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

Homework 5 (by Jinliang Wei) - Solutions
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 : 2015/11/06 13:03

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.

General Grading Policies

- -1 per query if the query plan doesn't match the query in question (due to typo, etc); no additional penalty if the answers match the execution plan.
- Who graded what (by last name):
  A-D: Jiaxi
  F-Liu: Yujing
  Long-T: Anna
  V-X: Dana
  Y-Z: Jinliang

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.

Solution:

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

QUERY PLAN

Seq Scan on business (cost=0.00..1411.80 rows=2731 width=53)
(actual time=0.010..6.492 rows=2724 loops=1)
Filter: (city = 'Pittsburgh'::text)
Rows Removed by Filter: 58460
Total runtime: 6.599 ms

(b) Based on the execution plan:

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

Solution: 1411.80 *the actual number may change, correct answer just has to match cost provided in the execution plan.*

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

Solution: 6.5999 *the actual number may change, correct answer just has to match total runtime provided in the execution plan.*

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

Solution:

```
CREATE INDEX business_city ON business(city);
```

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

Solution:

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

[1] Using the default PostgreSQL options.

QUERY PLAN

Bitmap Heap Scan on business (cost=61.42..742.56 rows=2731 width=53)
(actual time=0.426..1.077 rows=2724 loops=1)
Recheck Cond: (city = 'Pittsburgh'::text)
-¿ Bitmap Index Scan on business_city (cost=0.00..60.74 rows=2731 width=0)
(actual time=0.388..0.388 rows=2724 loops=1)
Index Cond: (city = 'Pittsburgh'::text)
Total runtime: 1.196 ms

(e) Based on the new execution plan:

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

Solution: 742.56

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

Solution: 1.196

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

Solution: 2731

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

Solution: 2724

(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.

Solution:

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

relpages
647

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

Solution:

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

reltuples
61184

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

Solution: No.

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

Solution:

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

relpages
217

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

Solution:

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

reltuples
61184

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';
```
Solution: yes

ii. **[1 point]**
```
SELECT * FROM business
WHERE city > 'B';
```
Solution: no

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

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

v. **[1 point]**
```
SELECT * FROM business
WHERE city BETWEEN 'B' AND 'G'
OR state = 'PA';
```
Solution: no

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

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

Solution: no

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

Solution:

```
CREATE INDEX business_name ON business(name);
```

(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';
```

Solution: (1) only business_city

ii. **[1 point]**

```
SELECT * FROM business
WHERE city BETWEEN 'B' AND 'G'
AND name < 'A' ;
```

Solution: (3) both

iii. **[1 point]**

```
SELECT * FROM business
WHERE city BETWEEN 'B' AND 'G'
AND name BETWEEN 'A' AND 'D';
```

Solution: (1) only business_city

iv. **[1 point]**

```
SELECT * FROM business
WHERE city BETWEEN 'B' AND 'G'
AND name BETWEEN 'A' AND 'B';
```

Solution: (2) only business_name

v. **[1 point]**

```
SELECT * FROM business
WHERE city > 'B'
AND name > 'G';
```

Solution: (4) neither

(d) For the query

[2] Using the default PostgreSQL options.

`SELECT * FROM business WHERE name BETWEEN 'A' AND 'B';` ,
answer the following questions:

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

Solution: yes

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

Solution: ∼6% (3866 out of 61184 tuples)

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

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

Solution: no

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

Solution: 99% (60654 out of 61184 tuples)

(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?

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: 768kB

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

Solution: 77.026 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?

Solution: quicksort

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

Solution: memory

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

Solution: 1889kB

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

Solution: 14.095 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;
```

Grading info:

- *-1 for missing the query plan for one sub-problem;*
- *-2 for missing the query plan for more than one sub-problems.*

(a) Provide the query plan for the query above, and answer the following questions:

Solution:

QUERY PLAN

```
Hash Join (cost=2621.64..12291.81 rows=176697 width=89)
(actual time=30.593..335.271 rows=176697 loops=1)
Hash Cond: (business_category.bid = business.bid)
-> Seq Scan on business_category (cost=0.00..3217.97 rows=176697 width=35)
(actual time=0.012..18.277 rows=176697 loops=1)
-> Hash (cost=1258.84..1258.84 rows=61184 width=54)
(actual time=27.498..27.498 rows=61184 loops=1)
Buckets: 2048 Batches: 8 Memory Usage: 660kB
-> Seq Scan on business (cost=0.00..1258.84 rows=61184 width=54)
(actual time=0.002..5.616 rows=61184 loops=1)
Total runtime: 341.784 ms
```

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

Solution: hash join

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

Solution: 12291.81

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

Solution: 341.784ms

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

Solution:

QUERY PLAN

```
Merge Join (cost=23451.51..31426.37 rows=176697 width=89)
(actual time=736.541..1246.393 rows=176697 loops=1)
Merge Cond: (business.bid = business_category.bid)
-> Index Scan using business_bid on business
(cost=0.00..4729.87 rows=61184 width=54)
(actual time=0.005..27.649 rows=61184 loops=1)
-> Materialize (cost=23451.42..24334.91 rows=176697 width=35)
(actual time=736.527..901.759 rows=176697 loops=1)
-> Sort (cost=23451.42..23893.17 rows=176697 width=35)
(actual time=736.524..880.134 rows=176697 loops=1)
Sort Key: business_category.bid
Sort Method: external merge Disk: 7832kB
-> Seq Scan on business_category
(cost=0.00..3217.97 rows=176697 width=35) (actual time=0.004..16.439 rows=1766
Total runtime: 1340.858 ms
```

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

Solution: merge

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

Solution: 31426.37

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

Solution: 1340.858ms

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

Solution:

```
CREATE INDEX business_category_bid on business_category(bid);
```

(d) Provide the new query plan with the index in place, and answer the following questions:

[3]Using the default PostgreSQL options.

Solution:

QUERY PLAN

```
Merge Join (cost=0.98..19062.27 rows=176697 width=89)
(actual time=0.014..415.155 rows=176697 loops=1)
Merge Cond: (business.bid = business_category.bid)
-> Index Scan using business_bid on business
(cost=0.00..4729.87 rows=61184 width=54)
(actual time=0.005..26.930 rows=61184 loops=1)
-> Index Scan using business_category_bid on business_category
(cost=0.00..11970.72 rows=176697 width=35)
(actual time=0.003..71.276 rows=176697 loops=1)
Total runtime: 422.077 ms
```

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

Solution: merge

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

Solution: 19062.28

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

Solution: 422.077ms

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

Solution:

QUERY PLAN

```
Nested Loop (cost=0.00..33532.63 rows=176697 width=89)
(actual time=0.041..468.408 rows=176697 loops=1)
-> Seq Scan on business (cost=0.00..1258.84 rows=61184 width=54)
(actual time=0.005..6.423 rows=61184 loops=1)
-> Index Scan using business_category_bid on business_category
(cost=0.00..0.50 rows=3 width=35)
(actual time=0.006..0.007 rows=3 loops=61184)
Index Cond: (bid = business.bid)
Total runtime: 475.811 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: 33532.63

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

Solution: 475.811ms

(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:

Solution:

QUERY PLAN

Nested Loop (cost=0.00..246665461.27 rows=176697 width=89)
Join Filter: (business.bid = business_category.bid)
-> Seq Scan on business
(cost=0.00..1258.84 rows=61184 width=54)
-> Materialize (cost=0.00..5482.45 rows=176697 width=35)
> Seq Scan on business_category
(cost=0.00..3217.97 rows=176697 width=35)

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

Solution: nested loop

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.)

Solution: 246665461.27

(g) **[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]

```
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.

Solution:

QUERY PLAN

```
Sort (cost=57702.07..57708.55 rows=2591 width=84)
(actual time=448.134..448.252 rows=2090 loops=1)
Sort Key: (avg(review.stars))
Sort Method: quicksort Memory: 260kB
-> HashAggregate (cost=57522.78..57555.17 rows=2591 width=84)
(actual time=447.296..447.677 rows=2090 loops=1)
-> Hash Join (cost=9278.84..57503.35 rows=2591 width=84)
(actual time=54.685..443.537 rows=9401 loops=1)
Hash Cond: (review.uid = yelp_user.uid)
-> Hash Join (cost=1450.91..47127.98 rows=80253 width=50)
(actual time=8.379..332.184 rows=66116 loops=1)
Hash Cond: (review.bid = business.bid)
-> Seq Scan on review (cost=0.00..35066.64 rows=1569264 width=50)
(actual time=0.050..146.259 rows=1569264 loops=1)
-> Hash (cost=1411.80..1411.80 rows=3129 width=23)
(actual time=8.317..8.317 rows=3041 loops=1)
Buckets: 1024 Batches: 1 Memory Usage: 164kB
-> Seq Scan on business (cost=0.00..1411.80 rows=3129 width=23)
(actual time=0.007..7.854 rows=3041 loops=1)
Filter: ((state)::text = 'PA'::text)
Rows Removed by Filter: 58143
-> Hash (cost=7528.94..7528.94 rows=11839 width=80)
(actual time=45.361..45.361 rows=11243 loops=1)
Buckets: 1024 Batches: 2 Memory Usage: 615kB
-> Seq Scan on yelp_user (cost=0.00..7528.94 rows=11839 width=80)
(actual time=0.012..39.017 rows=11243 loops=1)
Filter: (fans > 10)
Rows Removed by Filter: 355472 Total runtime: 448.388 ms
```

```
EXPLAIN ANALYZE 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);
```

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

Solution: 57708

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

Solution: 448.388ms

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

Solution: quick sort

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

Solution: memory

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

Solution:
```
CREATE INDEX yelp_user_fans on yelp_user(fans);
```

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

Solution:

QUERY PLAN

```
Sort (cost=49116.46..49118.87 rows=964 width=84)
(actual time=344.845..344.885 rows=922 loops=1)
Sort Key: (avg(review.stars))
Sort Method: quicksort Memory: 97kB
-> HashAggregate (cost=49056.63..49068.68 rows=964 width=84)
(actual time=344.503..344.698 rows=922 loops=1)
-> Hash Join (cost=5066.93..49046.99 rows=964 width=84)
(actual time=29.264..343.205 rows=2398 loops=1)
Hash Cond: (review.uid = yelp_user.uid)
-> Hash Join (cost=1450.91..44387.79 rows=29854 width=50)
(actual time=9.039..289.786 rows=22820 loops=1)
Hash Cond: (review.bid = business.bid)
-> Seq Scan on review (cost=0.00..38989.80 rows=583766 width=50)
(actual time=0.542..218.261 rows=579527 loops=1)
Filter: (stars = 5::double precision)
Rows Removed by Filter: 989737
-> Hash (cost=1411.80..1411.80 rows=3129 width=23)
(actual time=8.480..8.480 rows=3041 loops=1)
Buckets: 1024 Batches: 1 Memory Usage: 164kB
-> Seq Scan on business (cost=0.00..1411.80 rows=3129 width=23)
(actual time=0.009..7.961 rows=3041 loops=1)
Filter: ((state)::text = 'PA'::text)
Rows Removed by Filter: 58143
-> Hash (cost=3317.03..3317.03 rows=11839 width=80)
(actual time=19.277..19.277 rows=11243 loops=1)
Buckets: 1024 Batches: 2 Memory Usage: 615kB
-> Bitmap Heap Scan on yelp_user
(cost=224.04..3317.03 rows=11839 width=80)
(actual time=1.415..12.891 rows=11243 loops=1)
Recheck Cond: (fans ¿ 10)
-> Bitmap Index Scan on yelp_user_fans
(cost=0.00..221.08 rows=11839 width=0)
(actual time=1.084..1.084 rows=11243 loops=1)
Index Cond: (fans ¿ 10)
```

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

Solution: yes (but not significant)