Introduction to WITH queries and Materialization

Divya Sharma

Sr. Database Specialist SA
Amazon Web Services

We will be discussing

1. What are Common Table Expressions?
2. Advantages of using CTEs
3. What are Materialized CTEs?
4. Usage and impact of Materialized CTEs
5. Should I use CTEs?

Why are we discussing “Common Table Expressions”?

It all started with a customer question

Existing SQL	with cte_items as (select distinct c.p_id as p_id, ps.ps_sk as ps_sk from euA_staging.staging_nd_mk_c c join eu_report.v_ps ps on ps.p_id::integer=c.cm_id and c.p_cdr='AB NATIONAL') select
Updated SQL	with cte_items as MATERIALIZED (select distinct c.p_id as p_id, ps.ps_sk as ps_sk from euA_staging.staging_nd_mk_c c join eu_report.v_ps ps on ps.p_id::integer=c.cm_id and c.p_cdr='AB NATIONAL') select
Run time in new aroura version with "Materialized"	3 min 28 secs
Run time in new aroura version without "Materialized"	48 secs

What are “Common Table Expressions”?

WITH clause syntax

```
WITH cte_name AS (

CTE_query_definition

) statement;
```

What are “Common Table Expressions”?

```
WITH regional_sales AS (

   SELECT region, SUM(amount) AS total_sales
   FROM orders
   GROUP BY region ),

   top_regions AS (

   SELECT region
   FROM regional_sales
   WHERE total_sales > (SELECT SUM(total_sales)/10 FROM regional_sales) )

SELECT region, product, SUM(quantity) AS product_units, SUM(amount) AS
product_sales
FROM orders
WHERE region IN (SELECT region FROM top_regions) GROUP BY region, product;
```

Auxiliary statement 1

Auxiliary statement 2

Primary statement

What settings impact CTE’s

- work_mem – intermediate results
- enable_material - not related to CTE!
- Planner costing parameters

Advantages of using CTEs

Advantages of using CTEs

Using a sample table…

```
postgres=> \d orders
                       Table "public.orders"
  Column  |         Type          | Collation | Nullable | Default
----------+-----------------------+-----------+----------+---------
 region   | bigint                |           |          |
 product  | character varying(50) |           |          |
 quantity | bigint                |           |          |
 amount   | bigint                |           |          |
```

Advantages of using CTEs

Building a sample report

```
 region | product    | product_units | product_sales
--------+------------+---------------+----------------
      3 | blue shoe|              20 |            200
      3 | widget     |              7 |             77
      1 | widget     |              3 |             33
      1 | blue shoe|              15 |            150
      1 | tumbler    |              3 |            6
      4 | dice       |            100 |            100
      4 | tumbler    |              9 |            198

(7 rows)
```

Advantages of using CTEs

Readability

```
WITH regional_sales AS (

    SELECT region, SUM(amount) AS total_sales
    FROM orders
    GROUP BY region ),

    top_regions AS (

    SELECT region
    FROM regional_sales
    WHERE total_sales > (SELECT SUM(total_sales)/10 FROM regional_sales) )

SELECT region, product, SUM(quantity) AS product_units, SUM(amount) AS
product_sales
FROM orders
WHERE region IN (SELECT region FROM top_regions) GROUP BY region, product;
```

Advantages of using CTEs

Readability

```
SELECT region,
       product,
       SUM(quantity) AS product_units,
       SUM(amount) AS product_sales
FROM orders
WHERE region IN
  (
    SELECT region
    FROM orders
    GROUP BY region
    HAVING sum(amount) >
      (
        SELECT SUM(amount)/10
        FROM orders
      )
  )
GROUP BY region, product;
```

Advantages of using CTEs

Reusability

```
WITH regional_sales AS (
    SELECT region, SUM(amount) AS total_sales
    FROM orders
    GROUP BY region ),
    top_regions AS (
    SELECT region
    FROM regional_sales
    WHERE total_sales > (SELECT SUM(total_sales)/10 FROM regional_sales) )
SELECT region, product, SUM(quantity) AS product_units, SUM(amount) AS
product_sales
FROM orders
WHERE region IN (SELECT region FROM top_regions) GROUP BY region, product;
```

Advantages of using CTEs

Recursive

```
postgres=> WITH RECURSIVE t(n) AS (
SELECT 1
UNION ALL
SELECT n+1 FROM t WHERE n+1 <= 5
)
SELECT n FROM t;
n
---
 1
 2
 3
 4
 5
(5 rows)
```

Non-recursive part

Recursive part (with limit)

Advantages of using CTEs

Recursive (beware without termination)

```
 QUERY PLAN
---------------------------------------------------------------------------
 CTE Scan on t  (cost=4.28..6.30 rows=101 width=4)
   CTE t
     ->  Recursive Union  (cost=0.00..4.28 rows=101 width=4)
           ->  Result  (cost=0.00..0.01 rows=1 width=4)
           ->  WorkTable Scan on t t_1  (cost=0.00..0.23 rows=10 width=4)
(5 rows)
```

Not terminated

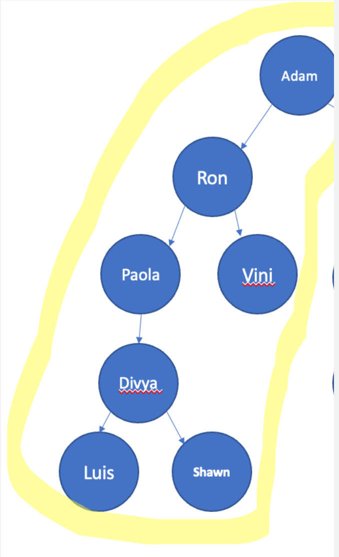
```
 QUERY PLAN
---------------------------------------------------------------------------
 Limit  (cost=4.28..4.38 rows=5 width=4)
   CTE t
     ->  Recursi
           ->  R
           ->  W
   ->  CTE Scan
(6 rows)
```

```
 QUERY PLAN
---------------------------------------------------------------------------
 CTE Scan on t  (cost=3.21..3.83 rows=31 width=4)
   CTE t
     ->  Recursive Union  (cost=0.00..3.21 rows=31 width=4)
           ->  Result  (cost=0.00..0.01 rows=1 width=4)
           ->  WorkTable Scan on t t_1  (cost=0.00..0.26 rows=3 width=4)
                 Filter: ((n + 1) <= 5)
(6 rows)
```

Limit or
Where clause

aws

Recursive Hierarchy




```
postgres=> select * from employees;
 id |   name    | salary |        job         | manager_id
----+-----------+--------+--------------------+------------
  1 | Adam      |  10000 | CEO                |
  4 | Divya     |   1800 | Manager            |          5
  6 | Vinicius  |   2000 | Database Engineer  |          7
  2 | Luis      |   1400 | Senior Developer   |          4
  5 | Paola     |   4000 | CTO                |          7
  3 | Shawn     |    500 | Developer          |          4
  7 | Ron       |   5000 | Vice President     |          1
(7 rows)

postgres=> WITH RECURSIVE managers AS (
   SELECT id, name, manager_id, job, 1 AS level
   FROM employees
   WHERE id = 1
UNION
   SELECT e.id, e.name, e.manager_id, e.job, managers.level + 1 AS level
   FROM employees e
   JOIN managers ON e.manager_id = managers.id
)
SELECT * FROM managers;
 id |   name    | manager_id |        job         | level
----+-----------+------------+--------------------+-------
  1 | Adam      |            | CEO                |     1
  7 | Ron       |          1 | Vice President     |     2
  6 | Vinicius  |          7 | Database Engineer  |     3
  5 | Paola     |          7 | CTO                |     3
  4 | Divya     |          5 | Manager            |     4
  2 | Luis      |          4 | Senior Developer   |     5
  3 | Shawn     |          4 | Developer          |     5
(7 rows)
```

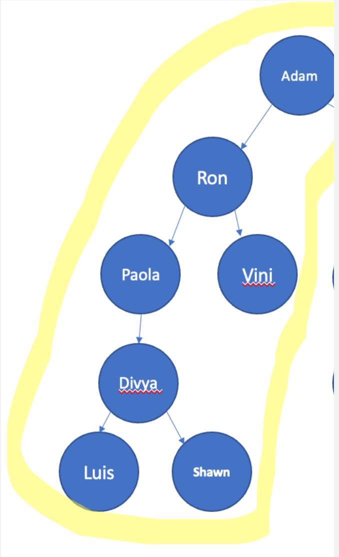
Non-Recursive part

Recursive part

Recursive Hierarchy

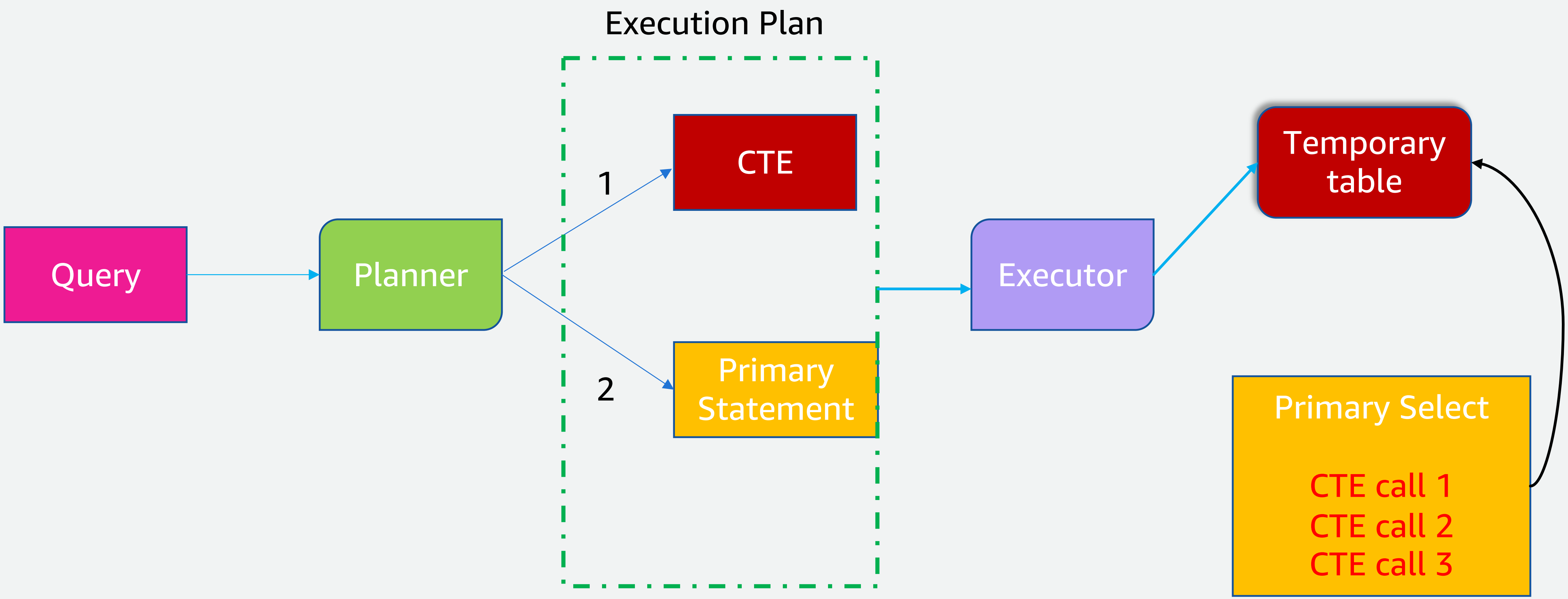
Postgres 14 feature

```
postgres=> WITH RECURSIVE managers AS (
   SELECT id, name, manager_id, job, 1 AS level
   FROM employees
   WHERE id = 1
UNION
   SELECT e.id, e.name, e.manager_id, e.job, managers.level + 1 AS level
   FROM employees e
   JOIN managers ON e.manager_id = managers.id
) SEARCH DEPTH FIRST BY id SET tree
SELECT * FROM managers;



id |   name    | manager_id |        job         | level |         tree
----+-----------+------------+--------------------+-------+-----------------------
  1 | Adam      |            | CEO                |     1 | {(1)}
  7 | Ron       |          1 | Vice President     |     2 | {(1),(7)}
  5 | Paola     |          7 | CTO                |     3 | {(1),(7),(5)}
  6 | Vinicius  |          7 | Database Engineer  |     3 | {(1),(7),(6)}
  4 | Divya     |          5 | Manager            |     4 | {(1),(7),(5),(4)}
  3 | Shawn     |          4 | Developer          |     5 | {(1),(7),(5),(4),(3)}
  2 | Luis      |          4 | Senior Developer   |     5 | {(1),(7),(5),(4),(2)}
(7 rows)
```



What is “Materialized” for CTEs?

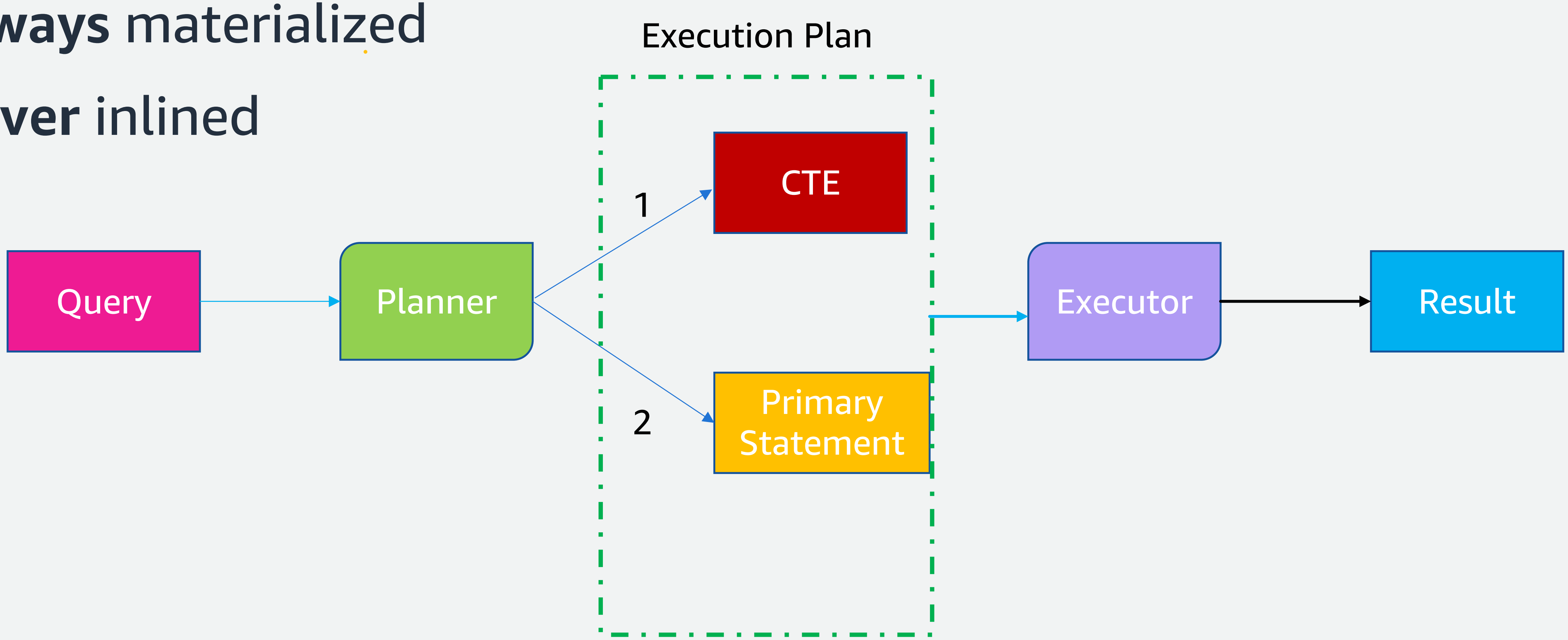
What's "Materialized" for CTEs?



CTE computation before version 12

Always materialized

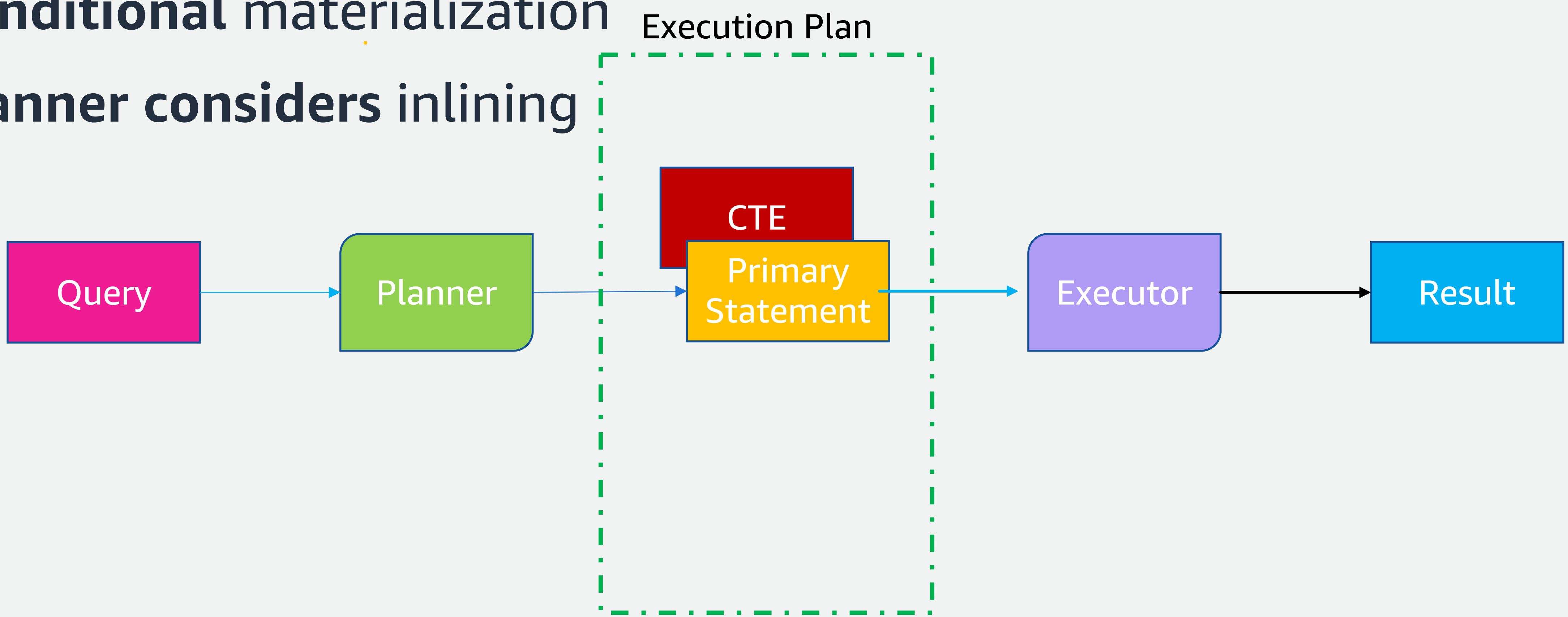
Never inlined



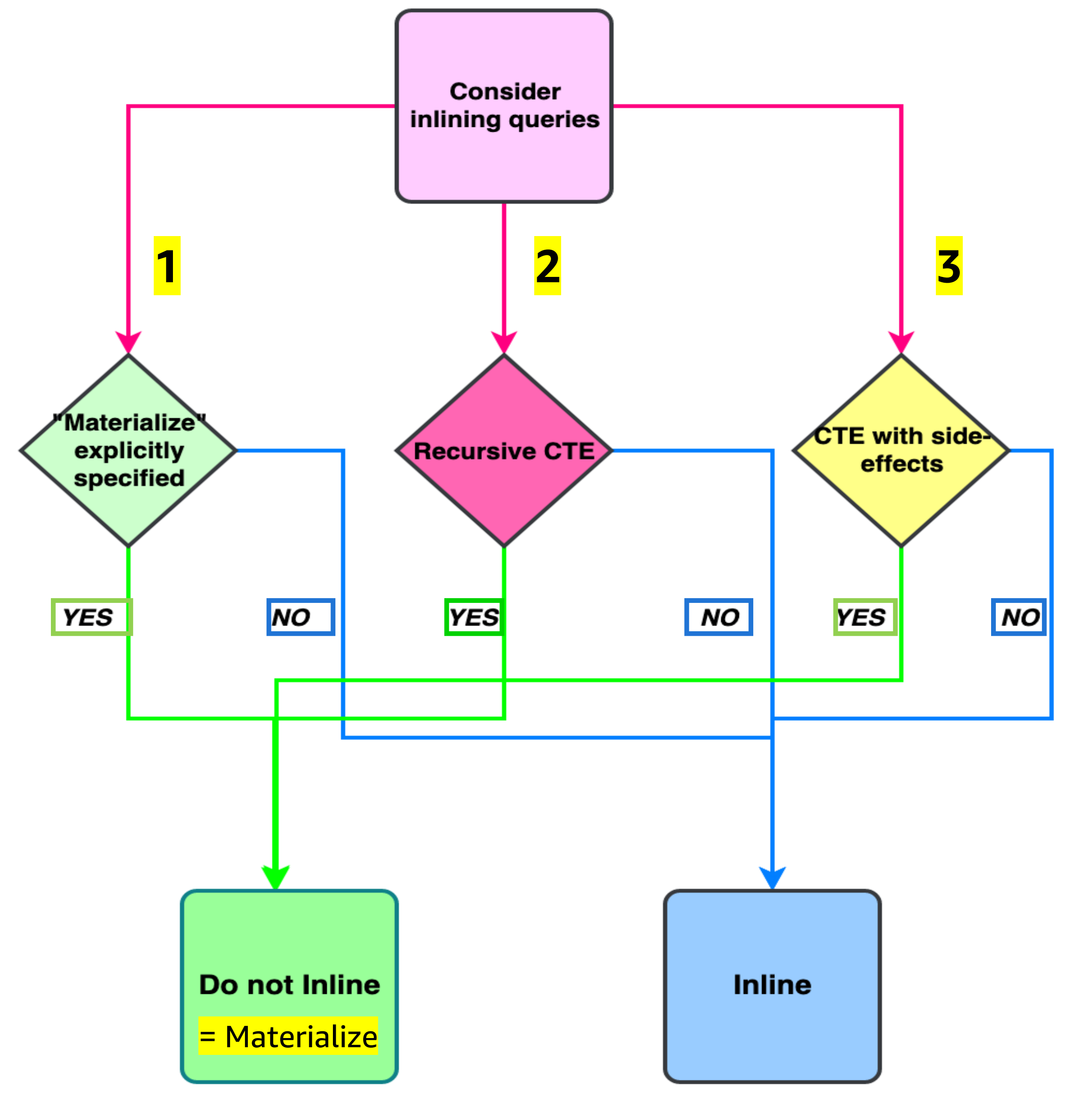
CTE computation in version 12+

Conditional materialization

Planner considers inlining



CTE computation in version 12+



CTE computation version 12+ : Explicitly mention "Materialize"

```
postgres=> create table big_table as select s, md5(random()::text) from generate_Series(1,1000000) s;
SELECT 1000000
postgres=>
postgres=>
```

```
postgres=> Explain Analyze  WITH w AS MATERIALIZED (
    SELECT * FROM big_table
)
SELECT * FROM w WHERE s=10;
                                                    QUERY PLAN
-------------------------------------------------------------------------------------------------------------------------------
 CTE Scan on w  (cost=18334.00..40834.00 rows=5000 width=36) (actual time=0.013..395.725 rows=1 loops=1)
   Filter: (s = 10)
   Rows Removed by Filter: 999999
   CTE w
     ->  Seq Scan on big_table  (cost=0.00..18334.00 rows=1000000 width=37) (actual time=0.006..98.871 rows=1000000 loops=1)
 Planning Time: 0.053 ms
 Execution Time: 404.407 ms
(7 rows)
```

CTE computation version 12+ : Explicitly mention "Materialize"

Important : Know your indexes!

```
postgres=> Explain Analyze  WITH w AS (
    SELECT * FROM big_table
)
SELECT * FROM w WHERE s=10;
                                                        QUERY PLAN
------------------------------------------------------------------------------------------------------------------------
 Index Scan using big_table_s_idx on big_table  (cost=0.42..8.44 rows=1 width=37) (actual time=0.062..0.062 rows=1 loops=1)
   Index Cond: (s = 10)
 Planning Time: 0.166 ms
 Execution Time: 0.075 ms
(4 rows)
```

```
postgres=> Explain Analyze  WITH w AS MATERIALIZED (
    SELECT * FROM big_table
)
SELECT * FROM w WHERE s=10;
                                                        QUERY PLAN
------------------------------------------------------------------------------------------------------------------------
 CTE Scan on w  (cost=18334.00..40834.00 rows=5000 width=36) (actual time=0.013..395.725 rows=1 loops=1)
   Filter: (s = 10)
   Rows Removed by Filter: 999999
   CTE w
     ->  Seq Scan on big_table  (cost=0.00..18334.00 rows=1000000 width=37) (actual time=0.006..98.871 rows=1000000 loops=1)
 Planning Time: 0.053 ms
 Execution Time: 404.407 ms
(7 rows)
```

CTE computation version 12+ Multiple Reference

```
postgres=> explain
WITH pgc_cte AS (
SELECT * FROM pg_class
)
SELECT * FROM pgc_cte AS pgc_cte1
JOIN pgc_cte AS pgc_cte2 ON pgc_cte1.relname = pgc_cte2.relname
WHERE pgc_cte2.relname = 'pg_class';
                              QUERY PLAN
--------------------------------------------------------------------------
 Nested Loop  (cost=17.60..38.36 rows=4 width=472)
   CTE pgc_cte
     ->  Seq Scan on pg_class  (cost=0.00..17.60 rows=460 width=279)
   ->  CTE Scan on pgc_cte pgc_cte1  (cost=0.00..10.35 rows=2 width=236)
         Filter: (relname = 'pg_class'::name)
   ->  CTE Scan on pgc_cte pgc_cte2  (cost=0.00..10.35 rows=2 width=236)
         Filter: (relname = 'pg_class'::name)
(7 rows)
```

CTE computation version 12+ : Multiple Reference

But…we can use NOT MATERIALIZED

```
postgres=> explain
WITH pgc_cte AS NOT MATERIALIZED (
SELECT * FROM pg_class
)
SELECT * FROM pgc_cte AS pgc_cte1
JOIN pgc_cte AS pgc_cte2 ON pgc_cte1.relname = pgc_cte2.relname
WHERE pgc_cte2.relname = 'pg_class';
                                                   QUERY PLAN
----------------------------------------------------------------------------------------------------
 Nested Loop  (cost=0.55..16.59 rows=1 width=558)
   ->  Index Scan using pg_class_relname_nsp_index on pg_class  (cost=0.27..8.29 rows=1 width=279)
         Index Cond: (relname = 'pg_class'::name)
   ->  Index Scan using pg_class_relname_nsp_index on pg_class pg_class_1  (cost=0.27..8.29 rows=1
width=279)
         Index Cond: (relname = 'pg_class'::name)
(5 rows)
```

CTE computation version 12+ : Inlining

```
postgres=> explain analyze
postgres-> WITH CTE_emp AS (
postgres(>     SELECT manager_id, employees.name FROM employees GROUP BY manager_id, employees.name
postgres(> )
postgres-> SELECT * FROM CTE_emp WHERE manager_id = 4;
                                              QUERY PLAN
-------------------------------------------------------------------------------------------------------------------------
 CTE Scan on cte_emp  (cost=11.75..13.32 rows=1 width=520) (actual time=0.022..0.026 rows=2 loops=1)
   Filter: (manager_id = 4)
   Rows Removed by Filter: 5
   CTE cte_emp
     ->  HashAggregate  (cost=11.05..11.75 rows=70 width=520) (actual time=0.021..0.022 rows=7 loops=1)
           Group Key: employees.manager_id, employees.name
           ->  Seq Scan on employees  (cost=0.00..10.70 rows=70 width=520) (actual time=0.012..0.013 rows=7 loops=1)
 Planning time: 0.090 ms
 Execution time: 0.108 ms
(9 rows)
```

Version 11

```
postgres=> explain analyze
postgres-> WITH CTE_emp AS (
postgres(>     SELECT manager_id, employees.name FROM employees GROUP BY manager_id, employees.name
postgres(> )
postgres-> SELECT * FROM CTE_emp WHERE manager_id = 4;
                                              QUERY PLAN
-------------------------------------------------------------------------------------------------------------------------
 Group  (cost=10.88..10.90 rows=1 width=520) (actual time=0.020..0.022 rows=2 loops=1)
   Group Key: employees.manager_id, employees.name
   ->  Sort  (cost=10.88..10.89 rows=1 width=520) (actual time=0.019..0.020 rows=2 loops=1)
         Sort Key: employees.name
         Sort Method: quicksort  Memory: 25kB
         ->  Seq Scan on employees  (cost=0.00..10.88 rows=1 width=520) (actual time=0.008..0.009 rows=2 loops=1)
               Filter: (manager_id = 4)
               Rows Removed by Filter: 5
 Planning Time: 0.093 ms
 Execution Time: 0.041 ms
(10 rows)
```

Version 12

Usage and impact of Materialized CTEs

Usage and Impact of Materializing CTEs

- It can avoid duplicate computation of an expensive WITH query
- Act as an “optimization fence” – black box to the planner
- Materialization itself can be a bit costly
- Indexes cannot be used efficiently after Materializing CTEs

Going back to the customer question

```
with cte_items as (
select distinct c.p_id as p_id, ps.ps_sk as ps_sk from ra.staging_nd_mk_c c join
ra_report.v_ps ps on ps.p_id::integer=c.cm_id and c.p_cdr='NATIONAL'
)
Select
```

Version 13 - Without "Materialized" keyword

```
with cte_items as MATERIALIZED (
select distinct c.p_id as p_id, ps.ps_sk as ps_sk from ra.staging_nd_mk_c c join ra_report.v_ps
ps on ps.p_id::integer=c.cm_id and c.p_cdr='NATIONAL'
)
Select
```

Version 13 – With "Materialize" keyword

PG13 execution time without "MATERIALIZED" : 48s
PG13 execution time with "MATERIALIZED" : 3m 28s

Sample Uses of CTEs

Autovacuum Eligible

```
WITH vbt AS (SELECT setting AS autovacuum_vacuum_threshold FROM pg_settings WHERE name =
'autovacuum_vacuum_threshold'),
     vsf AS (SELECT setting AS autovacuum_vacuum_scale_factor FROM pg_settings WHERE name =
'autovacuum_vacuum_scale_factor'),
     fma AS (SELECT setting AS autovacuum_freeze_max_age FROM pg_settings WHERE name = 'autovacuum_freeze_max_age'),

sto AS (select opt_oid, split_part(setting, '=', 1) as param,split_part(setting, '=', 2) as value from (select oid
opt_oid,unnest(reloptions) setting from pg_class) opt)
SELECT '"'||ns.nspname||'"."'||c.relname||'"' as relation,pg_size_pretty(pg_table_size(c.oid)) as
table_size,age(relfrozenxid) as xid_age, coalesce(cfma.value::float, autovacuum_freeze_max_age::float)
autovacuum_freeze_max_age,
(coalesce(cvbt.value::float, autovacuum_vacuum_threshold::float) +
coalesce(cvsf.value::float,autovacuum_vacuum_scale_factor::float) * c.reltuples)
AS autovacuum_vacuum_tuples, n_dead_tup as dead_tuples FROM
pg_class c join pg_namespace ns on ns.oid = c.relnamespace
join pg_stat_all_tables stat on stat.relid = c.oid join vbt on (1=1) join vsf on (1=1) join fma on (1=1)
left join sto cvbt on cvbt.param = 'autovacuum_vacuum_threshold' and c.oid = cvbt.opt_oid
left join sto cvsf on cvsf.param = 'autovacuum_vacuum_scale_factor' and c.oid = cvsf.opt_oid
left join sto cfma on cfma.param = 'autovacuum_freeze_max_age' and c.oid = cfma.opt_oid
WHERE c.relkind = 'r' and nspname <> 'pg_catalog'
AND (age(relfrozenxid) >= coalesce(cfma.value::float, autovacuum_freeze_max_age::float)
OR coalesce(cvbt.value::float, autovacuum_vacuum_threshold::float) +
coalesce(cvsf.value::float,autovacuum_vacuum_scale_factor::float) *
c.reltuples <= n_dead_tup)

ORDER BY age(relfrozenxid) DESC LIMIT 10;
```

https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/Appendix.PostgreSQL.CommonDBATasks.Autovacuum.html

Autovacuum Eligible

(cont.)

```
  relation                                          | table_size | xid_age | autovacuum_freeze_max_age | autovacuum_vacuum_tuples | dead_tuples
--------------------------------------------------+------------+---------+---------------------------+--------------------------+-------------
 "public"."employees"                             | 8192 bytes |     634 |                 200000000 |                    869.2 |           0
 "public"."mytable"                               | 2888 kB    |     574 |                 200000000 |                 295781.2 |       81329
 "public"."spatial_ref_sys"                       | 6976 kB    |     572 |                 200000000 |                 714392.4 |           0
 "hint_plan"."hints"                              | 8192 bytes |     572 |                 200000000 |                    869.2 |           0
(4 rows)
```

XID Wrap Estimate

```
WITH max_age AS ( SELECT 2000000000 as max_old_xid , setting AS
autovacuum_freeze_max_age FROM pg_catalog.pg_settings
WHERE name = 'autovacuum_freeze_max_age' ) ,

per_database_stats AS ( SELECT datname , m.max_old_xid::int ,
m.autovacuum_freeze_max_age::int , age(d.datfrozenxid) AS oldest_xid
FROM pg_catalog.pg_database d JOIN max_age m ON (true) WHERE
d.datallowconn )

SELECT max(oldest_xid) AS oldest_xid ,
max(ROUND(100*(oldest_xid/max_old_xid::float))) AS
percent_towards_wraparound
 , max(ROUND(100*(oldest_xid/autovacuum_freeze_max_age::float))) AS
percent_towards_emergency_autovac
 FROM per_database_stats ;
```

https://github.com/awslabs/pg-collector/blob/main/pg_collector.sql#L212

XID Wrap Estimate

(cont.)

```
 oldest_xid | percent_towards_wraparound | percent_towards_emergency_autovac
------------+----------------------------+-----------------------------------
  166708621 |                          8 |                                83
(1 row)
```

Index Size and Info

```
WITH index_size_info as
(
SELECT
schemaname,relname as "Table",
indexrelname AS indexname,
indexrelid,
pg_relation_size(indexrelid) index_size_byte,
pg_size_pretty(pg_relation_size(indexrelid)) AS index_size
FROM pg_catalog.pg_statio_all_indexes  ORDER BY 1,4 desc)
Select a.schemaname,
a.relname as "Table_Name",
a.indexrelname AS indexname,
b.index_size,
a.idx_scan,
a.idx_tup_read,
a.idx_tup_fetch
from pg_stat_all_indexes a ,  index_size_info b
where a.idx_scan >0
and a.indexrelid=b.indexrelid
and a.schemaname not in ('pg_catalog')
order by b.index_size_byte desc,a.idx_scan asc ;
```

https://github.com/awslabs/pg-collector/blob/main/pg_collector.sql#L1172

Index Size and Info

(cont.)

schemaname	Table_Name	indexname	index_size	idx_scan	idx_tup_read	idx_tup_fetch
public	big_table	big_table_s_idx	21 MB	1	1	1
pg_toast	pg_toast_432944	pg_toast_432944_index	1936 kB	136078	128136	128136
public	mytable	mytable_id	1336 kB	3	20008	4
pg_toast	pg_toast_430903	pg_toast_430903_index	104 kB	2005	10	10
public	employees	employee_idx	16 kB	1	1	1
pg_toast	pg_toast_3079	pg_toast_3079_index	16 kB	1	0	0
pg_toast	pg_toast_1255	pg_toast_1255_index	16 kB	15	20	20
public	blog	blog_pkey	16 kB	94	4	4
pg_toast	pg_toast_2619	pg_toast_2619_index	16 kB	194	247	247
pg_toast	pg_toast_2618	pg_toast_2618_index	16 kB	234	773	773

(10 rows)

Should I use CTE's?

- Will a subquery be worse?
- Will using temp tables be better?
- Will MATERIALIZE help or hurt?
- Will a view/materialized view be better?

As with most tuning decisions….it depends!

CTE's continue to improve

All of this is great. I'm kinda uneasy about the fact that by default CTEs will be run in NOT MATERIALIZED way, and if you want to preserve older way of working, you have to modify your queries. But – it's definitely a progress, so I can't really complain.

Thanks to all involved, great work.

- Hubert "depesz" Lubaczewski

https://www.depesz.com/2019/02/19/waiting-for-postgresql-12-allow-user-control-of-cte-materialization-and-change-the-default-behavior/

Thank you!