Carnegie Mellon University
Department of Computer Science
15-415/615 - Database Applications
C. Faloutsos & A. Pavlo, Fall 2015

Homework 5 (by Jinliang Wei)
Due: **hard copy, in class** at **3:00pm, on Wednesday, Oct. 28**
Due: **tarball, BlackBoard** at **3:00pm, on Wednesday, Oct. 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 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-1h for setting up postgres; approx. 1-2 hours for each question*

Revision : 2015/10/14 15:11

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

Preliminaries

Database set-up

In this homework, we will use Postgres and the Yelp review 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://www.cs.cmu.edu/~christos/courses/dbms.F15/hws/HW2/postgresql-setup.html` for setting up Postgres.

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: Oct. 28, 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. 28, 3:00pm
   - Where: on *Blackboard*, under 'Assignments'/'Homework #5'

Create the tarball for submission Obtain the HW5 template folder from `http://www.cs.cmu.edu/~christos/courses/dbms.F15/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 . . . . . [25 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 based on the query below:

```
SELECT * FROM business
WHERR city = 'Pittsburgh';
```

(a) **[3 points]** Provide the execution plan with the actual runtime of the query. Provide the SQL statement you used and its output.

(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 `city` on the table `business`.[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:

i. **[2 points]** How many pages are used to store the table `business`? Provide the answer and the query you use to generate the answer.

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

iii. **[2 points]** Is that number always equal to the result of running `SELECT COUNT(*) FROM business`?

iv. **[2 points]** How many pages are used to store the index you created?

v. **[2 points]** How many tuples are in the index?

[1] Using the default PostgreSQL options.

Question 2: Using indexes . . . . . [25 points]

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

Make sure you have an index on the column `business.city`, created in Q1-(c).

(a) For each of those queries, answer (yes) if the index you created for Q1, item (c) was used or (not) if it wasn't:

i. **[1 point]**
```
SELECT * FROM business
WHERE city = 'Pittsburgh';
```

ii. **[1 point]**
```
SELECT * FROM business
WHERE city > 'B';
```

iii. **[1 point]**
```
SELECT * FROM business
WHERE city BETWEEN 'B' AND 'G'
AND name > 'A';
```

iv. **[1 point]**
```
SELECT * FROM business
WHERE city BETWEEN 'B' AND 'G'
AND state = 'PA';
```

v. **[1 point]**
```
SELECT * FROM business
WHERE city BETWEEN 'B' AND 'G'
OR state = 'PA';
```

vi. **[1 point]**
```
SELECT * FROM business
WHERE city BETWEEN 'B' AND 'G';
```

vii. **[1 point]**
```
SELECT * FROM business
WHERE city != 'San Diego';
```

(b) **[1 point]** Create an index on the column `name` on the table `business`.[2] Provide the SQL command.

(c) For each of those queries, answer (1) if only the index on `business_city` was used, (2) if only the index on `business_name` was used, (3) if both were used, or (4) if neither one of the indexes were used:

i. **[1 point]**
```
SELECT * FROM business
WHERE city BETWEEN 'B' AND 'G'
AND name < 'T';
```

[2] Using the default PostgreSQL options.

ii. **[1 point]**
```
SELECT * FROM business
WHERE city BETWEEN 'B' AND 'G'
AND name < 'A' ;
```

iii. **[1 point]**
```
SELECT * FROM business
WHERE city BETWEEN 'B' AND 'G'
AND name BETWEEN 'A' AND 'D';
```

iv. **[1 point]**
```
SELECT * FROM business
WHERE city BETWEEN 'B' AND 'G'
AND name BETWEEN 'A' AND 'B';
```

v. **[1 point]**
```
SELECT * FROM business
WHERE city > 'B'
AND name > 'G';
```

(d) For the query
`SELECT * FROM business WHERE name BETWEEN 'A' AND 'B';` ,
answer the following questions:

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

ii. **[1 point]** What percentage of the total records in the table `business` was returned?

(e) For the query
`SELECT * FROM business WHERE name > 'A';` ,
answer the following questions:

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

ii. **[1 point]** What percentage of the total records in the table `clicks` was returned?

(f) For the query
```
SELECT * FROM business
WHERE city BETWEEN 'O' AND 'Q' ORDER BY city;
```
,
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 = '25MB';`.
For the same query as above, 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 . . . . . [25 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 business.*, business_category.*
FROM business, business_category
WHERE business.bid = business_category.bid;
```

(a) Provide the query plan for the query above, 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)

(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) **[5 points]** Create an index that improves the total runtime of this query.[3]. Provide the SQL statement.

(d) Provide the new query plan with the index in place, 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)

(e) 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)

(f) Execute the command `SET enable_indexscan = false; SET enable_bitmapscan = false;` to disable index scans (you don't have to actually run the query). Provide the new query plan, and answer the following questions:
   i. **[2 points]** Which join method was used – nested loop, merge, or hash?
   ii. **[2 points]** What was the estimated cost of the query? (Feel free to find out the actual time, but be aware that it takes a while.)

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

[3] Using the default PostgreSQL options.

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

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

```
SELECT business.bid, avg(review.stars), count(yelp_user.*)
FROM business, review, yelp_user
WHERE business.bid = review.bid
AND yelp_user.uid = review.uid
AND yelp_user.fans > 10
AND business.state = 'PA'
GROUP BY business.bid
ORDER BY avg(review.stars);
```

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

(b) **[5 points]** Provide the query plan for the query above along with the SQL query that generates the query plan.

(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. **[5 points]** What sorting algorithm was used for ordering by average number of reviews?

ii. **[5 points]** Where did the sort happen (disk or memory)?

(e) **[3 points]** Create an index for the fans column in yelp_user table. Provide the Postgres command.

(f) i. **[4 points]** What is the new execution plan after the index is created?

ii. **[1 point]** Did the index help reduce the estimated cost?