

# Kylin5 supports using computable columns as Join Key and partition columns

## 1 背景

支持将可计算列用作连接键和分区列

## 2 后端设计

可计算列作为关联键和分区列对于 KE 的影响包括建模、构建、查询。

### 2.1 建模

建模时，可计算列为什么不能用关联键？

在界面上设计模型的时候，如果用户添加可计算列，那么就要校验可计算列，提交的时候会把当前的信息封装在 ModelRequest 中，然后将它转换成模型。在转换的时候，它会根据 ModelRequest 拉取了多少表以及界面上的 join 关系信息获取项目中的表信息。此时的表信息其实是不包含可计算列信息的。如果需要用可计算列作为 join key，那么就需要让 CC 列出现在表信息中，因此需要对表信息进行扩展。从而在接下来选择 join key 的时候能选择 CC。

建模时，可计算列为什么不能用作分区列？

这是因为被前端限制，所以前端放开这个限制就可以。目前发现推断分区列类型的时候报错，两种方式处理这个问题：1) 报错提示更友好点；2) 允许推断可计算列的类型。

暂时不支持 cc 作为分区列时候的类型推断；

暂不支持 cc 作为分区列的 segment 裁剪；

### 2.2 构建

构建的时候要支持可计算列作为关联键，主要在于构造 SegmentFlatTable 这边要能够正确的处理 CC 列的情况。已有的代码构建平表时过滤了可计算列，现在放开就可以实现这个功能。

```
private def generateDatasetOnTable(ss: SparkSession, tableRef: TableRef): Dataset[Row] = {
-   val tableCols = tableRef.getColumns.asScala.map(_.getColumnDesc).filter(!_.isComputedColumn).toArray
+   val tableCols = tableRef.getColumns.asScala.map(_.getColumnDesc).toArray
    val structType = SchemaProcessor.buildSchemaWithRawTable(tableCols)
    val alias = tableRef.getAlias
    val dataset = ss.createDataFrame(Lists.newArrayList[Row], structType).alias(alias)
    SegmentFlatTable.wrapAlias(dataset, alias)
}
```

为什么删除了这行就能生效？

```
1 protected def generateFlatTablePart(): Dataset[Row] = {
2   val recoveredDS = tryRecoverFTDS()
3   if (recoveredDS.nonEmpty) {
4     return recoveredDS.get
5   }
6   var flatTableDS = if (needJoin) {
7     val lookupTableDSMap = generateLookupTables()
8     if (inferFiltersEnabled) {
9       FiltersUtil.initFilters(tableDesc, lookupTableDSMap)
10    }
11    val jointDS = joinFactTableWithLookupTables(fastFactTableDS, lookupTableDSMa
12    concatCCs(jointDS, factTableCCs)
13  } else {
14    fastFactTableDS
15  }
16  flatTableDS = applyFilterCondition(flatTableDS)
17  changeSchemeToColumnId(flatTableDS, tableDesc)
18 }
```

在修正前面的逻辑的情况下，jointDS 中就包含了 CC 的信息，接下来 concatCCs 就对 CC 做了替换处理

```
1 private def concatCCs(table: Dataset[Row],
2   computColumns: Set[TblColRef]): Dataset[Row] = {
3   val matchedCols = selectColumnsInTable(table, computColumns)
4   var tableWithCcs = table
5   matchedCols.foreach(m =>
6     tableWithCcs = tableWithCcs.withColumn(
7       convertFromDot(m.getBackTickIdentity),
8       expr(convertFromDot(m.getBackTickExpressionInSourceDB)))
9   )
10  tableWithCcs
11 }
```

F cc = f.c1 + b.c2. F join A on f.c1 + b.c2 = A.c3

A B

F cc = a.c3 + 1 F join A on a.c3 + 1 = A.c4

## 2.3 查询

CC 作为 join key, 那么查询时的 join key 需要使用 cc 名称, 用表达式目前已知的是在嵌套 cc 和 单列作为 cc 的时候会无法命中模型;

举例:

- A Join b on A.cc = B.col, cc = cc1 + 1, cc1 = A.a + A.b 可以命中模型
- A Join b on A.cc1 + 1 = B.col, cc = cc1 + 1, cc1 = A.a + A.b 无法命中模型
- A Join b on A.cc = B.col, cc = A.a 可以命中模型
- A Join b on A.a = B.col, cc = A.a 无法命中模型

CC 作为 join key, 跨表的 cc 目前暂时不支持;

举例:

- A Join b on A.cc = B.col, cc = A.a + C.d 不支持
- A Join b on A.cc = B.col, cc = B.b + 1 不支持

## 3 主要改动点

- 分离了 QueryUtil 和 PushdownUtil 的一些方法, 原则上 PushdownUtil 可以使用 QueryUtil 中的方法, 反过来不可以, 如果出现这种情况, 需要考虑方法放置的地方是否合适; 并且对于一些 transformer 类的初始化逻辑进行了改写, 修复一些 sonar 顽固问题;
- QueryAliasMatcher 中 resolveComputedColumnRef 方法中的 bug 修复, 3x 代码中修复了, 迁移过来;
- 将查询 Rule 相关的静态方法抽取出来了一个 RuleUtils 工具类, 而不是揉杂在 QueryUtil 里面;
- ColumnDesc 中分别定义了 getComputedColumnExpr 和 getDoubleQuoteInnerExpr 方法, 前者给下压用, 后者给 Calcite 使用;
- TblColRef 中的 getDoubleQuoteExp 调用了 ColumnDesc 中的 getDoubleQuoteInnerExpr, getBackTickExp 调用了 ColumnDesc 的 getComputedColumnExpr;

```
public String getDoubleQuoteExpressionInSourceDB() {
    if (column.isComputedColumn())
        return column.getComputedColumnExpr();
    return wrapIdentity(DOUBLE_QUOTE);
}

public String getBackTickExpressionInSourceDB() {
    if (column.isComputedColumn())
        return column.getComputedColumnExpr();
    return wrapIdentity(BACK_TICK);
}

public String getDoubleQuoteExp() {
    if (column.isComputedColumn())
        return column.getDoubleQuoteInnerExpr();
    return wrapIdentity(DOUBLE_QUOTE);
}

public String getBackTickExp() {
    return column.isComputedColumn() ? column.getComputedColumnExpr() : wrapIdentity(BACK_TICK);
}

public String getTable() {
    if (column.getTable() == null) {
```

- TableService 中的 getPartitionColumnFormat 增加了根据分区列表表达式的探测逻辑;

```
try {
    if (tableDesc.isKafkaTable()) {
        List<ByteBuffer> messages = kafkaService.getMessage(tableDesc.getKafkaConfig(), project);
        checkMessage(table, messages);
        Map<String, Object> resp = kafkaService.decodeMessage(messages);
        String message = ((List<String>) resp.get("message")).get(0);
        Map<String, Object> mapping = kafkaService.parseMessage(project, tableDesc.getKafkaConfig(), message);
        Map<String, Object> mappingAllCaps = new HashMap<>();
        mapping.forEach((key, value) -> mappingAllCaps.put(key.toUpperCase(Locale.ROOT), value));
        String cell = (String) mappingAllCaps.get(partitionColumn);
        return DateFormat.proposeDateFormat(cell);
    } else if (partitionExpression == null) {
        List<String> list = PushDownUtil.backtickQuote(partitionColumn.split("\\."));
        String cell = PushDownUtil.probeColFormat(table, String.join(".", list), project);
        return DateFormat.proposeDateFormat(cell);
    } else {
        String cell = PushDownUtil.probeExpFormat(table, partitionExpression, project);
        return DateFormat.proposeDateFormat(cell);
    }
}
```

- ComputedColumnEvalUtil 中评估 CC 列类型的逻辑和 IndexDependencyParser 中的逻辑重复较大, 统一了代码、简化了绝大部分冗余代码;
- ModelSemanticHelper 的 convertToDataModel 增加了表的 CC 到 tabledesc 中保证 Ndatamodel 的 initJoinDesc 正常。
- ModelService 中很多对于是使用 getDoubleQuoteExp 还是 getBackTickExp 的地方进行了纠正, messageModelFilterCondition 删掉了错误的改写导致 filterCondition 既不能用于下压, 也不能被 calcite 解析。
- timestampadd(day, 1, col). timestampadd('day', 1, col)
- JoinsGraph 修复了 columnDescEquals 方法, 这个方法原来可能导致有时候命中索引, 有时候下压, 有时候无法查询;
- SegmentFlatTable、FlatTableAndDictBase 对于单事实表的情况, 平表 dataset 中也加入了 cc 列;
- FlatTableHelper 不再过滤掉cc;