Alternate Join Syntax in Hive

Harish Butani

October 11, 2013

Certain tools still generate 'old style' Join queries where the join condition is in the *Where* clause. A related set of issues that can be addressed is that of pushing forward joining conditions; in a manner similar to the *Predicate Pushdown* feature of Hive.

In the following we give e.g.s of the issues and then propose a set of Steps to solve these problems.

1 Current behavior of Hive

1.1 Where clause predicates are not pushed down.

This is fairly obvious, for e.g.

```
select *
from R1 join R2
where R1.x = R2.y
```

Is handled as a *cross-product* on R1 with R2, followed by a *selection*.

1.2 Join conditions not pushed down either

We have a similar situation with such a query:

```
select *
from R1 join R2 join R3 on R1.x = R2.y and R2.y = R3.z
```

This is handled as:

- a *cross-product* on R1 and R2
- followed by a selection, applying $R1.x = R2.y$
- followed by a join with R3 on $R2.y = R3.z$

By not pushing down $R1.x = R2.y$ we also loose the opportunity to **Merge the 2 Joins into 1 MR Job**.

1.3 We don't infer joining conditions and push them down

Consider the following query:

```
select *
from R1 join R2 join R3 on R1.x = R3.z and R2.y = R3.z
```

This is handled as:

- a *cross product* on R1 and R2
- followed by a join on R3 on $R1.x = R3.z and R2.y = R3.z$

But from the 2 predicates on the second join, we can infer $R1.x = R2.y$ and push this predicate down to the first Join.

Further we can get rid of one of the predicates in the join of R3, so for e.g. we can treat the query as if the user specified

```
select *
from R1 join R2 on R1.x = R2.y join R3 on R1.x = R3.z
```

In this case the 2 joins will be merged into 1 MR join.

2 Proposed Changes

2.1 Notes on current implementation.

Before getting into the proposed changes documenting some details of the existing implementation:

QBJoinTree is the central data structure that captures a Join. It is a lefty tree that captures the chain of Joins specified. A QBJoinTree node captures:

Join Type INNER, LEFT OUTER, RIGHT OUTER,....

Left Aliases the tables/aliases to the left (ones that come before it).

Right Alias the table/alias that this Join is joining in.

Join Expressions the list of join predicates on this Join, broken down into left and right expressions.

Pushdown Filter expressions list of predicates identified as only applying to one side of a Join and can be pushed down. Broken down into left and right lists.

Filter expressions any predicates identified as only applying to one side of a Join; but must be applied during the Join operation(in case of Outer joins). Broken down into left and right lists.

So in this Query:

```
select *
from R1 join R2 join R3 on R1.x = R2.y and R2.y = R3.z
```

two QBJoinTrees are created:

```
- QBJoinTree 1
  - Join Type = INNER
  - Left Aliases = [R1]
  - Right Alias = R2
  - Join Expressions = []
  - Pushdown Filter Expressions = []
  - Filter expression = []

- QBJoinTree 2
  - Left Tree = QBJoinTree 1
  - Join Type = Inner
  - Left Aliases = [R1, R2]
  - Right Alias = R3
  - Join Expressions = [left : [R2.y], right : [R3.z]]
  - Pushdown Filter Expressions = [left: [R1.x = R2.y]]
  - Filter expression = []
```

2.2 Pushdown Join Expressions

We should further analyze *Pushdown Filter* predicates to check if they are candidate Join Expressions for the previous QBJoinTree. This is a simple check: in addition to only referring to left Aliases the expression must have the following properties:

- each side of the predicate must refer to at least one table.
- the set of aliases referred by the left and right sides must be different.

If this is the case, we pass this expression with the detailed information analyzed in *parseJoinCondition*: the left Sources and right Sources, the left and right expressions etc. to the previous QBJoinTree and have it either add a Join Expression or recursively pass further down. **In the example above**: for QBJoinTree 2 the $R1.x = R2.y$ left pushdown filter can be given to QBJoinTree 1, which should add it as a Join Expression. You end up with:

```
- QBJoinTree 1
  - Join Type = INNER
  - Left Aliases = [R1]
  - Right Alias = R2
  - Join Expressions = [left : [R1.x], right : [R2.y]]
  - Pushdown Filter Expressions = []
  - Filter expression = []

- QBJoinTree 2
  - Left Tree = QBJoinTree 1
  - Join Type = Inner
  - Left Aliases = [R1, R2]
  - Right Alias = R3
  - Join Expressions = [left : [R2.y], right : [R3.z]]
  - Pushdown Filter Expressions = []
  - Filter expression = []
```

2.3 Pushdown Predicates from the Where Clause

We break up the Where Clause into Conjuncts and look for Conjuncts that have the following properties:

- is a Equality Predicate
- there are no Unqualified Column references in the Conjunct
- the left and right sides of the Predicate each refers to at least 1 table.
- the set of table aliases on the left and right sides is different.

We hand each of these conjuncts to the top QBJoinTree and have it apply it as a Join Condition. We do this in the *genPlan* flow immediately after finishing generation of the QBJoinTree. This should happen before an attempt is made to merge QBJoinTree nodes.

We track the conjuncts that got pushed into QBJoinTree, so when the FilterDesc is created for the Where Clause, we should exclude these conjuncts.

So for the e.g.

```
select *
from R1 join R2
where R1.x = R2.y
```

Today you get a QBJoinTree that is:

```
- QBJoinTree 1
  - Join Type = INNER
  - Left Aliases = [R1]
  - Right Alias = R2
  - Join Expressions = []
  - Pushdown Filter Expressions = []
  - Filter expression = []
```

After applying this check, we end up with:

```
- QBJoinTree 1
  - Join Type = INNER
  - Left Aliases = [R1]
  - Right Alias = R2
  - Join Expressions = [left:[R1.x], right:[R2.y]]
  - Pushdown Filter Expressions = []
  - Filter expression = []
```

2.4 Allow unqualified column references in Join conditions.

Today we flag the following Query as an error. A related issue is that the second query below will not be converted to a Join.

```
-- assume schema R1(x,z); R2(y,a)
-- q1
select *
```

```
from R1 join R2 on x = y

-- q2
select *
from R1 join R2
where x = y
```

It is fairly easy to enhance the QBJoinTree generation code to resolve unqualified column references. At the time of QBJoinTree generation we have the RowResolvers of all the Table sources. We can check all of them of resolve an unqualified column, using the table alias from the RowResolver that returns a match; in case multiple matches are found we throw and Ambiguous Column Reference error.

Hive users are accustomed to qualifying columns in Join conditions. But extending this to where clause predicates (albeit the ones involving join predicates) may not be acceptable. Adding this feature will ensure users don't have to treat join predicates in a special way.

2.5 Infer Join conditions that can be pushed down.

After a QBJoinTree has been constructed. We can scan the list of right Join expressions and look for cases where they are the same. In those cases we can infer a equality condition between the 2 corresponding left Join expressions. This new predicate can be handed to the previous QBJoinTree to apply as a potential Join condition.

So for this query:

```
select *
from R1 join R2 join R3 on R1.x = R3.z and R2.y = R3.z
```

The QBJoinTree we get today is:

```
- QBJoinTree 1
  - Join Type = INNER
  - Left Aliases = [R1]
  - Right Alias = R2
  - Join Expressions = []
  - Pushdown Filter Expressions = []
  - Filter expression = []

- QBJoinTree 2
  - Left Tree = QBJoinTree 1
  - Join Type = Inner
  - Left Aliases = [R1, R2]
  - Right Alias = R3
  - Join Expressions = [left : [R1.x, R2.y], right : [R3.z, R3.z]]
  - Pushdown Filter Expressions = []
  - Filter expression = []
```

From the right Join Expressions in QBJoinTree 2 we can infer $R1.x = R2.y$ which can be pushed to QBJoinTree 1. So we end up with:

```
- QBJoinTree 1
  - Join Type = INNER
  - Left Aliases = [R1]
  - Right Alias = R2
  - Join Expressions = [left: [R1.x], right:[R2.y]]
  - Pushdown Filter Expressions = []
  - Filter expression = []

- QBJoinTree 2
  - Left Tree = QBJoinTree 1
  - Join Type = Inner
  - Left Aliases = [R1, R2]
  - Right Alias = R3
  - Join Expressions = [left : [R1.x, R2.y], right : [R3.z, R3.z]]
  - Pushdown Filter Expressions = []
  - Filter expression = []
```

In addition since from the first and second predicates in QBJoinTree 2 we successfully inferred a Join Condition, we can remove any one of them. So we end up with:

```
- QBJoinTree 1
  - Join Type = INNER
  - Left Aliases = [R1]
  - Right Alias = R2
  - Join Expressions = [left: [R1.x], right:[R2.y]]
  - Pushdown Filter Expressions = []
  - Filter expression = []

- QBJoinTree 2
  - Left Tree = QBJoinTree 1
  - Join Type = Inner
  - Left Aliases = [R1, R2]
  - Right Alias = R3
  - Join Expressions = [left : [R1.x], right : [R3.z]]
  - Pushdown Filter Expressions = []
  - Filter expression = []
```

This QBJoinTree is better than the previous one because now QBJoinTree 1 and 2 can be merged into 1 node during the merge step.