CTE support in CMU-DB

Project Presentation

Group XI
Members: Rohan Aggarwal, Gautam Jain, Preetansh Goyal
Mentor: Andy Pavlo

Problem Statement

In the current cascade-style query optimizer implementation of CMU-DB, add functionality of Common Table Expressions (CTE's) to support TPC-DS queries.

The goals for the project were:

1. Implement functionality of Non-Recursive CTE's and add support to query optimizer
2. Support TPC-DS like queries

CTEs (Common Table Expressions)

- CTEs help define temporary result which can be referenced later in a complex query
- Are alternatives to using nested queries or views
- Defined within a statement using the WITH operator

```
WITH EMPLOYEE AS
    (SELECT ID,NAME,AGE
     FROM COMPANY)
SELECT NAME, AGE
FROM EMPLOYEE;
```

Defining temp table using CTE

Referring to table in query

Source: https://www.essentialsql.com/introduction-common-table-expressions-ctes/

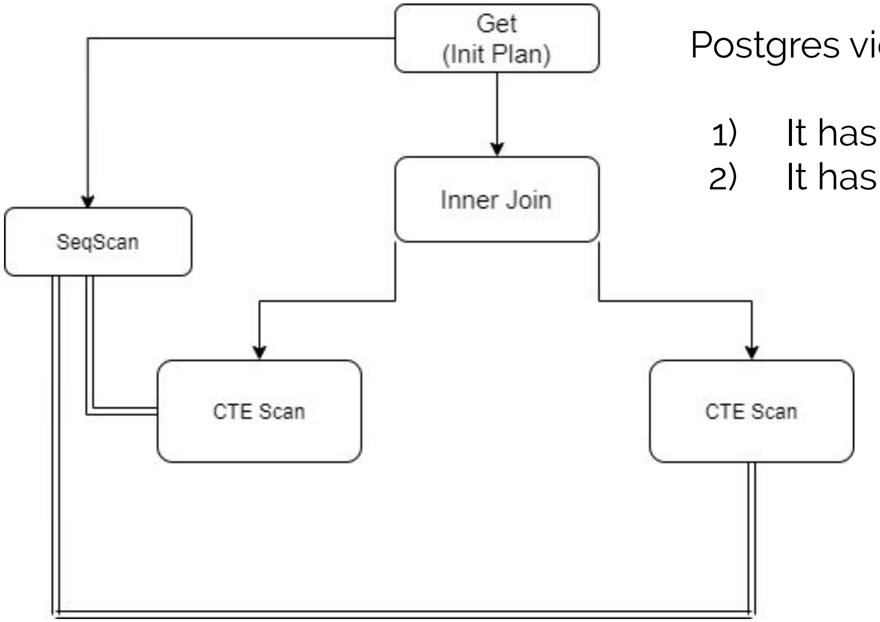
DEMO

We will demonstrate our progress:

1. Implementation of Non-Recursive CTE's
2. Support for TPC-DS like queries

PostgreSQL approach

WITH EMPLOYEE AS (SELECT NAME,AGE, SALARY FROM COMPANY)
SELECT E1.AGE,E2.SALARY FROM EMPLOYEE AS E1, EMPLOYEE AS E2
WHERE E1.NAME = E2.NAME;



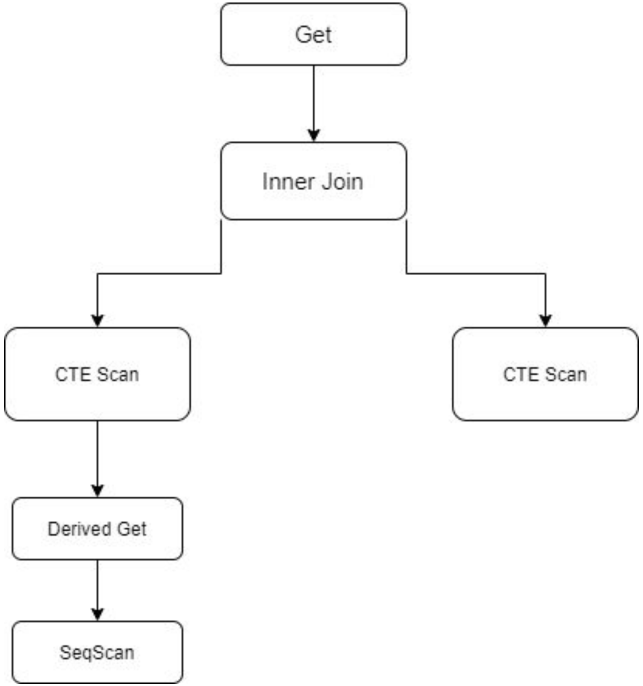
Postgres violates the Volcano Model.

1) It has child nodes on which it doesn't call Next.
2) It has child nodes which calls next on non - descendants.

PROBLEM:

Terrier uses a different system, how to get correct behaviour in the new setting ?

Optimizer (Logical Plan)



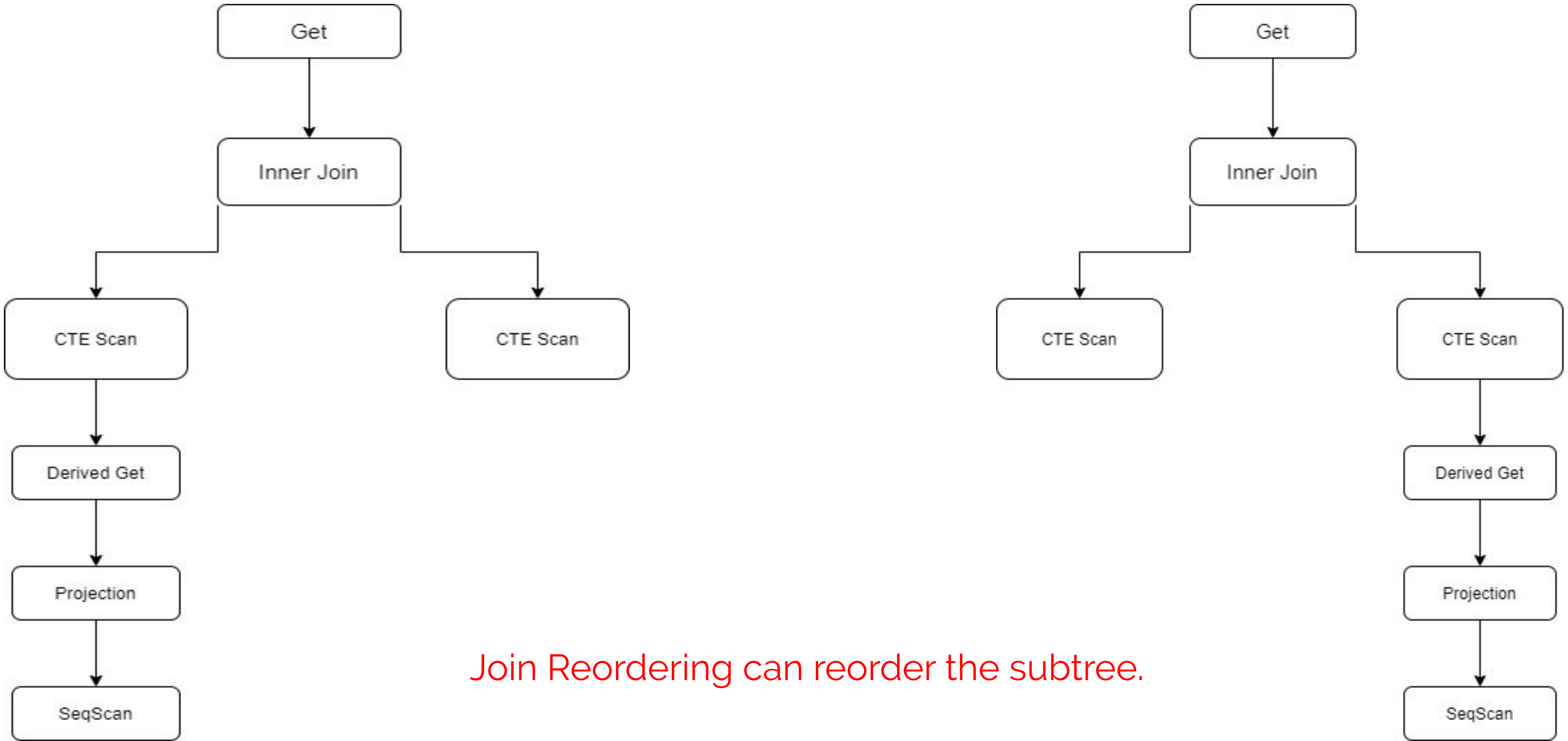
WITH EMPLOYEE AS (SELECT NAME,AGE, SALARY FROM COMPANY) SELECT E1.AGE,E2.SALARY FROM EMPLOYEE AS E1, EMPLOYEE AS E2 WHERE E1.NAME = E2.NAME;

Where to connect the sub tree ?

1. All CTE nodes
2. Pick any one
3. Choose some specific one, first or last

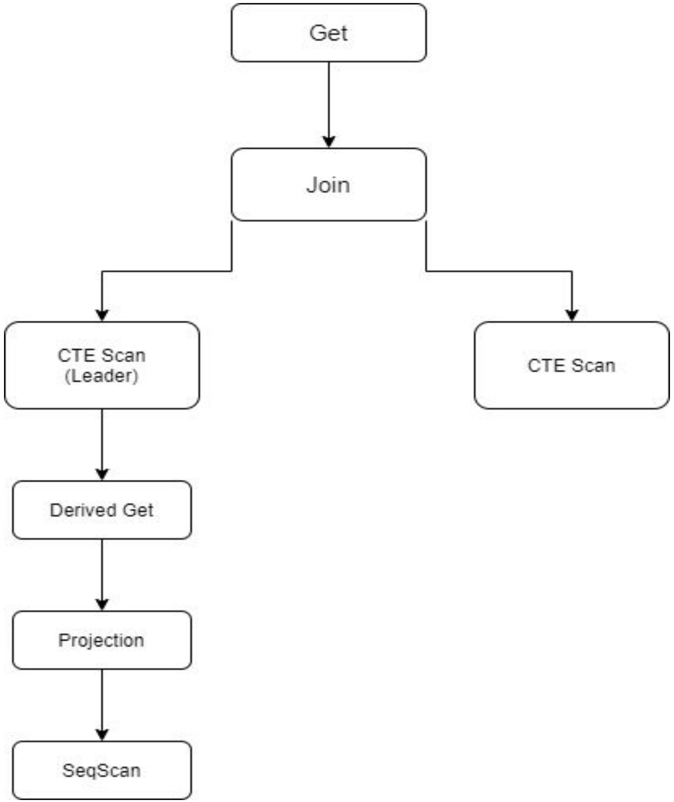
We pick the first cte to connect the sub tree.

Optimizer Output (Multiple Possible Physical Plans)



Join Reordering can reorder the subtree.

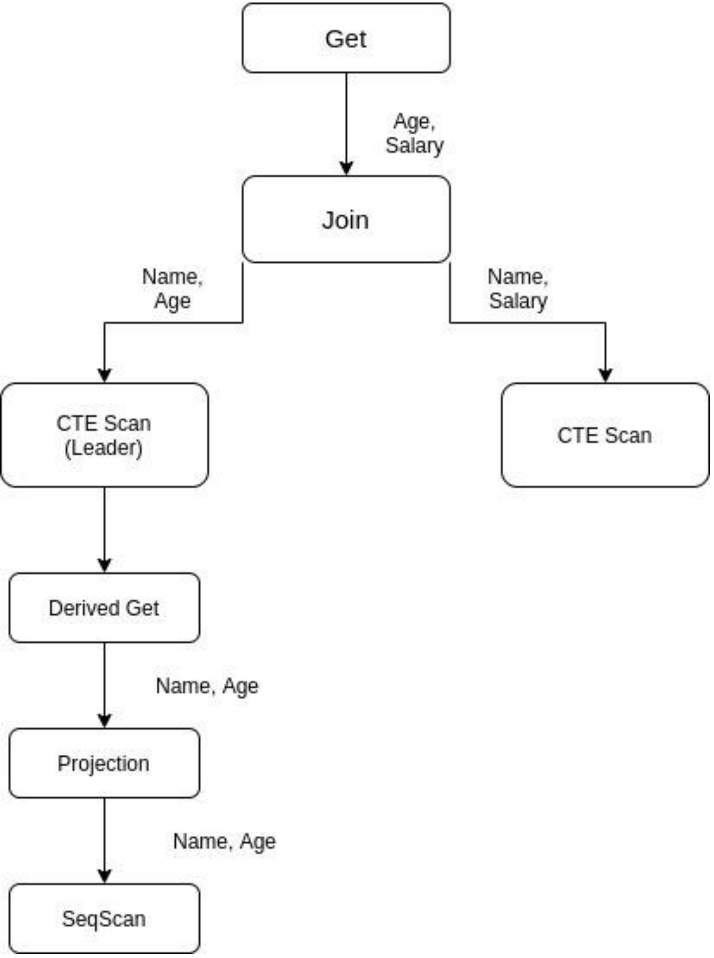
Optimizer (Physical Plan)



WITH EMPLOYEE AS (SELECT NAME,AGE, SALARY FROM COMPANY) SELECT E1.AGE,E2.SALARY FROM EMPLOYEE AS E1, EMPLOYEE AS E2 WHERE E1.NAME = E2.NAME;

- Perform a DFS of the tree
- Move the sub plan to the “appropriate” place and label the node as Leader
- Leader materializes the temp table and populates it

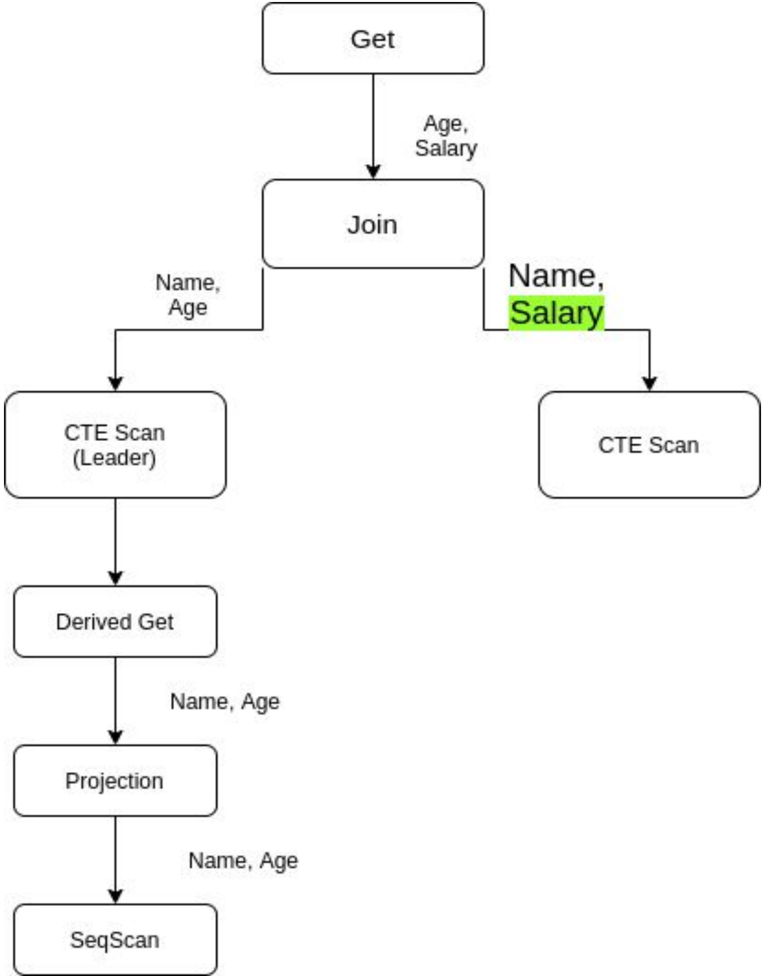
Output Schema Generation



WITH EMPLOYEE AS (SELECT NAME,AGE, SALARY FROM COMPANY) SELECT E1.AGE,E2.SALARY FROM EMPLOYEE AS E1, EMPLOYEE AS E2 WHERE E1.NAME = E2.NAME;

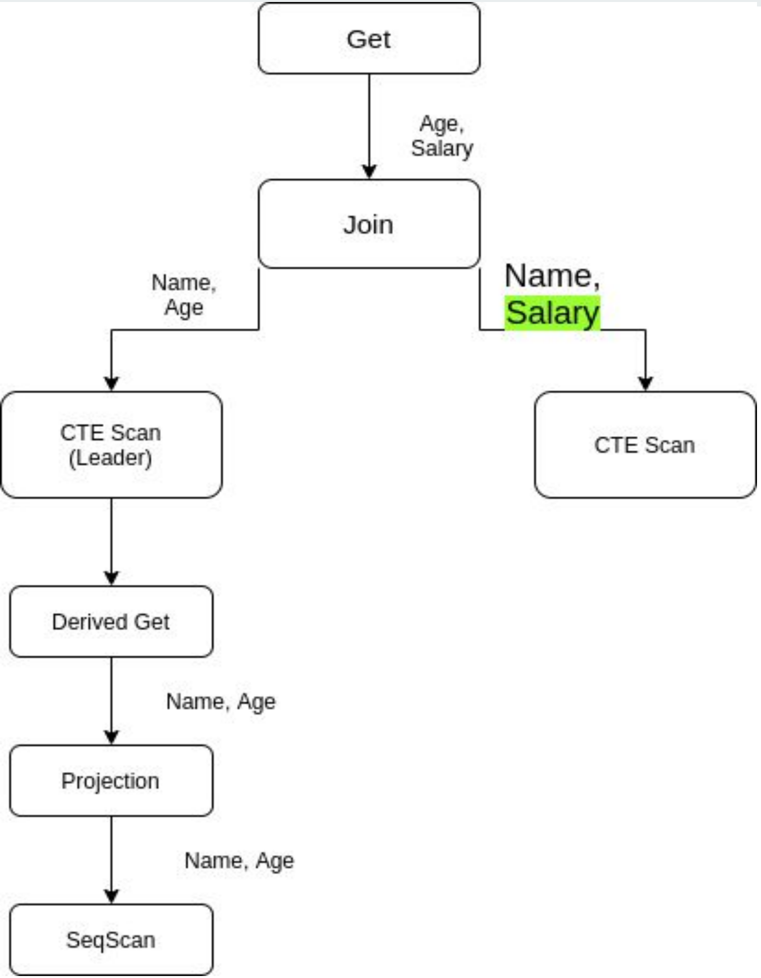
- Parent asks the child for the attributes it requires.
- This decides the output schema of the each node.

Output Schema Changes



WITH EMPLOYEE AS (SELECT NAME,AGE, SALARY FROM COMPANY) SELECT E1.AGE,E2.SALARY FROM EMPLOYEE AS E1, EMPLOYEE AS E2 WHERE E1.NAME = E2.NAME;

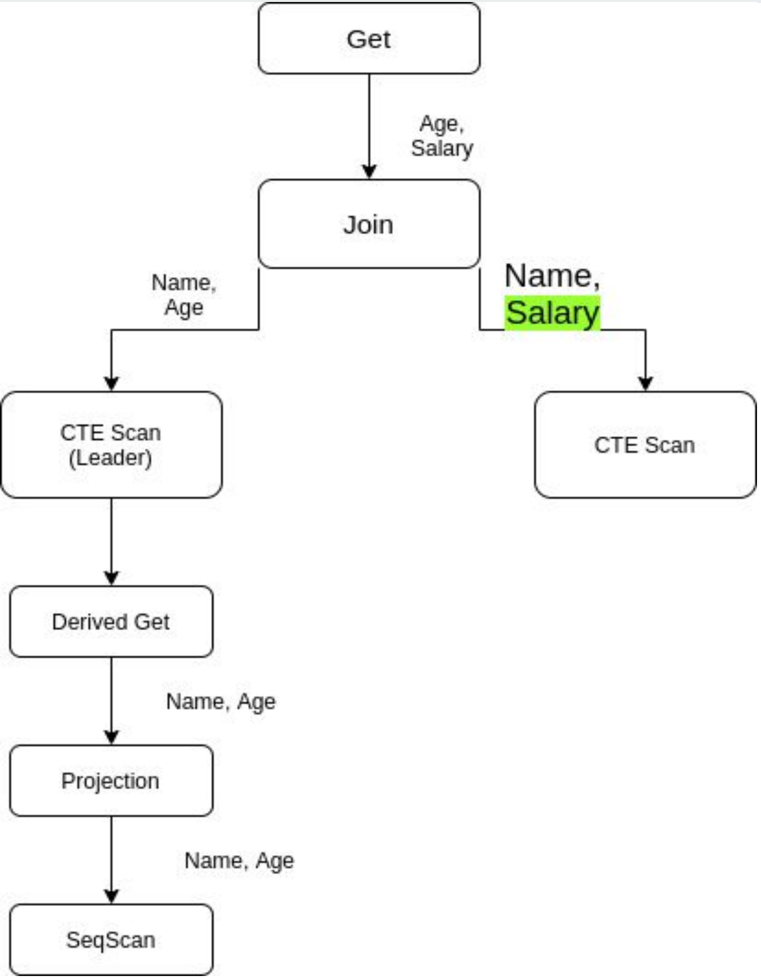
PROBLEM :
Salary never reached the Leader Node and hence sequential scan optimizes its output.
The materialized table will never have Salary.



Output Schema Changes

WITH EMPLOYEE AS (SELECT NAME,AGE, SALARY FROM COMPANY) SELECT E1.AGE,E2.SALARY FROM EMPLOYEE AS E1, EMPLOYEE AS E2 WHERE E1.NAME = E2.NAME;

WHAT TO DO...?????



Parser To the Rescue

WITH EMPLOYEE AS (SELECT NAME,AGE, SALARY FROM COMPANY) SELECT E1.AGE,E2.SALARY FROM EMPLOYEE AS E1, EMPLOYEE AS E2 WHERE E1.NAME = E2.NAME;

SOLUTION :

Parser provides Abstract expressions of the select query inside With Clause.

Store them inside cte node to create the correct table_schema

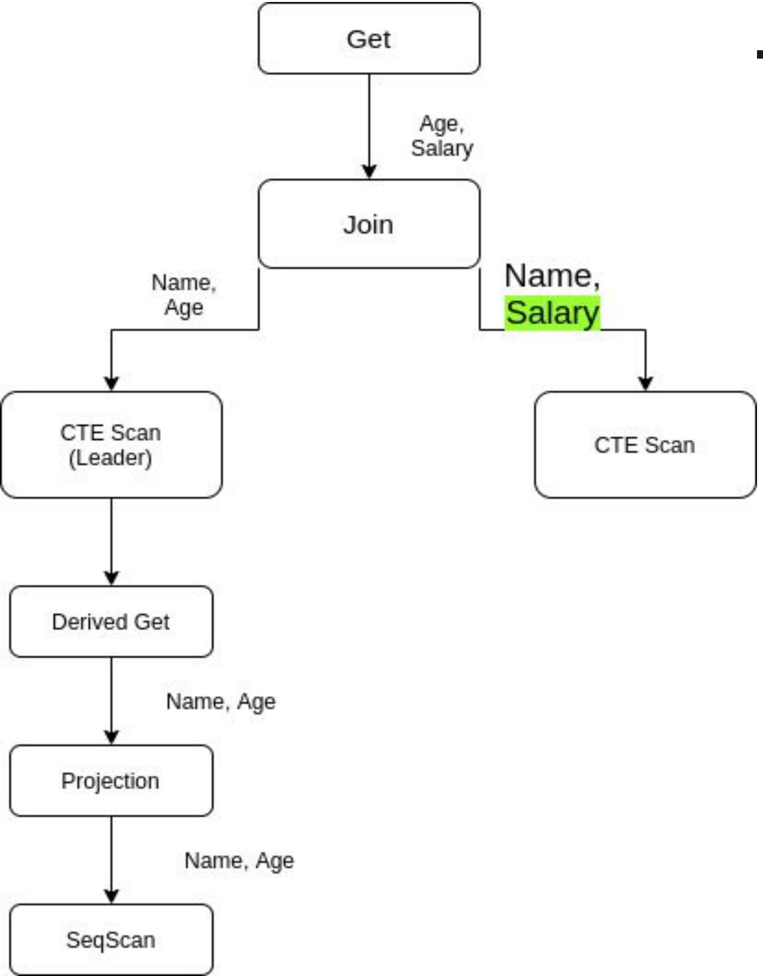


Table Schema and Output Schema

- Every CTE scan node has a uniform Table Schema and its own Output Schema.
- Leader creates a new table with the table schema and populates it.
- All nodes query each column in their Output Schema from the Table Schema and perform a sequential scan.

One Last Problem - Aliases in Derived Get

```
WITH EMPLOYEE AS (SELECT NAME, MAX(AGE) AS MXAGE, SALARY)
SELECT E1.MXAGE,E2.SALARY FROM EMPLOYEE AS E1, EMPLOYEE AS
E2 WHERE E1.NAME = E2.NAME
```

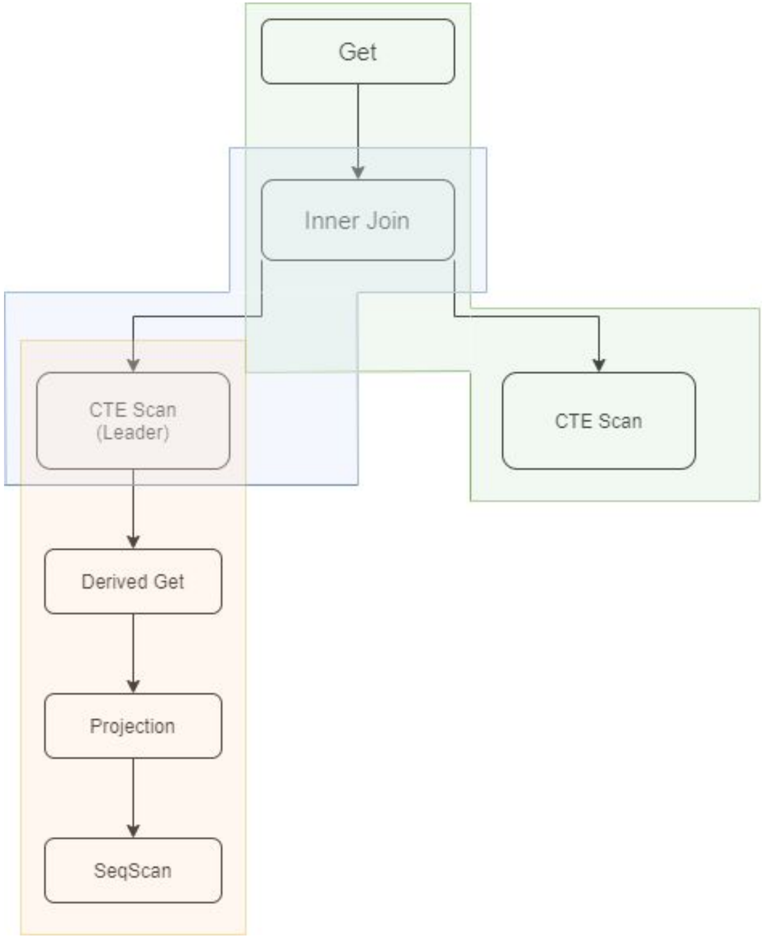
Derived Get requires column value expressions of the aliases.
Parser's output does not account for this.

Solution - Create new expressions accordingly and register their cleanup as a deferred action to the Garbage Collector.

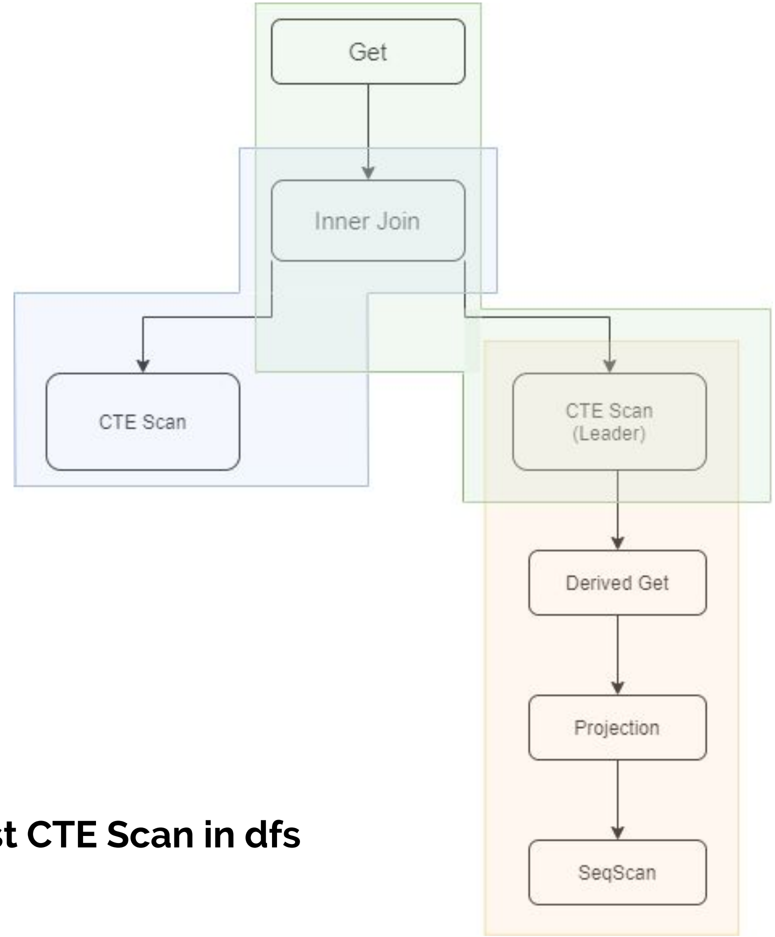
Execution Engine

Features Introduced in the execution engine:

- Creation of table in execution engine
- CTE Leader : Pipeline breaker
- CTE Scan : CTE Table iterator



Leader Election



- Blue Pipeline, executes first
- CTE Table not populated and CTE Scan returns no tuples
- Empty output

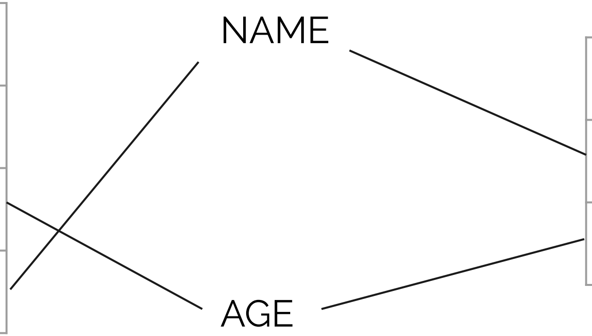
Population must happen at the **first CTE Scan in dfs**

Schema of the new table

1. Create schema of the table
2. Track output schema
3. Correct mapping of the columns

Table Schema

Column ID	Type
0	INT
1	INT
2	VARCHAR



Output Schema

Column ID	Type
0	VARCHAR
1	INT

Creating a table in Execution Engine

PostgreSQL

1. **Using temp table**
   A completely different table data structure stored in heap memory.

2. **Usage in materialized views**
   The temp table is optimized for temporary materialization.

3. **Population on demand**

4. **Deletion**
   Freeing up the heap usage

Terrier

1. **Normal table object "not in catalog"**
   Usage of existing table structure support albeit without the support of a catalog

2. **New support in execution engine**
   Unwrapping of the ddl executor to mimic it's behaviour in the execution engine.

3. **Populate completely**

4. **Deletion:**
   Rolling back of the data structures to ensure no leaks. (Double deferred action)

Deletion of table (Double deferred action)

NAME	AGE	SALARY
Gautam	23	0

SQL Table (Main Table)

DELTA	Pointers
AGE -> 23	

Previous

Delta Storage Segment

- Both need to be deleted
- Transaction can only delete delta storage when it's about to commit

Scenario 1 : Deletion of table when TPL finishes

NAME	AGE	SALARY
Gautam	23	0

SQL Table (Main Table)

DELTA	Pointers
AGE -> 23	

Previous

Delta Storage Segment

Transaction can't find the delta storage. :(

Solution : Defer the deletion of table to transaction -> no memory leaks

Testing

- JUnit Tests: Multiple queries involving CTEs testing the overall functionality
- Tests for TPC-DS like CTE queries
- Execution Engine tests for different possible CTE query plans
- Binder and Logical plan tests

Future Work

- Adding Merging and Pushdown optimizations to non-recursive CTE implementation
- Adding support for Recursive CTEs

Goals

75%

Add support for very basic non-recursive CTEs support in the Query engine ✓

100%

Add non-recursive CTE support using temporary tables similar to PostgreSQL ✓

125%

Add ~~Merging~~ ✓, Pushdown, and, Reuse optimizations ✓ to non-recursive CTE implementation