EXPLAIN Explained

Understanding the PostgreSQL planner better

Divya Sharma

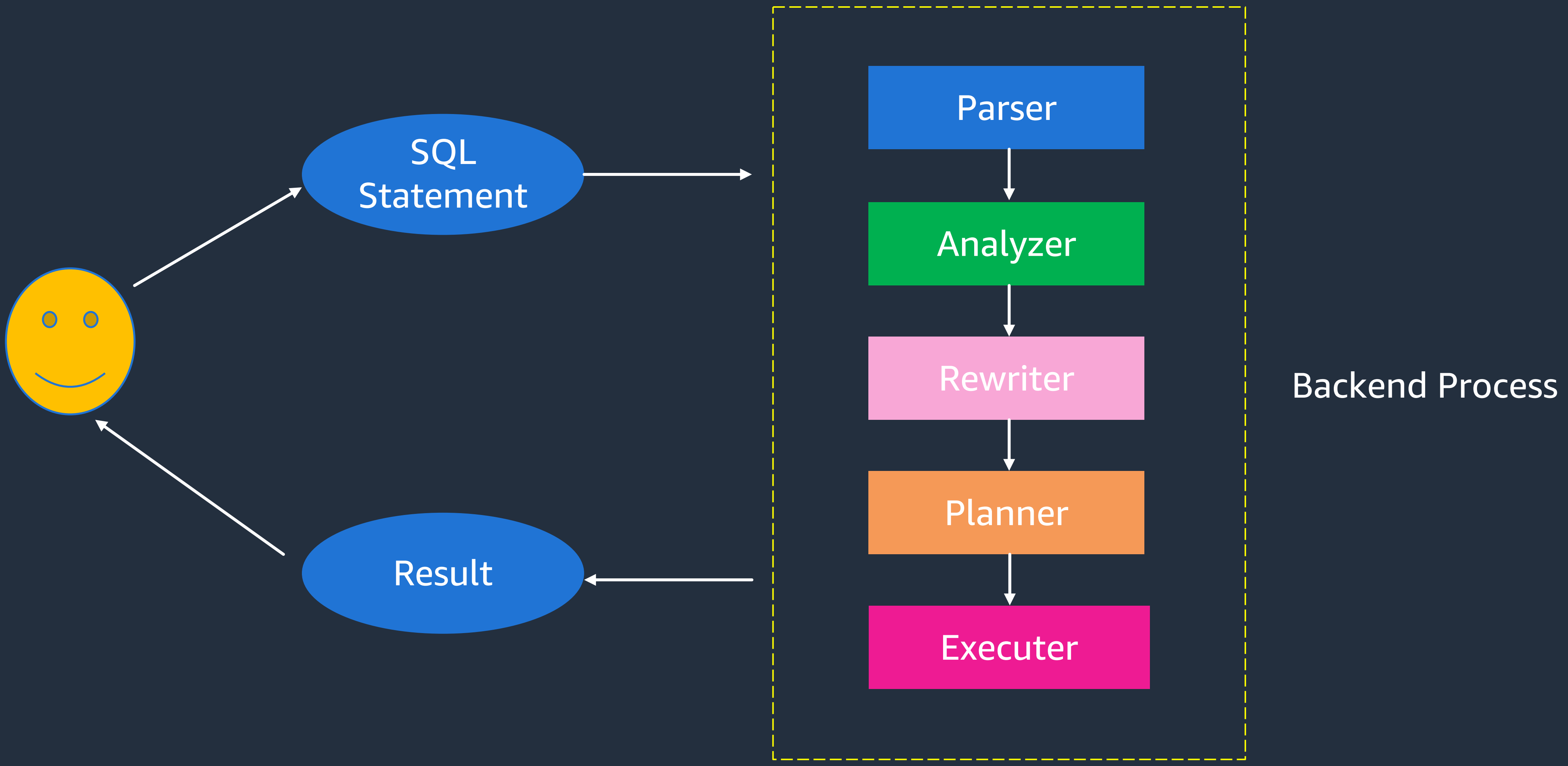
Sr. RDS PostgreSQL Solutions Architect

Agenda

- Understanding the stages for Query Planning
- Query Planning decision factors – cost of plan, statistics, parameter settings
- work_mem tuning indications
- Execution of prepared statements

Stages of Query Planning

Stages of Query Planning



How to get the explain plan?

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 }

EXPLAIN options

```
postgres=> EXPLAIN ( ANALYZE, WAL, 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

```
postgres=> EXPLAIN (ANALYZE, WAL, BUFFERS) DELETE FROM test WHERE random() < 0.5;

                                               QUERY PLAN

--------------------------------------------------------------------------------------------------------------

 Delete on test  (cost=0.00..14.80 rows=0 width=0) (actual time=0.424..0.425 rows=0 loops=1)

   Buffers: shared hit=283 dirtied=8

   WAL: records=269 fpi=7 bytes=22733

   ->  Seq Scan on test  (cost=0.00..14.80 rows=173 width=6) (actual time=0.006..0.058 rows=269 loops=1)

         Filter: (random() < '0.5'::double precision)

         Rows Removed by Filter: 251

         Buffers: shared hit=7

 Planning:

   Buffers: shared hit=11

 Planning Time: 0.100 ms

 Execution Time: 0.457 ms

(11 rows)
```

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

How to get the explain plan?

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

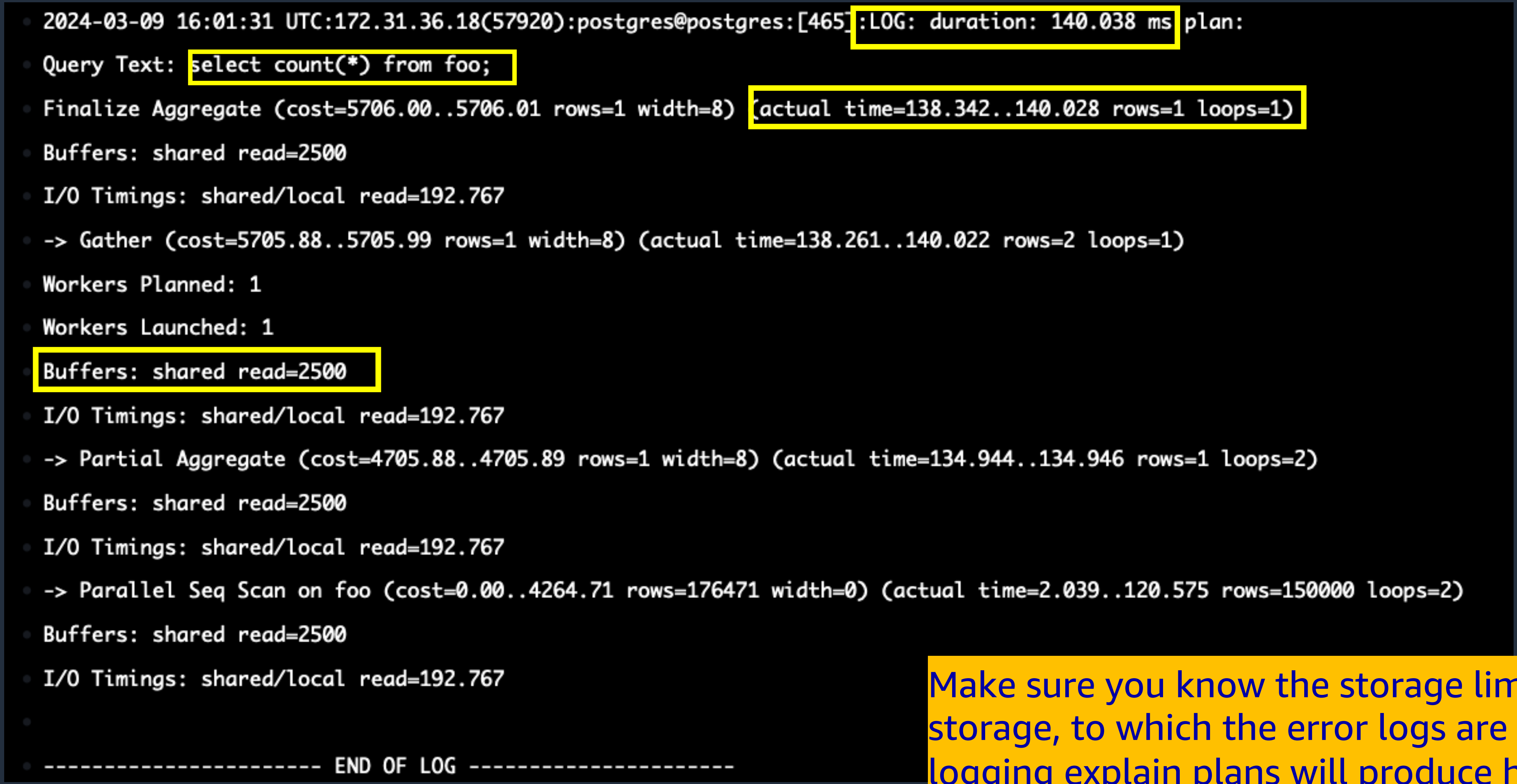
auto_explain.log_min_duration
auto_explain.log_analyze
auto_explain.log_buffers
auto_explain.log_wal
auto_explain.log_nested_statements etc.

How to get the explain plan?

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;
```

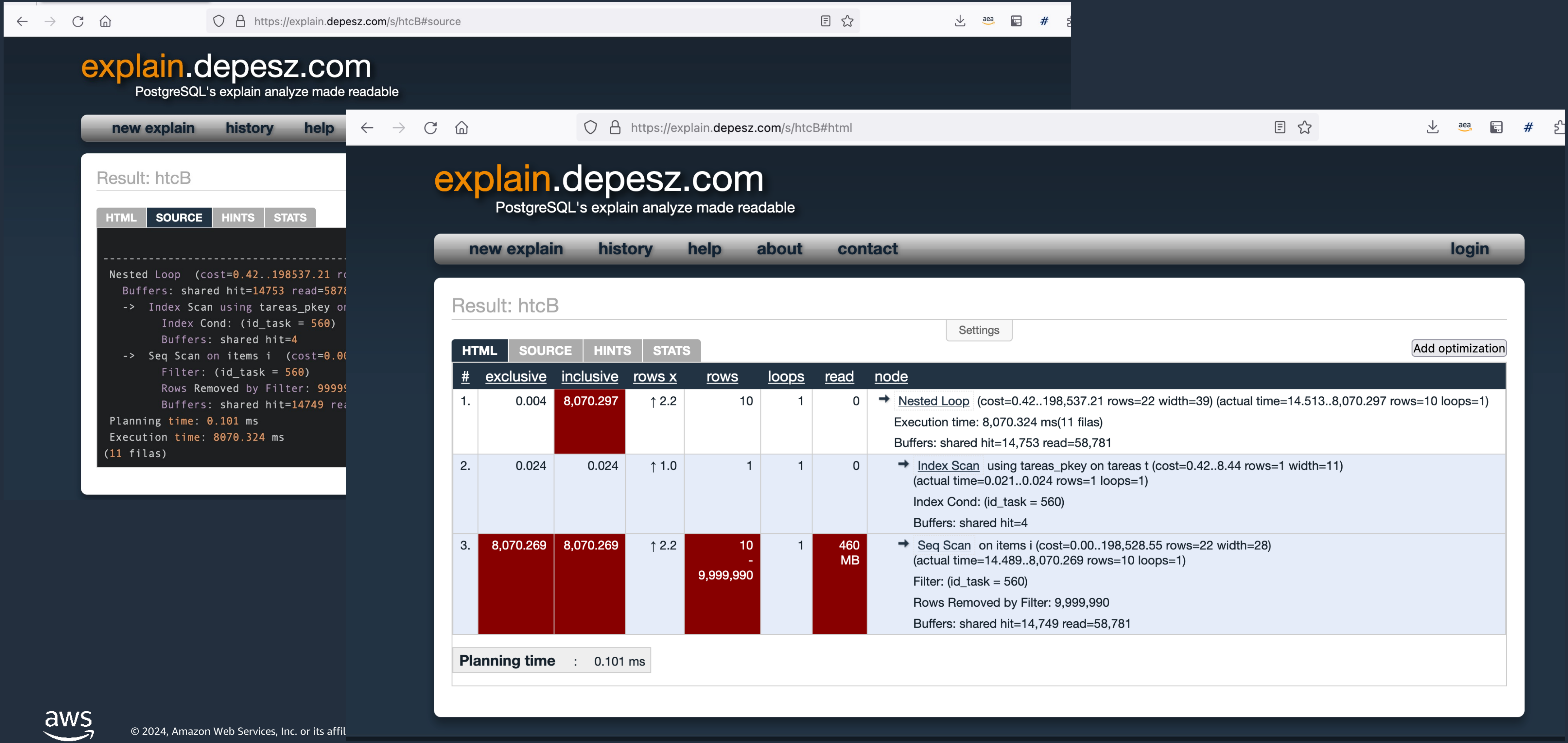
How to get the explain plan? - auto.explain



```
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


https://explain.depesz.com/s/htcB#source
explain.depesz.com
PostgreSQL's explain analyze made readable
new explain
history
help
Result: htcB
HTML
SOURCE
HINTS
STATS

```
-----------------------------------------
 Nested Loop  (cost=0.42..198537.21 ro
   Buffers: shared hit=14753 read=5878
   ->  Index Scan using tareas_pkey on
         Index Cond: (id_task = 560)
         Buffers: shared hit=4
   ->  Seq Scan on items i  (cost=0.00
         Filter: (id_task = 560)
         Rows Removed by Filter: 99999
         Buffers: shared hit=14749 rea
 Planning time: 0.101 ms
 Execution time: 8070.324 ms
(11 filas)
```

#	exclusive	inclusive	rows x	rows	loops	read	node
1.	0.004	8,070.297	↑ 2.2	10	1	0	→ Nested Loop (cost=0.42..198,537.21 rows=22 width=39) (actual time=14.513..8,070.297 rows=10 loops=1) Execution time: 8,070.324 ms(11 filas) Buffers: shared hit=14,753 read=58,781
2.	0.024	0.024	↑ 1.0	1	1	0	→ Index Scan using tareas_pkey on tareas t (cost=0.42..8.44 rows=1 width=11) (actual time=0.021..0.024 rows=1 loops=1) Index Cond: (id_task = 560) Buffers: shared hit=4
3.	8,070.269	8,070.269	↑ 2.2	10 - 9,999,990	1	460 MB	→ Seq Scan on items i (cost=0.00..198,528.55 rows=22 width=28) (actual time=14.489..8,070.269 rows=10 loops=1) Filter: (id_task = 560) Rows Removed by Filter: 9,999,990 Buffers: shared hit=14,749 read=58,781

Planning time : 0.101 ms

Query planner decision factors

- Cost of the plan
- Statistics stored in pg_statistics (pg_stats is accessible)
- Parameter settings – seq_page_cost, random_page_cost, enable_indexscan, enable_seqscan etc.

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=> EXPLAIN ( ANALYZE, WAL, 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)
```

- 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)* 300000 + 1* 2500

  = 6250

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.

Statistics used by the planner - ”ANALYZE”

- ANALYZE collects statistics about the contents of tables in the database – data distribution statistics
- Running ANALYZE (or VACUUM ANALYZE) ensures that the planner has up-to-date statistics about the table.
- Whenever you have significantly altered the distribution of data within a table, running **ANALYZE** is strongly recommended.
- Note that if the autovacuum daemon is enabled, it might run ANALYZE automatically
- Run ”Analyze” after a version upgrades and creation of indexes.

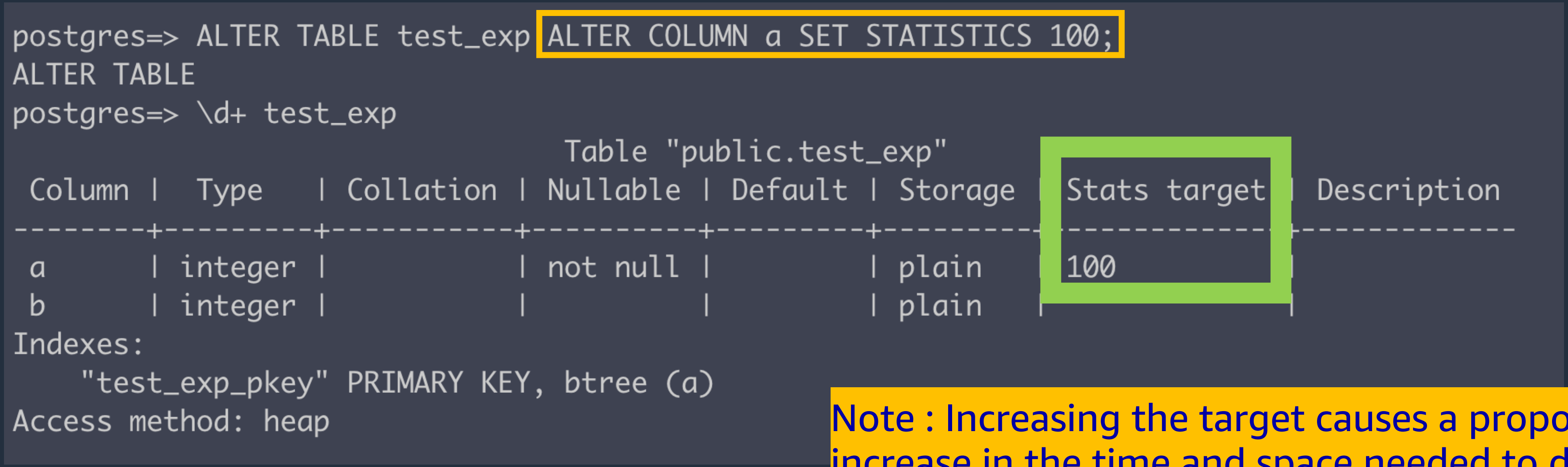
Know when to run ANALYZE (to update statistics)

```
postgres=> EXPLAIN ( ANALYZE, WAL, 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)
```

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.

default_statistics_target and n_distinct

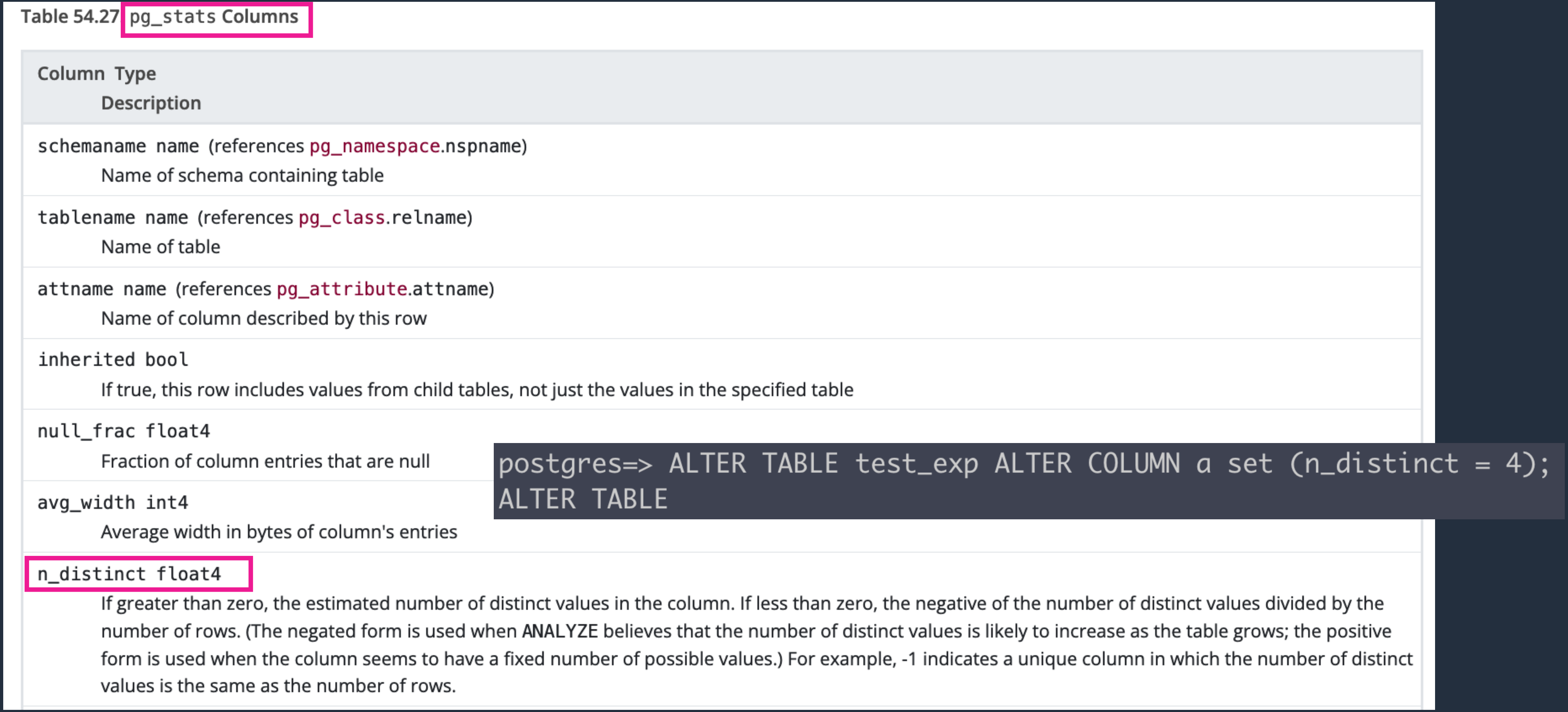


Table 54.27 pg_stats Columns

Column	Type	Description
schemaname	name (references pg_namespace.nspname)	Name of schema containing table
tablename	name (references pg_class.relname)	Name of table
attname	name (references pg_attribute.attname)	Name of column described by this row
inherited	bool	If true, this row includes values from child tables, not just the values in the specified table
null_frac	float4	Fraction of column entries that are null
avg_width	int4	Average width in bytes of column's entries
n_distinct	float4	If greater than zero, the estimated number of distinct values in the column. If less than zero, the negative of the number of distinct values divided by the number of rows. (The negated form is used when ANALYZE believes that the number of distinct values is likely to increase as the table grows; the positive form is used when the column seems to have a fixed number of possible values.) For example, -1 indicates a unique column in which the number of distinct values is the same as the number of rows.

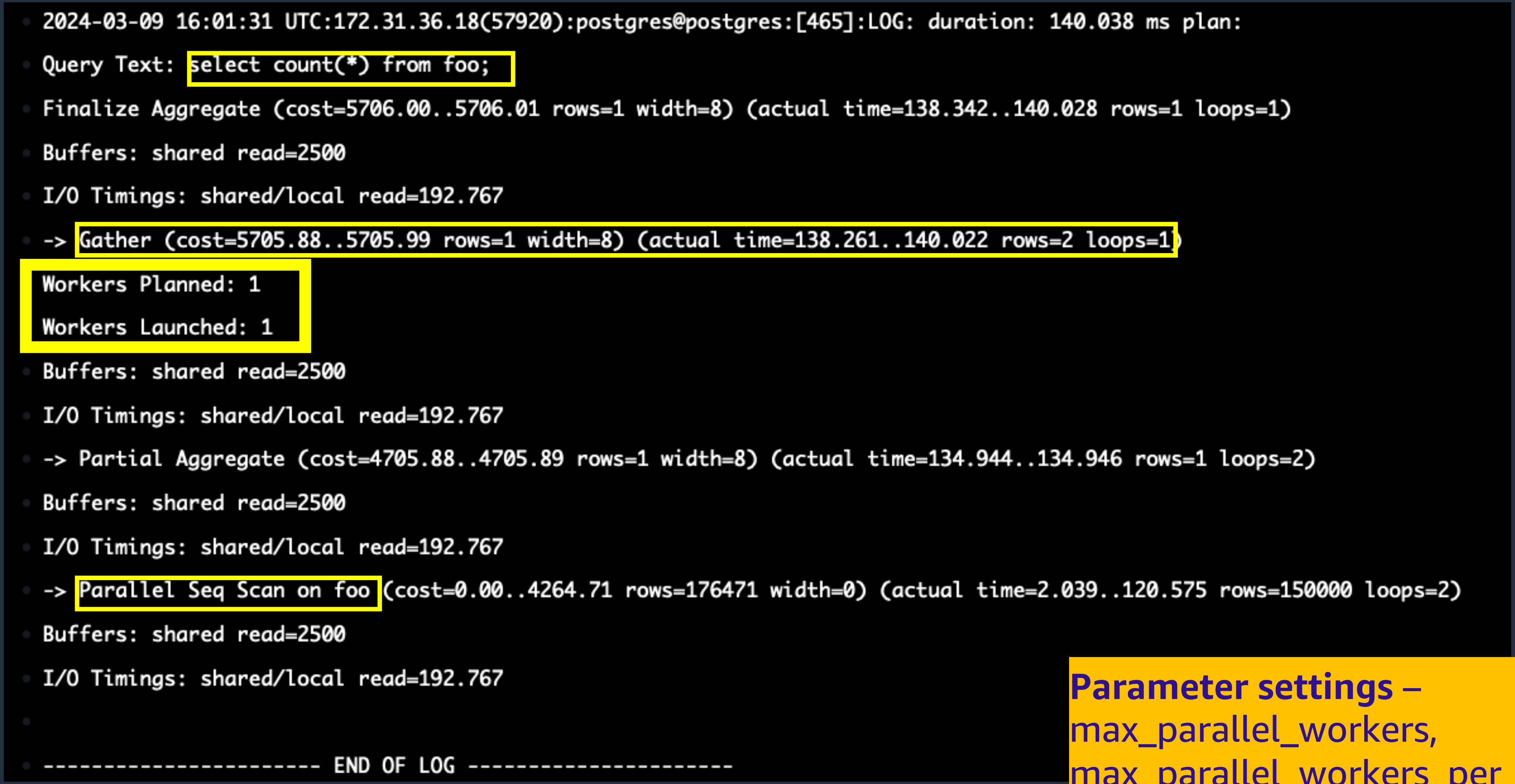
```
postgres=> ALTER TABLE test_exp ALTER COLUMN a set (n_distinct = 4);
ALTER TABLE
```

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.

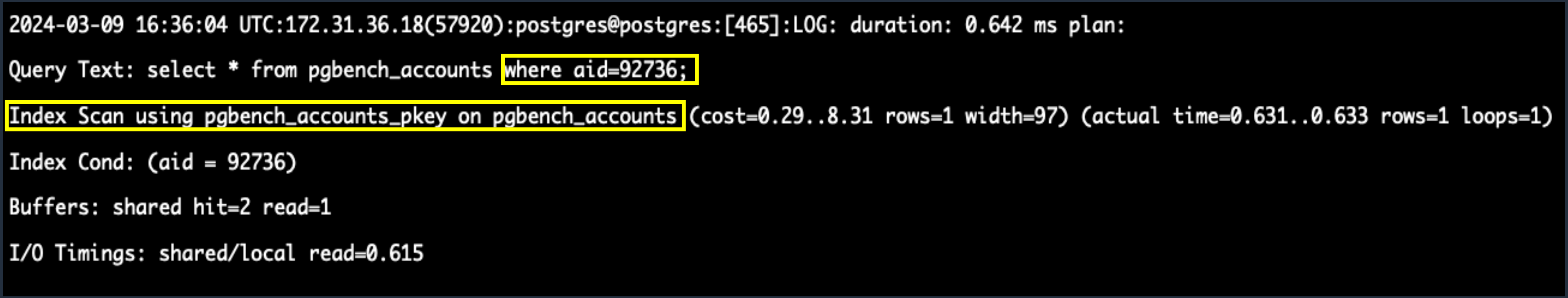
Scan Type – Sequential Scan



```
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 ----------------------
```

Parameter settings –
max_parallel_workers,
max_parallel_workers_per_gather
enable_seq_scan

Scan type – Index Scan

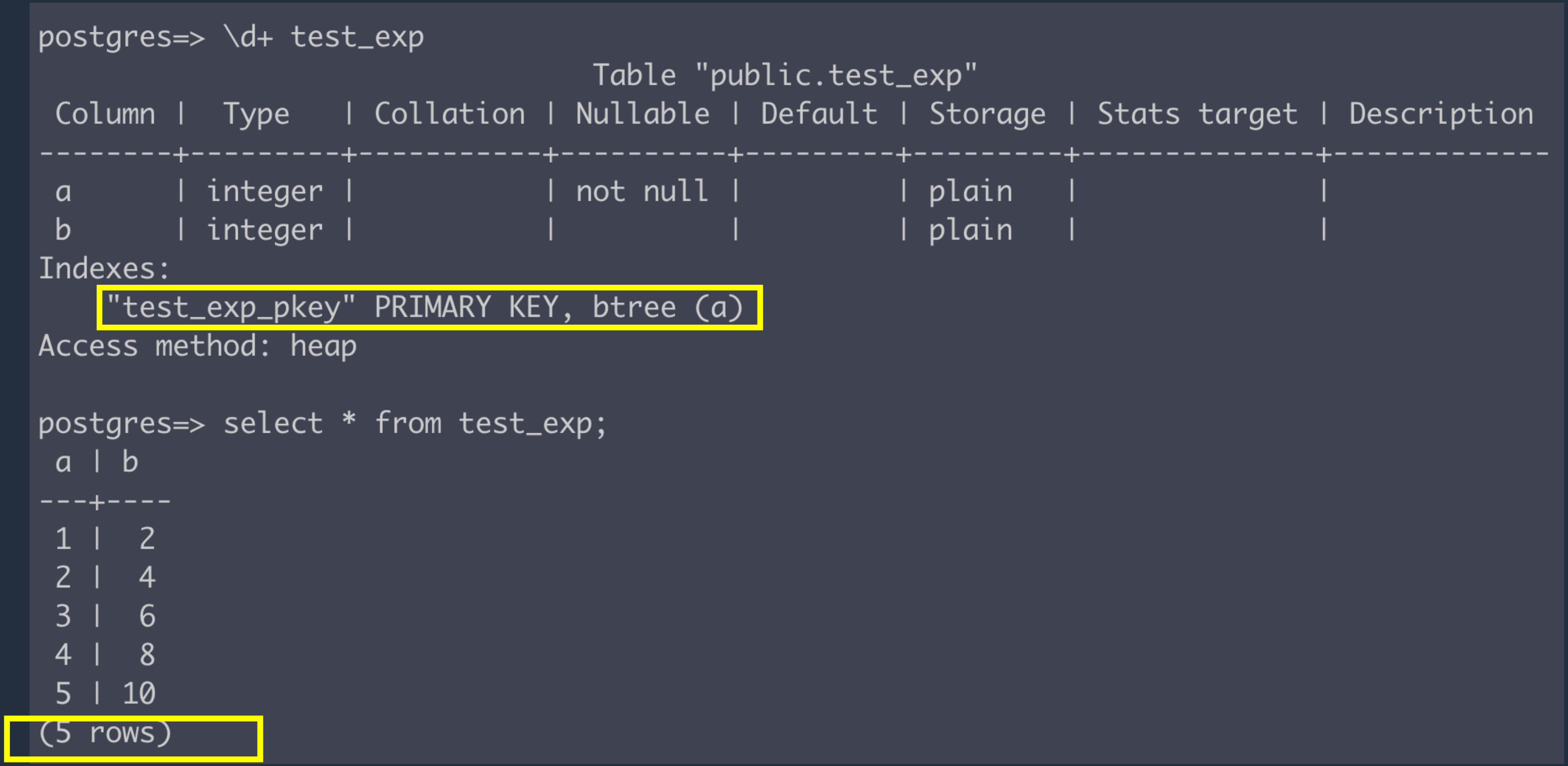


```
2024-03-09 16:36:04 UTC:172.31.36.18(57920):postgres@postgres:[465]:LOG: duration: 0.642 ms plan:
Query Text: select * from pgbench_accounts where aid=92736;
Index Scan using pgbench_accounts_pkey on pgbench_accounts (cost=0.29..8.31 rows=1 width=97) (actual time=0.631..0.633 rows=1 loops=1)
Index Cond: (aid = 92736)
Buffers: shared hit=2 read=1
I/O Timings: shared/local read=0.615
```

Note :

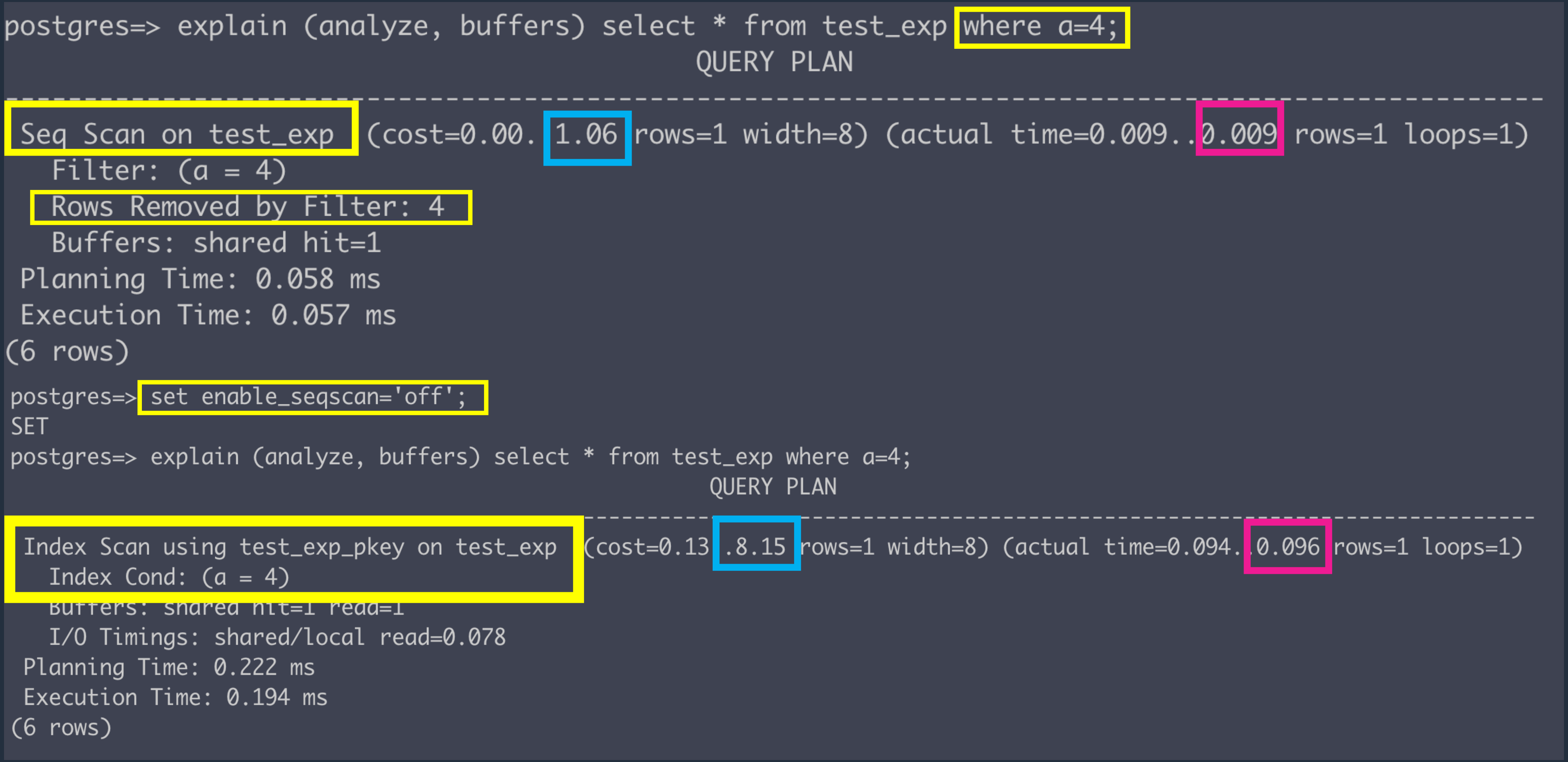
- **All indexes in PostgreSQL are *secondary* indexes**
- **Parallel Index scans are also supported, but only for b-tree indexes currently**

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)
```

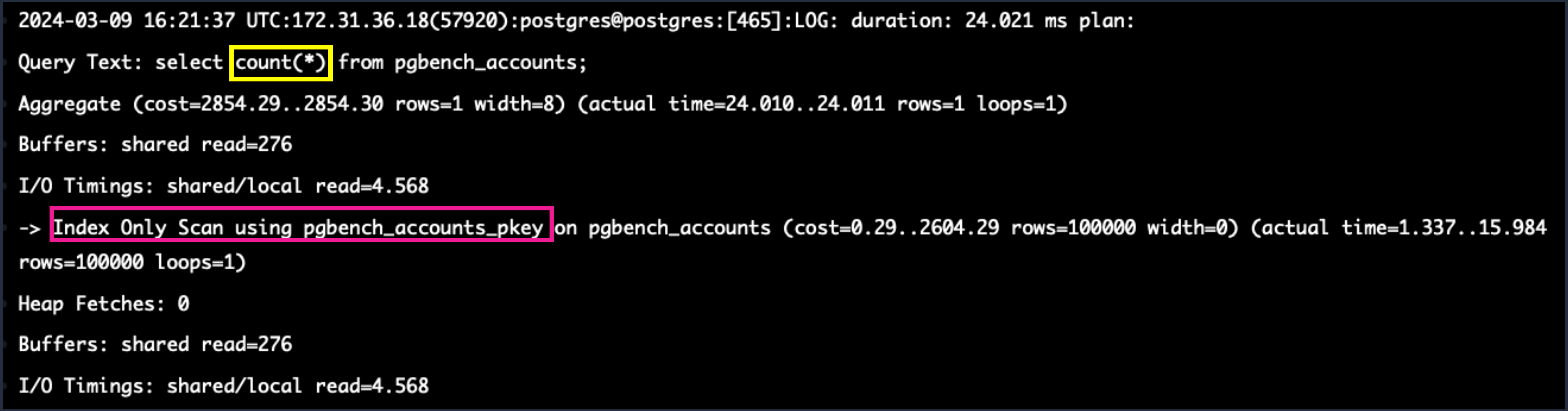
Scan type – Sequential and Index Scan

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!

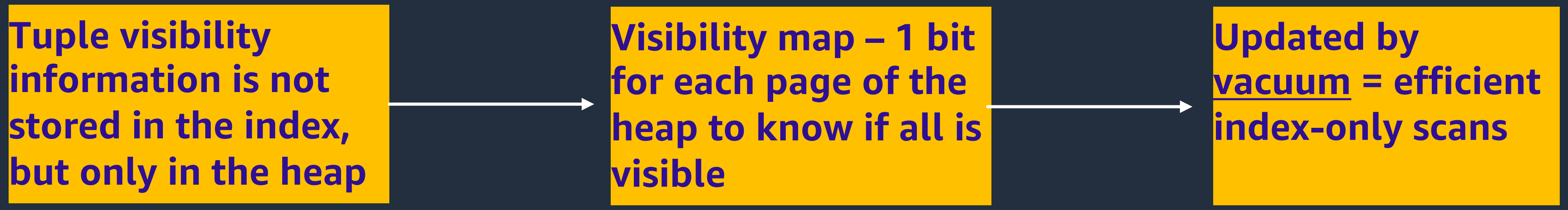
Scan type – Index Only Scan



```
2024-03-09 16:21:37 UTC:172.31.36.18(57920):postgres@postgres:[465]:LOG: duration: 24.021 ms plan:
Query Text: select count(*) from pgbench_accounts;
Aggregate (cost=2854.29..2854.30 rows=1 width=8) (actual time=24.010..24.011 rows=1 loops=1)
Buffers: shared read=276
I/O Timings: shared/local read=4.568
-> Index Only Scan using pgbench_accounts_pkey on pgbench_accounts (cost=0.29..2604.29 rows=100000 width=0) (actual time=1.337..15.984
rows=100000 loops=1)
Heap Fetches: 0
Buffers: shared read=276
I/O Timings: shared/local read=4.568
```

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

Covering Indexes

- You can also use covering indexes to benefit more from index-only scans :
  - CREATE INDEX tab_x_y ON tab(x) INCLUDE (y);
  - Remember, this is not equal to : CREATE INDEX tab_x_y ON tab(x, y) ;

Scan type – Bitmap scan

```
EXPLAIN SELECT * FROM tbl WHERE n < 100;

                           QUERY PLAN

-------------------------------------------------------------------------
Bitmap Heap Scan on tbl  (cost=2.37..232.35 rows=106 width=244)
   Recheck Cond: (n < 100)
   ->  Bitmap Index Scan on tbl_idx  (cost=0.00..2.37 rows=106 width=0)
         Index Cond: (n < 100)
```

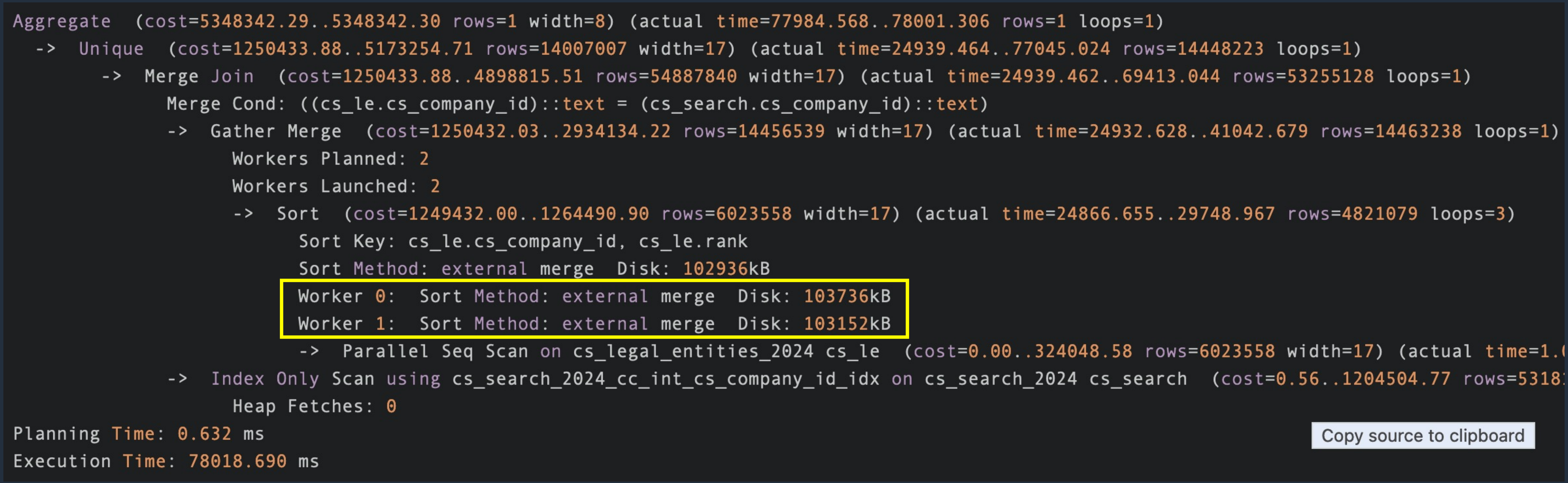
Bitmaps also help in combining multiple indexes (including multiple uses of the same index) to handle cases that cannot be implemented by single index scans.

Heap blocks : Exact and Lossy

```
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
```

Just one last note on : 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, statistics, and some parameter settings (eg. enable_seqscan)
- Running Analyze might not be enough – know about tuning default_statistics_target (and also n_distinct)
- Know the indexes you are creating for their optimal use – consider covering indexes if needed.
- 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). When the session ends, the prepared statement is forgotten.

Know if the statistics are updated – especially after a version upgrade or major data changes

Know about the indexes you create – only meaningful indexes add value!

Know if you need to tune memory parameters and parallel worker parameters!

Always have a direction/focus while troubleshooting issues in performance via explain plans!

Thank you!

Divya Sharma

Sr. RDS PostgreSQL Solutions Architect

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