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?
   The filtering is applied before the join, during the sequential scan on rental. This is done to speed up the join, as it already removes a large set of rows from the rental table.

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?
   The estimated startup cost is the cost before the first tuple can be output. Depending on the operations in the query, this can be fast or slow. Query 1 has an estimated startup cost of 0.00, since the only operation is a sequential scan and we can thus start returning tuples instantly. Query 4 has a sorting phase for the DISTINCT operation, and we can thus only start returning tuples when the sorting is complete. Such a 'blocking' operator causes the estimated startup cost to be higher.

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?
   The only difference between their respective execution plans is the presence of the

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

filtering during the table scan. Both queries will thus scan the whole table and have a similar speed (the query with the filter will be slightly longer). An index on the rating column of the film table could be used to speed up 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.
   The solution without subquery is faster than the one with. Indeed, the subquery has to be executed for every customer_id, which is slower than a simple combination of joins. An improved solution of the slower query could be:

```
SELECT c.first_name, c.last_name
FROM customer c
INNER JOIN rental r USING(customer_id)
GROUP BY c.customer_id
HAVING count(r.rental_id) >= ALL (
    SELECT count(*)
    FROM rental r2
    GROUP BY r2.customer_id
);
```

   The solution only has to run the subquery once, and has thus a similar performance as the other query.

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
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;
```

   Query 1 is executed using a hash join, whereas query 2 is executed using a nested loop (and an index scan). The optimizer chooses the join method based on the

cardinality of the tables used in the join. Since query 1 requires a join between two tables with a lot of rows (599 and 200 respectively), the hash join is preferred. Query 2 on the other hand joins a large table (customer) with a table containing only 2 entries (staff). The faster nested loop join is then preferred.

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;
```

The query plans are identical. This means that PostgreSQL handles CTE's as a single big query instead of two separate ones, and is thus able to optimize it correctly. Older versions of PostgreSQL did not have this feature, and using CTE's could thus sometimes be a problem for the optimizer.

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?
   PG-13

   ```
   SELECT most_common_vals, most_common_freqs
   FROM pg_stats
   WHERE tablename = 'film' AND attname = 'rating';
   ```

2. What fraction (in %) of the DVD's have not been returned yet (return_date is NULL)?
   Around 1.14%

   ```
   SELECT null_frac*100 FROM pg_stats
   WHERE tablename = 'rental' AND attname = 'return_date';
   ```

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?
   On average, there are 7 movies of the same length/duration.

   ```
   SELECT -1/n_distinct FROM pg_stats
   WHERE tablename = 'film' AND attname = 'length';
   ```

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