EXPLAIN: beyond the basics

Michael Christofides

Hi, I’m Michael

Half of the team behind pgMustard

Spent a lot of time looking into EXPLAIN

Background: product management, database tools

pgmustard.com/docs/explain

michael@pgmustard.com

michristofides

Picking up from other EXPLAIN talks

Not the basics*

1) Some of the less intuitive **arithmetic**

2) Some less well covered **issues**

* postgresql.org/docs/current/performance-tips
thoughtbot: reading EXPLAIN ANALYZE
YouTube: Josh Berkus Explaining EXPLAIN

Picking up from other EXPLAIN talks

Not the basics*

1) Arithmetic: **why** is this query slow?

2) Issues: **what** can we do about it?

* postgresql.org/docs/current/performance-tips
thoughtbot: reading EXPLAIN ANALYZE
YouTube: Josh Berkus Explaining EXPLAIN

Disclaimer: heavily doctored plans ahead, mistakes possible.

Arithmetic: loops

Many of the stats are a **per-loop average**

This includes costs, rows, timings

Watch out for rounding, especially to 0 rows

```
Nested Loop
(cost=0.84..209.82 rows=16 width=11)
(actual time=0.076..0.368 rows=86 loops=1)

  -> Index Only Scan using a on b
        (cost=0.42..4.58 rows=9 width=4)
        (actual time=0.013..0.019 rows=9 loops=1)

  -> Index Scan using x on y
        (cost=0.42..22.73 rows=7 width=15)
        (actual time=0.012..0.030 rows=10 loops=9)
```

Index Scan: 9 * 10 = 90 rows

Nested Loop: 86 rows

(Rounding not too bad here)

Arithmetic: threads

Costs, rows, and timings are also per-thread

Shown as loops

Threads = workers + 1 <- the leader

Tip: use VERBOSE

```
Parallel Seq Scan on table
 (cost=0.00..6772.21 rows=79521 width=22)
 (actual time=0.090..71.866 rows=63617 loops=3)

    Output: column1, column2, column3

    Worker 0:  actual time=0.111..66.325 rows=56225 loops=1

    Worker 1:  actual time=0.138..66.027 rows=58792 loops=1
```

```
Seq Scan: 63617 * 3 = 190851 rows

Leader:   190851 - 58792 - 56225
          = 75834 rows
```

Arithmetic: buffers

Buffer stats are a total, **not** per-loop

Inclusive of children

```
Nested Loop  (... loops=1)
  Buffers: shared hit=105
  ->  Index Only Scan using a on b  (... loops=1)
        Buffers: shared hit=4
  ->  Index Scan using x on y  (... loops=9)
        Buffers: shared hit=101
```

Nested Loop buffers:

105 - (101 + 4) = 0 blocks

Arithmetic: timings

Per-loop, per-thread

Inclusive of children

Per-node times can be tricky, even for tools

```
Nested Loop
 (cost=0.84..209.82 rows=16 width=11)
 (actual time=0.076..0.368 rows=86 loops=1)

  -> Index Only Scan using a on b
       (cost=0.42..4.58 rows=9 width=4)
       (actual time=0.013..0.019 rows=9 loops=1)

  -> Index Scan using x on y
       (cost=0.42..22.73 rows=7 width=15)
       (actual time=0.012..0.030 rows=10 loops=9)
```

```
Index Scan:  0.030 * 9 = 0.270 ms

Nested Loop: 0.368 - 0.270 - 0.019
             = 0.079 ms
```

```
WITH init AS (
 SELECT * FROM pg_sleep_for('100ms')
 UNION ALL
 SELECT * FROM pg_sleep_for('200ms')
)

(SELECT * FROM init LIMIT 1)
UNION ALL
(SELECT * FROM init);
```

Credit @felixge

```
Append (actual time=100.359..300.688 … )
  CTE init
    ->  Append (actual time=100.334..300.652 … )
          ->  Function Scan (actual time=100.333..100.335 … )
          ->  Function Scan (actual time=200.310..200.312 … )
  ->  Limit (actual time=100.358..100.359 … )
        ->  CTE Scan a (actual time=100.355..100.356 … )
  ->  CTE Scan b (actual time=0.001..200.322 … )

Execution Time: 300.789 ms
```

Some double-counting in this case.

Further reading:
flame-explain.com/docs/general/quirk-correction

Arithmetic: tools can help

```
eg    explain.depesz.com

      explain.dalibo.com

      explain.tensor.ru       <-  🇷🇺

      flame-explain.com       <-  fellow calculations nerd

      pgmustard.com           <-  👋
```

Summary: check the arithmetic

Watch out for loops and threads

Watch out for CTEs

Tools can help, if in doubt check two

Issues: quick recap of the basics

Seq Scans with large filters

Bad row estimates

Operations on disk rather than in memory

Issues: inefficient index scans

Looks out for lots of rows being filtered

Filters are **per-loop**

So again, watch out for rounding

```
-> Index Scan using x on y
    (cost=0.42..302502.05 rows=1708602 width=125)
    (actual time=172810.219..173876.540 rows=1000 loops=1)
      Index Cond: (id = another_id)
      Filter: (status = 1)
      Rows Removed by Filter: 3125626
```

Index efficiency: 1000/(1000+3125626) = 0.03%

Watch out for high loops

Issues: late filters

Row calculations important

Look for lots of rows being discarded

Filter earlier to avoid work

```
-> Sort (**rows=100** loops=1)
    -> Hash Join (**rows=44628** loops=1)
        -> ...
        -> ...
```

Discarded rows: 44628 - 100 = 44528 (99.8%)

Caveats: aggregation an exception

Issues: lots of data read

Requires BUFFERS

Lots of data being read for the amount returned

Can be a sign of bloat

Default block size: 8kB

```
-> Index Scan using x on y
    (cost=0.57..2.57 rows=1 width=8)
    (actual time=0.064..0.064 rows=1 loops=256753)
       Index Cond: (id = another_id)
       Filter: (status = 1)
       Buffers: shared hit=1146405 read=110636
```

Data read: (1146405 + 110636) * 8kB = 10GB

Data returned: 1 * 256753 * 8 bytes = 2MB

Caveats: width estimated, rows rounded

Issues: lossy bitmap scans

When bitmap would otherwise exceed work_mem

Point to a block rather than a row (Tuple Id)

Lossy blocks are a total (ie **not** per-loop)

```
-> Bitmap Heap Scan on table
     (cost=49153.29..4069724.27 rows=3105598 width=1106)
     (actual time=591.928..56472.895 rows=3853272 loops=1)
       Recheck Cond: (something > something_else)
       Rows Removed by Index Recheck: 5905323
       Heap Blocks: exact=14280 lossy=1951048
```

Lossy blocks: 1951048/(1951048+14280) = 99%

Extra rows read: 5.9 million

Issues: excessive heap fetches

Look out for heap fetches

When an index-only scan has to check the table

```
-> Index Only Scan using x on y
    (cost=0.42..28.52 rows=6 width=0)
    (actual time=0.007..0.037 rows=0 loops=87628)
        Index Cond: (a = (t.b))
        Heap Fetches: 19160
```

```
Time:              0.037 * 87628 = 3242 ms
Rows (max):          0.5 * 87628 = 43814
Heap fetches:      19160 / 43814 = 44% (at least)
```

Issues: planning time

At the end of the query plan

Not included in the execution time

Warning: not available via auto_explain

```
(...)
Planning Time: 27.844 ms
Execution Time: 11.162 ms
```

Planning proportion:

27.844/(27.844 + 11.162) = 71%

Issues: Just In Time compilation

At the end of the query plan

Included in execution time

On by default in PostgreSQL 12 and 13

```
Planning Time: 9.138 ms

JIT:
 Functions: 277
 Options: Inlining true, Optimization true, Expressions true,
          Deforming true
 Timing: Generation 31.602 ms, Inlining 253.114 ms, Optimization
         1498.268 ms, Emission 913.945 ms, Total 2696.929 ms

Execution Time: 5194.851 ms
```

JIT proportion:

2696.929/(9.138 + 5194.851) = 52%

```
-> Seq Scan on table
    (cost=0.00..3.57 rows=72 width=8)
    (actual time=2262.312..2262.343 rows=54 loops=1)
      Buffers: shared hit=3
```

Very suspicious actual start-up time from a JIT dominated plan.

Issues: triggers

At the end of the query plan

Total time across calls

Check foreign keys indexed

Before triggers vs after triggers

```
Planning Time: 0.227 ms

Trigger: RI_ConstraintTrigger_a_12345 on table
           time=83129.491 calls=2222623

Execution Time: 87645.739 ms
```

Trigger proportion:

83129.491/(0.227 + 87645.739) = 95%

Tip: use VERBOSE to see trigger names

Summary: keep rarer issues in mind

Check the end section first

Look out for filters, rechecks, lossy blocks, heap fetches, amount of data

Tools, mailing lists, and communities can help

Further reading

flame-explain.com/docs/general/quirk-correction

pgmustard.com/docs/explain

wiki.postgresql.org/wiki/Slow_Query_Questions

Thank you! Any questions?

michael@pgmustard.com

michristofides