Query Optimization - Part 1

INFO-H417: Lab Session 2

2023-2024

A) **The EXPLAIN statement in PostgreSQL returns the execution plan of a query. By making use of this EXPLAIN statement, answer the following questions.** For a detailed information on how to use the explain statement, refer to the PostgreSQL documentation[1]. The questions refer to queries of Lab 1.

1. Query 4 contains a filter on the *return_date* column. When is this filtering applied in the query plan, and why?

2. The EXPLAIN statement returns an estimate of the cost of the query (ex.: cost=0.00...1.00). The first value corresponds to the estimated startup cost (cost before we output the first tuple) and the second corresponds to the estimated total cost. In query 4, the startup cost is equivalent to the total cost, whereas Query 1 has an estimated startup cost of 0.00. What causes this difference?

3. Query 2 counts the movies rated PG-13. How does this query compare (in execution plan and expected duration) to a query counting all of the movies? Assuming that the number of movies rated PG-13 is very low compared to the total number of movies, what could be done to improve the execution speed of this query?

4. In Lab 1, two queries are given to answer question 9. Which one is faster and why? How can the slower solution be improved with a minimal amount of changes. Make use of the EXPLAIN ANALYZE statement to analyze the execution speed of the queries together with their query plan.

5. These two following queries are almost identical, but still have quite different execution plans. What are the differences and why does the optimizer choose these plans?

```
Query a:
SELECT a.first_name, a.last_name
FROM customer a, actor b
```

[1] https://www.postgresql.org/docs/current/using-explain.html

```
WHERE a.first_name = b.first_name AND a.last_name = b.last_name;
```

```
Query b:
SELECT a.first_name, a.last_name
FROM customer a, staff b
WHERE a.first_name = b.first_name AND a.last_name = b.last_name;
```

6. Using the EXPLAIN statement, examine the execution plan of the two following queries. What does this teach us about how PostgreSQL handles Common Table Expressions (CTE's)? Note that this behaviour is relatively new. Running the same experiment in older versions of PostgreSQL can return different results.

```
Query c:
WITH film_num_rentals(film_id, title, length, num_rentals) AS (
    SELECT film_id, title, length, count(*)
    FROM film
    INNER JOIN inventory USING(film_id)
    INNER JOIN rental USING(inventory_id)
    GROUP BY film_id, title, length
)
SELECT avg(num_rentals)
FROM film_num_rentals
WHERE length >= 180;
```

```
Query d:
WITH film_num_rentals(film_id, title, length, num_rentals) AS (
    SELECT film_id, title, length, count(*)
    FROM film
    INNER JOIN inventory USING(film_id)
    INNER JOIN rental USING(inventory_id)
    GROUP BY film_id, title, length
    HAVING length >= 180
)
SELECT avg(num_rentals)
FROM film_num_rentals;
```

B) To determine the best execution plan for a query, the optimizer makes use of statistics computed on the different tables. These statistics can be found in the *pg_statistic* table or the more user-friendly *pg_stats* view[2]. **Use the *pg_stats* view to answer the following questions. Additionally, give the query used to answer the question.**

1. What is the most common rating in the 'film' table?
2. What fraction (in %) of the DVD's have not been returned yet (return_date is NULL)?
3. On average (over every possible movie duration), how many movies have the same exact duration? That is, if we count the number of movies for every value in the *length* column of the *film* table, what would be the average of this number?

[2]https://www.postgresql.org/docs/current/view-pg-stats.html