

PostgreSQL Specific Query is Slow

Last updated by | Francisco Javier Pardillo Martin | Mar 7, 2023 at 7:03 AM PST

Please don't modify or move as this is part of GT , please contact haaqel@microsoft.com if needed

Specific Query is Slow

Monday, January 13, 2020
4:35 PM

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:

```
craig=# EXPLAIN ANALYZE SELECT last_name FROM employees where salary >= 50000;
                                QUERY PLAN
-----
Seq Scan on employees  (cost=0.00..35811.00 rows=1 width=6) (actual time=2.401..295.247 rows=1428 loops=1)
  Filter: (salary >= 50000)
  Total runtime: 295.379 ms
(3 rows)
```

Annotations: **Startup Cost** points to the cost values (0.00, 35811.00). **Max Time** points to the actual time (2.401). **Rows** points to the rows returned (1428).

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:

```
craig=# EXPLAIN ANALYZE SELECT last_name FROM employees where salary >= 50000;
                                QUERY PLAN
-----
Seq Scan on employees  (cost=0.00..35811.00 rows=1 width=6) (actual time=2.401..295.247 rows=1428 loops=1)
  Filter: (salary >= 50000)
  Total runtime: 295.379 ms
(3 rows)
```

Annotations: **Startup Cost** points to the cost values (0.00, 35811.00). **Max Time** points to the actual time (2.401). **Rows** points to the rows returned (1428).

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:

```
craig=# EXPLAIN ANALYZE SELECT last_name FROM employees where salary >= 50000;
                                QUERY PLAN
-----
Index Scan using idx_emps on employees  (cost=0.00..8.49 rows=1 width=6) (actual time=0.047..1.603 rows=1428 loops=1)
  Index Cond: (salary >= 50000)
  Total runtime: 1.771 ms
(3 rows)
```

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 :

```

1  EXPLAIN (ANALYZE, BUFFERS) SELECT count(*) FROM c WHERE pid = 1 AND cid > 200;
2
3
4  QUERY PLAN
5  -----
6  Aggregate  (cost=219.50..219.51 rows=1 width=8) (actual time=2.808..2.809 rows=1)
7  Buffers: shared read=45
8  I/O Timings: read=0.380
9  -> Seq Scan on c (cost=0.00..195.00 rows=9800 width=0) (actual time=0.083..0.083 rows=9800)
10  Filter: ((cid > 200) AND (pid = 1))
11  Rows Removed by Filter: 200
12  Buffers: shared read=45
13  I/O Timings: read=0.380
14
15  Planning:
16  Buffers: shared hit=48 read=29
17  I/O Timings: read=0.713
18  Planning Time: 1.673 ms
19  Execution Time: 3.096 ms
20  (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:

HTML	SOURCE	STATS					Add optimization
#	exclusive	inclusive	rows_x	rows	loops	node	
1.	0.859	2.809	↑ 1.0	1	1	→ Aggregate (cost=219.50..219.51 rows=1 width=8) (actual time=2.808..2.809 rows=1 loops=1) Buffers: shared read=45 I/O Timings: read=0.380 Planning: Buffers: shared hit=48 read=29 I/O Timings: read=0.713	
2.	1.950	1.950	↑ 1.0	9,800	1	→ Seq Scan on c (cost=0.00..195.00 rows=9,800 width=0) (actual time=0.083..0.083 rows=9,800 loops=1) Filter: ((cid > 200) AND (pid = 1)) Rows Removed by Filter: 200 Buffers: shared read=45 I/O Timings: read=0.380	
Planning time : 1.673 ms							
Execution time : 3.096 ms							

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:

Plan Raw Query Stats

Execution time: 3.1ms Planning time: **1.67ms** Triggers: N/A

time rows estimation

#1 AGGREGATE

#2 L SEQ SCAN

#1 AGGREGATE

#2 SEQ SCAN on c

Seq Scan Node finds relevant records by sequentially scanning the input record set. When reading from a table, Seq Scans (unlike Index Scans) perform a single read operation (only the table is read).

General IO & Buffers Output Workers Misc

Timing: **1.95ms** | 63%







Rows: 9,800 (Planned: 9,800)

Rows Removed by Filter: 200 | 2%

Cost: **195** (Total: 195)

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

More information

Please note that EXPLAIN ANALYZE will provide server side execution costs and timings, as explained here:

<https://www.postgresql.org/docs/current/using-explain.html> 

It's also important to realize that the cost only reflects things that the planner cares about. In particular, the cost does not consider the time spent transmitting result rows to the client, which could be an important factor in the real elapsed time

So, you could use EXPLAIN ANALYZE to get server side timings and compare with client side total execution time, difference between both execution times should be related to Network roundtrips, client side parsing time, ... so, not related to the Azure Database for PostgreSQL service itself, but other elements: Networking, Client side, ...

Created with Microsoft OneNote 2016.

How good have you found this content?

