Eyal Trabelsi - Apprently, you can debug your SQL queries

May 29, 2023

1 Apparently, you can "debug" your SQL queries

```
FROM users
SELECT subject,
       COUNT(*)
GROUP BY 1
WHERE twitter = 'donaldtrump';
```

- Missing records.
- Too many records.
- Duplications.
- Nulls.

2 Identify flaws in queries is tough

- Require skills and experience.
- Databases dont provide debuggers.
- Databases do provide execution plans.

3 Buzzword alert !!

4 Democratization of execution plans

5 Query Execution Flow

6 Let's explain Explain

- **explain**: show what the planner planned to do.
- **explain analyze**: what the planner did (**executes the query**)

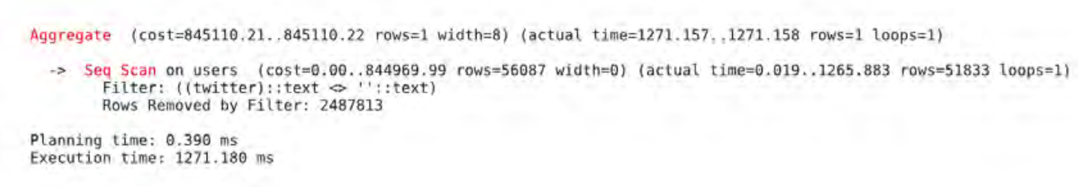
```
EXPLAIN [ ( option [, ...] ) ] statement
```

```
- ANALYZE [ boolean ]
- VERBOSE [ boolean ]
- COSTS [ boolean ]
- SETTINGS [ boolean ]
- BUFFERS [ boolean ]
- WAL [ boolean ]
- TIMING [ boolean ]
- SUMMARY [ boolean ]
- FORMAT { TEXT | XML | JSON | YAML }
```

Pro Tip : go over an execution plan at least once; similar across databases.

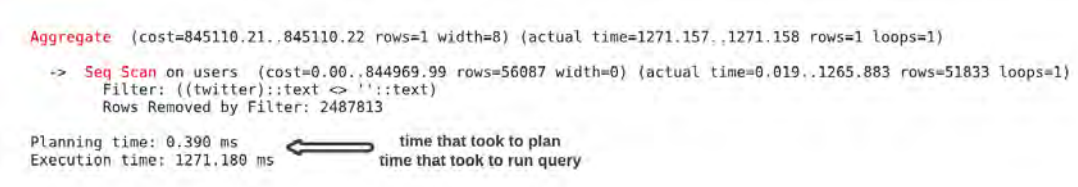
7 Explain Anatomy

```
EXPLAIN ANALYZE
SELECT COUNT(*) FROM users WHERE twitter != '';
```



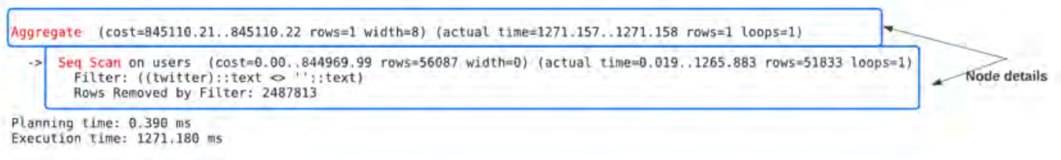
```
Aggregate  (cost=845110.21..845110.22 rows=1 width=8) (actual time=1271.157..1271.158 rows=1 loops=1)

  ->  Seq Scan on users  (cost=0.00..844969.99 rows=56087 width=0) (actual time=0.019..1265.883 rows=51833 loops=1)
        Filter: ((twitter)::text <> ''::text)
        Rows Removed by Filter: 2487813

Planning time: 0.390 ms
Execution time: 1271.180 ms
```

- Look crypted at first :(.
- It's longer than our query :((.
- Real-world execution plans are overwhelming:((.



```
Aggregate  (cost=845110.21..845110.22 rows=1 width=8) (actual time=1271.157..1271.158 rows=1 loops=1)

  ->  Seq Scan on users  (cost=0.00..844969.99 rows=56087 width=0) (actual time=0.019..1265.883 rows=51833 loops=1)
        Filter: ((twitter)::text <> ''::text)
        Rows Removed by Filter: 2487813

Planning time: 0.390 ms
Execution time: 1271.180 ms
```

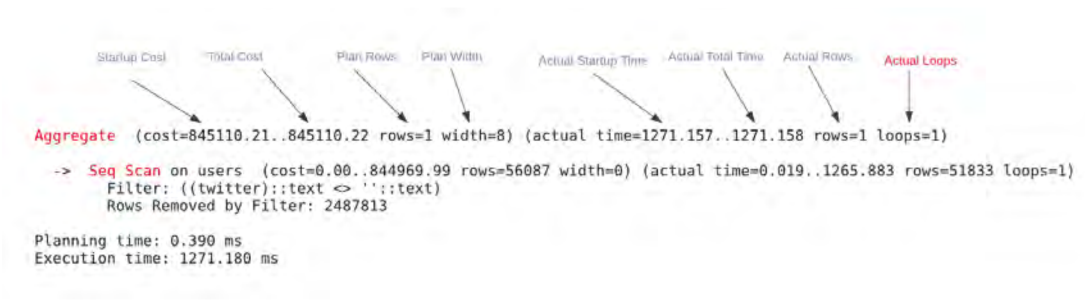
- Query execution took 1.27 seconds.
- Query planning took 0.4 millis.



```
Aggregate  (cost=845110.21..845110.22 rows=1 width=8) (actual time=1271.157..1271.158 rows=1 loops=1)
  ->  Seq Scan on users  (cost=0.00..844969.99 rows=56087 width=0) (actual time=0.019..1265.883 rows=51833 loops=1)
        Filter: ((twitter)::text <> ''::text)
        Rows Removed by Filter: 2487813
Planning time: 0.390 ms
Execution time: 1271.180 ms
```

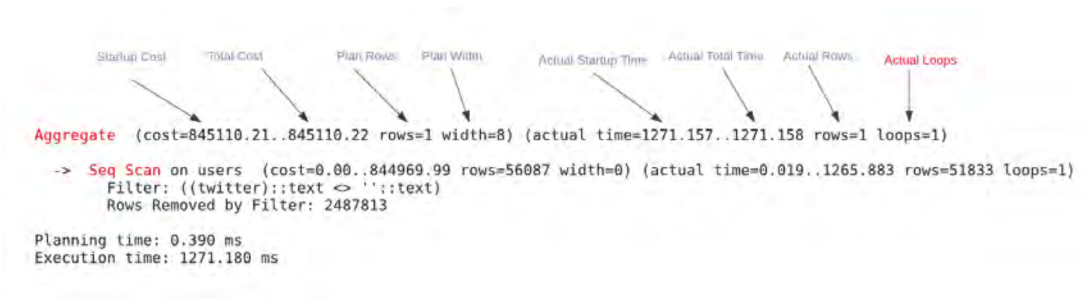
- Structured as inverse tree.
- Many operations: - 'Seq Scan', 'Values Scan', 'Sample Scan', 'Function Scan', 'CTEScan', 'Index Scan', 'Bitmap Heap Scan', 'Bitmap Index Scan', 'Index Only Scan','Subquery Scan', - 'Hash Join','Hash','Nested Loop', 'Merge Join', - GroupAggregate','Aggregate', 'HashAggregate', 'WindowAgg', - 'Gather', 'Gather Merge','Unique','Result', 'SetOp', ' 'Limit', 'Sort', 'materialize', 'LockRows', 'Append', 'Merge Append' etc.

Pro Tip : Cheat on your homework.



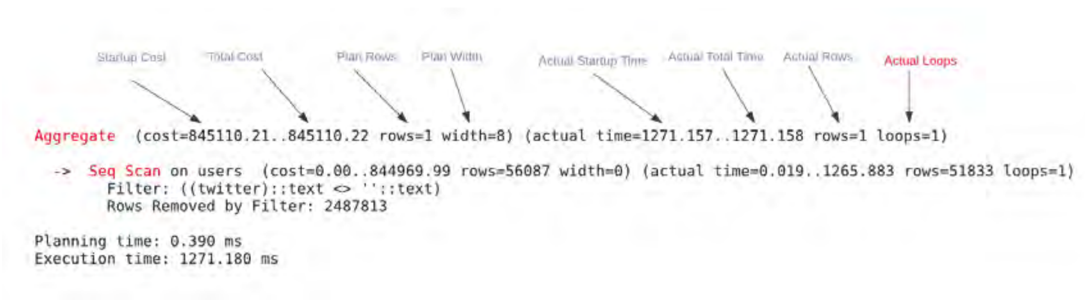
```
Aggregate  (cost=845110.21..845110.22 rows=1 width=8) (actual time=1271.157..1271.158 rows=1 loops=1)
  ->  Seq Scan on users  (cost=0.00..844969.99 rows=56087 width=0) (actual time=0.019..1265.883 rows=51833 loops=1)
        Filter: ((twitter)::text <> ''::text)
        Rows Removed by Filter: 2487813
Planning time: 0.390 ms
Execution time: 1271.180 ms
```

- **Plan Rows**: the estimated number of produced rows of Aggregate node is 1.
- **Actual Rows**: the actual number of produced rows of Aggregate node is 1 (per-loop average).
- **Plan Width**: the estimated average size of rows of Aggregate node is 8 bytes.



```
Aggregate  (cost=845110.21..845110.22 rows=1 width=8) (actual time=1271.157..1271.158 rows=1 loops=1)
  ->  Seq Scan on users  (cost=0.00..844969.99 rows=56087 width=0) (actual time=0.019..1265.883 rows=51833 loops=1)
        Filter: ((twitter)::text <> ''::text)
        Rows Removed by Filter: 2487813
Planning time: 0.390 ms
Execution time: 1271.180 ms
```

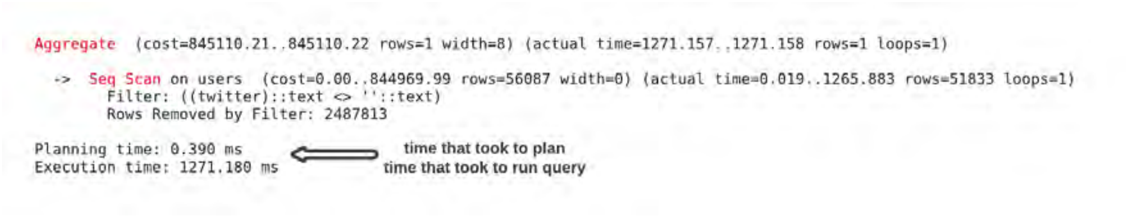
- **Startup Cost**: arbirary units that represent estimated time to return the first row of Aggregate is 845110 (total).
- **Total Cost**: arbirary units that represent estimated time to return all the rows of Aggregate is 845110 (total).
- **Actual Startup Time**: time to return the first row in ms of Aggregate is 1271.157 (total).
- **Actual Total Time**: time to return all the rows in ms of Aggregate is 1271.158 (per-loop average and total).



```
Aggregate  (cost=845110.21..845110.22 rows=1 width=8) (actual time=1271.157..1271.158 rows=1 loops=1)
  ->  Seq Scan on users  (cost=0.00..844969.99 rows=56087 width=0) (actual time=0.019..1265.883 rows=51833 loops=1)
        Filter: ((twitter)::text <> ''::text)
        Rows Removed by Filter: 2487813
Planning time: 0.390 ms
Execution time: 1271.180 ms
```

- **Actual Loops**: the number of loops the same node was executed is 1.
- To make the numbers comparable with the way the cost estimates are shown.
- To get the total time and rows, the actual time and rows need to be multiplied by loops values.

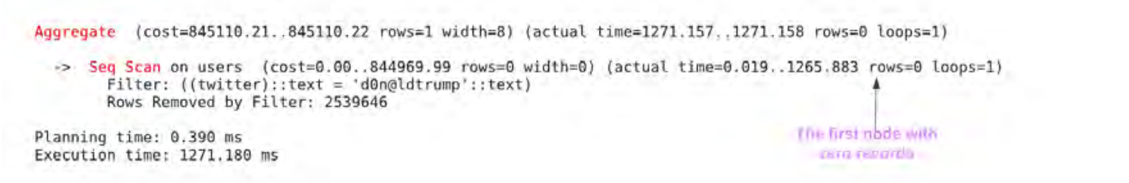
Pro Tip : every database has its wild card.



```
Aggregate  (cost=845110.21..845110.22 rows=1 width=8) (actual time=1271.157..1271.158 rows=1 loops=1)

  ->  Seq Scan on users  (cost=0.00..844969.99 rows=56087 width=0) (actual time=0.019..1265.883 rows=51833 loops=1)
        Filter: ((twitter)::text <> ''::text)
        Rows Removed by Filter: 2487813

Planning time: 0.390 ms
Execution time: 1271.180 ms
```

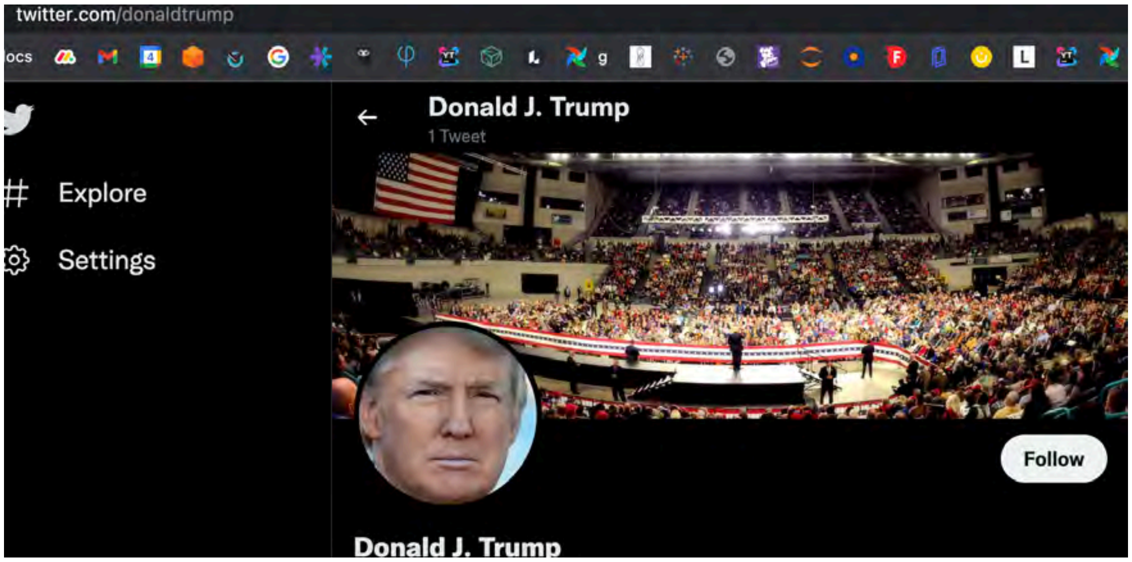
8 Example: Empty Results

```
EXPLAIN ANALYZE
SELECT COUNT(*) FROM users WHERE twitter = 'd0n@ldtrump';
```

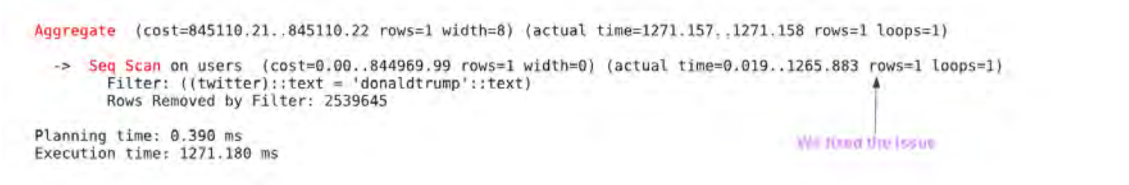


```
Aggregate  (cost=845110.21..845110.22 rows=1 width=8) (actual time=1271.157..1271.158 rows=0 loops=1)

  ->  Seq Scan on users  (cost=0.00..844969.99 rows=0 width=0) (actual time=0.019..1265.883 rows=0 loops=1)
        Filter: ((twitter)::text = 'd0n@ldtrump'::text)
        Rows Removed by Filter: 2539646

Planning time: 0.390 ms
Execution time: 1271.180 ms
```

- We perform a sequential scan on the users table.
- The scan filters out all rows using a Filter.
- in the first operation.
- so we dropped all the events at the donaldtrump predicate

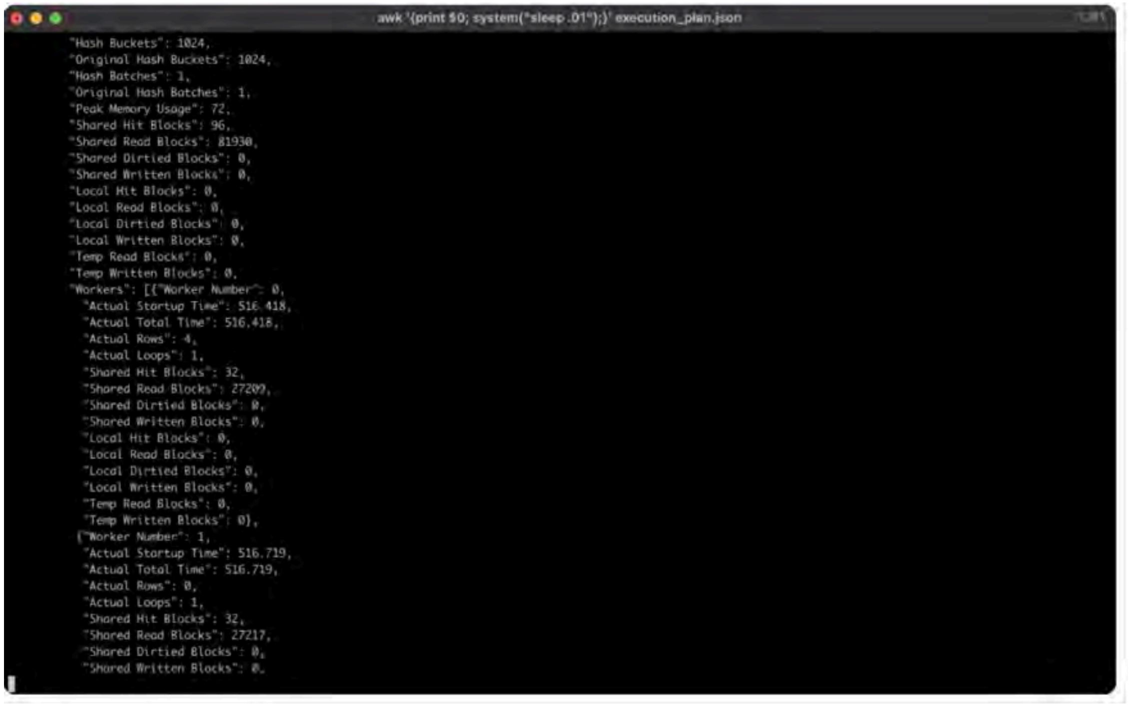


```
EXPLAIN ANALYZE
SELECT COUNT(*) FROM users WHERE twitter = 'donaldtrump';
```



```
Aggregate  (cost=845110.21..845110.22 rows=1 width=8) (actual time=1271.157..1271.158 rows=1 loops=1)

  ->  Seq Scan on users  (cost=0.00..844969.99 rows=1 width=0) (actual time=0.019..1265.883 rows=1 loops=1)
        Filter: ((twitter)::text = 'donaldtrump'::text)
        Rows Removed by Filter: 2539645

Planning time: 0.390 ms
Execution time: 1271.180 ms
```

- **Pro Tip** : in case we know a problem exists it is a productivity tool.
- **Pro Tip** : in case we don't know a problem exists it may protect us.



```
        "Hash Buckets": 1024,
        "Original Hash Buckets": 1024,
        "Hash Batches": 1,
        "Original Hash Batches": 1,
        "Peak Memory Usage": 72,
        "Shared Hit Blocks": 96,
        "Shared Read Blocks": 81930,
        "Shared Dirtied Blocks": 0,
        "Shared Written Blocks": 0,
        "Local Hit Blocks": 0,
        "Local Read Blocks": 0,
        "Local Dirtied Blocks": 0,
        "Local Written Blocks": 0,
        "Temp Read Blocks": 0,
        "Temp Written Blocks": 0,
        "Workers": [{"Worker Number": 0,
          "Actual Startup Time": 516.418,
          "Actual Total Time": 516.418,
          "Actual Rows": 4,
          "Actual Loops": 1,
          "Shared Hit Blocks": 32,
          "Shared Read Blocks": 27209,
          "Shared Dirtied Blocks": 0,
          "Shared Written Blocks": 0,
          "Local Hit Blocks": 0,
          "Local Read Blocks": 0,
          "Local Dirtied Blocks": 0,
          "Local Written Blocks": 0,
          "Temp Read Blocks": 0,
          "Temp Written Blocks": 0},
         {"Worker Number": 1,
          "Actual Startup Time": 516.719,
          "Actual Total Time": 516.719,
          "Actual Rows": 0,
          "Actual Loops": 1,
          "Shared Hit Blocks": 32,
          "Shared Read Blocks": 27217,
          "Shared Dirtied Blocks": 0,
          "Shared Written Blocks": 0,
```

9 Aren't there easier ways?!

- UI is nice.
- Hints why/where a particular issue orinated.
- Hints how how rewrite your queries.
- Saving traces for rainy day

9.1 QueryFlow

```
SELECT  title_id
FROM titles
INNER JOIN crew ON crew.title_id = titles.title_id
INNER JOIN people ON people.person_id = crew.person_id
WHERE crew.name = 'Rowan Atkinson'
```

```
SELECT *
FROM titles
INNER JOIN generes
ON generes.name like '%' || titles.genere_name || '%'
WHERE generes.safe_for_kids
```

```
+ Support multiple metrics/queries/engines.
+ Operations can be linkable with examples.
+ UI indicates the proportions of metrics and problematic operations.
```

```
- Not mature
- Very opinionated.
```

10 Optimistic Future

- Easy and intuitive.
- Integrated in IDEs
- Proactive