Extended support for subqueries in HIVE (HIVE-15456)

1 Description

This document intends to document the work done in HIVE-15456 by providing brief overview of the design and transformations implemented to extend support for subqueries. Prior subquery transformations and restrictions are documented at <LINK>.
This document will attempt to document the goal of the project, design choices made during the project and restrictions which HIVE enforces on subqueries.

2 Goal

The goal of the project was to continue and extend subquery support in WHERE/HAVING clause to enable multiple subqueries, nested subqueries and scalar subqueries by leverage Calcite. Prior to this HIVE supported only IN/NOT IN and EXISTS/NOT EXSITS queries.

3 Design

CalcitePlanner::genFilterRelNode() on receiving an AST with subqueries walks over it and generate logical plan for each and every subquery. Calcite uses RexSubquery node to encapsulate all kind of subqueries. genFilterRelNode generates a filter consiting of RexSubquery. This is also is where HIVE checks for various restrictions and tries to throw an error if subquery is not supported.
Once logical plan for the whole query is generated *HiveSubqueryRemoveRule* is invoked which rewrites the plan and remove RexSubquery node. Immediately after this rule *HiveRelDecorrelator::decorrelateQuery* is invoked which decorrelates correlated queries.

Following section categorizes subqueries to describe how HIVE plans and transform them

3.1 EXISTS/NOT EXISTS

3.1.1 EXISTS/NOT EXISTS, uncorrelated, with or without aggregate

Irrespective of existence of aggregate, such queries are transformed into cross join b/w inner and parent table. Inner table scan is succeeded by a *project{true}* and *group by{true}*.

e.g. Plan generated just after decorrelateQuery for query ***select p_name from part where exists (select p_size from part)*** is as follows:

```
HiveProject(p_name=[$1])
  HiveProject(p_partkey=[$0], p_name=[$1], p_mfgr=[$2], p_brand=[$3], p_type=[$4], p_size=[$5], p_container=[$6],
p_retailprice=[$7], p_comment=[$8], BLOCK__OFFSET__INSIDE__FILE=[$9], INPUT__FILE__NAME=[$10],
ROW__ID=[$11])
    HiveJoin(condition=[true], joinType=[inner], algorithm=[none], cost=[not available])
      HiveTableScan(table=[[default.part]], table:alias=[part])
      HiveAggregate(group=[{0}])
        HiveProject(i631=[true])
          HiveProject(p_size=[$5])
            HiveTableScan(table=[[default.part]], table:alias=[part])
```

3.1.2 EXISTS/NOT EXISTS, correlated, without aggregate

Such queries are similarly transformed into inner join with correlated columns as key. Inner table scan is similarly succeeded by a *project{true}* and *group by{true}*.
e.g.

```
select p_name from part where exists (select p_size from part p where p.p_type = part.p_type);
```

```
HiveProject(p_name=[$1])
  HiveProject(p_partkey=[$0], p_name=[$1], p_mfgr=[$2], p_brand=[$3], p_type=[$4], p_size=[$5],
p_container=[$6], p_retailprice=[$7], p_comment=[$8], BLOCK__OFFSET__INSIDE__FILE=[$9],
INPUT__FILE__NAME=[$10], ROW__ID=[$11])
    LogicalJoin(condition=[=($4, $12)], joinType=[inner])
      HiveTableScan(table=[[default.part]], table:alias=[part])
      LogicalProject(p_type=[$0], $f1=[true])
        HiveAggregate(group=[{0}])
          HiveProject(p_type=[$1], i841=[$0])
            HiveProject(i841=[true], p_type=[$1])
              HiveProject(p_size=[$5], p_type=[$4])
                LogicalFilter(condition=[=($4, $4)])
                  HiveTableScan(table=[[default.part]], table:alias=[p])
```

3.1.3 EXISTS/NOT EXISTS, correlated, with aggregate

Such queries are currently not supported. Since having an aggregate inside subquery will always produce at least one row, making EXISTS always true, transforming it into join could lead to wrong results. Therefore HIVE checks for them and throws an error. We plan to support this in future (HIVE-15757).

3.2 IN

3.2.1 Uncorrelated, without aggregate

Such queries are transformed into a group by{ subquery expression} on inner table succeeded by inner join on subquery expression. Note that later *HiveSemiJoinRule* attempts to convert such joins into left semi-join.

Example

```
select p_size from part where p_size IN (select p_size from part);
```

```
HiveProject(p_size=[$5])
  HiveProject(p_partkey=[$0], p_name=[$1], p_mfgr=[$2], p_brand=[$3], p_type=[$4], p_size=[$5],
p_container=[$6], p_retailprice=[$7], p_comment=[$8], BLOCK__OFFSET__INSIDE__FILE=[$9],
INPUT__FILE__NAME=[$10], ROW__ID=[$11])
    HiveJoin(condition=[=($5, $12)], joinType=[inner], algorithm=[none], cost=[not available])
      HiveTableScan(table=[[default.part]], table:alias=[part])
      HiveAggregate(group=[{0}])
        HiveProject(p_size=[$5])
          HiveTableScan(table=[[default.part]], table:alias=[part])
```

3.2.2 Uncorrelated, with aggregate

Same as 3.2.1

3.2.3 Correlated, without aggregate

Similarly to 3.2.1 this is transformed into a group by{subquery expression, correlated columns} on inner table succeeded by inner join on subquery expression, correlated column. This again later is attempted to be converted into Left-semi join via HiveSemiJoinRule.

Example

```
select *
from src b
where b.key in
      (select a.key
       from src a
       where b.value = a.value and a.key > '9')
```

```
HiveProject(key=[$0], value=[$1])
  HiveProject(key=[$0], value=[$1], BLOCK__OFFSET__INSIDE__FILE=[$2],
INPUT__FILE__NAME=[$3], ROW__ID=[$4])
    LogicalJoin(condition=[AND(=($1, $6), =($0, $5))], joinType=[inner])
      HiveTableScan(table=[[default.src]], table:alias=[b])
      HiveAggregate(group=[{0, 1}])
        HiveProject(key=[$0], value=[$1])
          HiveProject(key=[$0], value=[$1])
            LogicalFilter(condition=[AND(=($1, $1), >($0, _UTF-16LE'9'))])
              HiveTableScan(table=[[default.src]], table:alias=[a])
```

3.2.4 Correlated, with aggregate

This case has similar problem as mentioned in 3.1.3. Since having an aggregate inside subquery will always produce at least one row transforming it into join could lead to wrong results for empty groups (HIVE-15721). Note that this issue in this case occurs only with COUNT since with other aggregates HIVE produces NULL on empty groups. IN with NULL is false so transforming such cases into JOIN wouldn't produce wrong results. Given all of this such queries are categorized further into two cases:

3.2.4.1 COUNT

In this case we disallow queries with empty groups. This is done by generating an extra runtime check to make sure there is no empty groups coming out of inner query. This done using group by{correlated columns} succeeded by count() which is then passed to UDF sq_count_check(). This UDF throws an error if count() is zero, since this is the case which could lead to wrong results. The result from this UDF is further joined with parent/outer table and is discarded. Inner query is then transformed similar to 3.1.3
Example

```
select * from part where p_size IN
                (select count(*) from part pp where pp.p_type = part.p_type);
```

```
HiveProject(p_partkey=[$0], p_name=[$1], p_mfgr=[$2], p_brand=[$3], p_type=[$4], p_size=[$5],
p_container=[$6], p_retailprice=[$7], p_comment=[$8])
  HiveProject(p_partkey=[$0], p_name=[$1], p_mfgr=[$2], p_brand=[$3], p_type=[$4], p_size=[$5],
p_container=[$6], p_retailprice=[$7], p_comment=[$8], BLOCK__OFFSET__INSIDE__FILE=[$9],
INPUT__FILE__NAME=[$10], ROW__ID=[$11])
    LogicalJoin(condition=[AND(=($4, $15), =($5, $14))], joinType=[inner])
      LogicalFilter(condition=[>(sq_count_check($13, true), 0)])
        LogicalJoin(condition=[=($4, $12)], joinType=[left])
          HiveTableScan(table=[[default.part]], table:alias=[part])
          HiveAggregate(group=[{0}], cnt_in=[COUNT()])
            HiveProject(p_type=[$1], _o__c0=[$0])
              HiveProject(_o__c0=[$1], p_type=[$0])
                HiveAggregate(group=[{0}], agg#0=[count()])
                  HiveProject(p_type=[$1], $f0=[$0])
                    HiveProject($f0=[$0], p_type=[$4])
                      LogicalFilter(condition=[=($4, $4)])
                        HiveTableScan(table=[[default.part]], table:alias=[pp])
      HiveAggregate(group=[{0, 1}])
        HiveProject(_o__c0=[$0], p_type=[$1])
          HiveProject(_o__c0=[$1], p_type=[$0])
            HiveAggregate(group=[{0}], agg#0=[count()])
              HiveProject(p_type=[$1], $f0=[$0])
                HiveProject($f0=[$0], p_type=[$4])
                  LogicalFilter(condition=[=($4, $4)])
                    HiveTableScan(table=[[default.part]], table:alias=[pp])
```

3.2.4.2 Aggregates except COUNT

Same as 3.2.3 (correlated, without aggregate)

3.3 NOT IN

Since presence of NULL in inner query changes the semantic NOT IN is handled differently from IN

3.3.1 Uncorrelated, without aggregate

Such queries, similarly to IN are transformed into group by{subquery expression} on inner table succeeded by LEFT JOIN with outer table on subquery expression columns. To determine presence of NULLs count(), count(subquery expression) is done succeeded by cross join with outer table. Difference in count(), count(subquery expression) will indicate presence of NULL.

Example

```
select *
from src
where src.key not in
 ( select key  from src s1
   where s1.key > '2');
```

```
HiveProject(key=[$0], value=[$1])
  HiveProject(key=[$0], value=[$1], BLOCK__OFFSET__INSIDE__FILE=[$2],
INPUT__FILE__NAME=[$3], ROW__ID=[$4])
    HiveFilter(condition=[NOT(CASE(=($5, 0), false, IS NOT NULL($8), true, IS NULL($0), null, <($6,
$5), true, false))])
      HiveJoin(condition=[=($0, $7)], joinType=[left], algorithm=[none], cost=[not available])
        HiveJoin(condition=[true], joinType=[inner], algorithm=[none], cost=[not available])
          HiveTableScan(table=[[default.src]], table:alias=[src])
          HiveAggregate(group=[{}], c=[COUNT()], ck=[COUNT($0)])
            HiveProject(key=[$0])
              HiveFilter(condition=[>($0, _UTF-16LE'2')])
                HiveTableScan(table=[[default.src]], table:alias=[s1])
        HiveAggregate(group=[{0, 1}])
          HiveProject(key=[$0], i3=[true])
            HiveProject(key=[$0])
              HiveFilter(condition=[>($0, _UTF-16LE'2')])
                HiveTableScan(table=[[default.src]], table:alias=[s1])
```

3.3.2 Uncorrelated, with aggregate

Such queries are transformed same as 3.3.1 Uncorrelated without aggregate.

3.3.3 Correlated, without aggregate

Such queries, similarly to NOT IN, uncorrelated without aggregate are transformed into group by{subquery expression, correlated column} on inner table succeeded by LEFT JOIN on subquery expression, correlated column with outer table. Group by on {correlated column} is succeeded by count(), count(subquery expression) to determine presence of NULLs.

Example

```
select *
from src
where src.key not in
 ( select key  from src s1
   where s1.value = src.value and s1.key > '2'
 );
```

```
HiveProject(key=[$0], value=[$1])
  HiveProject(key=[$0], value=[$1], BLOCK__OFFSET__INSIDE__FILE=[$2],
INPUT__FILE__NAME=[$3], ROW__ID=[$4])
    LogicalFilter(condition=[NOT(CASE(=($6, 0), false, IS NULL($6), false, IS NOT NULL($10), true,
IS NULL($0), null, <($7, $6), true, false))])
      LogicalJoin(condition=[AND(=($0, $8), =($1, $9))], joinType=[left])
        LogicalJoin(condition=[=($1, $5)], joinType=[left])
          HiveTableScan(table=[[default.src]], table:alias=[src])
          HiveAggregate(group=[{0}], c=[COUNT()], ck=[COUNT($1)])
            HiveProject(value=[$1], key=[$0])
              HiveProject(key=[$0], value=[$1])
                LogicalFilter(condition=[AND(=($1, $1), >($0, _UTF-16LE'2'))])
                  HiveTableScan(table=[[default.src]], table:alias=[s1])
        LogicalFilter(condition=[=($0, $0)])
          LogicalProject(key=[$0], value=[$1], $f2=[true])
            HiveAggregate(group=[{0, 1}])
              HiveProject(key=[$0], value=[$2], i432=[$1])
                HiveProject(key=[$0], i432=[true], value=[$1])
                  HiveProject(key=[$0], value=[$1])
                    LogicalFilter(condition=[AND(=($1, $1), >($0, _UTF-16LE'2'))])
                      HiveTableScan(table=[[default.src]], table:alias=[s1])
```

3.3.4 Correlated, with aggregate

Such queries has same issue as 3.2.4 (IN, correlated, with aggregate). Similarly to 3.2.4, but for all aggregate this, along with count(), count(subquery expression) to indicate NULL presence, is transformed to generate an extra runtime check using sq_count_check() to throw an error for empty groups.

Example

```
select *
from src
where src.key not in
 ( select min(key) from src s1
   where s1.value = src.value and s1.key > '2'
 );
```

```
HiveProject(key=[$0], value=[$1])
  HiveProject(key=[$0], value=[$1], BLOCK__OFFSET__INSIDE__FILE=[$2], INPUT__FILE__NAME=[$3],
ROW__ID=[$4])
    LogicalFilter(condition=[NOT(CASE(=($8, 0), false, IS NULL($8), false, IS NOT NULL($12), true, IS NULL($0),
null, <($9, $8), true, false))])
      LogicalJoin(condition=[AND(=($0, $10), =($1, $11))], joinType=[left])
        LogicalJoin(condition=[=($1, $7)], joinType=[left])
          LogicalFilter(condition=[>(sq_count_check($6, true), 0)])
            LogicalJoin(condition=[=($1, $5)], joinType=[left])
              HiveTableScan(table=[[default.src]], table:alias=[src])
              HiveAggregate(group=[{0}], cnt_in=[COUNT()])
                HiveProject(value=[$1], _o__c0=[$0])
                  HiveProject(_o__c0=[$1], value=[$0])
                    HiveAggregate(group=[{0}], agg#0=[min($1)])
                      HiveProject(value=[$1], $f0=[$0])
                        HiveProject($f0=[$0], value=[$1])
                          LogicalFilter(condition=[AND(=($1, $1), >($0, _UTF-16LE'2'))])
                            HiveTableScan(table=[[default.src]], table:alias=[s1])
          HiveAggregate(group=[{0}], c=[COUNT()], ck=[COUNT($1)])
            HiveProject(value=[$1], _o__c0=[$0])
              HiveProject(_o__c0=[$1], value=[$0])
                HiveAggregate(group=[{0}], agg#0=[min($1)])
                  HiveProject(value=[$1], $f0=[$0])
                    HiveProject($f0=[$0], value=[$1])
                      LogicalFilter(condition=[AND(=($1, $1), >($0, _UTF-16LE'2'))])
                        HiveTableScan(table=[[default.src]], table:alias=[s1])
        LogicalFilter(condition=[=($0, $0)])
          LogicalProject(_o__c0=[$0], value=[$1], $f2=[true])
            HiveAggregate(group=[{0, 1}])
              HiveProject(_o__c0=[$0], value=[$2], i2247=[$1])
                HiveProject(_o__c0=[$0], i2247=[true], value=[$1])
                  HiveProject(_o__c0=[$1], value=[$0])
                    HiveAggregate(group=[{0}], agg#0=[min($1)])
                      HiveProject(value=[$1], $f0=[$0])
                        HiveProject($f0=[$0], value=[$1])
                          LogicalFilter(condition=[AND(=($1, $1), >($0, _UTF-16LE'2'))])
                            HiveTableScan(table=[[default.src]], table:alias=[s1])
```

3.4 Scalar subqueries

Scalar query's semantic dictates that subquery should produce at most one row. Since this would be impossible to determine at compile time HIVE generates a runtime check by generating count() on inner query which is passed to sq_count_check UDF. This UDF checks the value of count() and throws an error if it is greater than one. The result of this UDF is joined with outer query and discarded.

3.4.1 Uncorrelated, with or without aggregate

In addition to above mentioned runtime check such queries are transformed into LEFT JOIN with filter on top of it

Example

```
select * from part where
        p_size > (select avg(p_size) from part_null);
```

```
HiveProject(p_partkey=[$0], p_name=[$1], p_mfgr=[$2], p_brand=[$3], p_type=[$4], p_size=[$5],
p_container=[$6], p_retailprice=[$7], p_comment=[$8])
  HiveProject(p_partkey=[$0], p_name=[$1], p_mfgr=[$2], p_brand=[$3], p_type=[$4], p_size=[$5],
p_container=[$6], p_retailprice=[$7], p_comment=[$8], BLOCK__OFFSET__INSIDE__FILE=[$9],
INPUT__FILE__NAME=[$10], ROW__ID=[$11])
    HiveFilter(condition=[>(CAST($5):DOUBLE, $13)])
      HiveJoin(condition=[true], joinType=[left], algorithm=[none], cost=[not available])
        HiveJoin(condition=[true], joinType=[inner], algorithm=[none], cost=[not available])
          HiveTableScan(table=[[default.part]], table:alias=[part])
          HiveFilter(condition=[<=(sq_count_check($0), 1)])
            HiveAggregate(group=[{}], cnt=[COUNT()])
              HiveProject(_o__c0=[$0])
                HiveAggregate(group=[{}], agg#0=[avg($0)])
                  HiveProject($f0=[$5])
                    HiveTableScan(table=[[default.part_null]], table:alias=[part_null])
        HiveProject(_o__c0=[$0])
          HiveAggregate(group=[{}], agg#0=[avg($0)])
            HiveProject($f0=[$5])
              HiveTableScan(table=[[default.part_null]], table:alias=[part_null])
```

3.4.2 Correlated, without aggregate

This case is transformed similar to Uncorrelated with/without aggregate. Runtime check is done by doing count() preceded by group by on correlated columns. In addition this query is transformed into LEFT JOIN with filter on top of it.

Example

```
select * from part where p_size >
  (select p_size from part_null where part_null.p_type = part.p_type);
```

```
HiveProject(p_partkey=[$0], p_name=[$1], p_mfgr=[$2], p_brand=[$3], p_type=[$4], p_size=[$5],
p_container=[$6], p_retailprice=[$7], p_comment=[$8])
  HiveProject(p_partkey=[$0], p_name=[$1], p_mfgr=[$2], p_brand=[$3], p_type=[$4], p_size=[$5],
p_container=[$6], p_retailprice=[$7], p_comment=[$8], BLOCK__OFFSET__INSIDE__FILE=[$9],
INPUT__FILE__NAME=[$10], ROW__ID=[$11])
    LogicalJoin(condition=[AND(=($4, $15), >($5, $14))], joinType=[inner])
      LogicalJoin(condition=[=($4, $12)], joinType=[left])
        HiveTableScan(table=[[default.part]], table:alias=[part])
        LogicalFilter(condition=[<=(sq_count_check($1), 1)])
          HiveAggregate(group=[{0}], cnt=[COUNT()])
            HiveProject(p_type=[$1], p_size=[$0])
              HiveProject(p_size=[$5], p_type=[$4])
                LogicalFilter(condition=[=($4, $4)])
                  HiveTableScan(table=[[default.part_null]], table:alias=[part_null])
      HiveProject(p_size=[$5], p_type=[$4])
        LogicalFilter(condition=[=($4, $4)])
          HiveTableScan(table=[[default.part_null]], table:alias=[part_null])
```

3.4.3 Correlated with aggregate

This case has same problem as described earlier in NOT IN and IN correlated with aggregate, where in transforming such queries into JOIN could lead to wrong results for empty groups in inner query.
Such queries are transformed into group by {correlated columns} and aggregate is done on subquery expression. This is further LEFT joined with outer query on correlated columns succeeded by subquery filter on top. Filter on top takes care of the case with empty groups by doing comparison with either 0 or NULL (for count and rest of the aggregates respectively) if right side is NULL (indicating there is no corresponding group). If right side is not NULL then comparison is doing with actual value corresponding to subquery expression from right side.

Note that such transformation only works for equality correlated predicates. HIVE checks for such case and throws an error if it is not equality predicate.

Following example should make it clear.

Example

First example is with count aggregate

```
select * from part where p_size >
                (select count(p_size) from part_null where part_null.p_type = part.p_type);
```

```
HiveProject(p_partkey=[$0], p_name=[$1], p_mfgr=[$2], p_brand=[$3], p_type=[$4], p_size=[$5],
p_container=[$6], p_retailprice=[$7], p_comment=[$8])
  HiveProject(p_partkey=[$0], p_name=[$1], p_mfgr=[$2], p_brand=[$3], p_type=[$4], p_size=[$5],
p_container=[$6], p_retailprice=[$7], p_comment=[$8], BLOCK__OFFSET__INSIDE__FILE=[$9],
INPUT__FILE__NAME=[$10], ROW__ID=[$11])
    LogicalFilter(condition=[>(CAST($5):BIGINT, CASE(IS NULL($13), 0, $12))])
      LogicalJoin(condition=[=($4, $14)], joinType=[left])
        HiveTableScan(table=[[default.part]], table:alias=[part])
        HiveProject(_o__c0=[$0], alwaysTrue7106=[true], p_type=[$1])
          HiveProject(_o__c0=[$1], p_type=[$0])
            HiveAggregate(group=[{0}], agg#0=[count($1)])
              HiveProject(p_type=[$1], $f0=[$0])
                HiveProject($f0=[$5], p_type=[$4])
                  LogicalFilter(condition=[=($4, $4)])
                    HiveTableScan(table=[[default.part_null]], table:alias=[part_null])
```

Next example is with avg aggregate

```
select * from part where p_size >
  (select avg(p_size) from part_null where part_null.p_type = part.p_type);
```

```
HiveProject(p_partkey=[$0], p_name=[$1], p_mfgr=[$2], p_brand=[$3], p_type=[$4], p_size=[$5],
p_container=[$6], p_retailprice=[$7], p_comment=[$8])
  HiveProject(p_partkey=[$0], p_name=[$1], p_mfgr=[$2], p_brand=[$3], p_type=[$4], p_size=[$5],
p_container=[$6], p_retailprice=[$7], p_comment=[$8], BLOCK__OFFSET__INSIDE__FILE=[$9],
INPUT__FILE__NAME=[$10], ROW__ID=[$11])
    LogicalFilter(condition=[>(CAST($5):DOUBLE, CASE(IS NULL($13), null, $12))])
      LogicalJoin(condition=[=($4, $14)], joinType=[left])
        HiveTableScan(table=[[default.part]], table:alias=[part])
        HiveProject(_o__c0=[$0], alwaysTrue7515=[true], p_type=[$1])
          HiveProject(_o__c0=[$1], p_type=[$0])
            HiveAggregate(group=[{0}], agg#0=[avg($1)])
              HiveProject(p_type=[$1], $f0=[$0])
                HiveProject($f0=[$5], p_type=[$4])
                  LogicalFilter(condition=[=($4, $4)])
                    HiveTableScan(table=[[default.part_null]], table:alias=[part_null])
```

4 Future work

- Support scalar subqueries in other places beside WHERE/HAVING.
- Support correlated subqueries with windowing clause (HIVE-15759).
- Allow EXISTS/NOT EXISTS correlated queries with aggregate (HIVE-15757).
- Allow scalar correlated subqueries with aggregates with non-equi correlated predicates (HIVE-15758) .

- Improve IN/NOT IN correlated subqueries with aggregate to enable queries with empty groups.