Subquery Performance

by Sophia

WHAT'S COVERED

This lesson explores the use of the EXPLAIN command to analyze query performance, in two parts. Specifically, this lesson will cover:

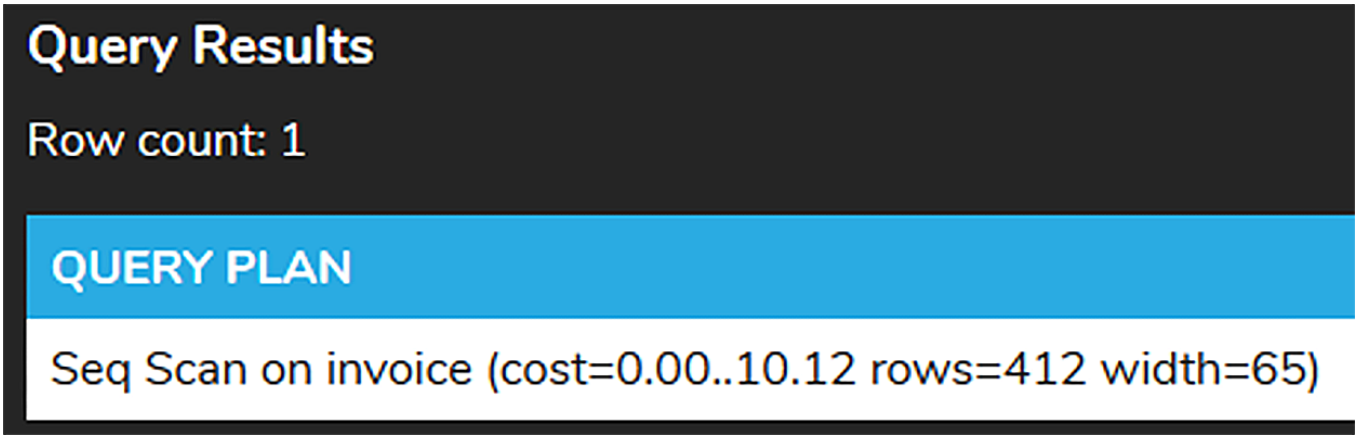
1. EXPLAIN Explained

When using subqueries, you have to consider performance as it relates to each part of the query. This can and will be different each time you run a query. However, you can get a general idea of how efficient a query will be in PostgreSQL because a **query plan** is created for every query that is attempted to run in the database. The database looks at the query structure and the properties of the data and tables to create a good plan before it is executed.

Understanding all of the details of a plan can be quite complex, so our focus is on the basics of what to look for within a query execution plan. We can use the **EXPLAIN** command on a query to display some of those details.

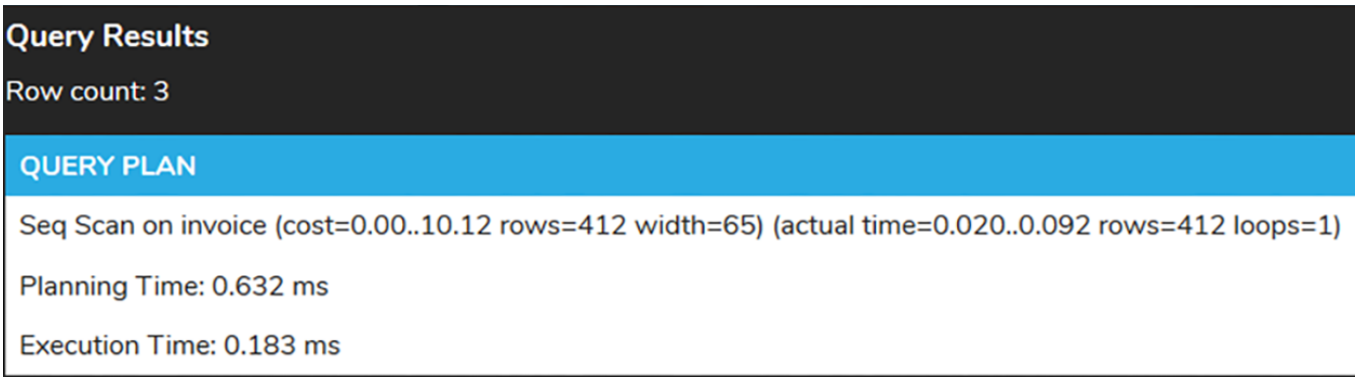
Let's query the invoice table and see what the results tell us:

```
EXPLAIN
SELECT *
FROM invoice;
```

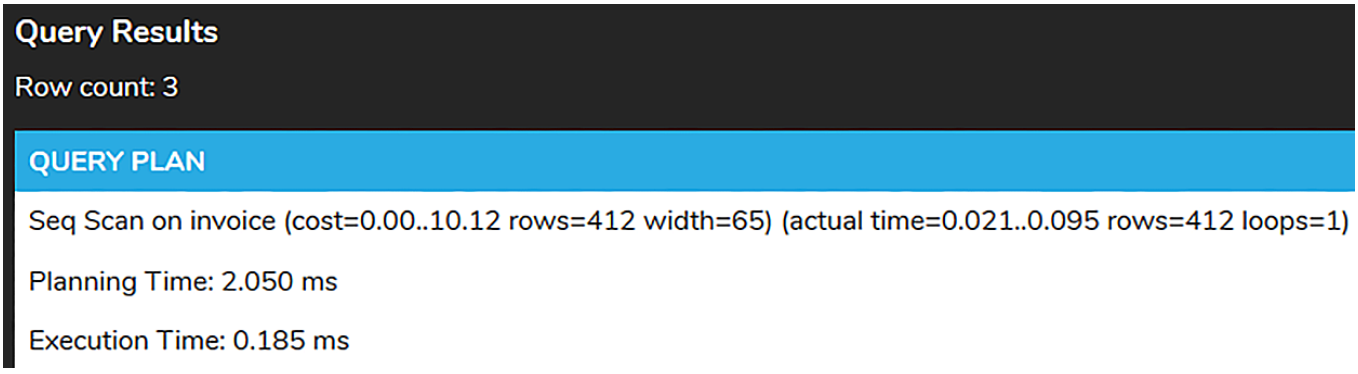


Using just EXPLAIN, the database will not run the query. It will create an estimation of the execution plan based on the statistics that it has. This means that the plan can differ a bit from reality. In PostgreSQL, though, we can execute the query as well. To do so, we can use the EXPLAIN ANALYZE at the beginning of the query instead of just EXPLAIN.

```
EXPLAIN ANALYZE
SELECT *
FROM invoice;
```



Note that this is an actual run process, so the timing will most likely be different each time you run it. For example, running it two more times yields the following results:



Query Results

Row count: 3

QUERY PLAN
Seq Scan on invoice (cost=0.00..10.12 rows=412 width=65) (actual time=0.018..0.088 rows=412 loops=1)
Planning Time: 0.353 ms
Execution Time: 0.170 ms

As such, we must rely on more than just the planning and execution time when comparing various statements.

Each query plan consists of **nodes** that can be nested and executed from the inside out. This means that the innermost node is executed before the outer node. Each node has a set of associated statistics, like the cost, the number of rows that are returned, the number of loops that are performed (if needed), and some other choices. From the last execution above, we can see that the cost shows 0.00..10.12, and we estimate that there are 412 rows returned. The width also shows the estimated width of each row in bytes, which in our case is 65. The cost field shows how expensive the node was—in other words, how much processing time and other system resources it required to execute. This can be a bit complex, as it is measured using different settings that start to become quite technical. The type of node appears at the left end of the first line. In the above example, it is "Seq Scan."

TERMS TO KNOW

Query Plan

The results of an EXPLAIN command, describing the cost of each operation in a query.

EXPLAIN

A command that provides insight into the query execution plan for a given SQL query.

Node

A section of a query plan that describes a single operation within the query.

1a. Node Types

There are different types of nodes, and some are more efficient than others, all other factors being equal. Here is a generalized ranking from most efficient to least efficient:

- Index Only Scan: This is the most efficient because it accesses all the required columns directly from their indexes without needing to access any unindexed table data. This reduces processing time.
- Index Scan: This type is typically efficient, especially when the query can leverage an index to quickly locate and retrieve the necessary rows based on the indexed columns.
- Bitmap Index Scan: Bitmap index scans can be efficient when querying multiple columns or performing set operations (AND, OR, NOT) that can leverage bitmap indexes effectively.
- Hash JOIN (Hash Map): A hash JOIN is a type of join that combines data from two tables based on a join condition. It is typically faster than a sequential scan and is quite effective for joining large data sets.

- Sequential Scan: This is typically the least efficient because it involves scanning the entire table sequentially. This can be slow, especially for large tables.

Because index scans are the most efficient, queries are most efficient when they reference only indexed columns, or when the queries are able to pull the needed records from the table based only on indexed columns. Keep in mind that primary key columns are automatically indexed, so querying on primary key columns can be very efficient compared to querying on other columns.

1b. Analyzing a Query Plan

When reading the cost in a query plan, note that it is expressed as two numbers separated by two dots, like this: 0.00..10.12. The first number is the lower bound—in other words, the minimum amount it could cost. The second number is the upper bound—in other words, the maximum amount it could cost. When comparing the cost of two different queries, you would want to look both at the upper bound (that is, the worst-case scenario) and at the size of the range between lower and upper bounds.

When comparing the overall cost of two queries, you could sum all of the lower bounds for the various nodes and then sum all of the upper bounds, to derive an expected range for the entire query. However, this does not always produce accurate information because a query execution plan can involve multiple operations that are not directly additive, because the operations may interact with each other in nonlinear ways.

A better way to use cost data is to see which of the nodes has the highest cost and then look at the clause in the query that it represents to see if there is any way to make that clause more efficient, such as by substituting a non-indexed column for an indexed one.

---

2. Subqueries vs. JOIN

Subqueries are more complex because you are using nested queries. For example, here is a subquery similar to what we used in the prior lesson:

```
EXPLAIN
SELECT invoice_id, invoice_date,customer_id, total
FROM invoice
WHERE customer_id IN
(SELECT customer_id FROM customer
WHERE city LIKE '%A');
```

Query Results

Row count: 6

QUERY PLAN
Hash Join (cost=2.81..15.49 rows=50 width=140)
Hash Cond: ((invoice.billing_city)::text = (customer.city)::text)
-> Seq Scan on invoice (cost=0.00..10.12 rows=412 width=8)
-> Hash (cost=2.74..2.74 rows=6 width=140)
-> Seq Scan on customer (cost=0.00..2.74 rows=6 width=140)
Filter: ((country)::text ~~ '%m'::text)

Now let's try getting the same data using a subquery:

```
EXPLAIN
SELECT *
FROM customer
WHERE city IN
(SELECT billing_city
FROM invoice
WHERE COUNTRY like '%m')
```

As you can see below, this version has a much higher cost. The first Seq Scan node's upper bound is 331.81. That's a lot higher than any of the upper bounds in the other version.

Query Results

Row count: 6

QUERY PLAN
Seq Scan on customer (cost=0.00..331.81 rows=30 width=140)
Filter: (SubPlan 1)
SubPlan 1
-> Result (cost=0.00..10.12 rows=412 width=8)
One-Time Filter: ((customer.country)::text ~~ '%m'::text)
-> Seq Scan on invoice (cost=0.00..10.12 rows=412 width=8)

Your turn! Open the SQL tool by clicking on the LAUNCH DATABASE button below. Then, enter in one of the examples above and see how it works. Next, try your own choices for which columns you want the query to provide.

SUMMARY

In this lesson, you learned that the **EXPLAIN** command provides insight into the query execution plan for a given query. Each query plan consists of different **types of nodes** that can be nested and executed from the inside out. You explored **analyzing a query plan** by adding the EXPLAIN command to the beginning of a query. The database engine uses this information to determine the order in which table scans, joins, and indexes are used to retrieve the requested data most efficiently. Lastly, you compared **subqueries vs. JOIN**, understanding that you can optimize query performance by modifying clauses so that they have a lower cost as reported in the query plan.

Source: THIS TUTORIAL WAS AUTHORED BY DR. VINCENT TRAN, PHD (2020) AND FAITHE WEMPEN (2024) FOR SOPHIA LEARNING. PLEASE SEE OUR TERMS OF USE.

TERMS TO KNOW

EXPLAIN
A command that provides insight into the query execution plan for a given SQL query.

Node
A section of a query plan that describes a single operation within the query.

Query Plan Node
The results of an EXPLAIN command, describing the cost of each operation in a query.