

KYLIN-5397 半累加度量 Design

Background

半累加度量是一种特殊的聚合度量，这个度量在聚合时有两个输入：时间维度和需要聚合的维度，计算度量时需要考虑时间的变化。

这个设计主要是用于实现半累加度量中的Last Child函数，这个函数的聚合规则是，当输入一个较新的时间值，则将度量值更新为该时间对应的维度值；如果输入一个相同的时间值，则将输入的维度值与当前的度量值相加。典型的应用场景是银行对余额变化的统计分析。

kylin4中有半累加(Last Child)功能，而kylin5目前无法支持。**要求在保证查询性能的前提下，补齐kylin5对半累加度量的支持，当前设计通过对该度量进行预计算达到预期的目的。**

Design

1. 元数据

kylin4使用额外的结构体semi_additive_info补充半累加度量的信息，以区分半累加和普通累加，但与kylin4不同的是，kylin5中半累加Last Child函数独立使用SUM_LC表示，使用expression的值与普通sum度量进行区分，因此不再需要该结构体。kylin5中模型元数据只需要在parameters增加一列，表示时间维度列即可。时间维度列提供对date和timestamp数据类型的支持。

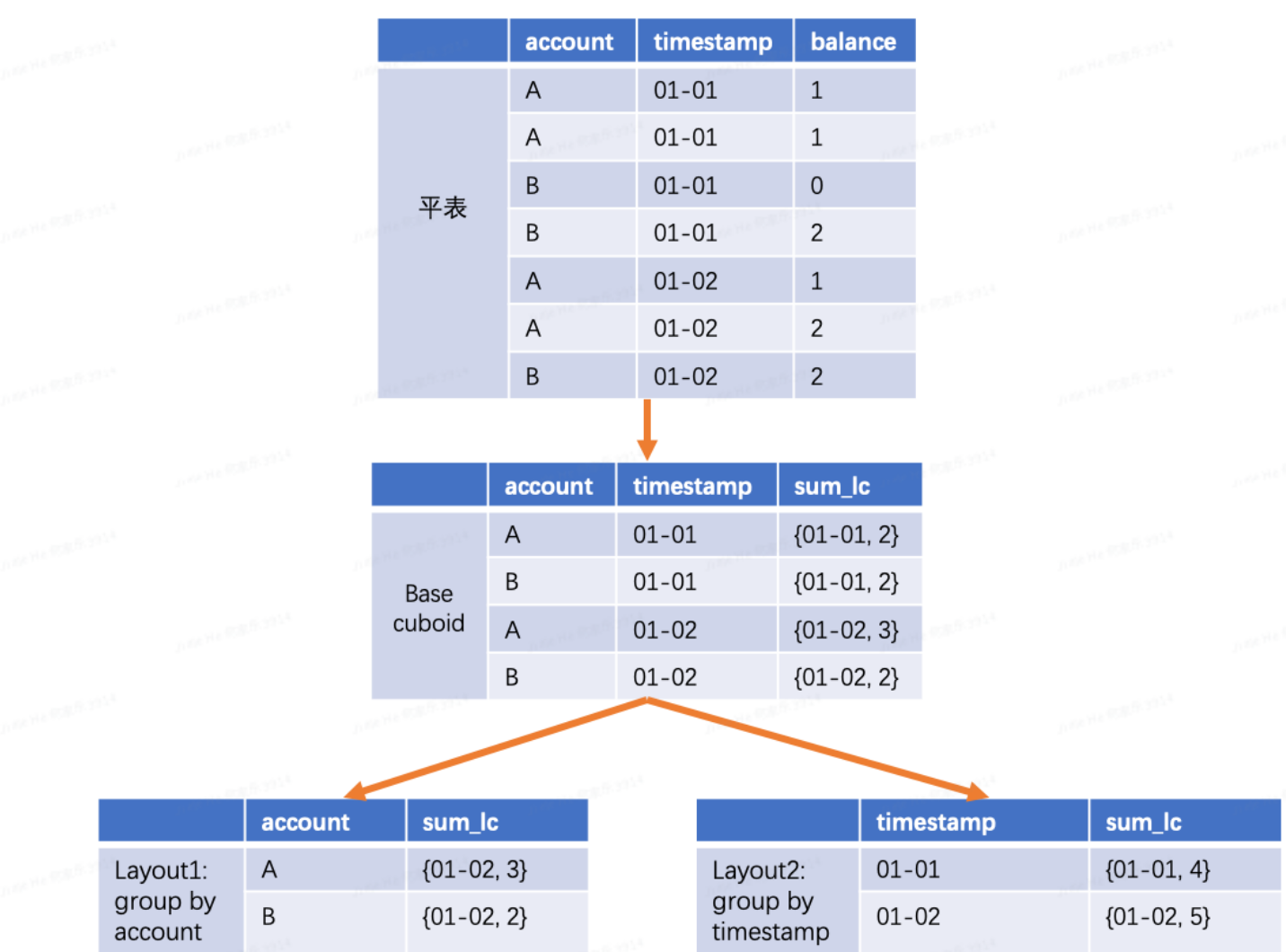
```
1  "all_measures": [{
2    "name": "SEMI_SUM_NAME",
3    "function": {
4      "expression": "SUM_LC",
5      "parameters": [{
6        "type": "column",
7        "value": "TARGET_COLUMN_NAME"
8      }, {
9        "type": "column",
10       "value": "ORDER_DATE_COLUMN_NAME"
11     }],
12     "returntype": "TARGET_COLUMN_DATA_TYPE"
13   },
14   "column" : null,
15   "comment" : "",
16   "id" : 100004,
17   "type" : "NORMAL",
18   "internal_ids" : [ ]
19 }]
```

2. 构建

该度量的构建通过新增一个spark udaf函数SumLC来实现，将layout的DataSet中的每一行数据作为这个函数的输入，其原理是在构建的过程中维护一个SumLCCounter，每一行数据传进来都会使用时间维度列的时间值判断要替换counter中的当前值还是与当前值进行累加，构建结束后输出SumLCCounter。在存储和传输的过程中，SumLCCounter包含时间值和累加值，将其作为整体进行序列化和反序列化。

分层构建会将base cuboid切分出包含不同维度的layout，对于切分出的layout，会利用到上一层计算出的半累加度量结果得到度量值。

举例说明：如下图所列数据，对account进行聚合的layout1，这个layout不包含时间维度列，在计算其度量的过程中，我们首先对A聚合，其上一层base cuboid半累加度量为{01-01, 2}和{01-02, 3}，按照上图中的计算逻辑，因为01-02大于01-01，对A聚合后直接取后者作为度量值，即{01-02, 3}。同理，对B聚合后，得到的度量值为{01-02, 2}。



3. 查询

与kylin4不同的是，带有半累加度量查询的sql语句需要带上时间列，结合时间列用于匹配模型，这个带来的好处是：

- 解除时间维度列的限制，也就是同一个项目下，允许同一个列结合不同的时间维度列，来构建不同的半累加度量。

- 支持查询下压，和击中索引或快照的查询，因为可以结合这个时间列将SUM_LC改写成LAST_VALUE(col)。
- 智能推荐同理可支持。

出于查询准确性的考虑，查询带有半累加度量时，禁用segment pruner。这是因为存在多个segment时，如果因为账户删除等业务场景，造成各个segment数据对不齐。如果此时segment pruner将部分segment过滤掉之后，会有可能造成查询结果不全。

查询的过程同样是利用半累加度量的聚合规则，这里复用SumLCCounter。反序列化后的SumLCCounter同时包装了时间值与半累加的数值，通过比较其中的时间值，按照半累加度量的计算规则，计算得到最后的度量值。

不同情景下的查询过程，以下述数据和查询为例，依次进行说明。

A. 同一个segment下的查询

- 要分析截止到01-03每个账户下的账户余额，使用的查询语句为：`select account, sum_lc(balance,timestamp) from table where timestamp < '01-03' group by account`，这个查询击中聚合了account和timestamp的layout，在通过filter将01-01和01-02的两天的数据过滤出来后，如下方左侧表格所示，对account进行一次聚合，对于账户A，存在01-01和01-02两日的数据，按照聚合规则，最终保留{01-02, 3}；同理，对于账户B，只保留{01-02, 2}。

	A	B	C	D
1	base cuboid	account	timestamp	sum_lc
2		A	01-01	{01-01, 2}
3		B	01-01	{01-01, 2}
4		A	01-02	{01-02, 3}
5		B	01-02	{01-02, 2}

得到的结果为：

A, 3
B, 2

- 要分析每个账户下的账户余额，使用的查询语句为：`select account, sum_lc(balance,timestamp) from table group by account`，这个语句可直接击中下方左侧对account聚合的layout，这个layout在构建过程中已经计算出账户A和B对应的Last Value。将其处理后作为结果返回。

	A	B	C
1	segment0	account	sum_lc
2		A	{01-02, 3}
3		B	{01-02, 2}

得到的结果为：

A, 3
B, 2

- 要分析1月份所有账户的总余额，使用的查询语句为：`select sum_lc(balance,timestamp) from table where MONTH(timestamp) = 1`，对于如下layout，会按照半累加度量的计算规则，逐行判断直到扫描到最后一行数据，逐行扫描中间结果保存在SumLCCounter中。最后将半累加后的结果取出。

	A	B	C
1	account	timestamp	sum_lc
2	A	01-01	{01-01, 2}
3	B	01-01	{01-01, 2}
4	A	01-02	{01-02, 3}
5	B	01-02	{01-02, 2}

逐行扫描中间结果： 得到的结果为：

扫描第一行后： 5

{01-01, 2}

扫描第二行后：

{01-01, 4}

扫描第三行后：

{01-02, 3}

扫描第四行后：

{01-02, 5}

B. 多个segment的查询

- 相较于单segment，多个segment时会可能增加一个后聚合的过程，我们还以分析1月份所有账户的总余额，使用的查询语句为：`select sum_lc(balance,timestamp) from table where MONTH(timestamp) = 1`，对于如下layout，每个segment得到一个中间结果，需要拿到这个中间结果再次进行聚合，再次聚合的过程同样符合半累加度量的计算规则，选择其中日期较新的数据，即为{01-02, 5}。

	A	B	C	D
1		account	timestamp	sum_lc
2	seg0	A	01-01	{01-01, 2}
3		B	01-01	{01-01, 2}
4	seg1	A	01-02	{01-02, 3}
5		B	01-02	{01-02, 2}

每个segment中间结果：

seg0: {01-01, 4}

seg1: {01-02, 5}

再次聚合：

{01-02, 5}

得到的结果：

5

Limitation

- kylin4中存在部分限制，这部分限制在kylin4中通过crossCheckSemiAdditiveConflict方法在新建或者更新Cube时进行校验，对于kylin5，我们也通过代码在新建或者更新模型的接口处进行相同的限制。
 - （限制）只有聚合函数为SUM的度量才可以设置半累加度量，半累加函数当前只支持LastChild
- 半累加列只支持可以进行累加计算的数值类型，这个规则同样会在添加该度量时进行column data type校验。
- 如果事实表原表数据不完整，会造成结果与预期不符合的情况。客户的构建要确保发生在客户的结算日期后，目的是保证源数据是完整的，否则数据不完整，查询结果会与预期有偏差。
- 同一日期下如果存在多条数据，也可能造成聚合结果是不正确的。客户需要对源数据进行清洗，去掉同一天的多条冗余数据，否则冗余数据会造成歧义，构建的结果有可能是错误的。
- 其他产品线影响：分层存储的clickhouse支持last_value函数，考虑通过改写来实现，同下压查询。

Test Suggestion

1. 查询结果符合预期，不会因击中不同模型不同结构的layout导致查询结果变化。
2. 使用不带/带有半累加度量的查询做对比，观察性能的变化，因为计算过程存在额外的编解码，带有半累加度量时性能存在一定下降。
3. 在相关接口处做的限制条件没有遗漏的情况。