Table Pushdown Query Optimizer

Objective:

Design of a new query rewrite mechanism called Table Pushdown. The query must have a subquery within it and the outer table must have a join with the inner table specified in the subquery. The rewrite **transforms the query such that the outer table is pushed inside the subselect**.[1]

Outcome:

By rewriting the query as such, the ultimate goal is to generate bushy plans faster. The immediate outcome is to generate queries which reduce the overall query runtime.

Conditions under which Rewrite is triggered[1]-

1. There must be a Subquery-
    a. If true, the subquery must be fetching data from only 1 table.
    b. There must be a join condition between one column in the Subquery relation and another in the Outer Table.
2. The Outer Table should satisfy the following-
    a. There is a join with the Subquery on the primary key column*[* User's query must ensure that the join column is either unique or is PK for the table as currently Presto Connectors/Engine do not provide a way to reason about uniqueness of a column*].
    b. There should not be any GROUP BY/ORDER BY or similar clauses.
3. If the above two are satisfied, then having a GROUP BY clause in the inner Subquery on the join column will not affect the semantics of the grouping.

***Special Comment Format=**

/* #distinct@ tab1 = col1,col2#*/

REGEX-

"^.*/*\\s*#distinct\\@\\s*(\\p{Alnum}+\\s*=\\s*(\\p{Alnum}+)(\\s*,\\s*\\p{Alnum}+)*)*\\s*#\\s**/.*$"

State after the Rewrite has been successful-

1. The Outer Table has been moved inside the Subquery.
2. The join clause on the Subquery column and the Outer Table primary key column is moved inside the Subquery. The actual column name in the Subquery is used here, not the alias.
3. All predicate clauses on the outer table are also moved into the Subquery(WHERE clauses).
4. If the outer table has any other join conditions, these are replaced by the subquery join column name as the outer table has been moved inside and is now not known outside.

A sample Query from TPC-H (Q17[2]) is shown below in its original form and its rewritten form satisfying the above steps-

Original Query-

```
SELECT Sum(l_extendedprice) / 7.0 AS avg_yearly
FROM lineitem, part,
    ( SELECT 0.2 * Avg(l_quantity) AS s_avg, l_partkey AS s_partkey
    FROM lineitem
    GROUP BY l_partkey ) sub
WHERE p_partkey = l_partkey
AND p_brand = 'Brand#43'
AND p_container = 'LG PACK'
AND p_partkey = s_partkey
AND l_quantity < s_avg;
```

Rewritten Query-

```
SELECT Sum(l_extendedprice) / 7.0 AS avg_yearly
FROM lineitem,
    ( SELECT 0.2 * Avg(l_quantity) AS s_avg, l_partkey AS s_partkey
    FROM lineitem, part
    WHERE p_brand = 'Brand#43'
    AND p_container = 'LG PACK'
    AND p_partkey = l_partkey
    GROUP BY l_partkey ) sub
WHERE s_partkey = l_partkey
AND l_quantity < s_avg;
```

Design Diagrams-

In Presto, the Optimizer can be implemented in two ways-

One is by implementing the PlanOptimizer interface which is a visitor based mechanism and calls various visitXXX( ) methods to perform the query optimization. Examples- PredicatePushdown.

The other way is to implement the Rule<T> interface where T is the type of the node on which this is called. This method is based on matching patterns, and creating new nodes and returning to the calling function. Here, we perform the optimization using the second method. We implement the Rule for TablePushdown on the JoinNode as this is the first node that is to be evaluated and encountered while traversing the plan tree from the root OutputNode-

A new class TablePushdown, which implements Rule<JoinNode> is to be created.

Next, we present the the class diagram of the TablePushdown class along with the methods that will be called.

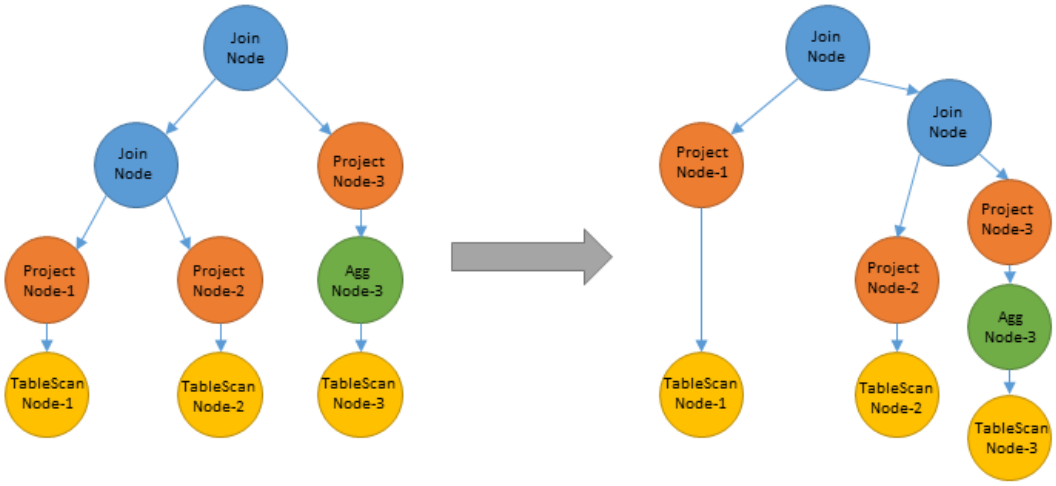
The apply( ) method enters at the first JoinNode encountered from the root of the Plan Tree.

Within the apply method, **verifyJoinCondition()** identifies if the conditions for the rule are satisfied by the query-

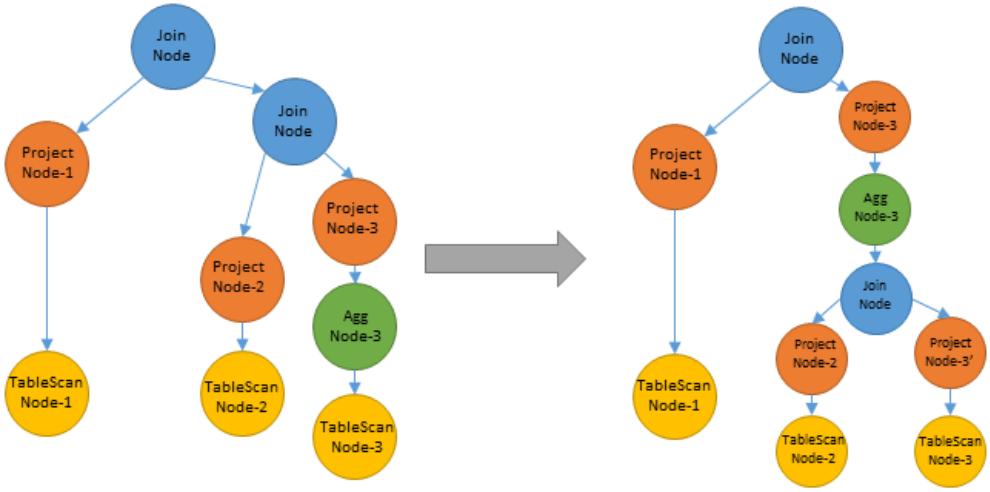
1. **verifyJoinOnUniqueColumn()**- Within the JoinNode, we get the join condition and extract the column aliases for the join condition and check if the column in the outer node has unique values or not by comparing against the user input column name.
2. **verifyPresenceOfJoinsAndGroupBy()**- This method verifies two things-
    a. Both children's paths from the JoinNode cannot have a JoinNode
    b. Only one path from the JoinNode has the GROUP BY clause (subquery table).

Once these conditions are verified, the rewrite mechanism is executed by calling the **planRewriter()** method-

1. First, the outer table is shifted to its intended position in the plan tree (effectively pushed down into the subquery and the join conditions and other associated values are updated).

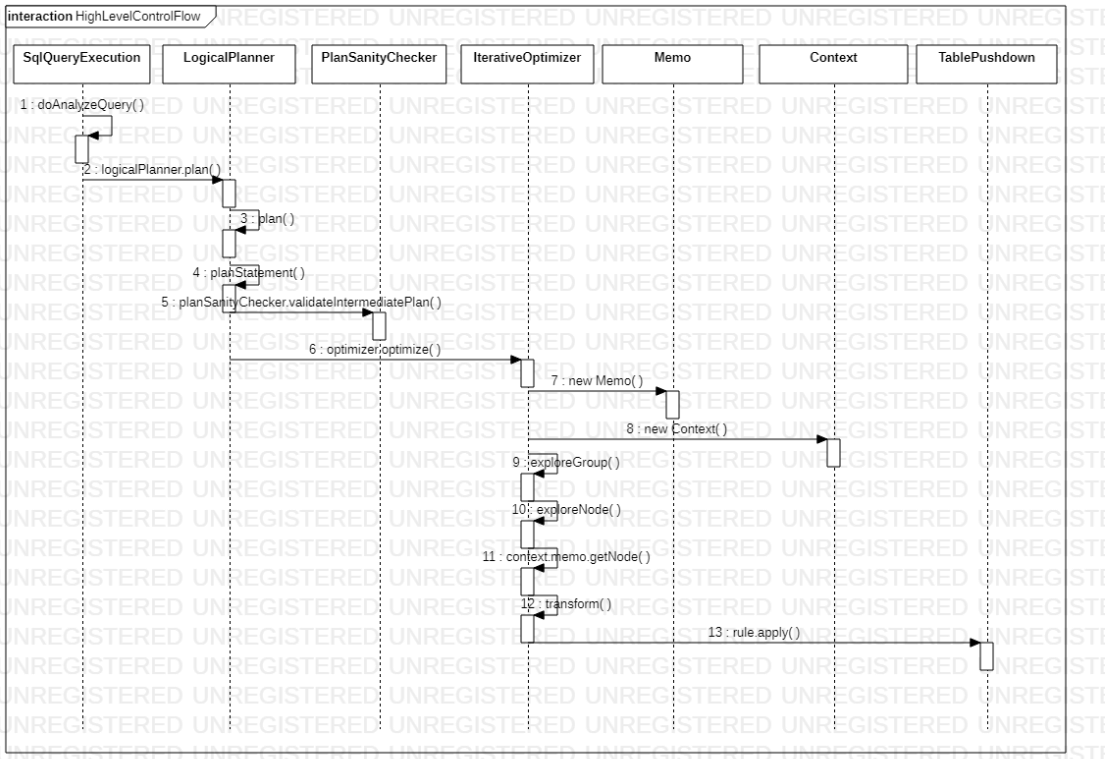


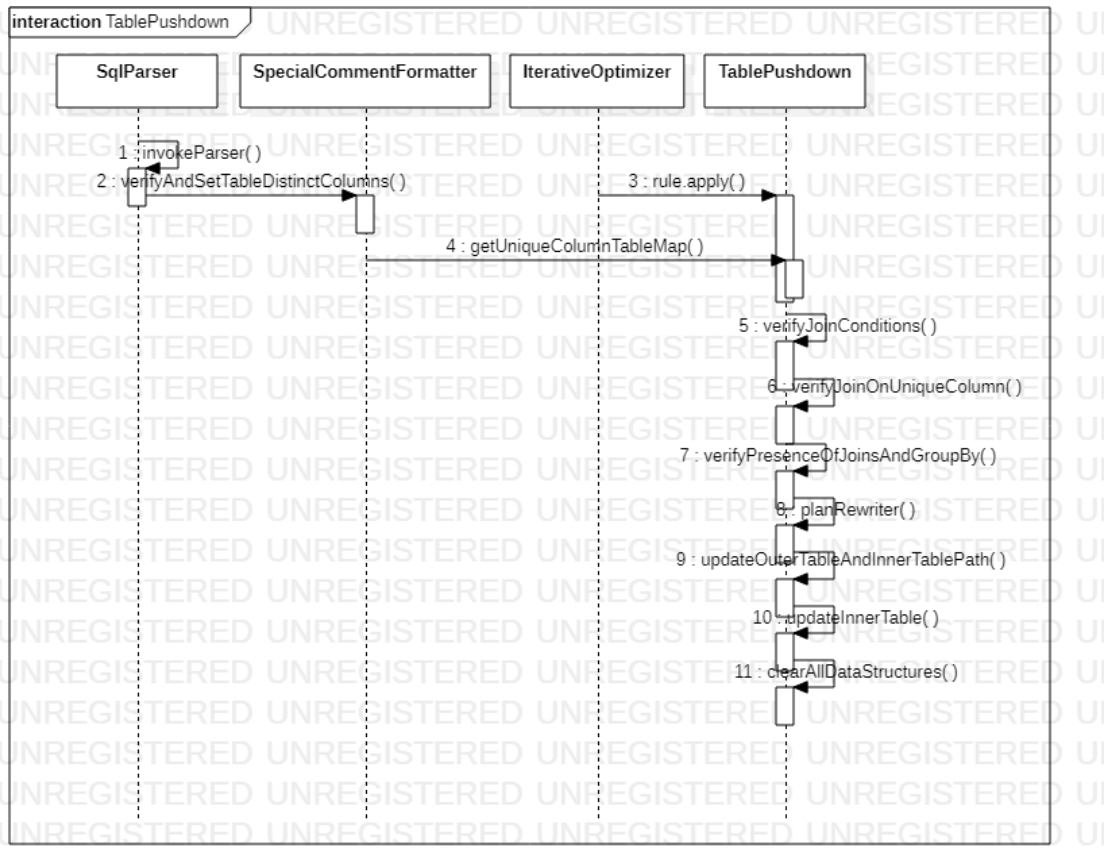
2. The inner subquery table's join conditions are updated and any preexisting aggregations are pulled above the new inner join as follows-



We also propose to have a session parameter for enabling/disabling this Rule called "`optimizer.push-table-through-subquery`".

Finally, the sequence of operations that should be performed to rewrite the query are shown as follows-





Tasks not handled as part of this feature design-

1. Recursive Subquery Table Pushdown.
2. Table Pushdown in the presence of - Multiple Outer table with inner table join

References:

1. https://15721.courses.cs.cmu.edu/spring2017/papers/15-optimizer2/a8-sen.pdf

2. http://www.tpc.org/tpc_documents_current_versions/pdf/tpc-h_v2.18.0.pdf