Recursive CTE in GPDB

嘉宾：苑海胜

公司：Pivotal Software Inc.

Who am I

苑海胜

Joined Pivotal at 10/2015

Staff software engineer

Team lead of query processing team

What is CTE?

Common Table Expression

A common table expression (CTE) can be thought of as a temporary result set that is defined within the execution scope of a single SELECT, INSERT, UPDATE, DELETE, or CREATE VIEW statement.

```
WITH cte AS (SELECT a, b FROM foo)
SELECT * FROM cte WHERE a > 0;
```

How does Postgres implement CTE?

WITH cte AS (SELECT a, b FROM foo)
SELECT * FROM cte as t1, cte as t2
WHERE t1.a = t2.b;

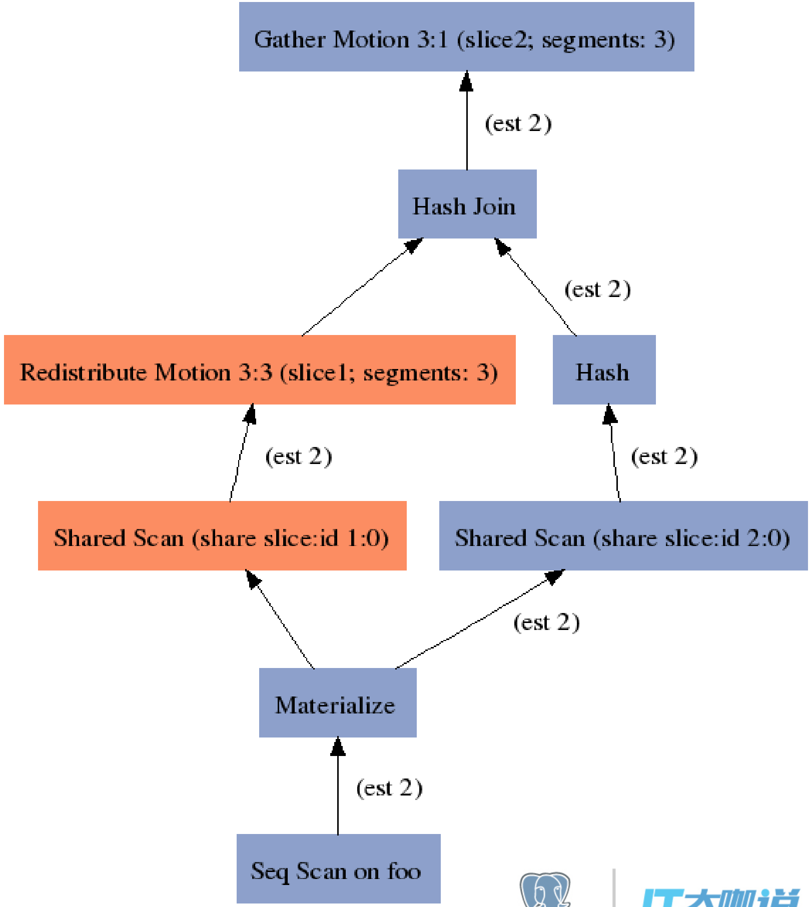
```
                            QUERY PLAN
------------------------------------------------------------------
 Hash Join  (cost=1.21..1.38 rows=5 width=16)
   Hash Cond: (t1.a = t2.b)
   CTE cte
     ->  Seq Scan on foo  (cost=0                        )
   ->  CTE Scan on cte t1  (cost=0.00..0.10 rows=5 width=8)
   ->  Hash  (cost=0.10..0.10 rows=5 width=8)
         ->  CTE Scan on cte t2  (cost=0.00..0.10 rows=5 width=8)
(7 rows)
```

Init Plan (CTE Definition)

How does GPDB impl

WITH cte AS (SELECT a, b FROM foo)
SELECT * FROM cte as t1, cte as t2 WHERE t1.a = t2.b;

```
                             QUERY PLAN
---------------------------------------------------
 Gather Motion 3:1  (slice2; segments: 3)  (cost=0.1
   ->  Hash Join  (cost=0.16..0.44 rows=2 width=16)
         Hash Cond: share0_ref2.b = share0_ref1.a
         ->  Redistribute Motion 3:3  (slice1; segm
               Hash Key: share0_ref2.b
               ->  Shared Scan (share slice:id 1:0)
         ->  Hash  (cost=0.10..0.10 rows=2 width=8)
               ->  Shared Scan (share slice:id 2:0)
                     ->  Materialize  (cost=2.05..2
                           ->  Seq Scan on foo  (co
 Settings:  gp_cte_sharing=on; optimizer=off
```


Gather Motion 3:1 (slice2; segments: 3)
(est 2)
Hash Join
(est 2)
Redistribute Motion 3:3 (slice1; segments: 3)
Hash
(est 2)
(est 2)
Shared Scan (share slice:id 1:0)
Shared Scan (share slice:id 2:0)
(est 2)
Materialize
(est 2)
Seq Scan on foo

What is the difference?

- Inlining CTE

  GPDB always inline CTE where there is only 1 reference

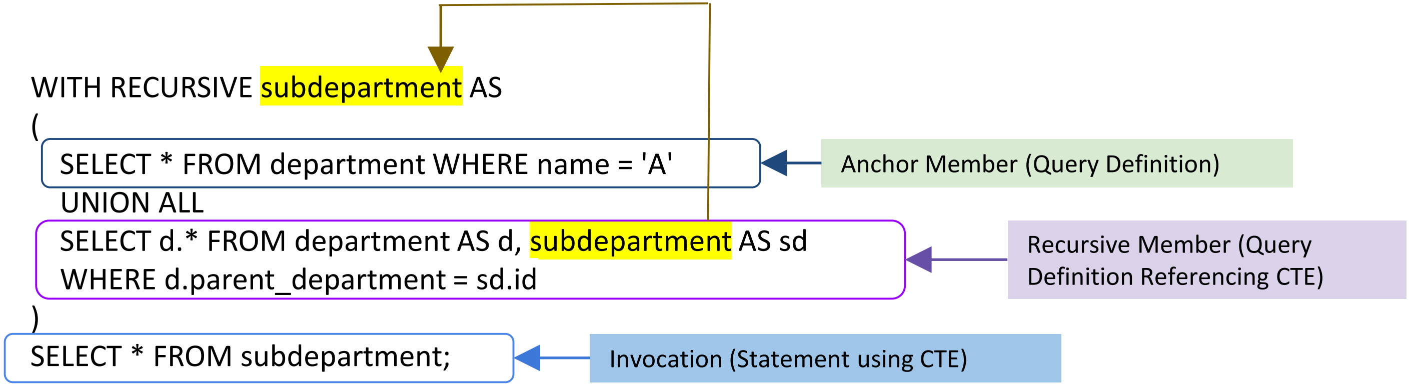
  E.g. Limit on CTE

- Predicate pushdown (Orca only)

$$\sigma_{a=1}(CTE) \text{ and } \sigma_{a=2}(CTE) \rightarrow \sigma_{a=1 \text{ or } a=2}(CTE)$$

What is Recursive CTE?

- Recursive CTEs are special in the sense they are allowed to reference themselves!
- Recursive CTEs are really good at working with hierarchical data such as org charts for bill of materials.



```
WITH RECURSIVE subdepartment AS
(
  SELECT * FROM department WHERE name = 'A'
  UNION ALL
  SELECT d.* FROM department AS d, subdepartment AS sd
  WHERE d.parent_department = sd.id
)
SELECT * FROM subdepartment;
```

Anchor Member (Query Definition)

Recursive Member (Query Definition Referencing CTE)

Invocation (Statement using CTE)

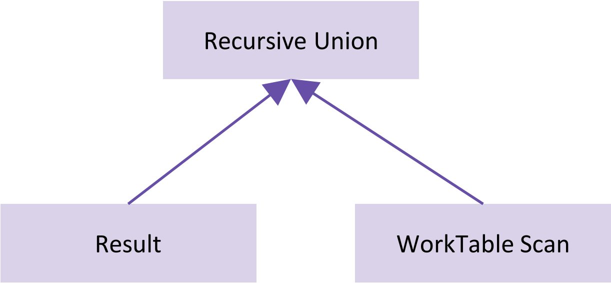
How does Postgres implement recursive CTE?

```
WITH RECURSIVE t(n) AS (
    SELECT 1
  UNION ALL
    SELECT n+1 FROM t WHERE n < 5
)
SELECT * FROM t;
```

```
                            QUERY PLAN
---------------------------------------------------------------------
 CTE Scan on t  (cost=2.95..3.57 rows=31 width=4)
   CTE t
     ->  Recursive Union  (cost=0.00..2.95 rows=31 width=4)
           ->  Result  (cost=0.00..0.01 rows=1 width=0)
           ->  WorkTable Scan on t  (cost=0.00..0.23 rows=3 width=4)
                 Filter: (t.n < 5)
```

How does recursive CTE work?

```
WITH RECURSIVE t(n) AS (
        SELECT 1
    UNION ALL
        SELECT n+1 FROM t WHERE n < 5
)
SELECT * FROM t;
```



1). RT = {1}, OUT = {1}

2). WT = RT = {1}, RT = {}

3). WT = {1}, RT = {2}, OUT = {1, 2}

4). WT = RT = {2}, RT = {}

5). WT = {2}, RT = {3}, OUT = {1, 2, 3}

6). WT = RT = {3}, RT = {}

7). WT = {3}, RT = {4}, OUT = {1, 2, 3, 4}

8). WT = RT = {4}, RT = {}

9). WT = {4}, RT = {5}, OUT = {1, 2, 3, 4, 5}

10). WT = RT = {5}, RT = {}

11). WT = {}, RT = {}, OUT = {1, 2, 3, 4, 5}

Another Recursive CTE Example

```
CREATE TABLE department (
  id INT PRIMARY KEY,
  parent_department INT REFERENCES department,
  name TEXT
);

INSERT INTO department VALUES (0, NULL, 'ROOT');
INSERT INTO department VALUES (1, 0, 'A');
INSERT INTO department VALUES (2, 1, 'B');
INSERT INTO department VALUES (3, 2, 'C');
INSERT INTO department VALUES (4, 2, 'D');
INSERT INTO department VALUES (5, 0, 'E');
INSERT INTO department VALUES (       6, 4, 'F');
INSERT INTO department VALUES (7, 4, 'G');
```

This will represent a tree structure of an organization:

```
ROOT ---> A ---> B ---> C ---> F
|                |
|                +----> D
|
+-----> E ---> G
```

Another Recursive CTE Example

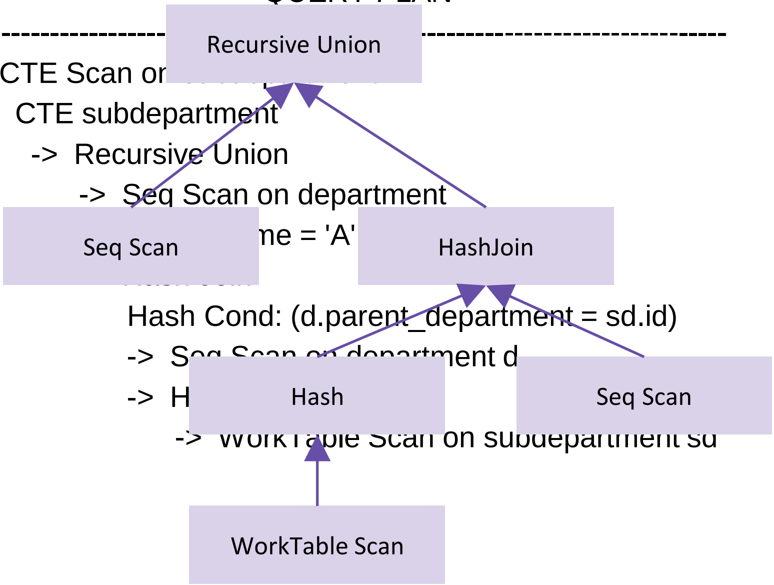
```
WITH RECURSIVE subdepartment AS
(
        -- non recursive term
        SELECT name as root_name, * FROM department
          WHERE name = 'A'

        UNION ALL

        -- recursive term
        SELECT sd.root_name, d.* FROM department AS d,
                     subdepartment AS sd
          WHERE d.parent_department = sd.id
)
SELECT * FROM subdepartment;
```

```
                         QUERY PLAN
--------------------                    --------------------------------
CTE Scan or
  CTE subdepartment
    ->  Recursive Union
          ->  Seq Scan on department
                        me = 'A'

                Hash Cond: (d.parent_department = sd.id)
                ->  Seq Scan on department d
                ->  H
                      ->  WorkTable Scan on subdepartment sd
```



What is wrong in MPP environment?

1). Recursive Union operator is rescan driven.

2). Recursive Union and WorkTable Scan share tuple store.

3). Motion is not rescannable!

```
Gather Motion 3:1  (slice2; segments: 3)
  ->  Recursive Union
        ->  Seq Scan on department
              Filter: name = ’A’::text
        ->  Nested Loop
              Join Filter: d.parent_department = sd.id
              -> Seq Scan on department d
              ->  Materialize
                    ->  Broadcast Motion 3:3  (slice1; segments: 3)
                          ->  WorkTable Scan on subdepartment sd
```

How to make it work in GPDB?

Don’t generate plan that has motion between WorkTableScan and RecursiveUnion.

1). Always gather on master

2). Always broadcast non-worktablescan side of join in recursive member.

```
Gather Motion 3:1  (slice2; segments: 3)
  ->  Recursive Union
        ->  Seq Scan on department
              Filter: name = 'A'::text
        ->  Nested Loop
              Join Filter: d.parent_department = sd.id
              -> WorkTable Scan on subdepartment sd
              ->  Materialize
                    ->  Broadcast Motion 3:3  (slice1; segments: 3)
                          ->  Seq Scan on department d
```

Another problem

When do we put WorkTableScan on outer or inner side of Join?

WorkTableScan on outer side of Join

hash table in Hash node will materialize the broadcast motion.

for next recursion, just need to rescan WTS, no need to rebuild hash table.

cost of building hash table on broadcast motion + number of recursion * average cost of WorkTableScan

```
Gather Motion 3:1
  ->  Recursive Union
        ->  Seq Scan on department  --- non-recursive part
              Filter: name = 'A'::text
        ->  Hash Join                --- recursive part
              Hash Cond: sd.id = d.parent_department
              ->  WorkTable Scan on subdepartment sd
              ->  Hash
                    ->  Broadcast Motion 3:3
                          ->  Seq Scan on department d
```

WorkTableScan on inner side of Join

materialize the broadcast motion on the outer side.

rebuild hash table on WTS for every recursion.

cost of materializing broadcast motion + number of recursion * (average cost of WorkTableScan + average cost of building hash table on WorkTableScan + cost of scanning materialize of the motion)

```
Gather Motion 3:1
    ->  Recursive Union
        ->  Seq Scan on department  --- non-recursive part
            Filter: name = 'A'::text
        ->  Hash Join               --- recursive part
            Hash Cond: sd.id = d.parent_department
            ->  Materialize
                ->  Broadcast Motion 3:3
                    ->  Seq Scan on department d
            ->  Hash
                WorkTable Scan on subdepartment sd
```

Thanks!