# Slow query performance

Last updated by | Hamza Aqel | Jan 9, 2023 at 5:19 AM PST

Before moving forward, it is important to identify if this is related to all queries or specific query to get started, use the below steps as a guidance:

- Check if either CPU or Memory resources are maxed out using our TSG, please check our CPU TSG
- Enable query store to get more details about the query executions on customer environment, for more details check our QueryStore details.
- If this is related to a specific query, you can check the below as example ..

The below instructions assuming you have a specific query , and try to figure out the reason od slowness :

**Step 1 :**

If a specific query is running slower, let's assume the slow query is the following:

select * from tablename<<<< change this to the query that is slow, ask the customer to run the following

1. \timing
   select * from tablename;
   EXPLAIN(analyze, buffers, verbose, costs, timing) select * from tablename;

2. Compare counter before and after running a query to see if we have cache hit or miss
   *SELECT*
   *heap_blks_read,*
   *heap_blks_hit,*
   *CURRENT_TIMESTAMP*
   *FROM*
   *pg_statio_user_tables*
   *Where*
   *relname=*"tablename";

   Then save the time and the queries and then run the query that is slow (when there is a repro)

   select * from tablename;

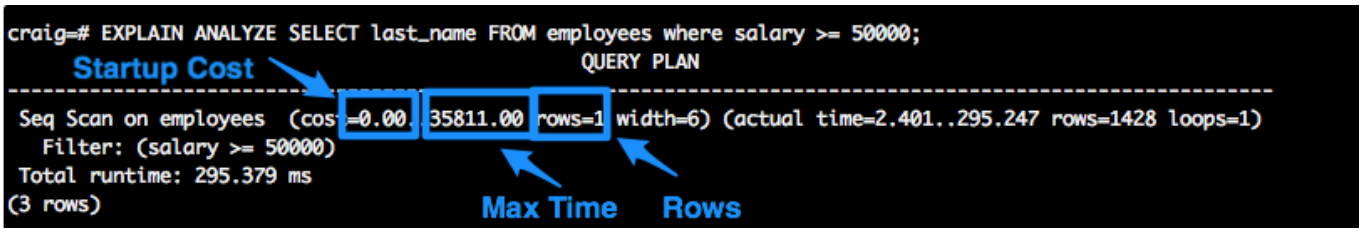
   *SELECT*
   *heap_blks_read,*
   *heap_blks_hit,*
   *CURRENT_TIMESTAMP*
   *FROM*
   *pg_statio_user_tables*
   *Where*
   *relname=*"tablename";

the idea of the above queries is to check if the query executions will be the same whatever the number of executios , as for the first time running , the cache might need to warm up .

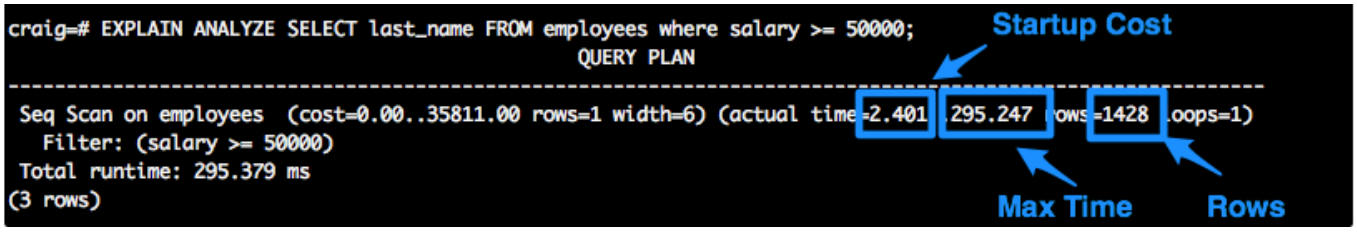
3- from step 1 ask the customer to share the execution plan and share with the customer that we are doing our best effort on such cases , to understand the execution plan you can refer to PostgreSQL Documentations , and see the below example (assuming we have a slowness with this query (SELECT last_name FROM employees where salary >= 50000;) :

```
EXPLAIN ANALYZE SELECT last_name FROM employees where salary >= 50000;
                                       QUERY PLAN
-----------------------------------------------------------------------------------------------------------
 Seq Scan on employees  (cost=0.00..16.50 rows=173 width=118) (actual time=0.018..0.018 rows=0 loops=1)
   Filter: (salary >= 50000)
 Total runtime: 0.053 ms
```

But lets take a look at what it actually means:



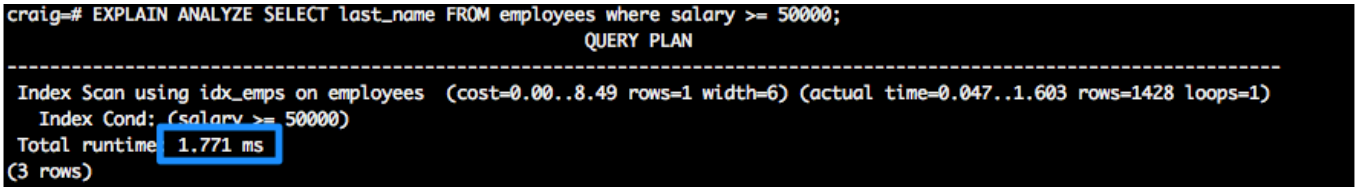

There's a couple of key items here. Often times you want to look for when a sequential scan is occurring, but more importantly you want to look at what the three items above are. The startup time, the maximum time and finally the number of rows returned. In this case, because we ran EXPLAIN ANALYZE, we have not only the estimated on the left, but the actual on the right as well:




In this case we see there's a high time spent and a sequential scan. As a result we may want to try to add an index and examine the results:

```
CREATE INDEX idx_emps on employees (salary);
```

With this we've now cut our query time from 295 ms to 1.7 ms:




Explain has many options , the most important options are :

- ANALYZE: with this keyword, EXPLAIN does not only show the plan and PostgreSQL's estimates, but it also executes the query (so be careful with UPDATE and DELETE!) and shows the actual execution time and row count for each step. This is indispensable for analyzing SQL performance.
- BUFFERS: You can only use this keyword together with ANALYZE, and it shows how many 8kB-blocks each step reads, writes and dirties. You always want this.
- VERBOSE: if you specify this option, EXPLAIN shows all the output expressions for each step in an execution plan. This is usually just clutter, and you are better off without it, but it can be useful if the executor spends its time in a frequently-executed, expensive function.

Typically, the best way to call EXPLAIN is: EXPLAIN (ANALYZE, BUFFERS) /* SQL statement */; for example :

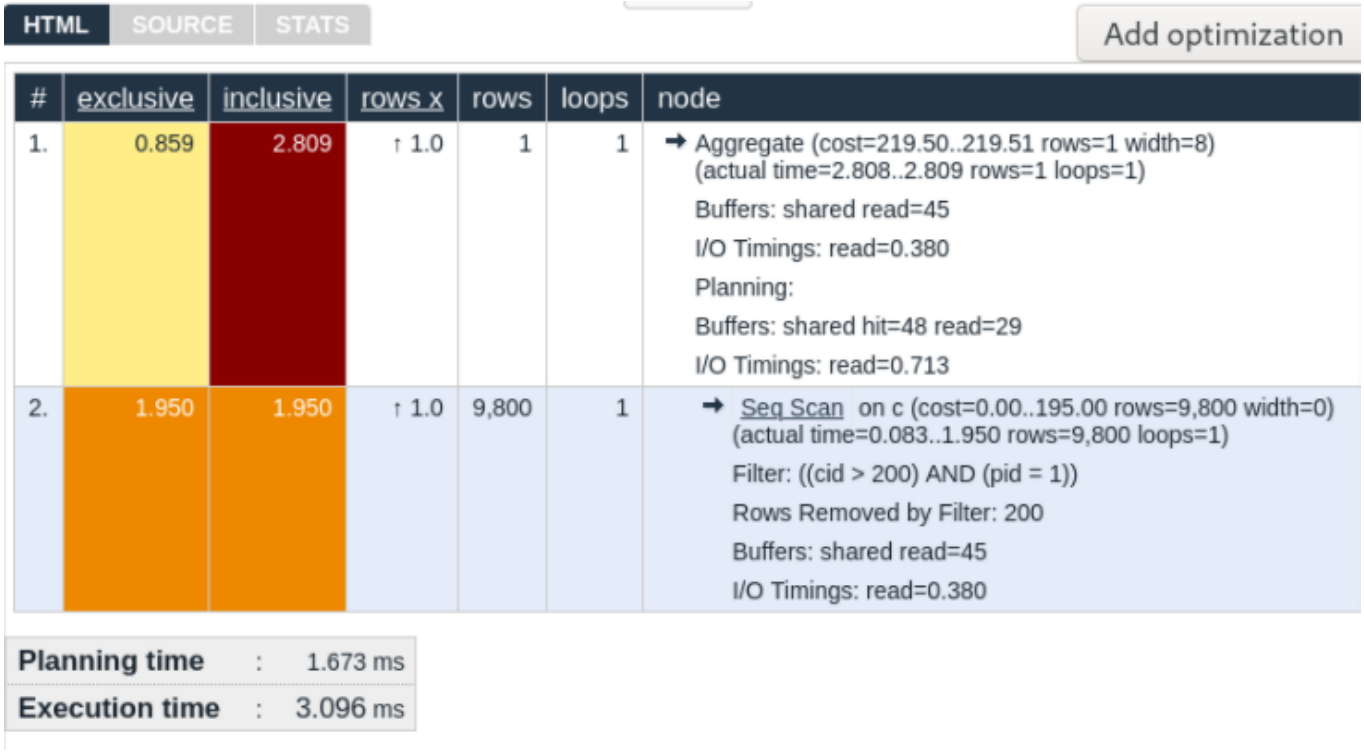
```
EXPLAIN (ANALYZE, BUFFERS) SELECT count(*) FROM c WHERE pid = 1 AND cid > 200;

                                                    QUERY PLAN
-------------------------------------------------------------------------------------------------
 Aggregate  (cost=219.50..219.51 rows=1 width=8) (actual time=2.808..2.809 rows
   Buffers: shared read=45
   I/O Timings: read=0.380
   ->  Seq Scan on c  (cost=0.00..195.00 rows=9800 width=0) (actual time=0.083.
         Filter: ((cid > 200) AND (pid = 1))
         Rows Removed by Filter: 200
         Buffers: shared read=45
         I/O Timings: read=0.380
 Planning:
   Buffers: shared hit=48 read=29
   I/O Timings: read=0.713
 Planning Time: 1.673 ms
 Execution Time: 3.096 ms
(13 rows)
```

**Tools to interpret EXPLAIN ANALYZE output**:

Since reading a longer execution plan is quite cumbersome, there are a few tools that attempt to visualize this:

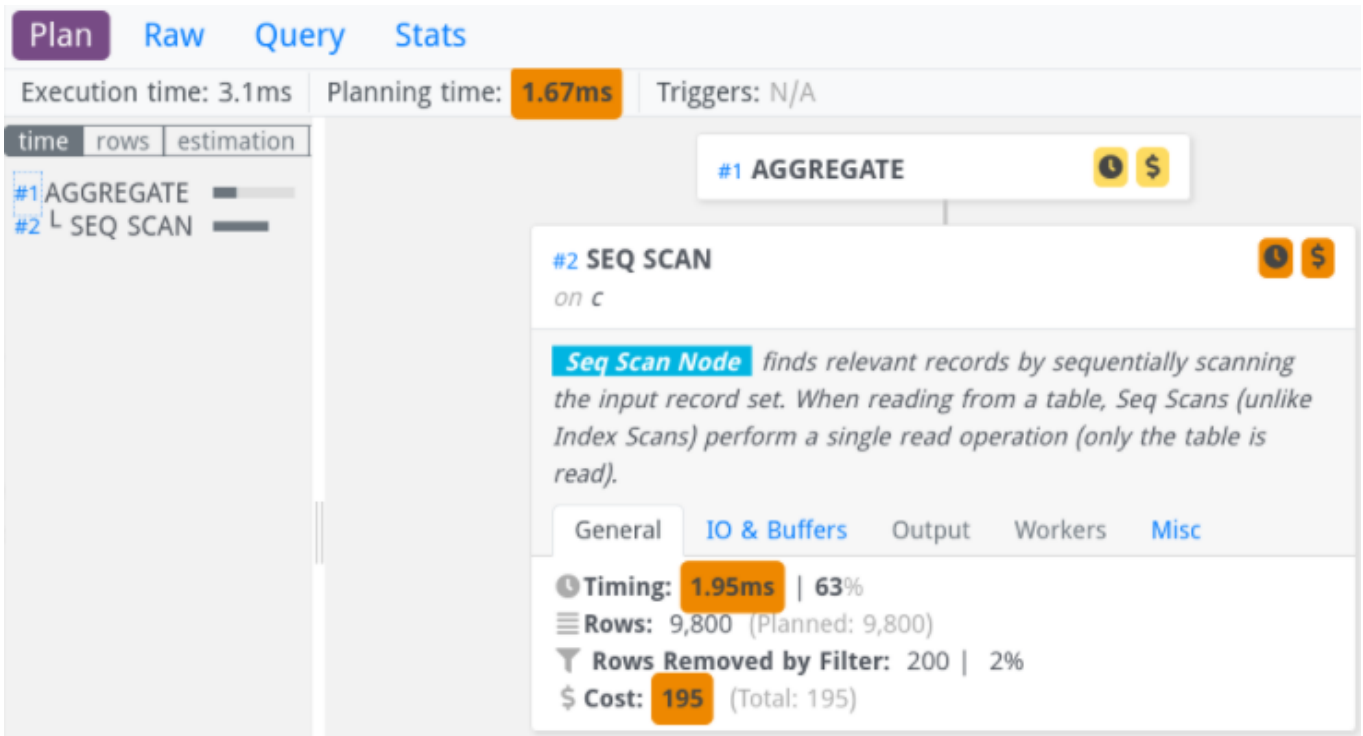
Depesz' EXPLAIN ANALYZE visualizer:

This tool can be found at https://explain.depesz.com/. If you paste the execution plan in the text area and hit "Submit", you will get output like this:



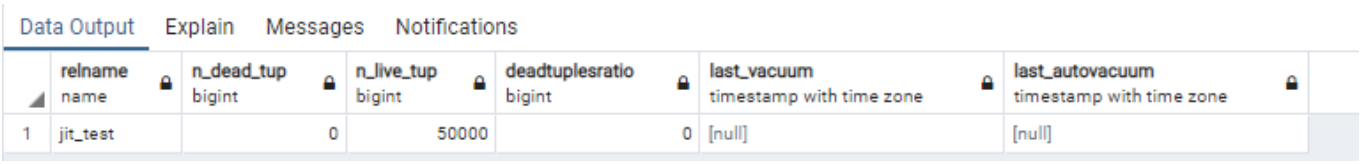

Dalibo's EXPLAIN ANALYZE visualizer:

This tool can be found at https://explain.dalibo.com/ . Again, you paste the raw execution plan and hit "Submit". The output is presented as a tree:




4 - check table bloats , in some cases high table bloats or fragmentation can affect or impact the execution plan , especially the vacuuming is not running as expected and will impact the table statistics which will impact the execution plan , run the below query on customer side to check the table bloats on the customer tables that are participated in the query :

SELECT relname, n_dead_tup, n_live_tup, (n_dead_tup/ n_live_tup) AS DeadTuplesRatio, last_vacuum, last_autovacuum FROM pg_catalog.pg_stat_all_tables WHERE relname in ('table1','table2') order by n_dead_tup DESC;



Data Output | Explain | Messages | Notifications

| | relname name | n_dead_tup bigint | n_live_tup bigint | deadtuplesratio bigint | last_vacuum timestamp with time zone | last_autovacuum timestamp with time zone |
|---|---|---|---|---|---|---|
| 1 | jit_test | 0 | 50000 | 0 | [null] | [null] |

if you noticed high ratio in deadtuplesratio column , you can run vacuum command on these tables ( refer to PostgreSQL documentaion for more details) :

PSQL> VACUUM (VERBOSE, ANALYZE) table_name;

and run the explain command to see if any changes on execution plan