** *table* ***(table trip o)*** *or* ***filtering (bike_id < 20)*** *as the most impacted part.* ***Only mentioning seq scan on trip is replaced with bitmap heap scan gets 0 out of 3.***
- *if index is created on end_time in (e), get 2 scores for "NO" (and get 0 for the rest of this question);*