Query plan changes

This document summarizes the Hive query plan changes seen after upgrading the Calcite version from 1.25.0 to 1.33.0.

Sections that are highlighted with orange indicate behavior changes and sections with red colour indicate subtle regressions.

Redundant IS NOT NULL predicate

In some cases an extra IS NOT NULL predicate appears in the plan, which is redundant. The extra predicate appears mostly when there are BETWEEN and IN clauses.

ql/src/test/results/clientpositive/tez/explainanalyze_4.q.out
ql/src/test/results/clientpositive/llap/explainuser_4.q.out
cint BETWEEN 1000000 AND 3000000
cint BETWEEN 1000000 AND 3000000 and cint is not null

ql/src/test/results/clientpositive/llap/sharedwork.q.out
ql/src/test/results/clientpositive/llap/pcs.q.out
filterExpr: (ds) IN ('2000-04-09', '2000-04-08') (type: boolean)
filterExpr: (ds is not null and (ds) IN ('2000-04-08', '2000-04-09')) (type: boolean)

This is a small regression tracked under:
https://issues.apache.org/jira/browse/CALCITE-6787
https://issues.apache.org/jira/browse/HIVE-28910

The extra IS NOT NULL predicate appears mostly due to the presence and expansion of the SEARCH operator. The removal could be done as part of the simplification process or during the expansion thus possibly both of the aforementioned tickets are necessary.
Nevertheless, the performance impact of the extra IS NOT NULL predicate is probably

Inequalities to ranges

The transformation is performed during expression simplification.

```
HiveFilter(condition=[AND(<>($0, 2), <>($0, 5))])
  HiveTableScan(table=[[default, t1]], table:alias=[t1])
HiveFilter(condition=[SEARCH($0, Sarg[(-∞..2), (2..5), (5..+∞)])])
  HiveTableScan(table=[[default, t1]], table:alias=[t1])
```

The CBO example above does not appear yet in the tests but the simplification can be observed in some physical plans such as the ones that follow.

ql/src/test/results/clientpositive/llap/ppd_join.q.out

```
HiveProject(c1=[$0], c4=[$2])
  HiveJoin(condition=[AND(=($0, $1), BETWEEN(true, _UTF-16LE'50':VARCHAR(2147483647)
CHARACTER SET "UTF-16LE", $1, $0))], joinType=[inner], algorithm=[none], cost=[not
available])
    HiveProject(c1=[$0])
      HiveFilter(condition=[AND(OR(<($1, _UTF-16LE'val_50'), >($0, _UTF-16LE'2')),
SEARCH($0, Sarg[(_UTF-16LE'20':VARCHAR(2147483647) CHARACTER SET
```

"UTF-16LE"..._UTF-16LE'4':VARCHAR(2147483647) CHARACTER SET "UTF-16LE"), (_UTF-16LE'4':VARCHAR(2147483647) CHARACTER SET "UTF-16LE"..._UTF-16LE'400':VARCHAR(2147483647) CHARACTER SET "UTF-16LE"); NULL AS FALSE]:VARCHAR(2147483647) CHARACTER SET "UTF-16LE"))])
HiveTableScan(table=[[default, src]], table:alias=[src])
HiveProject(key=[$0], value=[$1])
HiveFilter(condition=[SEARCH($0, Sarg[(_UTF-16LE'20':VARCHAR(2147483647) CHARACTER SET "UTF-16LE"..._UTF-16LE'4':VARCHAR(2147483647) CHARACTER SET "UTF-16LE"), (_UTF-16LE'4':VARCHAR(2147483647) CHARACTER SET "UTF-16LE"..._UTF-16LE'400':VARCHAR(2147483647) CHARACTER SET "UTF-16LE"); NULL AS FALSE]:VARCHAR(2147483647) CHARACTER SET "UTF-16LE")])
HiveTableScan(table=[[default, src]], table:alias=[src])

filterExpr: ((key > '20') and (key < '400') and ((value < 'val_50') or (key > '2')) and (key <> '4')) (type: boolean)
filterExpr: (((value < 'val_50') or (key > '2')) and key is not null and (((key > '20') and (key < '4')) or ((key > '4') and (key < '400')))) (type: boolean)

The new expression in the physical plan is the expanded form of the highlighted HiveFilter of the logical plan.

ql/src/test/results/clientpositive/llap/ppd_join3.q.out
ql/src/test/results/clientpositive/llap/ppd_join2.q.out
(key < '400') and
(key <> '14') and
(key <> '305') and
(key <> '302') and
(key <> '311')
key is not null and
((key < '14') or
((key > '14') and (key < '302')) or
((key > '302') and (key < '305')) or
((key > '305') and (key < '311')) or
((key > '311') and (key < '400')))

At the logical level using the SEARCH representation for inequalities is beneficial since probably we can achieve more powerful simplifications in the presence of more complex expressions. However, at the physical level the new plan is slower in terms of CPU/Memory cost since it contains more predicates that need to be evaluated on each incoming row. In most cases, the performance degradation will probably go unnoticed unless the query contains a large number of inequality predicates. The change also affects the computation of statistics since more predicates are added in the plan.

We logged https://issues.apache.org/jira/browse/HIVE-28911 for tracking further improvements that we could do to restore the inequalities in the plan.

Residual predicates in joins

In some cases extra/residual predicates appear inside the join condition that were not there before the upgrade.

ql/src/test/results/clientpositive/llap/ppd_join.q.out
ql/src/test/results/clientpositive/llap/ppd_gby_join.q.out
outputColumnNames: _col0, _col1, _col2
residual filter predicates: {'50' NOT BETWEEN _col1 AND _col0}
Statistics: Num rows: 64 Data size: 16960 Basic stats: COMPLETE Column stats: COMPLETE

SELECT src1.c1, src2.c4
FROM
(SELECT src.key as c1, src.value as c2 from src where src.key > '1') src1
JOIN
(SELECT src.key as c3, src.value as c4 from src where src.key > '2') src2
ON src1.c1 = src2.c3 AND src1.c1 < '400'
WHERE src1.c1 > '20' and (src1.c2 < 'val_50' or src1.c1 > '2') and (src2.c3 > '50' or src1.c1 < '50') and (src2.c3 <> '4')

The CBO plans before and after the upgrade for the query are shown below. For readability purposes types (VARCHAR etc.) and charset (_UTF-16LE) specifiers were removed manually.

HiveProject(c1=[$0], c4=[$2])
  HiveJoin(**condition=[=($0, $1)]**, joinType=[inner], algorithm=[none], cost=[not available])
    HiveProject(c1=[$0])
      HiveFilter(condition=[AND(>($0, '20'), <($0, '400'), OR(<($1, 'val_50'), >($0, '2')), <>($0, '4'))])
        HiveTableScan(table=[[default, src]], table:alias=[src])
    HiveProject(key=[$0], value=[$1])
      HiveFilter(condition=[AND(>($0, '20'), <($0, '400'), <>($0, '4'))])
        HiveTableScan(table=[[default, src]], table:alias=[src])
HiveProject(c1=[$0], c4=[$2])
  HiveJoin(**condition=[AND(=($0, $1), BETWEEN(true, '50', $1, $0))]**, joinType=[inner], algorithm=[none], cost=[not available])
    HiveProject(c1=[$0])
      HiveFilter(condition=[AND(OR(<($1, 'val_50'), >($0, '2')), SEARCH($0, Sarg[('20'..'4'), ('4'..'400'); NULL AS FALSE]))])
        HiveTableScan(table=[[default, src]], table:alias=[src])
    HiveProject(key=[$0], value=[$1])
      HiveFilter(condition=[SEARCH($0, Sarg[('20'..'4'), ('4'..'400'); NULL AS FALSE])])

```
    HiveTableScan(table=[[default, src]], table:alias=[src])
```

The focus in this case is the JOIN condition that after the upgrade contains the BETWEEN(true, '50', $1, $0) predicate that we can see as a residual predicate in the physical plan. This predicate seems slightly awkward but it is valid and in plain SQL it would be something like:

WHERE ‘50’ NOT BETWEEN src1.key AND src2.key

The BETWEEN predicate is a result of applying the HivePointLookupOptimizerRule on the following join expression:

HiveJoin(condition=[AND(=($0, $2), OR(>($2, _UTF-16LE'50'), <($0, _UTF-16LE'50')))], …)

The rule transforms the OR predicate to the aforementioned BETWEEN by factoring out the literal ‘50’. In terms of correctness the transformation is valid and the plan with the BETWEEN predicate remains correct. I am not sure if the point lookup transformation based on literals is useful but this is out of the scope of this discussion.
Before the upgrade the JOIN condition was simplified by the JoinReduceExpressionsRule.

```
AND(=($0, $2), OR(>($2, _UTF-16LE'50'), <($0, _UTF-16LE'50')))
=($0, $2)
```

The OR term was removed by exploiting the pulled up predicates coming from the join inputs. Note that OR cannot be removed if we don’t have the extra information about the predicates.

After the upgrade and the introduction of the SEARCH operator (CALCITE-4173) the simplification is not performed. The problem is tracked under CALCITE-6793.
negligible thus it's not blocking for the upgrade.

Join disjunction pushdown

The HiveJoinPushTransitivePredicatesRule tries to extract predicates from one side of a join and push them down on the other side if possible. In the past, we had various problems when pushing down predicates with disjunction so disabled the disjunction pushdown via HIVE-28310 and other related tickets.

After the Calcite upgrade and the introduction of the SEARCH operator, which among other things models disjunction with inequalities, the pushdown kicks-in and by-passes the existing checks for disjunctive predicates.

The pushdown is correct and basically reverts the effect of HIVE-28310. In addition, all known issues (i.e., cbo_join_transitive_pred_loop*) pass successfully irrespective of the value of the hive.optimize.join.disjunctive.transitive.predicates.pushdown property. At this stage it seems that

we can lift the limitation of disjunctive predicate pushdown and remove the property and related code altogether (https://issues.apache.org/jira/browse/HIVE-28924).

ql/src/test/results/clientpositive/llap/cbo_join_transitive_pred_loop_1.q.out
HiveFilter(condition=[IS NOT NULL($0)])
HiveFilter(condition=[AND(IS NOT NULL($0), IN($0, 202110, 202503))])
ql/src/test/results/clientpositive/llap/cbo_join_transitive_pred_loop_3.q.out
ql/src/test/results/clientpositive/llap/cbo_join_transitive_pred_loop_4.q.out

ql/src/test/results/clientpositive/llap/materialized_view_rewrite_7.q.out
ql/src/test/results/clientpositive/llap/correlationoptimizer8.q.out
ql/src/test/results/clientpositive/llap/join35.q.out
ql/src/test/results/clientpositive/llap/join34.q.out
INNER JOIN (SELECT `key`, `value`
FROM `default`.`src1`
WHERE `key` IS NOT NULL) AS `t6` ON `t4`.`key` = `t6`.`key`
INNER JOIN (SELECT `key`, `value`
FROM `default`.`src1`
WHERE **(CAST(`key` AS DOUBLE) < 20 OR CAST(`key` AS DOUBLE) > 100)** AND `key` IS NOT NULL) AS `t6` ON `t4`.`key` = `t6`.`key`

ql/src/test/results/clientpositive/llap/materialized_view_rewrite_by_text_9.q.out
HiveSemiJoin(condition=[=($0, $1)], joinType=[semi])
  HiveProject(col0=[$0])
HiveFilter(condition=[IS NOT NULL($0)])
HiveFilter(condition=[AND(IS NOT NULL($0), IN($0, 1, 2))])

The changes in stats/ratios/factors are a result of the filter pushdown since they also appeared in previous changes when the pushdown was removed (HIVE-28310).

Unconditional OR to IN/BETWEEN transformation

OR expressions comprising equalities of a column with literals are combined into IN predicates unconditionally.

ql/src/test/results/clientpositive/llap/in_typecheck_char.q.out
ql/src/test/results/clientpositive/llap/pointlookup3.q.out
filterExpr: ((key = 1) or (key = 2)) (type: boolean)
filterExpr: (key) IN (1, 2) (type: boolean)

Before the upgrade, the OR to IN/BETWEEN transformation was performed only via the HivePointLookupOptimizerRule that is controlled by the following properties:

- hive.optimize.point.lookup
- hive.optimize.point.lookup.min

After the upgrade, the RexSimplify can collapse OR comprising equalities with literals to the new internal SEARCH operator. The simplifier runs in various places during the optimization phase and cannot be switched off via a property.

Towards the end of the optimization phase we expand SEARCH to IN, BETWEEN, and other comparison operators. During the expansion we always favor IN and BETWEEN since they are more efficient and produce better statistics which also explains partly why the HivePointLookupOptimizerRule was introduced.

The point lookup rule is always on and by default triggers when there are two elements in a disjunction. In most cases the unconditional rewrite of OR to IN/BETWEEN should not be a problem although as indicated by the aforementioned tests the transformation cannot be switched off by tuning the point lookup properties.

At some point we considered making the expansion of SEARCH to OR (instead of IN) configurable based on some property (existing or new) but it didn't make much sense given that we already paid the cost of collapsing OR to SEARCH.

With the introduction of the SEARCH and the new simplifications some parts of HivePointLookupOptimizerRule are redundant thus we logged https://issues.apache.org/jira/browse/HIVE-28907 to track further improvements around this area.

Extra fields/nullable entries in JSON CBOPlan

The JSON serialization of the CBOPlan appears when we run EXPLAIN FORMATTED with a SQL query. After the upgrade the rowType JSON object contains some new "fields" and "nullable" entries (highlighted with light green below) when we have fields of type STRUCT. Every table has a ROW__ID field that is of type STRUCT so all EXPLAIN FORMATTED outputs are affected.

ql/src/test/results/clientpositive/llap/concat_op.q.out

```
"rowType": {
      "fields": [
        {
          "type": "VARCHAR",
          "nullable": true,
          "precision": 2147483647,
          "name": "key"
        },
        …
        {
          "fields": {
            "fields": [
              {
                "type": "BIGINT",
```

```
            "nullable": true,
            "name": "writeid"
          },
          {
            "type": "INTEGER",
            "nullable": true,
            "name": "bucketid"
          },
          {
            "type": "BIGINT",
            "nullable": true,
            "name": "rowid"
          }
        ],
        "nullable": true
      },
      "nullable": true,
      "name": "ROW__ID"
    },
    {
      "type": "BOOLEAN",
      "nullable": true,
      "name": "ROW__IS__DELETED"
    }
  ],
  "nullable": false
}
```

The new entries are redundant and can be dropped. This is tracked under CALCITE-6832.

This behavior change can potentially break 3rd-party applications that could rely on the JSON serialization but such usages should be pretty limited so it is not considered a blocker at this stage.

EXPLAIN EXTENDED (OPTIMIZED SQL)

The EXPLAIN EXTENDED command prints various information including an OPTIMIZED SQL entry that is the SQL serialization of the CBO plan.

New entry

In case the serialization fails, a warning with the exception is printed in the HS2 logs and the entry is missing in the output. As part of the upgrade some bugs were fixed in JdbcImplementor/SqlToRelConverter thus serialization that was failing before is not failing now leading to new OPTIMIZED SQL entries in the output.

ql/src/test/results/clientpositive/llap/pointlookup3.q.out
ql/src/test/results/clientpositive/llap/pcs.q.out
explain extended select /*+ MAPJOIN(pcs_t1) */ a.ds, b.key from pcs_t1 a join pcs_t1 b on

a.ds=b.ds where struct(a.ds, a.key, b.ds) in (struct('2000-04-08',1, '2000-04-09'), struct('2000-04-09',2, '2000-04-08'))
OPTIMIZED SQL: SELECT `t0`.`ds`, `t2`.`key`
FROM (SELECT `key`, `ds`
FROM `default`.`pcs_t1`
WHERE `key` IN (1, 2) AND `ds` IN ('2000-04-08', '2000-04-09')) AS `t0`
INNER JOIN (SELECT `key`, `ds`
FROM `default`.`pcs_t1`
WHERE `ds` IN ('2000-04-08', '2000-04-09')) AS `t2` ON `t0`.`ds` = `t2`.`ds` AND (`t0`.`key`, `t0`.`ds`, `t2`.`ds`) IN ((1, '2000-04-08', '2000-04-09'), (2, '2000-04-09', '2000-04-08'))

Redundant parentheses

ql/src/test/results/clientpositive/llap/auto_join_reordering_values.q.out
WHERE `date` IS NOT NULL AND `dealid` IS NOT NULL AND (`cityid` IS NOT NULL AND `userid` IS NOT NULL)) AS `t2` INNER JOIN (SELECT `date`
WHERE `date` IS NOT NULL AND `dealid` IS NOT NULL AND `cityid` IS NOT NULL AND `userid` IS NOT NULL) AS `t2` INNER JOIN (SELECT `date`

ql/src/test/results/clientpositive/llap/bucket_map_join_tez2.q.out
WHERE `fiscal_year` = '2015' AND (`accounting_period` = 10 AND `join_col` IS NOT NULL)) AS `t0`
WHERE `fiscal_year` = '2015' AND `accounting_period` = 10 AND `join_col` IS NOT NULL) AS `t0`

Deletion of extra spaces

ql/src/test/results/clientpositive/llap/pcs.q.out
itests/hive-blobstore/src/test/results/clientpositive/insert_into_table.q.out:53
FROM TABLE(INLINE(ARRAY[(1)]))
FROM TABLE(INLINE(ARRAY[(1)]))

Filter predicate order

The ordering of predicates inside the filter is different. This change affects both logical and physical plans.

ql/src/test/results/clientpositive/llap/correlationoptimizer9.q.out
ql/src/test/results/clientpositive/llap/correlationoptimizer10.q.out
hbase-handler/src/test/results/positive/hbase_custom_key2.q.out
filterExpr: ((key.col1 < '27') and (key.col1 >= '165')) (type: boolean)
filterExpr: ((key.col1 >= '165') and (key.col1 < '27')) (type: boolean)

iceberg/iceberg-handler/src/test/results/positive/delete_iceberg_copy_on_write_unpartitioned.q.out

filterExpr: (((b <> 'one') and (b <> 'four') and (a <> 22)) or ((a = 22) or (b) IN ('one', 'four')) is null or (b) IN ('one', 'four') or (a = 22)) (type: boolean)
filterExpr: (((b) IN ('four', 'one') or (a = 22)) is null or ((b <> 'four') and (b <> 'one') and (a <> 22)) or (b) IN ('four', 'one') or (a = 22)) (type: boolean)
ql/src/test/results/clientpositive/llap/explainuser_1.q.out
ql/src/test/results/clientpositive/llap/sharedwork.q.out
ql/src/test/results/clientpositive/llap/subquery_multi.q.out
ql/src/test/results/clientpositive/llap/subquery_in.q.out
(_col13 is not null or _col5 is null or (_col11 < _col10))
(_col5 is null or _col13 is not null or (_col11 < _col10))

ql/src/test/results/clientpositive/llap/fouter_join_ppr.q.out

filterExpr: ((UDFToDouble(key) < 20.0D) and (UDFToDouble(key) > 15.0D)) (type: boolean)
filterExpr: ((UDFToDouble(key) > 15.0D) and (UDFToDouble(key) < 20.0D)) (type: boolean)

ql/src/test/results/clientpositive/llap/pcs.q.out
filterExpr: ((struct(key,ds)) IN (const struct(1,'2000-04-08'), const struct(2,'2000-04-09')) and (ds = '2008-04-08')) (type: boolean)
filterExpr: ((ds = '2008-04-08') and (struct(key,ds)) IN (const struct(1,'2000-04-08'), const struct(2,'2000-04-09'))) (type: boolean)

Quite often we see that IS NULL appears before other predicates (notably IS NOT NULL). This is probably caused by a change in RexSimplify that was introduced as part of CALCITE-4159. There is not much we can do to restore the previous order and it's not necessary either.

We can also observe that EQUALS comes before IN. Note sure where this is coming from but it is not that important to identify the root cause.

IN clause literals sorted alphabetically

The literals inside the IN predicate are now sorted alphabetically.

ql/src/test/results/clientpositive/perf/tpcds30tb/tez/query8.q.out

```
IN ('89436', '30868', '65085', '22977', '83927', '77557', '58429', … )
IN ('10094', '10391', '10502', '10688', '10827', '10866', '11110', … )
```

Introduction of the SEARCH/Sarg operator in CBO plan.

Expression simplification

There are many predicates that are getting simplified due to improvements in RexSimplify and the new SEARCH representation.

Merge disjunctions with IN, BETWEEN, and comparison operators

The introduction of the SEARCH operator allows more powerful simplifications as it allows the unification of disjunctions with IN, BETWEEN, and comparisons operators (<,>,>=,<=,=,<>).

ql/src/test/results/clientpositive/perf/tpcds30tb/tez/cbo_query34.q.out
AND(IN($6, 2000, 2001, 2002), OR(BETWEEN(false, $9, 1, 3), BETWEEN(false, $9, 25, 28)), OR(<=(1, $9), <=($9, 3), <=(25, $9), <=($9, 28)))
AND(IN($6, 2000, 2001, 2002), OR(BETWEEN(false, $9, 1, 3), BETWEEN(false, $9, 25, 28)), IS NOT NULL($9))
ql/src/test/results/clientpositive/perf/tpcds30tb/tez/query88.q.out
AND(IN($3, 3, 0, 1), OR(AND(=($3, 3), <=($4, 5)), AND(=($3, 0), <=($4, 2)), AND(=($3, 1), <=($4, 3))), OR(<=($4, 5), <=($4, 2), <=($4, 3)))
AND(<=($4, 5), IN($3, 0, 1, 3), OR(AND(=($3, 3), IS NOT NULL($4)), AND(=($3, 0), <=($4, 2)), AND(=($3, 1), <=($4, 3))))

ql/src/test/results/clientpositive/llap/pcs.q.out
(key <> 2) and (key = 3)
key = 3

iceberg/iceberg-handler/src/test/results/positive/delete_iceberg_mixed.q.out
((((id <= 4) and (id <> 2)) or ((id > 4) or (id = 2)) is null) and ((id <= 4) or (id <> 2) or ((id > 4) or (id = 2)) is null) and FILE__PATH is not null)
((((id = 2) or (id > 4)) is null or (id < 2) or ((id > 2) and (id <= 4))) and FILE__PATH is not null)

ql/src/test/results/clientpositive/llap/bucketsortoptimize_insert_7.q.out
filterExpr: ((key < 8) and (key) IN (0, 5)) (type: boolean)
filterExpr: (key) IN (0, 5) (type: boolean)

ql/src/test/results/clientpositive/llap/cbo_rp_simple_select.q.out
filterExpr: (c_int is not null or (c_int = 0)) (type: boolean)
filterExpr: c_int is not null (type: boolean)
ql/src/test/results/clientpositive/llap/filter_cond_pushdown.q.out
SELECT f.key, f.value, m.value
FROM src f JOIN src m ON(f.key = m.key AND m.value is not null AND m.value !='')
WHERE (f.value IN ('2008-04-08','2008-04-10') AND f.value IN ('2008-04-08','2008-04-09') AND m.value='2008-04-10') OR (m.value='2008-04-08')

```
expressions: key (type: string), value (type: string), (value) IN ('2008-04-08', '2008-04-10') (type: boolean), (value) IN ('2008-04-08', '2008-04-09') (type: boolean)
expressions: key (type: string), value (type: string), (value = '2008-04-08') (type: boolean)
```

f.value IN ('2008-04-08','2008-04-10') AND f.value IN ('2008-04-08','2008-04-09') is getting simplified to f.value = '2008-04-08'.

```
filterExpr: ((value) IN ('2008-04-10', '2008-04-08') and (value <> '') and key is not null)
(type: boolean)
filterExpr: ((value) IN ('2008-04-08', '2008-04-10') and key is not null) (type: boolean)
```

The redundant check for "value <> ‘’" was removed.

ql/src/test/results/clientpositive/llap/subquery_ALL.q.out

```
CBO PLAN:
HiveAggregate(group=[{}], agg#0=[count()])
  HiveJoin(condition=[OR($2, AND(>($0, $1), $3, IS NOT TRUE(OR(<=($0, $1), $4))))],
joinType=[inner], algorithm=[none], cost=[not available])
  HiveJoin(condition=[OR($2, AND(>($0, $1), IS NOT TRUE(OR(<=($0, $1), $3))))],
joinType=[inner], algorithm=[none], cost=[not available])
    HiveProject(p_partkey=[$0])
      HiveTableScan(table=[[default, part]], table:alias=[part])
    HiveProject(m=[$0], ==[=($1, 0)], <>=[<>($1, 0)], >=[>($1, $2)])
    HiveProject(m=[$0], EXPR$0=[=($1, 0)], EXPR$1=[>($1, $2)])
      HiveAggregate(group=[{}], m=[MAX($0)], c=[COUNT()], d=[COUNT($0)])
        HiveTableScan(table=[[default, part]], table:alias=[part])
```

The changes are due to simplification of expressions. Mostly expressions like OR(X, Y, !X) are getting simplified to OR(X, Y), which seems reasonable

ql/src/test/results/clientpositive/perf/tpcds30tb/tez/cbo_query13.q.out
ql/src/test/results/clientpositive/perf/tpcds30tb/tez/cbo_query85.q.out
ql/src/test/results/clientpositive/perf/tpcds30tb/tez/query13.q.out
ql/src/test/results/clientpositive/perf/tpcds30tb/tez/query85.q.out

```
OR(
<=(100:DECIMAL(3, 0), $12),
<=($12, 150:DECIMAL(3, 0)),
<=(50:DECIMAL(2, 0), $12),
<=($12, 100:DECIMAL(3, 0)),
<=(150:DECIMAL(3, 0), $12),
<=($12, 200:DECIMAL(3, 0)))
IS NOT NULL ($12)
```

ql/src/test/results/clientpositive/llap/pcr.q.out

```
WHERE `ds` >= '2000-04-08' OR `ds` < '2000-04-10'
WHERE `ds` IS NOT NULL
```

After merging the comparison operators the whole disjunction can be simplified to IS NOT NULL.

Misc

ql/src/test/results/clientpositive/llap/subquery_multi.q.out

Select Operator
 expressions: (**p_type <> '1'**) (type: boolean)
 expressions: **p_type is not null** (type: boolean)

Above the Select Operator there is a Filter Operator with the same predicate so the simplification is valid.

ql/src/test/results/clientpositive/llap/subquery_select.q.out

SELECT p_size, p_size IN (
SELECT MAX(p_size) FROM part p where p.p_type = part.p_type)
FROM part
HiveProject(p_size=[$1], _o__c1=[OR(AND(IS NOT NULL($8), **IS NOT TRUE(OR($5, IS NULL($4)))**), AND(IS TRUE(OR($2, $6)), null, IS NOT TRUE(OR($5, IS NULL($4))), IS NULL($8)))])
 HiveJoin(condition=[AND(=($9, $0), =($1, $7))], joinType=[left], algorithm=[none], cost=[not available])
  HiveJoin(condition=[=($3, $0)], joinType=[left], algorithm=[none], cost=[not available])
   HiveProject(p_type=[$4], p_size=[$5], IS NULL=[IS NULL($5)])
    HiveTableScan(table=[[default, part]], table:alias=[part])
   HiveProject(p_type=[$0], c=[$1], **==[=($1, 0)]**, <=[<($2, $1)])
HiveProject(p_size=[$1], _o__c1=[OR(AND(IS NOT NULL($8), **IS NOT NULL($4), $5**), AND(IS TRUE(OR($2, $6)), null, IS NOT NULL($4), $5, IS NULL($8)))])
 HiveJoin(condition=[AND(=($9, $0), =($1, $7))], joinType=[left], algorithm=[none], cost=[not available])
  HiveJoin(condition=[=($3, $0)], joinType=[left], algorithm=[none], cost=[not available])
   HiveProject(p_type=[$4], p_size=[$5], EXPR$0=[IS NULL($5)])
    HiveTableScan(table=[[default, part]], table:alias=[part])
   HiveProject(p_type=[$0], c=[$1], **EXPR$1=[<>($1, 0)]**, EXPR$13=[<($2, $1)])

The expressions in the projects are introduced by the sub-query extraction logic (HiveSubQueryRemoveRule) when handling the IN sub-query. The initial expression is the following CASE statement:

CASE(=($13, 0), false, IS NULL($13), false, IS NOT NULL($16), true, IS NULL($5), null:BOOLEAN, <($14, $13), null:BOOLEAN, false)

The expression is simplified by RelBuilder and RexSimplify to:

OR(AND(IS NOT NULL($16), IS NOT NULL($13), <>($13, 0)), AND(IS TRUE(OR(IS NULL($5), <($14, $13))), null, IS NOT NULL($13), <>($13, 0), IS NULL($16)))

Later on during the optimization phase some projections, notably <>($13, 0), are pushed down in the tree and eventually lead to EXPR$1=[<>($1, 0)] that we see in the final plan. Before the upgrade some simplifications were not performed, thus the difference between =($1, 0) and <>($1, 0) that were observed in the final plan.

On the top projection AND(IS NOT NULL($8), IS NOT TRUE(OR($5, IS NULL($4)))) becomes AND(IS NOT NULL($8), IS NOT NULL($4), $5), where $5 refers to the highlighted portion of the last line in the corresponding plans, i.e., =($1, 0) and <>($1, 0) respectively.

Expansion of NOT BETWEEN

NOT BETWEEN predicates are always expanded using the greater and less than operators. The change also affects the display of OPTIMIZED SQL.

ql/src/test/results/clientpositive/llap/filter_numeric.q.out
filterExpr: hr NOT BETWEEN 12 AND 14 (type: boolean)
filterExpr: ((hr < 12) or (hr > 14)) (type: boolean)

ql/src/test/results/clientpositive/llap/sharedwork.q.out
CAST(`col_3` AS DATE) BETWEEN DATE '2018-07-01' AND DATE '2019-01-23'
CAST(`col_3` AS DATE) >= DATE '2018-07-01' AND CAST(`col_3` AS DATE) <= DATE '2019-01-23'

With this change NOT BETWEEN disappears completely from the plans. The new version is equally efficient and easier to handle since it relies on primitive operators that are handled better in Calcite and Hive. Since NOT BETWEEN disappears more code could be potentially dropped but it is outside the scope of this commit and upgrade.

Explicit CASTs

Some plans contain new CAST operators while some in some other CASTs are removed.

CAST addition

ql/src/test/results/clientpositive/llap/cbo_aggregate_reduce_functions_rule.q.out
COALESCE($0, 0E0:DOUBLE)
CAST(COALESCE($0, 0E0:DOUBLE)):DOUBLE
EXPLAIN CBO SELECT `$SUM0`(c_numeric) FROM test;
HiveProject(_o__c0=[CAST(COALESCE($0, 0E0:DOUBLE)):DOUBLE])
  HiveAggregate(group=[{}], agg#0=[sum($0)])
    HiveTableScan(table=[[default, test]], table:alias=[test])

There is an extra cast to DOUBLE around COALESCE that is added when the HiveAggregateReduceFunctionsRule is executed to reduce SUM0 to SUM with COALESCE. The extra cast is added for nullability purposes since the return type of COALESCE is a NOT NULLABLE DOUBLE while previously it was NULLABLE DOUBLE.

The difference is caused by the way RexBuilder creates literals (in this case the zero literal) that changed with the following commit.

Refactor: Change return type of RelBuilder.literal from RexNode to RexLiteral (commit 56a86a032ac05ed522846910eea6f884b31820e3)

The return type of COALESCE is inferred from its inputs and they changed slightly after the upgrade.
COALESCE($0, CAST(0E0:DOUBLE):DOUBLE)
COALESCE($0, 0E0:DOUBLE)

An extra nullability cast is not a big deal so its not blocking for the upgrade. The issue is tracked under https://issues.apache.org/jira/browse/HIVE-28925

CAST Removal

ql/src/test/results/clientpositive/llap/vector_case_when_2.q.out
```
    expressions: q548284 (type: int), CAST( CASE WHEN ((q548284 = 4)) THEN (0.8) WHEN ((q548284 = 5)) THEN (1) ELSE (8) END AS decimal(11,1)) (type: decimal(11,1))
    expressions: q548284 (type: int), CASE WHEN ((q548284 = 4)) THEN (0.8) WHEN ((q548284 = 5)) THEN (1) ELSE (8) END (type: decimal(2,1))
```

The change in the physical plan is a direct consequence of a respective change in the logical (CBO) plan where the CAST is removed.

select q548284, CASE WHEN ((q548284 = 4)) THEN (0.8) WHEN ((q548284 = 5)) THEN (1) ELSE (8) END from foo order by q548284 limit 1

```
HiveSortLimit(sort0=[$0], dir0=[ASC], fetch=[1])
  HiveProject(q548284=[$0], _o__c1=[CAST(CASE(=($0, 4), 0.8:DECIMAL(1, 1), =($0, 5), 1:DECIMAL(11, 1), 8:DECIMAL(11, 1))):DECIMAL(11, 1)])
    HiveTableScan(table=[[default, foo]], table:alias=[foo])
HiveSortLimit(sort0=[$0], dir0=[ASC], fetch=[1])
  HiveProject(q548284=[$0], _o__c1=[CASE(=($0, 4), 0.8:DECIMAL(1, 1), =($0, 5), 1:DECIMAL(11, 1), 8:DECIMAL(11, 1))])
    HiveTableScan(table=[[default, foo]], table:alias=[foo])
```

I haven't confirmed but I guess the CASE expression is already of type DECIMAL(11,1) thus the CAST is removed.

The remaining changes in the vectorized plan output are due to the removal of the cast.

The type of the vectorized column changes from *DECIMAL(11,1)* to *DECIMAL(2,1)* but this seems OK since all the possible values (i.e., 0.8, 1, 8) fit in a DECIMAL(2,1).

Worth mentioning that the new changes in this file seem to restore a regression from HIVE-23100 that is tracked under HIVE-23223.

LEFT OUTER to INNER joins

[CALCITE-5247] FilterJoinRule cannot simplify left join to inner join for `WHERE RHS.C1 IS NOT NULL OR RHS.C2 IS NOT NULL`

CALCITE-5247 allows the conversion of some LEFT OUTER joins to INNER joins.

ql/src/test/results/clientpositive/llap/subquery_notin.q.out
ql/src/test/results/clientpositive/llap/subquery_ANY.q.out

The join type changed in the query:

```
select * from part where p_partkey > ANY (select p_partkey from part p where
p.p_type = part.p_type)
```

The plan before decorrelation

```
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],
ROW__IS__DELETED=[$12])
   HiveFilter(condition=[OR(AND(>($0, $13), IS NOT TRUE(OR(IS NULL($16), =($14,
0)))), AND(>($0, $13), IS NOT TRUE(OR(IS NULL($16), =($14, 0))), IS NOT
TRUE(>($0, $13)), IS NOT TRUE(>($14, $15))))])
     LogicalCorrelate(correlation=[$cor0], joinType=[left],
requiredColumns=[{4}])
       HiveTableScan(table=[[default, part]], table:alias=[part])
       HiveProject(m=[$0], c=[$1], d=[$2], trueLiteral=[true])
         HiveAggregate(group=[{}], m=[MIN($0)], c=[COUNT()], d=[COUNT($0)])
           HiveProject(p_partkey=[$0])
             HiveFilter(condition=[=($4, $cor0.p_type)])
               HiveTableScan(table=[[default, part]], table:alias=[p])
```

We have a Left Correlate and a Filter on top of it

```
   HiveFilter(condition=[OR(AND(>($0, $13), IS NOT TRUE(OR(IS NULL($16), =($14,
0)))), AND(>($0, $13), IS NOT TRUE(OR(IS NULL($16), =($14, 0))), IS NOT
TRUE(>($0, $13)), IS NOT TRUE(>($14, $15))))])
     LogicalCorrelate(correlation=[$cor0], joinType=[left],
requiredColumns=[{4}])
```

The predicate

```
OR(AND(>($0, $13), IS NOT TRUE(OR(IS NULL($16), =($14, 0)))), AND(>($0, $13),
IS NOT TRUE(OR(IS NULL($16), =($14, 0))), IS NOT TRUE(>($0, $13)), IS NOT
TRUE(>($14, $15))))
```

is never true when the input columns coming from the right are NULLs. This is happening when there is no matching row on the right side of the join. From nullability standpoint the expression can be simplified to

```
OR(AND(>($0, $13), …), AND(>($0, $13), …)))
```
When $13 is NULL the whole expression is NULL. Since this is in a Filter operator NULL is treated as false hence the left join and the filter together will produce the same result set as an inner join thus the conversion is correct and useful.

Join ordering

Cardinality estimation

[CALCITE-4208] Improve metadata row count for Join

After CALCITE-4208, the cardinality estimation (row count) for ANTI/LEFT/RIGHT/FULL joins has changed.

ANTI

ql/src/test/results/clientpositive/perf/tpcds30tb/tez/cbo_query78.q.out

Join ordering is mainly determined by LoptOptimizeJoinRule. During the creation of the join tree, the rule has some swapping logic that estimates the cost of the left and right input and puts the smaller always on the right.

The combination of CALCITE-4208 with the existing metadata methods in Hive leads to dramatic changes in the row count of ANTI joins.

Consider for instance the following ANTI join between store_sales and store_returns tables that appears in TPC-DS query78. The join condition (ss_item_sk = sr_item_sk AND ss_ticket_number= sr_ticket_number) is between the primary keys of the two tables. The query is returning all store_sales except those items that were returned.

```
HiveAntiJoin(condition=[AND(=($8, $2), =($0, $7))], joinType=[anti])
    HiveProject(ss_item_sk=[$1], ss_customer_sk=[$2], ss_ticket_number=[$8],
ss_quantity=[$9], ss_wholesale_cost=[$10], ss_sales_price=[$12],
ss_sold_date_sk=[$22])
      HiveFilter(condition=[IS NOT NULL($22)])
        HiveTableScan(table=[[default, store_sales]],
table:alias=[store_sales])
    HiveProject(sr_item_sk=[$1], sr_ticket_number=[$8])
      HiveTableScan(table=[[default, store_returns]],
table:alias=[store_returns])
```

The cardinalities of the tables in scale factor 30TB are the following:

- store_sales: 82,510,879,939
- store_returns: 8,634,166,995

In an ideal world with a perfect cost model, the row count of the ANTI join would be 73,876,712,944 (82,510,879,939 - 8,634,166,995). The actual row counts between the version are shown below:

- Before upgrade: 1
- After upgrade: 82,510,879,939

Query 78, contains an INNER join from the ANTI join result above and the date_dim table that has a cardinality of 73,049.

LEFT/RIGHT

ql/src/test/results/clientpositive/llap/auto_join30.q.out
ql/src/test/results/clientpositive/llap/subquery_join_rewrite.q.out

The query plan has one LEFT join and one INNER join so the LoptOptimizeJoinRule must decide which one to perform first.

Before CALCITE-4208, an INNER join and LEFT/RIGHT join over the same inputs and the same condition returns exactly the same cardinality so all plans that are generated by the rule have exactly the same cost and there is no clear winner. The rule generates the following candidate plans and both have exactly the same cost (i.e., 18.5).

```
TopTree =
HiveJoin(condition=[true], joinType=[inner], algorithm=[none], cost=[{9.0 rows, 0.0 cpu, 0.0 io}])
  HiveJoin(condition=[=($0, $2)], joinType=[left], algorithm=[none], cost=[{9.5 rows, 0.0 cpu, 0.0
io}])
    HiveProject(ws_order_number=[$0], ws_warehouse_sk=[$1])
      HiveTableScan(table=[[default, web_sales]], table:alias=[ws1])
    HiveProject(ws_order_number=[$0], literalTrue=[true])
      HiveFilter(condition=[IS NOT NULL($0)])
        HiveSortLimit(sort0=[$0], dir0=[ASC], offset=[2], fetch=[1])
          HiveProject(ws_order_number=[$0])
            HiveTableScan(table=[[default, web_sales]], table:alias=[ws2])
  HiveAggregate(group=[{}], c=[COUNT()], ck=[COUNT($0)])
    HiveSortLimit(sort0=[$0], dir0=[ASC], offset=[2], fetch=[1])
      HiveProject(ws_order_number=[$0])
        HiveTableScan(table=[[default, web_sales]], table:alias=[ws2])

PushDownTree =
HiveJoin(condition=[=($0, $4)], joinType=[left], algorithm=[none], cost=[{9.5 rows, 0.0 cpu, 0.0
io}])
  HiveJoin(condition=[true], joinType=[inner], algorithm=[none], cost=[{9.0 rows, 0.0 cpu, 0.0 io}])
    HiveProject(ws_order_number=[$0], ws_warehouse_sk=[$1])
      HiveTableScan(table=[[default, web_sales]], table:alias=[ws1])
```

```
    HiveAggregate(group=[{}], c=[COUNT()], ck=[COUNT($0)])
      HiveSortLimit(sort0=[$0], dir0=[ASC], offset=[2], fetch=[1])
        HiveProject(ws_order_number=[$0])
          HiveTableScan(table=[[default, web_sales]], table:alias=[ws2])
  HiveProject(ws_order_number=[$0], literalTrue=[true])
    HiveFilter(condition=[IS NOT NULL($0)])
      HiveSortLimit(sort0=[$0], dir0=[ASC], offset=[2], fetch=[1])
        HiveProject(ws_order_number=[$0])
          HiveTableScan(table=[[default, web_sales]], table:alias=[ws2])
```

The final selection between the two depends entirely on the order that the candidates are generated and [HepPlanner happens to retain the first](#).

After CALCITE-4208, the LEFT join has a bigger cardinality estimation (row count) thus performing the LEFT join after the INNER join leads to a better plan that is reflected in the cost. The join ordering rule will only retain the best candidate that is the following.

```
HiveJoin(condition=[=($0, $4)], joinType=[left], algorithm=[none], cost=[{9.5 rows, 0.0 cpu, 0.0
io}])
  HiveJoin(condition=[true], joinType=[inner], algorithm=[none], cost=[{9.0 rows, 0.0 cpu, 0.0 io}])
    HiveProject(ws_order_number=[$0], ws_warehouse_sk=[$1])
      HiveTableScan(table=[[default, web_sales]], table:alias=[ws1])
    HiveAggregate(group=[{}], c=[COUNT()], ck=[COUNT($0)])
      HiveSortLimit(sort0=[$0], dir0=[ASC], offset=[2], fetch=[1])
        HiveProject(ws_order_number=[$0])
          HiveTableScan(table=[[default, web_sales]], table:alias=[ws2])
  HiveProject(ws_order_number=[$0], literalTrue=[true])
    HiveFilter(condition=[IS NOT NULL($0)])
      HiveSortLimit(sort0=[$0], dir0=[ASC], offset=[2], fetch=[1])
        HiveProject(ws_order_number=[$0])
          HiveTableScan(table=[[default, web_sales]], table:alias=[ws2])
```

ql/src/test/results/clientpositive/llap/vector_join30.q.out
The query joins the same table 3 times on the same column. One join is an INNER join and the other is LEFT join. After the upgrade the LEFT join is performed after the INNER join.

Costing ties

ql/src/test/results/clientpositive/perf/tpcds30tb/tez/cbo_query85.q.out

```
HiveJoin(condition=[AND(=($16, $13), =($17, $14), =($15, $1))], joinType=[inner],
algorithm=[none], cost=[not available])
  HiveJoin(condition=[=($12, $3)], joinType=[inner], algorithm=[none], cost=[not available])
    HiveJoin(web_returns with customer_address)
    HiveProject(cd_demo_sk=[$0], cd_marital_status=[$2], cd_education_status=[$3])
```

HiveFilter(condition=[AND(IN($2, 'M', 'D', 'U'), IN($3, '4 yr Degree', 'Primary', 'Advanced Degree'))])
HiveTableScan(table=[[default, customer_demographics]], **table:alias=[cd2]**)
HiveProject(cd_demo_sk=[$0], cd_marital_status=[$2], cd_education_status=[$3], ==[=($2, 'M')], =4=[=($3, '4 yr Degree')], =5=[=($2, 'D')], =6=[=($3, 'Primary')], =7=[=($2, 'U')], =8=[=($3, 'Advanced Degree')])
HiveFilter(condition=[AND(IN($2, 'M', 'D', 'U'), IN($3, '4 yr Degree', 'Primary ', 'Advanced Degree'))])
HiveTableScan(table=[[default, customer_demographics]], **table:alias=[cd1]**)
HiveJoin(condition=[AND(=($16, $13), =($17, $14), =($15, $1))], joinType=[inner], algorithm=[none], cost=[not available])
HiveJoin(condition=[=($12, $3)], joinType=[inner], algorithm=[none], cost=[not available])
HiveJoin(web_returns with customer_address)
HiveProject(cd_demo_sk=[$0], cd_marital_status=[$2], cd_education_status=[$3], ==[=($2, 'M')], =4=[=($3, '4 yr Degree')], =5=[=($2, 'D')], =6=[=($3, 'Primary')], =7=[=($2, 'U')], =8=[=($3, 'Advanced Degree')])
HiveFilter(condition=[AND(IN($2, 'M', 'D', 'U'), IN($3, '4 yr Degree', 'Primary ', 'Advanced Degree'))])
HiveTableScan(table=[[default, customer_demographics]], **table:alias=[cd1]**)
HiveProject(cd_demo_sk=[$0], cd_marital_status=[$2], cd_education_status=[$3])
HiveFilter(condition=[AND(IN($2, 'M', 'D', 'U'), IN($3, '4 yr Degree', 'Primary', 'Advanced Degree'))])
HiveTableScan(table=[[default, customer_demographics]], **table:alias=[cd2]**)

Basically the cost (rowcount) of subplans cd1 and cd2 is identical (493920.00000000006) so the order they appear in the join tree is purely based on the data structures used by the join ordering rule. The different number of columns in the HiveProject operator does not alter the row count thus does not impact the cost.

Column aliases

Different aliases for expressions inside projects. The changes can be seen in query plans and also in OPTIMIZED SQL since the latter is generated from them.

ql/src/test/results/clientpositive/llap/cardinality_preserving_join_opt.q.out
HiveProject(ss_customer_sk=[$1], *=[*(+($0, $0), $2)], CAST=[CAST($1):BIGINT])
HiveProject(ss_customer_sk=[$1], EXPR$0=[*(+($0, $0), $2)], EXPR$1=[CAST($1):BIGINT])

ql/src/test/results/clientpositive/encrypted/encryption_join_unencrypted_tbl.q.out
FROM (SELECT `key`, `value`, CAST(`key` AS DOUBLE) AS `CAST`
FROM (SELECT `key`, `value`, CAST(`key` AS DOUBLE) AS `EXPR$0`

Basic Statistics

There are some changes in basic statistics such as “Num rows” and “Data size” that occur as a side-effect of other plan changes listed in this document such as:

- Redundant IS NOT NULL predicate
- Residual predicates in JOIN condition
- Disjunctive predicate pushdown

ql/src/test/results/clientpositive/llap/explainuser_4.q.out
ql/src/test/results/clientpositive/llap/correlationoptimizer8.q.out

CBO Cost

Slight changes in the estimated cost in CBO plans.

ql/src/test/results/clientpositive/perf/tpcds30tb/tez/cbo_ext_query1.q.out
HiveJoin(condition=[AND(=($3, $7), >($4, $6))], joinType=[inner], algorithm=[none], cost=[{13.1400**73024144858 rows**, 0.0 cpu, 0.0 io}])
HiveJoin(condition=[AND(=($3, $7), >($4, $6))], joinType=[inner], algorithm=[none], cost=[{13.1400**68448808671 rows**, 0.0 cpu, 0.0 io}])

Shuffle join warning message

The table aliases that take part in the shuffle join appear differently in the warning message. Given that the plan itself appears unchanged the change is completely safe.

ql/src/test/results/clientpositive/smb_mapjoin_47.q.out
Warning: Shuffle Join JOIN[12][tables = [hdt_1, hdt_2, hdt_0]] in Stage 'Stage-1:MAPRED' is a cross product
Warning: Shuffle Join JOIN[12][tables = [hdt_0, hdt_1, hdt_2]] in Stage 'Stage-1:MAPRED' is a cross product