PG CONFERENCE GERMANY 2025

EXPLAIN Explained

Understanding the PostgreSQL planner better

Divya Sharma

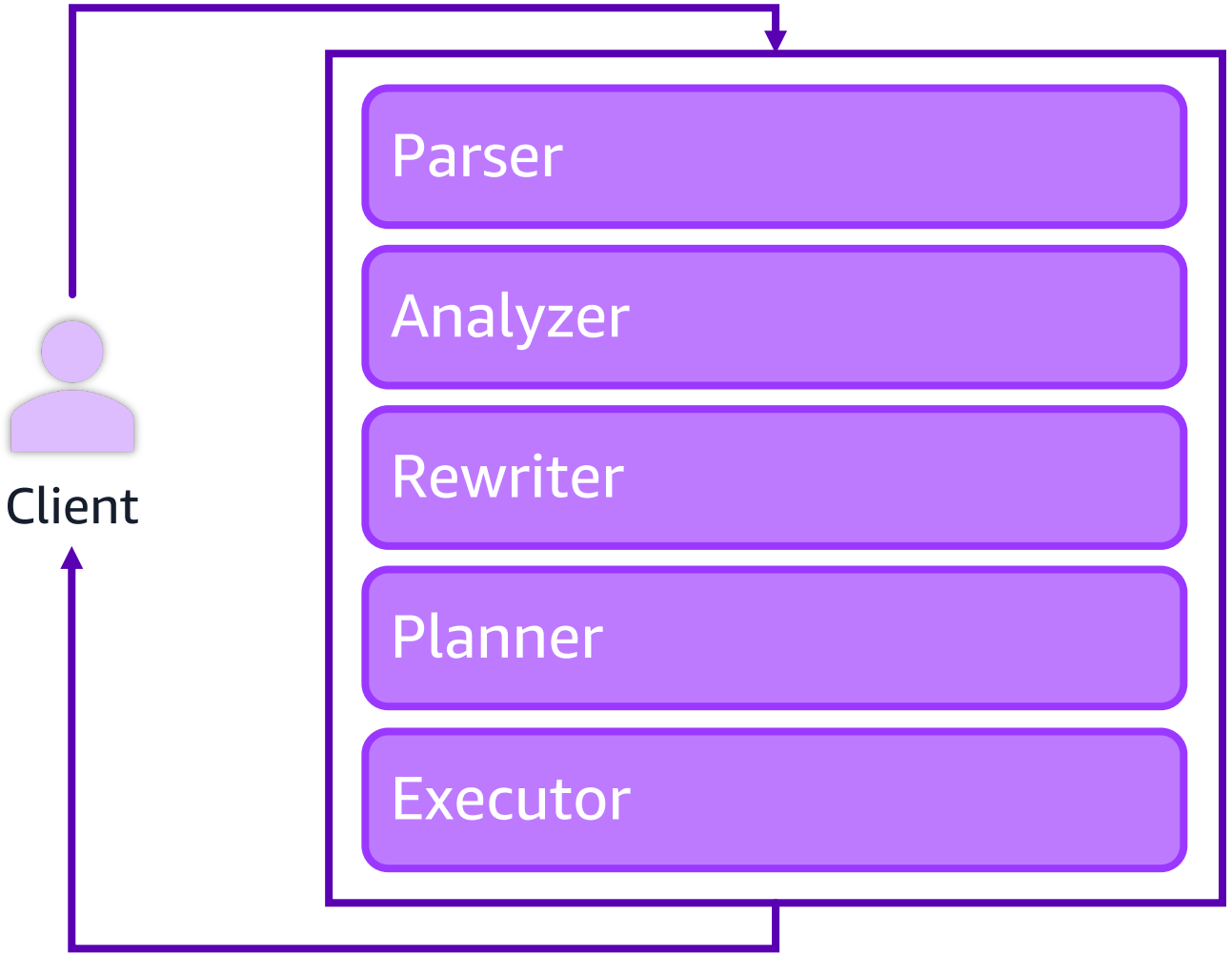
Sr. RDS PostgreSQL SA

Agenda

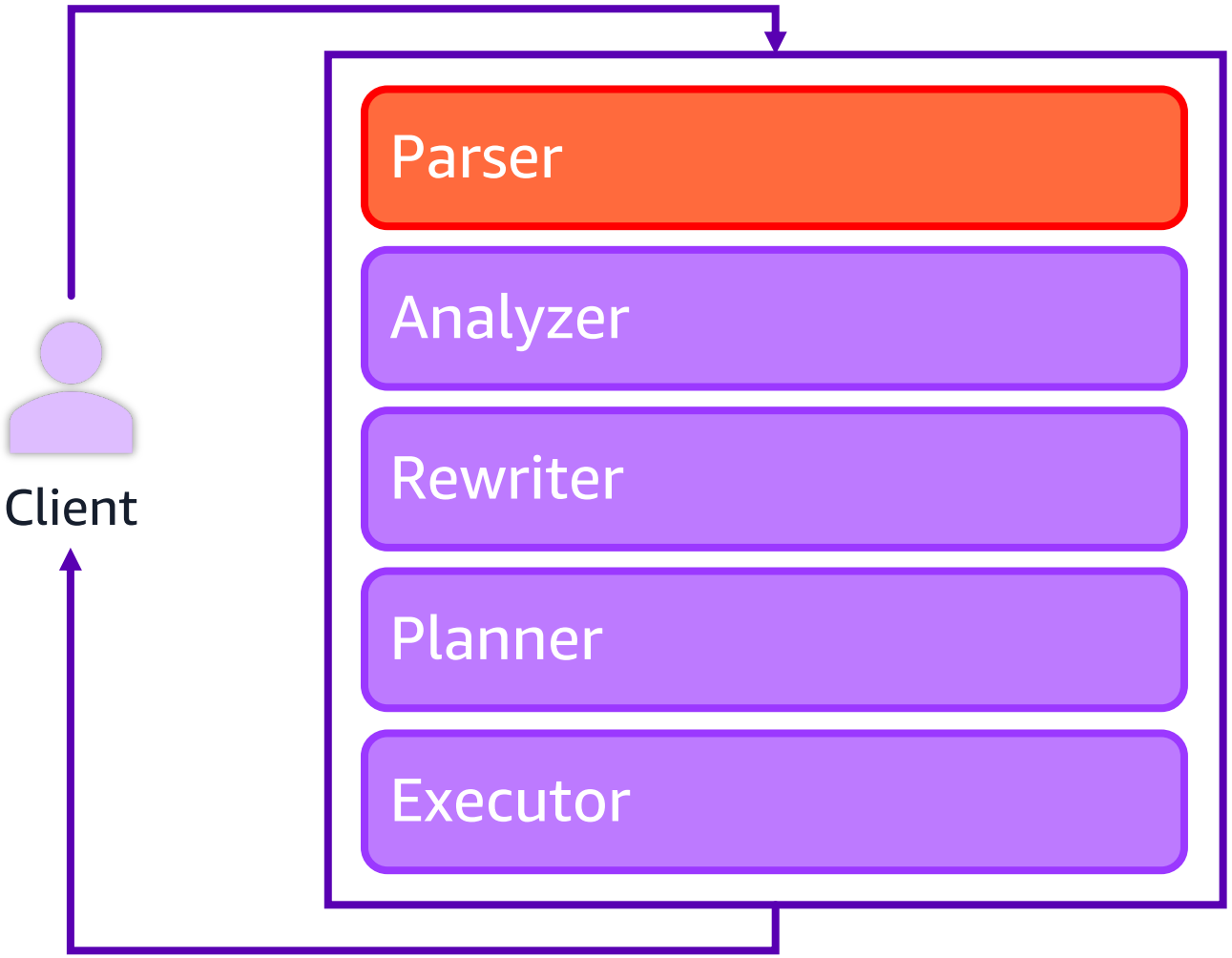
- Query Processing in PostgreSQL
- Viewing EXPLAIN plans
- Query planner decision factors
- Cost of the plan
- The “Analyze” command
- Query Tuning Cycle

Query Processing in PostgreSQL

Query Processing

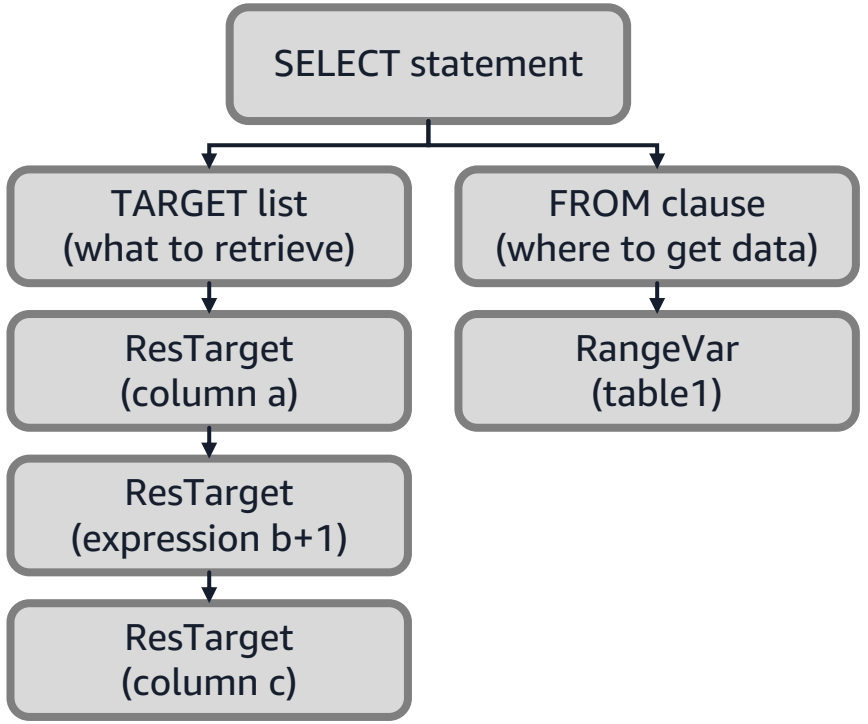


Query Processing

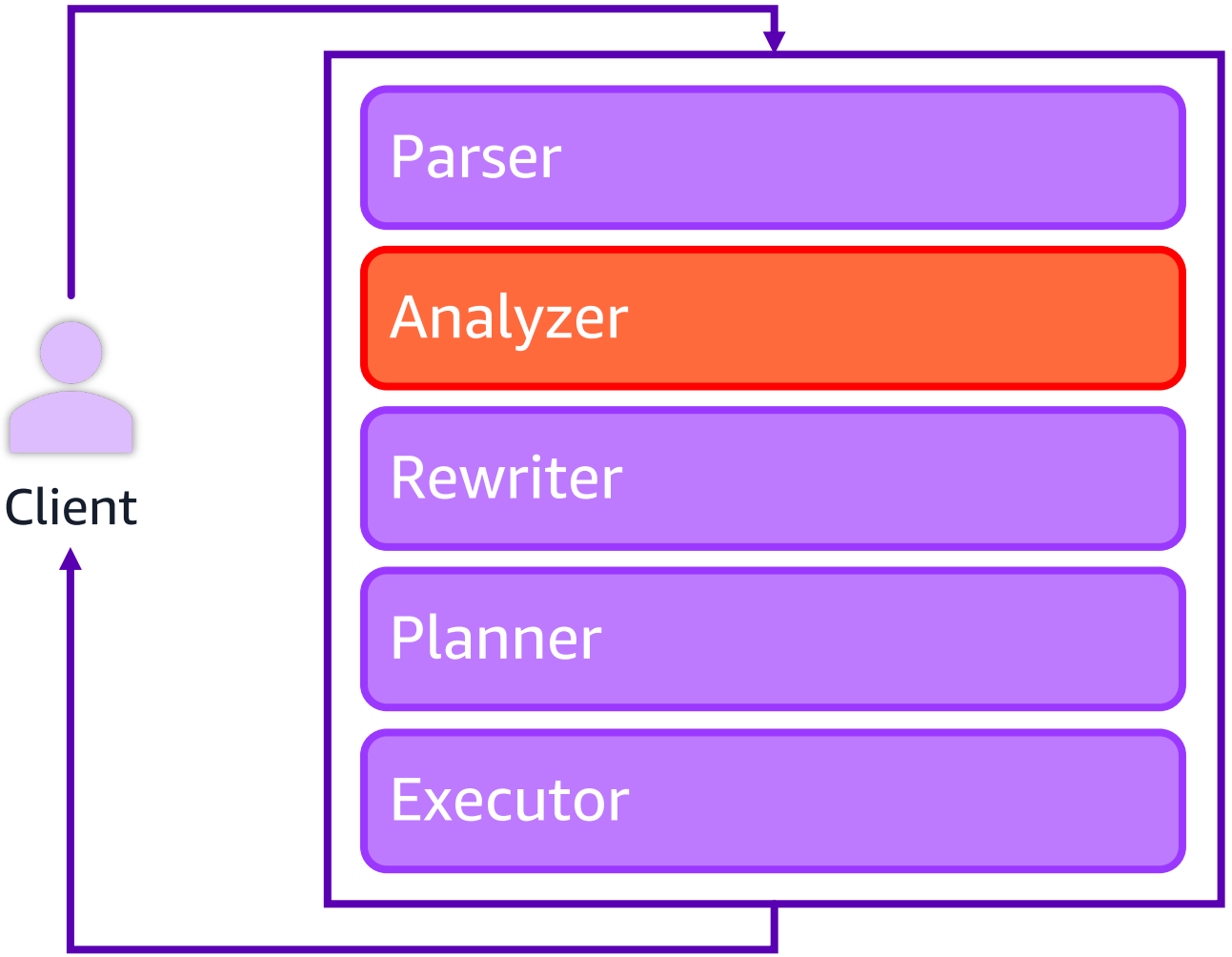


Query (example):

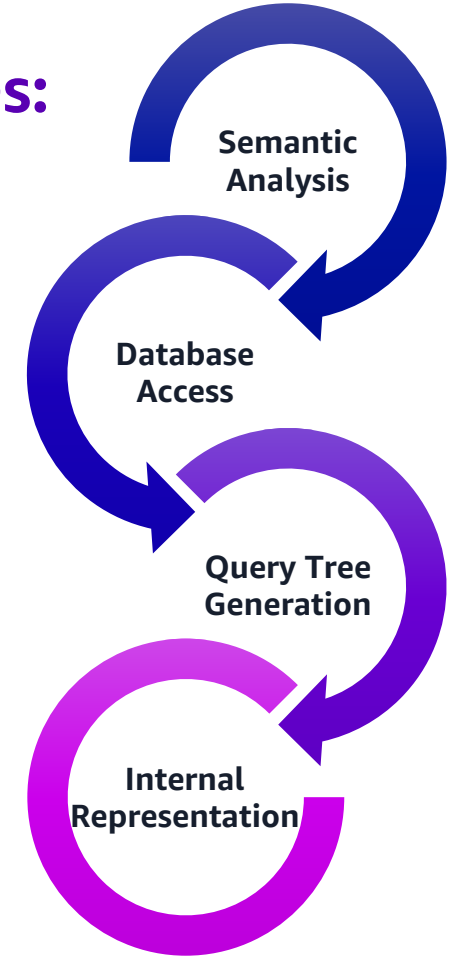
```
SELECT a, b + 1, c
FROM table1;
```



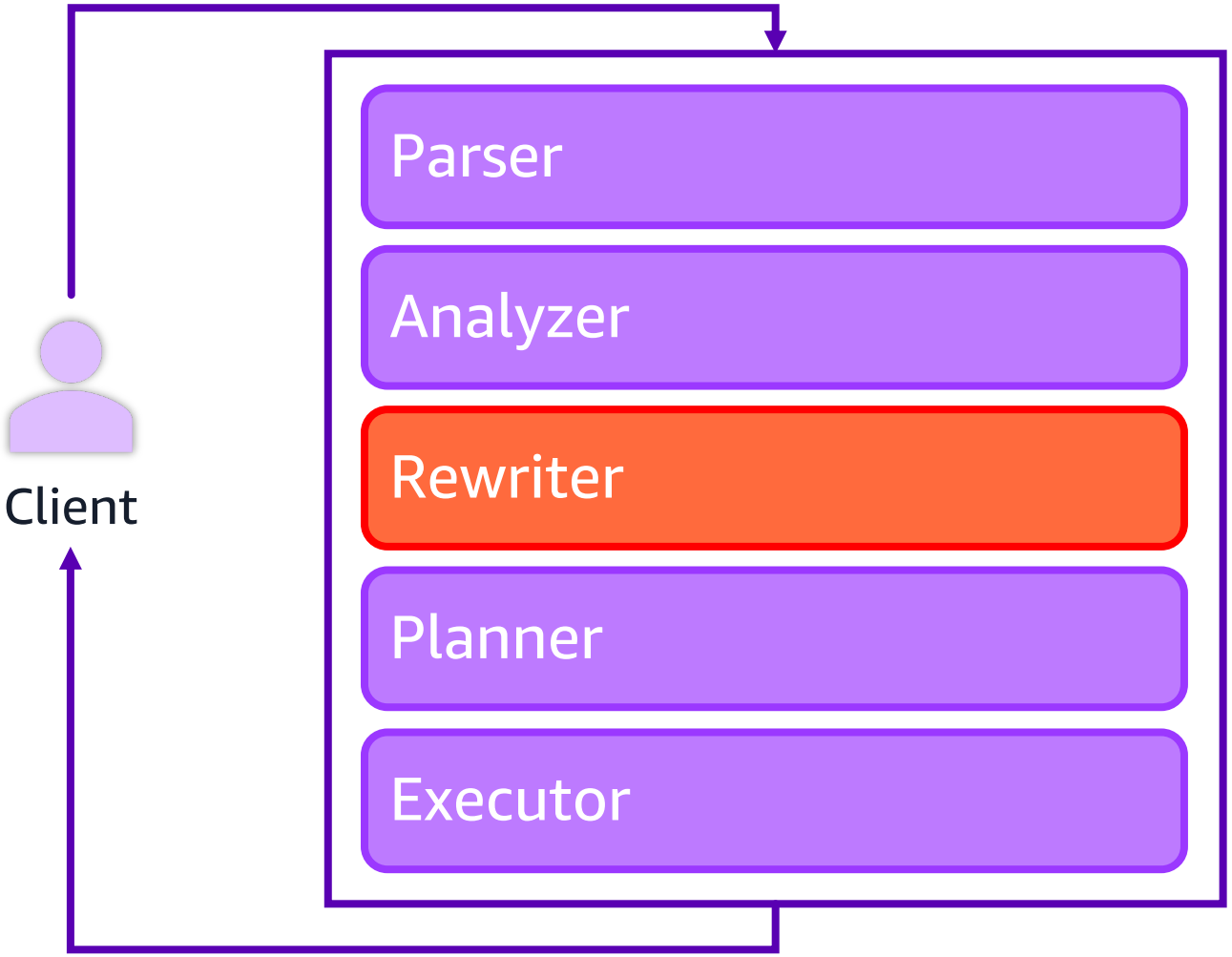
Query Processing



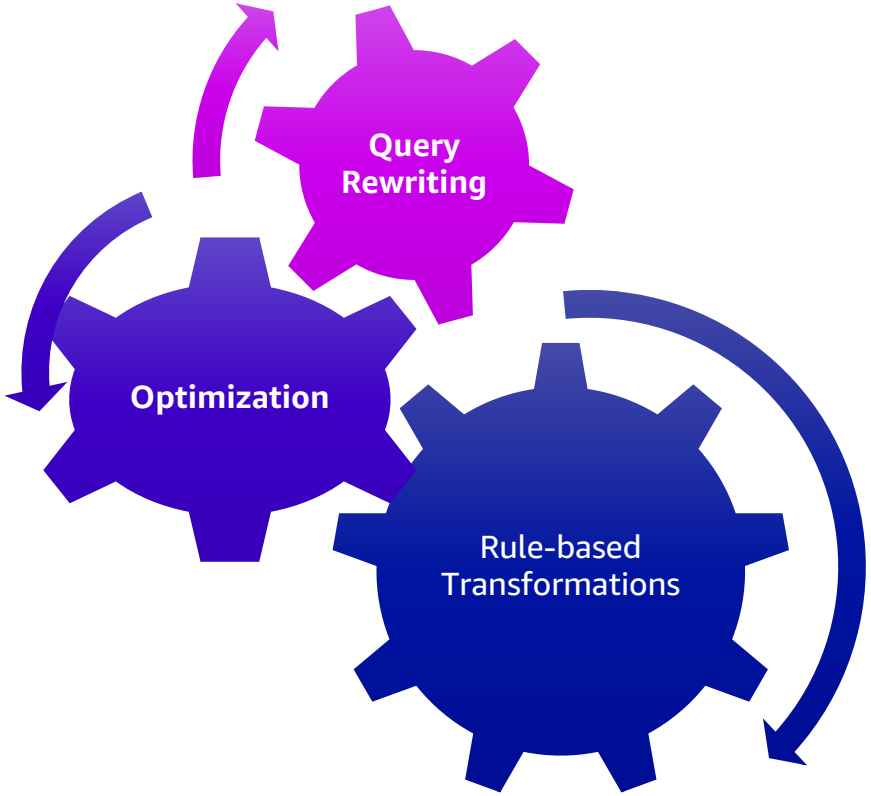
Key responsibilities:



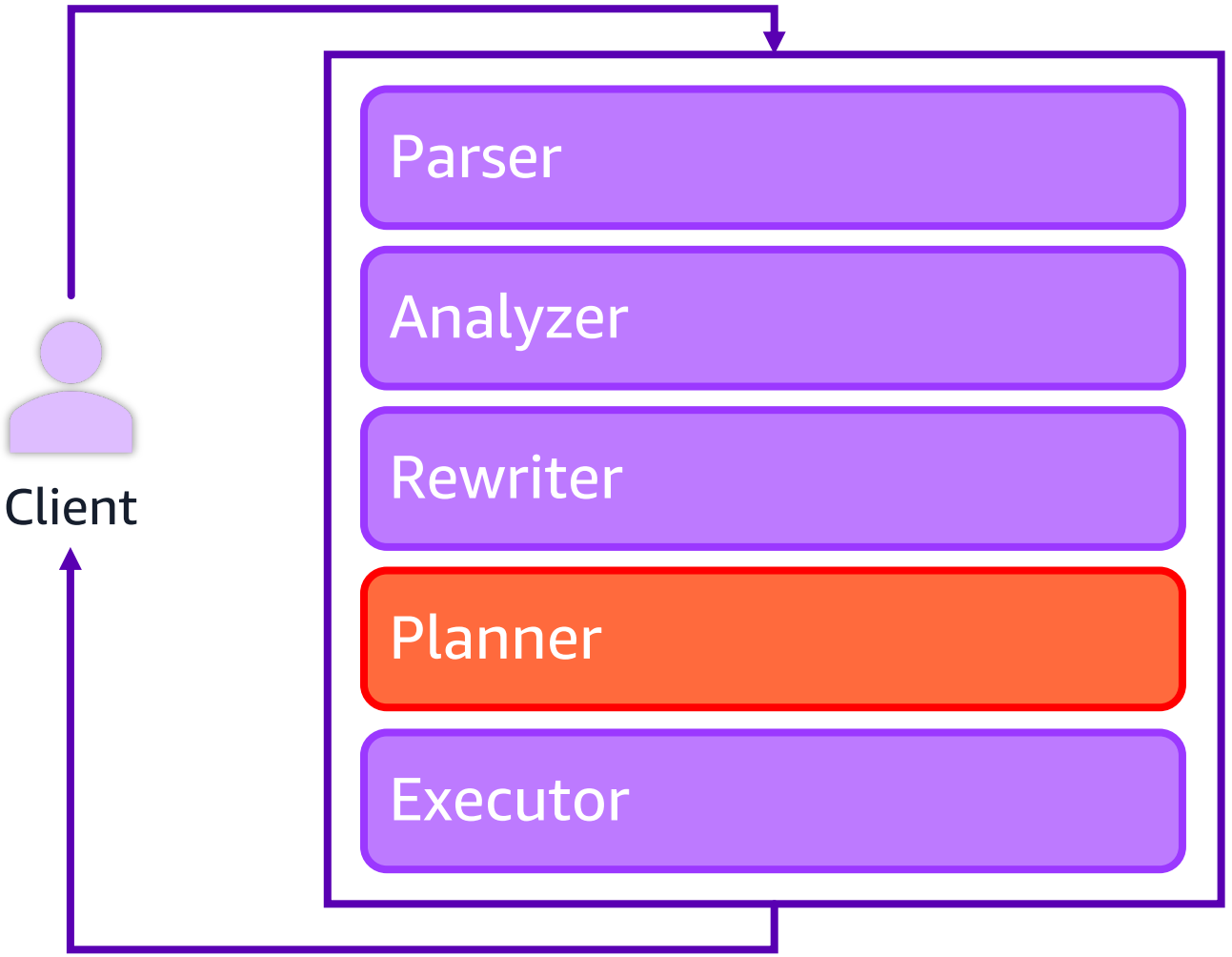
Query Processing



Key responsibilities:



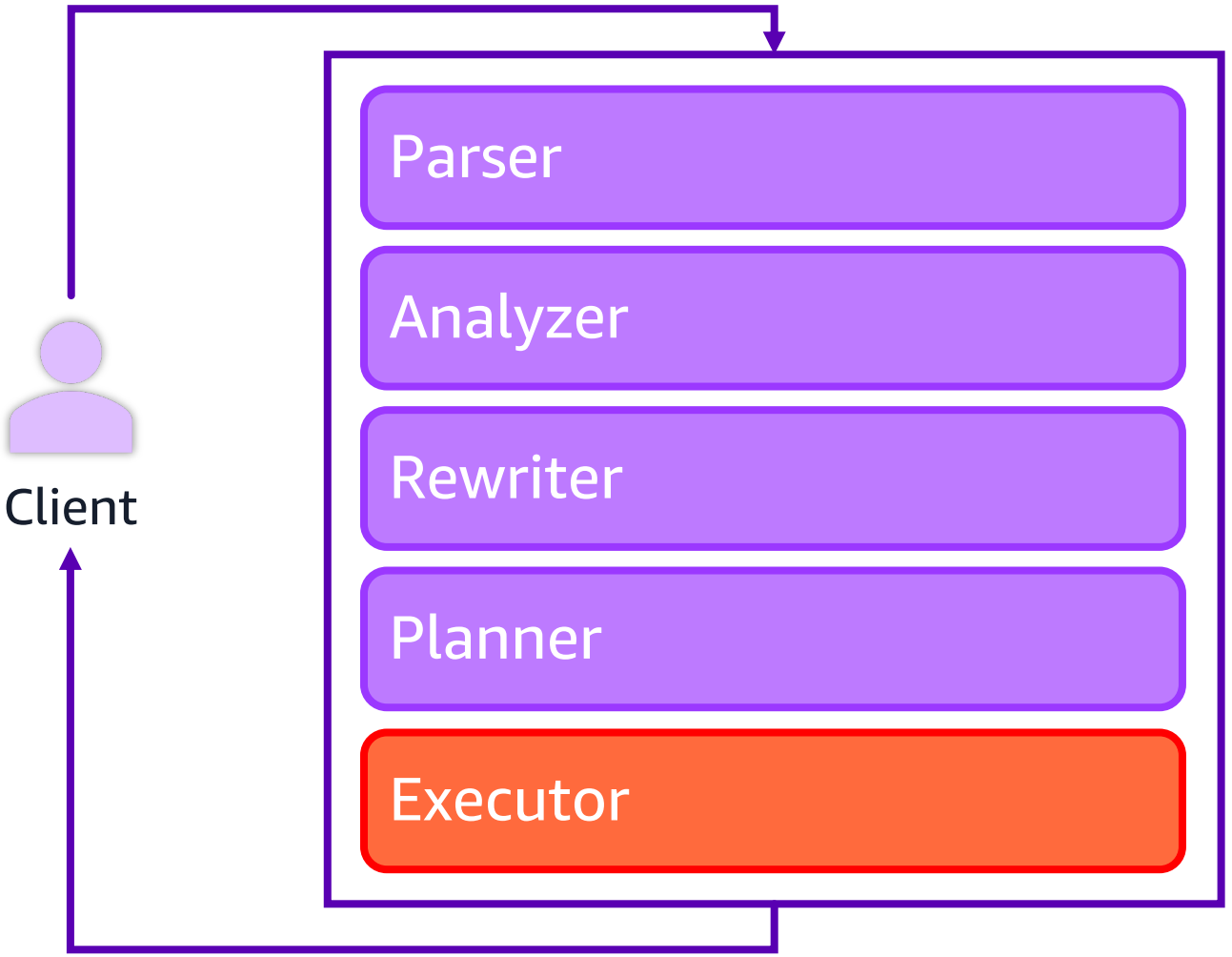
Query Processing



Key responsibilities:

- Cost estimation
- Path identification
- Plan selection
- Plan generation

Query Processing



Key responsibilities:

- Execution plan interpretation
- Data retrieval
- Result generation
- Result delivery

Viewing EXPLAIN plans

Viewing EXPLAIN plans

1. **Explain (with options)** – you have to run manually
2. **auto.explain module** – can automatically log plans for you in the error log

EXPLAIN options

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

where *option* can be one of:

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

SERIALIZE [{ NONE | TEXT | BINARY }]
MEMORY [*boolean*]

EXPLAIN options

```
postgres=> EXPLAIN ( ANALYZE, BUFFERS ) select * from foo where i<2432318;
                                  QUERY PLAN
------------------------------------------------------------------------------
--
 Seq Scan on foo  (cost=0.00..6250.00 rows=299970 width=37)
 (actual time=0.007..32.372 rows=300000 loops=1)
    Filter: (i < 2432318)
    Buffers: shared hit=2500
 Planning Time: 0.041 ms
 Execution Time: 46.293 ms
(5 rows)
```

EXPLAIN options – WAL

```
postgres=> select * from t;
 id | price
----+-------
  1 | 10.00
  2 |
(2 rows)
```

```
postgres=> EXPLAIN (ANALYZE, WAL, BUFFERS) update t set price = 20 where id=2;
                                                  QUERY PLAN
-------------------------------------------------------------------------------------------------------------
 Update on t  (cost=0.00..1.02 rows=0 width=0) (actual time=0.804..0.805 rows=0 loops=1)
   Buffers: shared hit=3 read=1 dirtied=2
   I/O Timings: shared read=0.655
   WAL: records=1 fpi=1 bytes=192
   ->  Seq Scan on t  (cost=0.00..1.02 rows=1 width=18) (actual time=0.007..0.009 rows=1 loops=1)
         Filter: (id = 2)
         Rows Removed by Filter: 1
         Buffers: shared hit=1
 Planning:
   Buffers: shared hit=3
 Planning Time: 0.072 ms
 Execution Time: 0.914 ms
(12 rows)
```

EXPLAIN options - WAL

```
postgres=> EXPLAIN (ANALYZE, WAL, BUFFERS) update t set price = 30 where id=2;
                                                 QUERY PLAN
-------------------------------------------------------------------------------------------------------------
 Update on t  (cost=0.00..1.02 rows=0 width=0) (actual time=0.043..0.044 rows=0 loops=1)
   Buffers: shared hit=3
   WAL: records=1 bytes=75
   ->  Seq Scan on t  (cost=0.00..1.02 rows=1 width=18) (actual time=0.020..0.021 rows=1 loops=1)
         Filter: (id = 2)
         Rows Removed by Filter: 1
         Buffers: shared hit=1
 Planning Time: 0.068 ms
 Execution Time: 0.058 ms
(9 rows)
```

The information about WALs can be most useful for understanding the generation of ‘full page images’ and hence, tuning checkpoints.

Using auto.explain module

auto.explain module – can automatically log plans for you (based on some parameters) in the error log

auto_explain.log_min_duration
auto_explain.log_analyze
auto_explain.log_buffers
auto_explain.log_wal
auto_explain.log_nested_statements etc.

Using auto.explain module

auto.explain module

```
postgres=# LOAD 'auto_explain’;
postgres=# SET auto_explain.log_min_duration = 0;
postgres=# SET auto_explain.log_analyze = true;

postgres=# SELECT count(*) FROM foo;
```

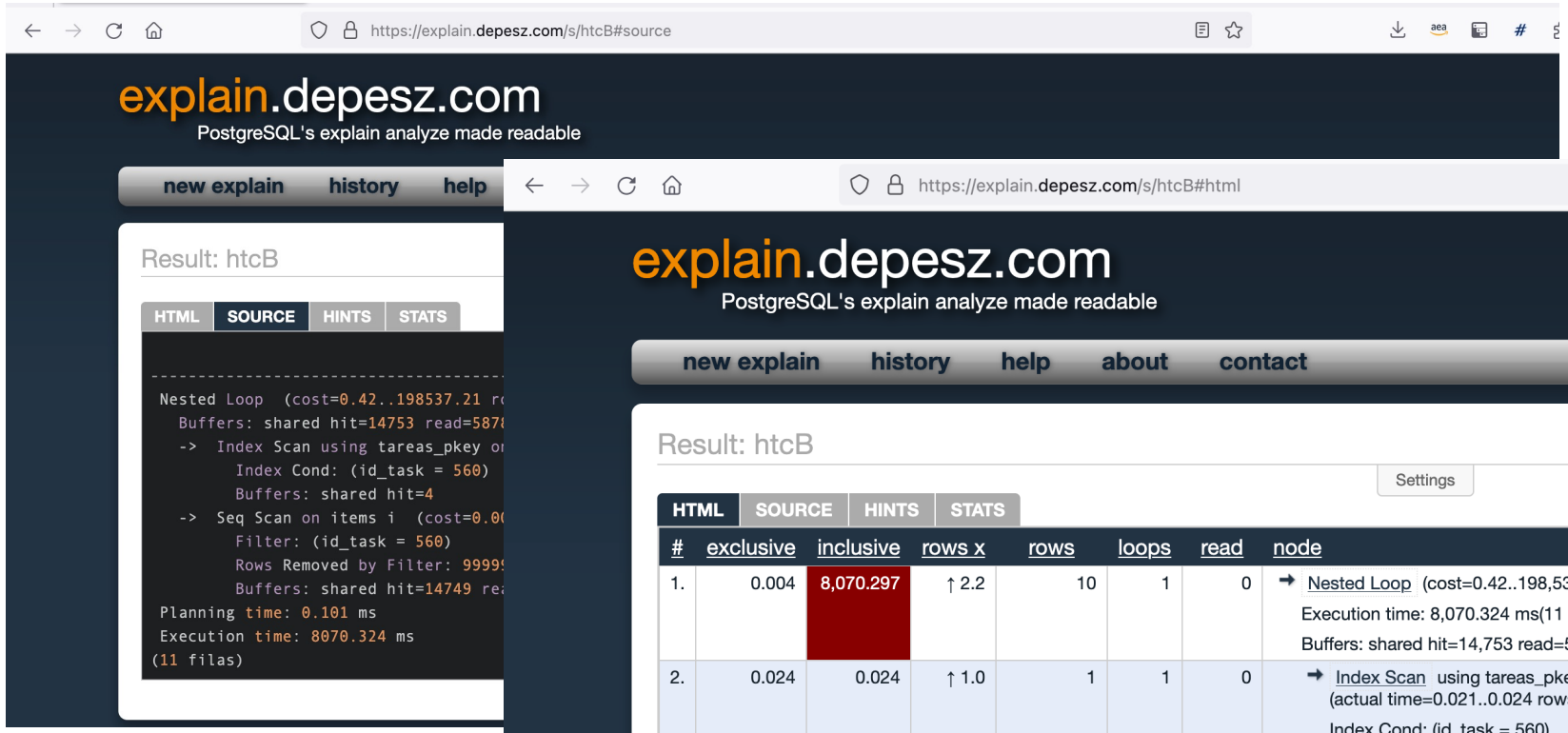
Using auto.explain module

```
2024-03-09 16:01:31 UTC:172.31.36.18(57920):postgres@postgres:[465]:LOG: duration: 140.038 ms plan:
Query Text: select count(*) from foo;
Finalize Aggregate (cost=5706.00..5706.01 rows=1 width=8) (actual time=138.342..140.028 rows=1 loops=1)
Buffers: shared read=2500
I/O Timings: shared/local read=192.767
-> Gather (cost=5705.88..5705.99 rows=1 width=8) (actual time=138.261..140.022 rows=2 loops=1)
Workers Planned: 1
Workers Launched: 1
Buffers: shared read=2500
I/O Timings: shared/local read=192.767
-> Partial Aggregate (cost=4705.88..4705.89 rows=1 width=8) (actual time=134.944..134.946 rows=1 loops=2)
Buffers: shared read=2500
I/O Timings: shared/local read=192.767
-> Parallel Seq Scan on foo (cost=0.00..4264.71 rows=176471 width=0) (actual time=2.039..120.575 rows=150000 loops=2)
Buffers: shared read=2500
I/O Timings: shared/local read=192.767

---------------------- END OF LOG ----------------------
```

Make sure you know the storage limits of the storage, to which the error logs are stored on, as logging explain plans will produce huge log files.

Understanding the explain plan



Query planner decision factors

Query planner decision factors

- Cost of the plan
- Statistics stored in pg_statistics (pg_stats is accessible)

Query planner decision factors

- **Cost of the plan**
- Statistics stored in pg_statistics (pg_stats is accessible)

Cost of the plan

1. Type of operation – Sequential scan, Index scan, sort
2. Parameter settings - seq_page_cost, random_page_cost, cpu_tuple_cost, parallel_setup_cost, effective_cache_size etc.

Cost for a Sequential Scan

```
postgres=> select * from t;
 id | price
----+-------
  1 | 10.00
  2 |
(2 rows)
```

```
postgres=> EXPLAIN (ANALYZE, WAL, BUFFERS) select * from t;
                                                 QUERY PLAN
-------------------------------------------------------------------------------------------------------------
 Seq Scan on t  (cost=0.00..1.02 rows=2 width=9) (actual time=0.005..0.006 rows=2 loops=1)
   Buffers: shared hit=1
 Planning Time: 0.035 ms
 Execution Time: 0.017 ms
(4 rows)
```

- Startup cost = 0 for Sequential Scan
- Run cost = (CPU run cost) + (Disk run cost)
  = (cpu_tuple_cost + cpu_operator_cost) * no. of tuples + seq_page_cost * no. of pages
  = (0.01 + 0.0025)* 2 + 1* 1
  = 0.025 + 1 = 1.025

Note, PostgreSQL assumes that all pages will be read from storage. In other words, PostgreSQL does not consider whether the scanned page is in the shared buffers or not.

Types of Scan

- Sequential Scan
- Index Scan
- Index Only Scan
- Bitmap Heap Scan

Parameter settings – seq_page_cost, random_page_cost, enable_indexscan, enable_seqscan etc.

Scan Type – Sequential Scan

```
postgres=> EXPLAIN ANALYZE (select count(*) from big_table);
                                                                  QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------------------------------
 Finalize Aggregate  (cost=14542.55..14542.56 rows=1 width=8) (actual time=64.874..65.823 rows=1 loops=1)
   ->  Gather  (cost=14542.33..14542.54 rows=2 width=8) (actual time=62.532..65.811 rows=3 loops=1)
         Workers Planned: 2
         Workers Launched: 2
         ->  Partial Aggregate  (cost=13542.33..13542.34 rows=1 width=8) (actual time=56.349..56.350 rows=1 loops=3)
               ->  Parallel Seq Scan on big_table  (cost=0.00..12500.67 rows=416667 width=0) (actual time=0.010..33.251 rows=333333 loops=3)
 Planning Time: 0.055 ms
 Execution Time: 65.867 ms
(8 rows)
```

Parameter settings –
max_parallel_workers,
max_parallel_workers_per_gather
enable_seq_scan

Scan type – Sequential and Index Scan

```
postgres=> \d+ test_exp
                                    Table "public.test_exp"
 Column |  Type   | Collation | Nullable | Default | Storage | Stats target | Description
--------+---------+-----------+----------+---------+---------+--------------+-------------
 a      | integer |           | not null |         | plain   |              |
 b      | integer |           |          |         | plain   |              |
Indexes:
    "test_exp_pkey" PRIMARY KEY, btree (a)
Access method: heap

postgres=> select * from test_exp;
 a | b
---+----
 1 |  2
 2 |  4
 3 |  6
 4 |  8
 5 | 10
(5 rows)
```

Scan type – Sequential and Index Scan

```
postgres=> explain (analyze, buffers) select * from test_exp where a=4;
                                                 QUERY PLAN
-------------------------------------------------------------------------------------------------------------
 Seq Scan on test_exp  (cost=0.00..1.06 rows=1 width=8) (actual time=0.009..0.009 rows=1 loops=1)
   Filter: (a = 4)
   Rows Removed by Filter: 4
   Buffers: shared hit=1
 Planning Time: 0.058 ms
 Execution Time: 0.057 ms
(6 rows)

postgres=> set enable_seqscan='off';
SET
postgres=> explain (analyze, buffers) select * from test_exp where a=4;
                                                 QUERY PLAN
-------------------------------------------------------------------------------------------------------------
 Index Scan using test_exp_pkey on test_exp  (cost=0.13..8.15 rows=1 width=8) (actual time=0.094..0.096 rows=1 loops=1)
   Index Cond: (a = 4)
   Buffers: shared hit=1 read=1
   I/O Timings: shared/local read=0.078
 Planning Time: 0.222 ms
 Execution Time: 0.194 ms
(6 rows)
```

Scan type – Sequential and Index Scan

Trying to hint with pg_hint_plan

```
postgres=> CREATE EXTENSION pg_hint_plan;
CREATE EXTENSION
postgres=> set enable_seqscan=on;
SET

postgres=> explain analyze /*+ Indexscan (test_exp test_exp_pkey) */ select * from
test_exp where a=4;
QUERY PLAN
------------------------------------------------------------------------------
--------------
Seq Scan on pg_hint_test (cost=0.00..38.25 rows=1 width=8) (actual time=0.009..0.009
rows=1 loops=1)
Filter: (a = 4)
Rows Removed by Filter: 4
Planning Time: 0.054 ms
Execution Time: 0.023 ms
(5 rows)
```

The planner can ignore hints from pg_hint_plan, provided it knows there are other better plans than the ones you are hinting towards!

Scan type – Index Only Scan

```
postgres=> \d+ pgbench_accounts
                                          Table "public.pgbench_accounts"
  Column  |     Type      | Collation | Nullable | Default | Storage  | Compression | Stats target | Description
----------+---------------+-----------+----------+---------+----------+-------------+--------------+-------------
 aid      | integer       |           | not null |         | plain    |             |              |
 bid      | integer       |           |          |         | plain    |             |              |
 abalance | integer       |           |          |         | plain    |             |              |
 filler   | character(84) |           |          |         | extended |             |              |
Indexes:
    "pgbench_accounts_pkey" PRIMARY KEY, btree (aid)
Access method: heap
Options: fillfactor=100, autovacuum_enabled=false

postgres=> EXPLAIN ANALYZE (select count(*) from pgbench_accounts);
                                                                             QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------------------------------------------------
 Aggregate  (cost=2846.86..2846.87 rows=1 width=8) (actual time=14.745..14.746 rows=1 loops=1)
   ->  Index Only Scan using pgbench_accounts_pkey on pgbench_accounts  (cost=0.29..2597.35 rows=99804 width=0) (actual time=0.018..9.945 rows=99804 loops=1)
         Heap Fetches: 0
 Planning Time: 0.060 ms
 Execution Time: 14.790 ms
(5 rows)
```

Scan type – Index Only Scan

- Index type must support index-only scans
- The query must reference only reference the columns stored in the index
  - Table having columns : x, y, z where (x, y) is the index
  - SELECT x FROM tab WHERE x = 'key' AND y < 42;
  - SELECT x FROM tab WHERE x = 'key' AND z < 42;

Tuple visibility information is not stored in the index, but only in the heap → **Visibility map – 1 bit for each page of the heap to know if all is visible** → **Updated by vacuum = efficient index-only scans**

Scan type – Bitmap scan

```
postgres=> \d+ pgbench_accounts
                                        Table "public.pgbench_accounts"
  Column  |     Type      | Collation | Nullable | Default | Storage  | Compression | Stats target | Description
----------+---------------+-----------+----------+---------+----------+-------------+--------------+-------------
 aid      | integer       |           | not null |         | plain    |             |              |
 bid      | integer       |           |          |         | plain    |             |              |
 abalance | integer       |           |          |         | plain    |             |              |
 filler   | character(84) |           |          |         | extended |             |              |
Indexes:
    "pgbench_accounts_pkey" PRIMARY KEY, btree (aid)
    "pgbench_accounts_bid_idx" btree (bid)
Access method: heap
postgres=> EXPLAIN ANALYZE  SELECT * FROM pgbench_accounts  WHERE aid < 100 or bid > 285;
                                                                 QUERY PLAN
---------------------------------------------------------------------------------------------------------------------------------------------
 Bitmap Heap Scan on pgbench_accounts  (cost=9.24..277.40 rows=81 width=97) (actual time=0.016..0.027 rows=49 loops=1)
   Recheck Cond: ((aid < 100) OR (bid > 285))
   Heap Blocks: exact=5
   ->  BitmapOr  (cost=9.24..9.24 rows=81 width=0) (actual time=0.010..0.010 rows=0 loops=1)
         ->  Bitmap Index Scan on pgbench_accounts_pkey  (cost=0.00..4.90 rows=81 width=0) (actual time=0.006..0.006 rows=49 loops=1)
               Index Cond: (aid < 100)
         ->  Bitmap Index Scan on pgbench_accounts_bid_idx  (cost=0.00..4.30 rows=1 width=0) (actual time=0.003..0.003 rows=0 loops=1)
               Index Cond: (bid > 285)
 Planning Time: 0.094 ms
 Execution Time: 0.045 ms
(10 rows)
```

Bitmaps also help in combining multiple indexes (including multiple uses of the same index) to handle cases that cannot be implemented by single index scans.

Bitmap scans: Exact and Lossy Heap blocks

```
EXPLAIN (ANALYZE) SELECT * FROM person WHERE age = 20 ;

                                        QUERY PLAN
-------------------------------------------------------------------------------------------------------------
Gather  (cost=3682.90..212050.63 rows=97334 width=126) (actual time=46.142..221.876 rows=101476 loops=1)
  Workers Planned: 2
  Workers Launched: 2
  ->  Parallel Bitmap Heap Scan on person  (cost=2682.90..201317.23 rows=40556 width=126) (actual
time=24.783..189.769 rows=33825 loops=3)
        Recheck Cond: (age = 20)
        Rows Removed by Index Recheck: 534475
        Heap Blocks: exact=17931 lossy=12856
        ->  Bitmap Index Scan on idx_person(cost=0.00..2658.57 rows=97334 width=0) (actual
time=36.926..36.926 rows=101476 loops=1)
              Index Cond: (age = 20)
Planning Time: 0.122 ms
Execution Time: 225.554 ms
```

Increasing work_mem until the scan uses mostly Exact Heap Blocks should improve performance, but be careful if you are making this change globally.

Another indication to tune work_mem

```
Aggregate  (cost=5348342.29..5348342.30 rows=1 width=8) (actual time=77984.568..78001.306 rows=1 loops=1)
  ->  Unique  (cost=1250433.88..5173254.71 rows=14007007 width=17) (actual time=24939.464..77045.024 rows=14448223 loops=1)
        ->  Merge Join  (cost=1250433.88..4898815.51 rows=54887840 width=17) (actual time=24939.462..69413.044 rows=53255128 loops=1)
              Merge Cond: ((cs_le.cs_company_id)::text = (cs_search.cs_company_id)::text)
              ->  Gather Merge  (cost=1250432.03..2934134.22 rows=14456539 width=17) (actual time=24932.628..41042.679 rows=14463238 loops=1)
                    Workers Planned: 2
                    Workers Launched: 2
                    ->  Sort  (cost=1249432.00..1264490.90 rows=6023558 width=17) (actual time=24866.655..29748.967 rows=4821079 loops=3)
                          Sort Key: cs_le.cs_company_id, cs_le.rank
                          Sort Method: external merge  Disk: 102936kB
                          Worker 0:  Sort Method: external merge  Disk: 103736kB
                          Worker 1:  Sort Method: external merge  Disk: 103152kB
                          ->  Parallel Seq Scan on cs_legal_entities_2024 cs_le  (cost=0.00..324048.58 rows=6023558 width=17) (actual time=1.(
              ->  Index Only Scan using cs_search_2024_cc_int_cs_company_id_idx on cs_search_2024 cs_search  (cost=0.56..1204504.77 rows=5318
                    Heap Fetches: 0
Planning Time: 0.632 ms
Execution Time: 78018.690 ms
```

Copy source to clipboard

Query planner decision factors

- Cost of the plan
- **Statistics stored in pg_statistics (pg_stats is accessible)**

Statistics used by the planner – ”ANALYZE”

- Collects statistics about the contents of tables in the database – data distribution statistics
- Ensures that the planner has up-to-date statistics about the table.
- strongly recommended to run ANALYZE whenever you have significantly altered the distribution of data within a table.
- If rows planned are very different from actual rows returned, run ANALYZE.
- If autovacuum daemon is enabled, it might run ANALYZE automatically
- Run ”Analyze” after a version upgrades for PostgreSQL version 17 and below. PostgreSQL 18 introduces the ability to keep planner statistics through a major version upgrade.

To have better planner statistics

1. Consider setting an optimal value for default_statistics_target – default is 100, max allowed value is 10000.
2. default_statistics_target – can be set per column basis or globally for the entire database

```
postgres=> ALTER TABLE test_exp ALTER COLUMN a SET STATISTICS 100;
ALTER TABLE
postgres=> \d+ test_exp
                                   Table "public.test_exp"
 Column |  Type   | Collation | Nullable | Default | Storage | Stats target | Description
--------+---------+-----------+----------+---------+---------+--------------+-------------
 a      | integer |           | not null |         | plain   | 100          |
 b      | integer |           |          |         | plain   |              |
Indexes:
    "test_exp_pkey" PRIMARY KEY, btree (a)
Access method: heap
```

Note : Increasing the target causes a proportional increase in the time and space needed to do ANALYZE.

Understanding PostgreSQL Statistics

https://aws.amazon.com/blogs/database/understanding-statistics-in-postgresql/

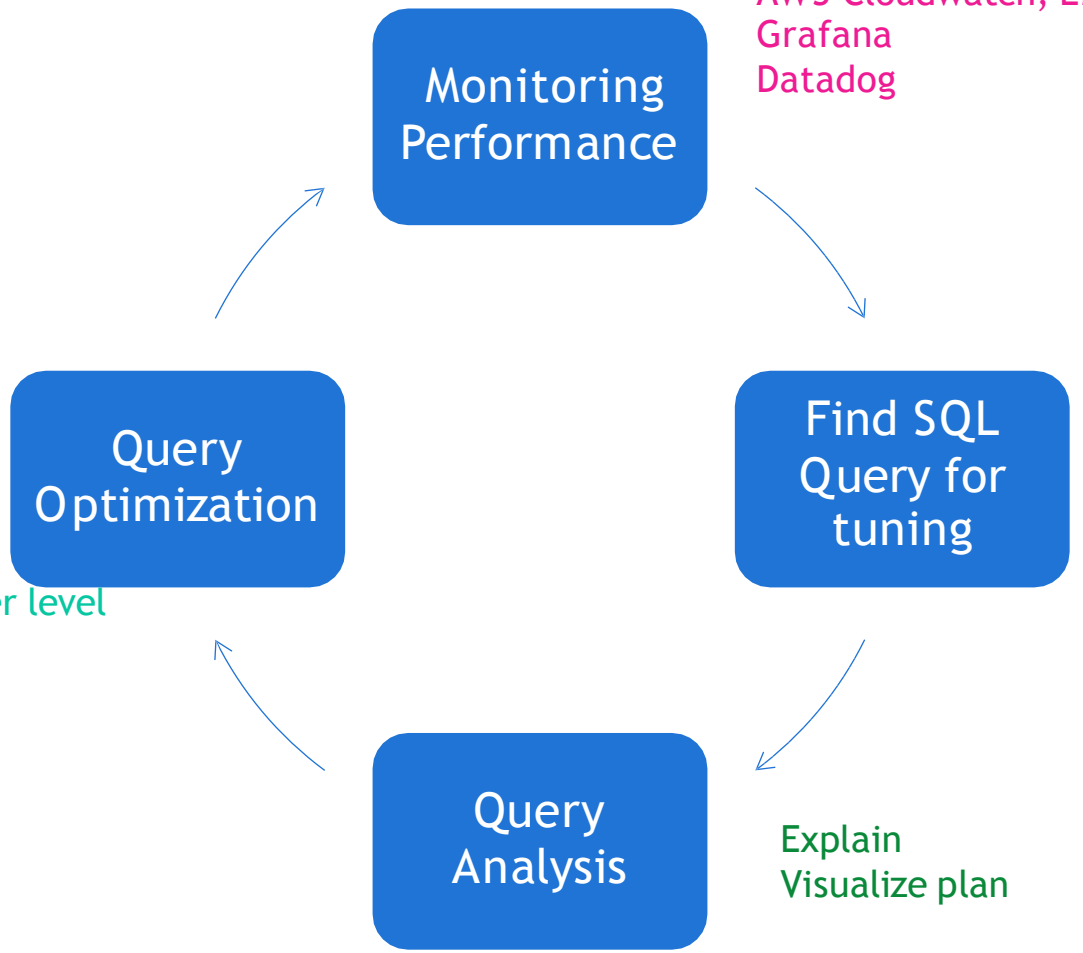
Planning Prepared statements

- A prepared statement is a server-side object that can be used to optimize performance.
- PREPARE = specified statement is parsed, analyzed, and rewritten
- EXECUTE (subsequently issued) = the prepared statement planned and executed
- ‘generic plan’ or ‘custom plan’ – planners waits for 5 executions
- plan_cache_mode = default value is auto;
  can also be set to force_generic_plan or force_custom_plan

Key Takeaways

- Plan can be captured manually using ‘EXPLAIN’ or by using the ‘auto.explain’ module
- Selecting a plan is dependent on cost of plan and planner statistics
- Running Analyze might not be enough – know about tuning default_statistics_target.
- Know the indexes you are creating for their optimal use
- Bitmap heap scan – can be exact or lossy – consider increasing work_mem
- Another indication of tuning work_mem would be visible disk usage in the plan
- Prepared statements – custom or generic plans (the latter after 5 executions). the prepared statement is forgotten at session termination.

Query Tuning Cycle



AWS Cloudwatch, EM and PI for RDS/ Aurora
Grafana
Datadog

Logging paramters/Error Log
pg_stat_statement
auto_explain extension

Explain
Visualize plan

Collecting more statistics
Relevant indexes (create/drop)
Update parameters database/session/user level

Thank you!

Divya Sharma

https://www.linkedin.com/in/divyasharma95/

Your feedback is very valuable to us!