

Kylin5 supports using computable columns as Join Key and partition columns

1 Background

Support for computed columns as join keys and partitioned columns

2 Backend design

The impact of computable columns as associated keys and partitioned columns on KE includes modeling, building, and querying.

2.1 The modeling

Why can't associated keys be used for computable columns when modeling?

When designing a model on the interface, if the user adds a computable column, it is necessary to verify the computable column. When submitting, the current information will be encapsulated in the ModelRequest, and then converted into a model. When converting, it will obtain the table information in the project according to how many tables are pulled by the ModelRequest and the join relationship information on the interface. The table information at this time does not actually contain the computable column information. If you need to use a computable column as a join key, you need to let CC be listed in the table information, so you need to expand the table information. So that you can select CC when you select the join key next.

Why can't computable columns be used as partition columns when modeling?

This is because it is restricted by the front end, so the front end can release this restriction. At present, when it is found that an error is reported when inferring the type of partition column, there are two ways to deal with this problem: 1) The error prompt is more friendly; 2) Allow inference of the type of computable columns.

Temporarily does not support type inference when cc is used as a partition column;

Segment clipping cc as a partition column is not currently supported;

2.2 Build

When building, it is necessary to support computable columns as associated keys, mainly to be able to correctly handle the situation of CC columns when constructing the SegmentFlatTable. The existing code filters the computable columns when building the flat table, and this function can be realized now.

```
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)  
}
```

Why does deleting this line take effect?

```
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 }
```

In the case of modifying the previous logic, jointDS contains CC information, and then concatCCs replaces CC

```
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.withColumnn(  
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 Query

CC as a join key, then the query when the join key needs to use cc name, with the expression is currently known in the nested cc and, when a single column as cc will not hit the model;

Example:

- 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 As a join key, cross-table cc is currently not supported;

Example:

- 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 Main changes

- Some methods of QueryUtil and PushdownUtil are separated. In principle, PushdownUtil can use the methods in QueryUtil, but not vice versa. If this happens, you need to consider whether the place where the methods are placed is appropriate; and the initialization logic of some transformer classes has been rewritten to fix some sonar stubborn problems;
- Bug fix in resolveComputedColumnRef method in QueryAliasMatcher, fixed in 3x code, migrated over;
- A RuleUtils utility class is extracted from the static methods related to the query Rule, rather than mixed in QueryUtil;
- ColumnDesc defines getComputedColumnExpr and getDoubleQuoteInnerExpr methods, the former for the lower pressure, the latter for Calcite use;

- getDoubleQuoteExp in TblColRef calls getDoubleQuoteInnerExpr in ColumnDesc, and getBackTickExp calls getComputedColumnExpr in ColumnDesc;

| | | |
|---|--|---|
| <pre> 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); } </pre> | <pre> 254 251 255 252 256 253 257 254 258 255 259 256 260 257 261 258 262 259 263 260 264 261 265 262 266 263 </pre> | <pre> 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) { </pre> |
|---|--|---|

- The getPartitionColumnFormat in TableService adds probing logic based on partition column expressions;

```

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);
    }
}

```

- The logic of evaluating CC column type in ComputedColumnEvalUtil and the logic in IndexDependencyParser are duplicated greatly, which unifies the code and simplifies most of the redundant code;
- The convertToDataModel of ModelSemanticHelper adds table CC to tabledesc to ensure that the initJoinDesc of Ndatamodel is normal.
- A lot of corrections have been made in ModelService for whether to use getDoubleQuoteExp or getBackTickExp, and massageModelFilterCondition removes the wrong rewriting, resulting in filterCondition being neither used for downloading nor parsed by calcite.
- timestampadd(day, 1, col). timestampadd('day', 1, col)
- JoinsGraph fix columnDescEquals method, this method may lead to the original sometimes hit the index, sometimes depressed, sometimes unable to query;
- SegmentFlatTable, FlatTableAndDictBase For the case of a single fact table, cc columns are also added to the flat table dataset;
- FlatTableHelper no longer filters out cc ;