

CMU - SCS
15-415/15-615 Database Applications
Spring 2013, C. Faloutsos
Homework 5: Query Optimization
Released: Tuesday, 02/26/2013
Deadline: Tuesday, 03/19/2013

Reminders - IMPORTANT:

- Like all homework, it has to be done **individually**.
- Please submit your answers in a **hard copy, in class**, on Tuesday, 03/19/2013, 1:30pm. The hard copy must include
 - **your answers** to the questions, as well as
 - **the outputs** of postgres for the queries you are asked to run.
- **Separate pages** per question: As before, for ease of grading, please print each of the four questions on a separate page, i.e., four pages in total for this homework. Again, please type your **name and andrew ID on each** of the four pages.

Reminders - FYI:

- Weight: 15% of homework grade.
- The points of this homework add up to 100.
- Rough time estimates: 3 - 6 hours.

Database setup - very similar to HW2

Again, we will work with PostgreSQL. Most of the instructions are similar to homework 2, and repeated for your convenience. We have put up a readme file on the course homepage with detailed instructions here:

<http://www.cs.cmu.edu/~christos/courses/dbms.S13/hws/HW2/PostgreSQLReadme.html>

Here is the quick summary to get you started:

1. We use the machine `newcastle.db.cs.cmu.edu`, as in homework 2. We assume that you already have your user-id and password from the previous homework.
2. On first login, run `../dkoutra415/setup_db2.sh`, press “y” to continue when prompted.
3. Run `pg_ctl start -o -i` and when script is done, press “Enter” again.
4. Run `psql`.
5. Sanity check: Run `SELECT COUNT(*) FROM movies;`. The count should equal 2,680.
6. Sanity check: Run `SELECT COUNT(*) FROM play_in2;`. The count should equal 74,772.
7. Run `\q` to quit PostgreSQL.
8. **IMPORTANT:** Please **stop the server** using `pg_ctl stop` before logging out.

Getting Started

We use the database tables `movies` and `play_in2`, which are described below. Notice that the `movies` table is the same as the one in homework 2, but the `play_in2` table has one extra field – `year` – compared to the `play_in` table in homework 2.

```
movies (mid, title, year, num_ratings, rating)
play_in2 (mid, name, year, cast_position)
```

For your convenience, we repeat the attributes of the tables here:

- In the table `movies`: `mid` is the unique identifier for each movie; `title` is the movie’s title; `year` is the movie’s year-of-release; `num_ratings` is the total number of ratings that a movie receives from the users; and `rating` is the movie’s average rating on a scale of 0.0-10.0.
- The table `play_in2` contains the main cast of movies: `name` is the actor’s name (assume each actor has an unique name); `year` is the movie’s release year; and `cast_position` is the order in which the actor appears in the movie cast list. For example, in the movie *Titanic*, the `cast_positions` of Leonardo DiCaprio and Kate Winslet are 1 and 2, respectively.

Resources for EXPLAIN, ANALYZE etc.

The following documents are useful for the purpose of this assignment.

- **Syntax of EXPLAIN command:** <http://www.postgresql.org/docs/8.3/static/sql-explain.html>
- **How to use EXPLAIN command and understand its output:** <http://www.postgresql.org/docs/8.3/static/performance-tips.html>
- **Create an Index for a table:** <http://www.postgresql.org/docs/8.3/static/sql-createindex.html>
- **Query Planner in PostgreSQL:** <http://www.postgresql.org/docs/8.3/static/runtime-config-query.html>

Index Management in PostgreSQL

Check the commands `CREATE INDEX` and `DROP INDEX`. To list all the indexes created, use the `\di` command in postgresql.

Query Planner in PostgreSQL

The `SET` command can be used to change the behavior of the query planner. For example,
`SET enable_hashjoin=false;`
 in postgresql disables the query planner's use of hash join plans.

Grading Schema

Each question is worth 2 points, unless marked with *, in which case it is worth 4 points.

Q1. Basic Query Optimization [20 points] - SUBMIT ON SEPARATE PAGE

Purpose: The goal of this question is make us familiar with the plans that the query optimizer is using, and how (and when) an index affects the execution plan.

We will focus on the query that finds the name and cast position of the actors that participated in movies that were released in 1988:

```
SELECT name, cast_position
FROM play_in2
WHERE year = 1988;
```

Q1.1 Using the `EXPLAIN` and `ANALYZE` statements, execute the above query.

*1.1.1 What is the execution plan of this query? Please also give the output of postgresql concerning the execution plan.

1.1.2 What is the estimated total cost of the plan?

1.1.3 What is the actual total cost of the plan?

Q1.2 Build an index on the field `year`.

1.2.1 Re-run the query, and report its execution plan again. Also provide the output of postgresql concerning the execution plan.

1.2.2 What is the estimated cost of the plan with the built index?

1.2.3 What is the actual cost of the plan with the built index?

1.2.4 Did the estimated and actual costs change? (yes/no)

1.2.5 If they changed, how did they change? (increased, decreased, remained the same)

1.2.6 Explain why the costs remained the same or changed after building the index.

Q2. Understanding the Statistics [20 points] - SUBMIT ON SEPARATE PAGE

Purpose: The goal of this exercise is to understand the statistics that the query planner is using.

Please `DROP` the index you created in the previous question. We will focus on the following query:

```
SELECT name, cast_position
FROM play_in2
WHERE cast_position > 1;
```

Q2.1 Using the `EXPLAIN` and `ANALYZE` statements, execute the given query.

- 2.1.1 What is the execution plan of the query? Also provide the output of postgresql concerning the execution plan.
- 2.1.2 What is the estimated total cost of the query?
- 2.1.3 Build an index on the field `name`. Re-run the query, and give its execution plan again.
- 2.1.4 What is the estimated total cost of the query?
- 2.1.5 How did the index affect the execution time of the query (decreased, increased, or remained the same)? Please provide a 1-sentence explanation.

Q2.2 Let's see the statistics that the query planner is using by checking the built-in table `pg_class`.

- 2.2.1 How many pages does the table `play_in2` occupy?
- 2.2.2 How many pages are used to store the table `movies`?
- 2.2.3 How many pages are used to store the index on `name`?
- 2.2.4 What is the planner's estimate for the number of tuples of the output?
- 2.2.5 What is the actual number of tuples of the output?

Q3. ORDER BY Query Optimization [20 points] - SUBMIT ON SEPARATE PAGE

Purpose: The goal of this exercise is to see different sorting algorithms and their impact.

Please DROP the index you created in the previous question. In this question we will focus on two `ORDER BY` queries. The first query finds the title, year and rating of the movies that have rating at least 5. The result is sorted on the rating.

```
[Query1:]    SELECT title, year, rating
              FROM movies
              WHERE rating >= 5
              ORDER BY rating;
```

The second query finds the name and the release year of all movies, sorted on the release year.

```
[Query2:]    SELECT name, year
              FROM play_in2
              ORDER BY year;
```

Q3.1 Using the `EXPLAIN` and `ANALYZE` statements, execute QUERY 1.

- 3.1.1 What is the execution plan of QUERY 1? Also provide the output of postgresql concerning the execution plan.

3.1.2 Where is the sort of QUERY 1 done? (e.g., on the disk? in main memory? somewhere else?)

3.1.3 For QUERY 1, which algorithm is used for sorting?

Q3.2 Make sure that you have dropped the index on year that you created for Q1. Using the EXPLAIN and ANALYZE statements, execute QUERY 2.

3.2.1 What is the execution plan of QUERY 2?

3.2.2 For QUERY 2, where is the sort done?

3.2.3 Which sort algorithm is used for QUERY 2?

3.2.4 Is there a difference in the way sorting is done for the two queries? (yes/no)

3.2.5 Explain why there is or there is not any difference.

Q3.3 Create an index on the year of the play_in2 table.

3.3.1 What is the execution plan of QUERY 2 now? Also provide the output of postgresql concerning the execution plan.

3.3.2 How is the sort done for QUERY 2 after building the index?

Q4. JOIN Query Optimization [40 points] - SUBMIT ON SEPARATE PAGE

Purpose: In this question, we want to become familiar with the join execution plans, as well as learn how to disable a specific plan.

Please DROP the index you created in the previous question. We will study the execution plan of the following query:

```
SELECT play_in2.name, movies.title
FROM movies, play_in2
WHERE movies.year = play_in2.year;
```

Q4.1 Execute the query above.

4.1.1 What is the execution plan of the query? Also provide the output of postgresql concerning the execution plan.

4.1.2 What is the **estimated** total cost of the plan?

4.1.3 What is the join algorithm that the planner uses to execute the query?

Q4.2 Create an index on movies.year, and execute the query again.

- 4.2.1 What is the execution plan now? Also provide the output of postgresql concerning the execution plan.
- 4.2.2 What is the **estimated** total cost of the plan?
- 4.2.3 What join algorithm does the planner use to execute the query?
- 4.2.4 Did the plan of the query change? (yes/no)
- 4.2.5 Explain why execution plan changed or remained the same.

Q4.3 Now create an index on `play_in2.year` too, and execute the query again.

- 4.3.1 What is the execution plan now? Also provide the output of postgresql concerning the execution plan.
- 4.3.2 What is the **estimated** total cost of the plan?
- 4.3.3 What join algorithm does the planner use to execute the query?
- 4.3.4 Did the plan of the query change after the creation of the second index? (yes/no)
- 4.3.5 Explain why the execution plan changed or remained the same after the creation of both indices.

Q4.4 Using the command `SET`, disable the join algorithm that you answered in Q4.3.3.

- *4.4.1 Re-execute the query and give its execution plan. Also provide the output of postgresql concerning the execution plan.
- 4.4.2 What join algorithm did the planner use?
- 4.4.3 What is the **actual** total cost of the execution?

Q4.5 Now disable the join algorithm you answered in Q4.4.2, and re-execute the query.

- 4.5.1 Give the new execution plan. Also provide the output of postgresql concerning the execution plan.
- 4.5.2 What join algorithm did the planner use?
- 4.5.3 What is the **actual** total cost of the execution?