EDB
Postgres® for the AI Generation

EXPLAIN Explained:
Making Sense of PostgreSQL Query Plans

April 25th, 2025. PGDay Pune

Jeevan Chalke

Principal Engineer, Engineering Manager

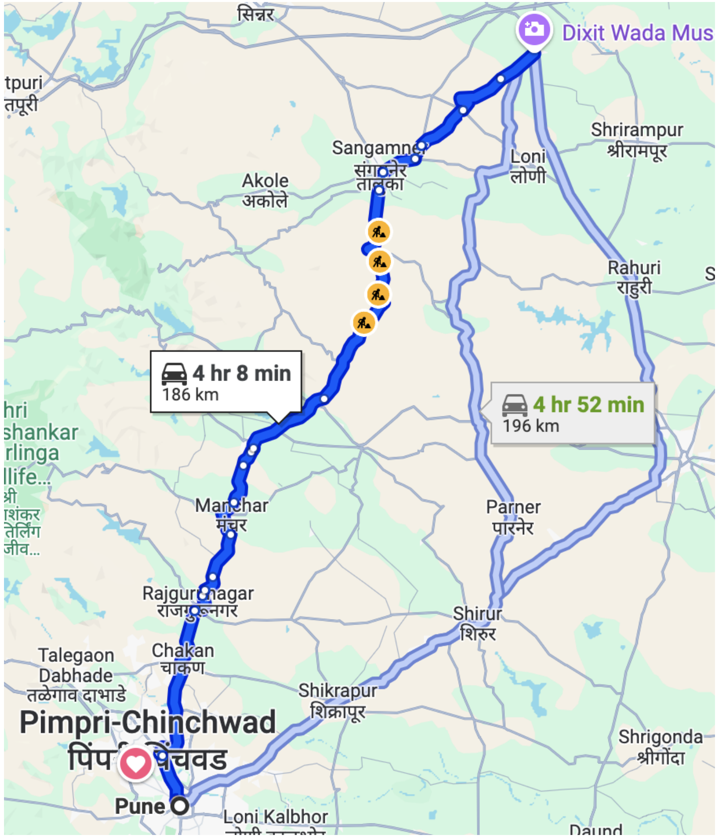
Jeevan.chalke@enterprisedb.com

Agenda

- Brief on optimizer/planner
- Paths and Plan
- EXPLAIN and EXPLAIN ANALYZE
  - Scans - seq, index
  - Joins
  - Sort
  - Grouping
  - Parallelism, Partitioning, Foreign Scan - Summary
- Options used with EXPLAIN

Brief on optimizer/planner

- Statistics
  - ANALYZE table
- Creates different Paths
  - The way query can be evaluated to give the desired output
- Gives metrics to them
- Chooses the cheapest path
- Converts Path to the Plan
- Executor then executes the query according to the chosen plan



EDB
Postgres for the AI Generation

Paths and Plan

- Costs
  - cpu_tuple_cost
  - cpu_operator_cost
  - …
- Rows
  - Estimated
  - Actual
- Width
- Time
  - Actual
  - Planning
  - Execution

EXPLAIN works with SELECT, INSERT, UPDATE, DELETE, VALUES, EXECUTE, DECLARE, CTAS, or CREATE MATERIALIZED VIEW queries

Basic EXPLAIN with seq scan, setup

```
# create table mytab1(a int, b int, c int);
# create table mytab2(x int, y int);
# insert into mytab1 select i, i%50, i%100 from generate_series(1, 1000000) i;
# insert into mytab2 select i, i%30 from generate_series(1, 1000) i;
# analyze mytab1;
# analyze mytab2;

# select relname, relpages, reltuples from pg_class where relname like 'mytab%';
 relname | relpages | reltuples
---------+----------+-----------
 mytab1  |     5406 |     1e+06
 mytab2  |        5 |      1000
(2 rows)

# select name, setting from pg_settings where name in ('cpu_tuple_cost', 'seq_page_cost');
      name      | setting
----------------+---------
 cpu_tuple_cost | 0.01
 seq_page_cost  | 1
(2 rows)
```

Basic EXPLAIN with seq scan

- Syntax
  - EXPLAIN [(option [, ...])] statement
  - EXPLAIN [ANALYZE] [VERBOSE] statement

```
# explain
  select * from mytab1;
                            QUERY PLAN
-------------------------------------------------------------------
 Seq Scan on mytab1  (cost=0.00..15406.00 rows=1000000 width=12)
(1 row)
```

- For a seq scan; the planner has to do two things
  - Read all the pages
  - Read all the tuples from each page
- So the cost will be
  - 5406 (relpages) * 1 (seq_page_cost) + 1000000 (reltuples) * 0.01 (cpu_tuple_cost) = **15406.00**

With WHERE clause and INDEX

```
# explain select * from mytab1 where a < 800000;
                           QUERY PLAN
-----------------------------------------------------------------
 Seq Scan on mytab1  (cost=0.00..17906.00 rows=798548 width=12)
   Filter: (a < 800000)
(2 rows)
```

- Filter is added in the output showing the condition; Number of rows reduced
- Cost increased: cpu_operator_cost = 0.0025 (+2500)

- Let's add an Index and run the same query

```
# create index myidx1 on mytab1(a);
```

```
# explain select * from mytab1 where a < 800000;
                           QUERY PLAN
-----------------------------------------------------------------
 Seq Scan on mytab1  (cost=0.00..17906.00 rows=798548 width=12)
   Filter: (a < 800000)
(2 rows)
```

Same result?

Let's understand
EXPLAIN ANALYZE
first…

EXPLAIN ANALYZE

- Gives the EXPLAIN plan
- Executes the query; discards the output
- Gives actual timing and row details
- Gives a few more finer details; like loops and allows using BUFFERS option
- Provides summary showing planning and execution time
- Risky with DML commands
  - For example, an EXPLAIN ANALYZE DELETE… will actually delete the rows from a table

With an Index and EXPLAIN ANALYZE

```
# explain (analyze) select * from mytab1 where a < 800000;
                                      QUERY PLAN
------------------------------------------------------------------------------------------
 Seq Scan on mytab1  (cost=0.00..17906.00 rows=799624 width=12) (actual
time=0.019..134.772 rows=799999 loops=1)
   Filter: (a < 800000)
   Rows Removed by Filter: 200001
 Planning Time: 0.085 ms
 Execution Time: 170.264 ms
(5 rows)

# set enable_seqscan to off; -- Forces Index scan
# explain (analyze) select * from mytab1 where a < 800000;
                                      QUERY PLAN
------------------------------------------------------------------------------------------
 Index Scan using myidx1 on mytab1  (cost=0.42..27099.84 rows=799624 width=12) (actual
time=0.045..251.022 rows=799999 loops=1)
   Index Cond: (a < 800000)
 Planning Time: 0.083 ms
 Execution Time: 285.570 ms
(4 rows)
```

With an Index and EXPLAIN ANALYZE (continued…)

```
# set enable_seqscan to on;

# explain (analyze) select * from mytab1 where a > 800000;
                                      QUERY PLAN
-----------------------------------------------------------------------------------------
 Index Scan using myidx1 on mytab1  (cost=0.42..6797.99 rows=200375 width=12) (actual
time=0.070..62.023 rows=200000 loops=1)
   Index Cond: (a > 800000)
 Planning Time: 0.177 ms
 Execution Time: 69.694 ms
(4 rows)
```

- So, having an index doesn’t always improve the performance.
- But it does in most of the cases and you have to figure it out by looking at your data.

With an Index, and specific column

```
# explain (analyze) select a from mytab1 where a > 800000;
                                      QUERY PLAN
-----------------------------------------------------------------------------------------
 Index Only Scan using myidx1 on mytab1  (cost=0.42..5710.99 rows=200375 width=4) (actual
time=0.051..31.107 rows=200000 loops=1)
   Index Cond: (a > 800000)
   Heap Fetches: 75
 Planning Time: 0.091 ms
 Execution Time: 38.271 ms
(5 rows)
```

- Index Scan is changed to Index Only Scan
- Required column is part of an index itself so no need to scan the actual table

- Available planner options
  - enable_seqscan, enable_indexscan, enable_indexonlyscan, enable_bitmapscan

JOINs

```
# explain (analyze, costs off) select a,x from mytab1 t1 left join mytab2 t2 on (t1.a=t2.x);
                                   QUERY PLAN
--------------------------------------------------------------------------------
 Hash Left Join (actual time=0.316..281.785 rows=1000000 loops=1)
   Hash Cond: (t1.a = t2.x)
   ->  Seq Scan on mytab1 t1 (actual time=0.014..88.353 rows=1000000 loops=1)
   ->  Hash (actual time=0.288..0.290 rows=1000 loops=1)
         Buckets: 1024  Batches: 1  Memory Usage: 44kB
         ->  Seq Scan on mytab2 t2 (actual time=0.013..0.132 rows=1000 loops=1)
 Planning Time: 0.572 ms
 Execution Time: 321.697 ms

# explain (costs off)select a from mytab1 t1 where a in (select x from mytab2);
                   QUERY PLAN
-------------------------------------------------
 Merge Semi Join
   Merge Cond: (t1.a = mytab2.x)
   ->  Index Only Scan using myidx1 on mytab1 t1
   ->  Sort
         Sort Key: mytab2.x
         ->  Seq Scan on mytab2
```

JOINs Summary

- Merge Join
  - Sorts, and then merges
  - Faster for bigger data-set
- Hash Join
  - Works with equality constraints
  - Faster provided we have enough memory
  - Mostly used where one table is smaller
- Nested Loop
  - For smaller data-set
  - Mostly used for cross joins
- Join Types: Left/Right/Full/Anti/Semi/Inner
- Planner parameters
  - enable_hashjoin, enable_mergejoin, enable_nestloop

SORT

```
# explain (analyze, buffers) select * from mytab1 t1 where a < 200000 order by b;
                                      QUERY PLAN
------------------------------------------------------------------------------------------
 Sort  (cost=28089.70..28594.44 rows=201894 width=12) (actual time=109.382..143.465
rows=199999 loops=1)
   Sort Key: b
   Sort Method: external merge  Disk: 4320kB
   Buffers: shared hit=1631, temp read=540 written=542
   ->  Index Scan using myidx1 on mytab1 t1  (cost=0.42..6848.57 rows=201894 width=12)
(actual time=0.046..61.180 rows=199999 loops=1)
         Index Cond: (a < 200000)
         Buffers: shared hit=1631
 Planning Time: 0.101 ms
 Execution Time: 156.481 ms
(9 rows)
```

- Sort Key
- Sort Method is external merge sort
- Buffers option, temp read/written in blocks (8K)

SORT (Continued...)

```
# set work_mem to '16MB';
# explain (analyze, buffers, settings) select * from mytab1 t1 where a < 200000 order by b;
                                                QUERY PLAN
-------------------------------------------------------------------------------------------
 Sort  (cost=24638.70..25143.44 rows=201894 width=12) (actual time=111.089..137.246
rows=199999 loops=1)
   Sort Key: b
   Sort Method: quicksort  Memory: 12394kB
   Buffers: shared read=1631
   ->  Index Scan using myidx1 on mytab1 t1  (cost=0.42..6848.57 rows=201894 width=12)
(actual time=3.728..82.659 rows=199999 loops=1)
         Index Cond: (a < 200000)
         Buffers: shared read=1631
 Settings: enable_mergejoin = 'off', work_mem = '16MB'
 Planning Time: 0.118 ms
 Execution Time: 152.970 ms
(10 rows)
```

- Sort Method is quicksort due to increased work_mem sorting now done in-memory.

GROUPING

```
# explain (analyze, memory)
  select count(*) from mytab2 t1 group by y having sum(x) > 17000;
                                     QUERY PLAN
-----------------------------------------------------------------------------------------
 HashAggregate  (cost=22.50..22.88 rows=10 width=12) (actual time=0.389..0.394 rows=5
loops=1)
   Group Key: y
   Filter: (sum(x) > 17000)
   Batches: 1  Memory Usage: 24kB
   Rows Removed by Filter: 25
   ->  Seq Scan on mytab2 t1  (cost=0.00..15.00 rows=1000 width=8) (actual
time=0.015..0.110 rows=1000 loops=1)
 Planning:
   Memory: used=14kB  allocated=16kB
 Planning Time: 0.132 ms
 Execution Time: 0.435 ms
```

- Group keys are displayed, and Filter shows the Having clause
- Parameter enable_hashagg can be used to disable HashAggregate

GROUPING with Parallelism

```
# explain verbose select count(*) from mytab3 t1 group by y having sum(x) > 17000;
                                       QUERY PLAN
------------------------------------------------------------------------------------------
 Finalize GroupAggregate  (cost=132749.06..132756.89 rows=10 width=12)
   Output: count(*), y
   Group Key: t1.y
   Filter: (sum(t1.x) > 17000)
   ->  Gather Merge  (cost=132749.06..132756.06 rows=60 width=20)
         Output: y, (PARTIAL count(*)), (PARTIAL sum(x))
         Workers Planned: 2
         ->  Sort  (cost=131749.04..131749.11 rows=30 width=20)
               Output: y, (PARTIAL count(*)), (PARTIAL sum(x))
               Sort Key: t1.y
               ->  Partial HashAggregate  (cost=131748.00..131748.30 rows=30 width=20)
                     Output: y, PARTIAL count(*), PARTIAL sum(x)
                     Group Key: t1.y
                     ->  Parallel Seq Scan on public.mytab3 t1  (cost=0.00..94248.00
rows=5000000 width=8)
                           Output: x, y
```

Parallelism, Partitioning, Foreign Scan - Summary

- Parallelism
  - Look for parameters related to Parallelism
    - parallel_leader_participation, parallel_setup_cost, parallel_tuple_cost, max_parallel_workers, max_parallel_workers_per_gather, etc.
  - New nodes and details in EXPLAIN
    - Gather or Gather Merge, Workers Planned
- Partitioning
  - Look for parameters related to Partitioning
    - enable_partition_pruning, enable_partitionwise_aggregate, enable_partitionwise_join
  - New node Append
- Foreign Scan
  - Details in EXPLAIN
    - Relations, Remote SQL

Options Supported with EXPLAIN

- **ANALYZE, VERBOSE, COSTS, SETTINGS, BUFFERS, MEMORY**

- **GENERIC_PLAN** [boolean] => Allow the statement to contain parameter placeholders like $1, but still generate a generic plan. Cannot be used together with ANALYZE
- **SERIALIZE** [{ NONE | TEXT | BINARY }] => Include information on the cost of serializing the query's output data. Used only with ANALYZE, Default TEXT
- **WAL** [boolean] => Include information on WAL record generation. Used only with ANALYZE
- **TIMING** [boolean] => Show actual times. Used only with ANALYZE, default TRUE
- **SUMMARY** [boolean] => Gives summary on planning and execution times
- **FORMAT** { TEXT | XML | JSON | YAML } => Output display format. Default TEXT

Format

```
# explain (format json, verbose off,
    analyze, summary, timing off, costs off,
    buffers off, wal)
  select * from mytab1;
            QUERY PLAN
----------------------------------
 [                               +
   {                             +
     "Plan": {                   +
       "Node Type": "Seq Scan",  +
       "Parallel Aware": false,  +
       "Async Capable": false,   +
       "Relation Name": "mytab1",+
       "Alias": "mytab1",        +
       "Actual Rows": 1000001,   +
       "Actual Loops": 1,        +
       "WAL Records": 0,         +
       "WAL FPI": 0,             +
       "WAL Bytes": 0            +
     },                          +
     "Planning Time": 0.056,     +
     "Triggers": [               +
     ],                          +
     "Execution Time": 87.329    +
   }                             +
 ]
```

```
# explain (format xml, verbose off,
    analyze, summary, timing off, costs off,
    buffers off, wal)
  select * from mytab1;
                        QUERY PLAN
----------------------------------------------------------
 <explain xmlns="http://www.postgresql.org/2009/explain">+
   <Query>                                               +
     <Plan>                                              +
       <Node-Type>Seq Scan</Node-Type>                   +
       <Parallel-Aware>false</Parallel-Aware>            +
       <Async-Capable>false</Async-Capable>              +
       <Relation-Name>mytab1</Relation-Name>             +
       <Alias>mytab1</Alias>                             +
       <Actual-Rows>1000001</Actual-Rows>                +
       <Actual-Loops>1</Actual-Loops>                    +
       <WAL-Records>0</WAL-Records>                      +
       <WAL-FPI>0</WAL-FPI>                              +
       <WAL-Bytes>0</WAL-Bytes>                          +
     </Plan>                                             +
     <Planning-Time>0.105</Planning-Time>                +
     <Triggers>                                          +
     </Triggers>                                         +
     <Execution-Time>87.991</Execution-Time>             +
   </Query>                                              +
 </explain>
```

EDB
Postgres® for the AI Generation

THANK YOU

References

PostgreSQL Documentation

- https://www.postgresql.org/docs/17/sql-explain.html
- https://www.postgresql.org/docs/current/using-explain.html

My same talk (PGConf India 2019)

- https://pgconf.in/files/presentations/2019/02-0102-ExplainByJeevanChalkeEDB.pdf

Query Planning Gone Wrong by Robert Haas

- http://rhaas.blogspot.com/2013/05/query-planning-gone-wrong.html

Other web links

- https://momjian.us/main/writings/pgsql/optimizer.pdf
- https://neon.tech/postgresql/postgresql-tutorial/postgresql-explain
- https://wiki.postgresql.org/images/4/45/Explaining_EXPLAIN.pdf