实时数仓|Flink SQL之维表join

维表是数仓中的一个概念，维表中的维度属性是观察数据的角度，在建设离线数仓的时候，通常是将维表与事实表进行关联构建星型模型。在实时数仓中，同样也有维表与事实表的概念，其中事实表通常存储在kafka中，维表通常存储在外部设备中(比如MySQL，HBase)。对于每条流式数据，可以关联一个外部维表数据源，为实时计算提供数据关联查询。维表可能是会不断变化的，在维表JOIN时，需指明这条记录关联维表快照的时刻。需要注意是，目前Flink SQL的维表JOIN仅支持对当前时刻维表快照的关联(处理时间语义)，而不支持事实表rowtime所对应的的维表快照(事件时间语义)。通过本文你可以了解到：

- 如何使用Flink SQL创建表
- 如何定义Kafka数据源表
- 如何定义MySQL数据源表
- 什么是Temporal Table Join
- 维表join的案例

Flink SQL创建表

注意：本文的所有操作都是在Flink SQL cli中进行的

创建表的语法

```
CREATE TABLE [catalog_name.][db_name.]table_name
  (
    { <column_definition> | <computed_column_definition> }[ , ...n]
    [ <watermark_definition> ]
  )
  [COMMENT table_comment]
  [PARTITIONED BY (partition_column_name1, partition_column_name2, ...)]
  WITH (key1=val1, key2=val2, ...)
-- 定义表字段
<column_definition>:
  column_name column_type [COMMENT column_comment]
-- 定义计算列
<computed_column_definition>:
  column_name AS computed_column_expression [COMMENT column_comment]
-- 定义水位线

```

```
17 | <watermark_definition>:
     WATERMARK FOR rowtime_column_name AS watermark_strategy_expression
```

解释

COMPUTED COLUMN(计算列)

计算列是一个通过`column_name AS computed_column_expression`生成的虚拟列，产生的计算列不是物理存储在数据源表中。一个计算列可以通过原有数据源表中的某个字段、运算符及内置函数生成。比如，定义一个消费金额的计算列(cost)，可以使用表的价格(price)*数量(quantity)计算得到。

计算列常常被用在定义时间属性(见另一篇文章Flink Table API&SQL编程指南之时间属性(3)，可以通过PROCTIME()函数定义处理时间属性，语法为`proc AS PROCTIME()`。除此之外，计算列可以被用作提取事件时间列，因为原始的事件时间可能不是TIMESTAMP(3)类型或者是存在JSON串中。

> **尖叫提示**：
> 1.在源表上定义计算列，是在读取数据源之后计算的，计算列需要跟在SELECT查询语句之后；
> 2.计算列不能被INSERT语句插入数据，在INSERT语句中，只能包括实际的目标表的schema，不能包括计算列

水位线

水位线定义了表的事件时间属性，其语法为:

```
1 | WATERMARK FOR rowtime_column_name AS watermark_strategy_expression
```

其中`rowtime_column_name`表示表中已经存在的事件时间字段，值得注意的是，该事件时间字段必须是TIMESTAMP(3)类型，即形如`yyyy-MM-dd HH:mm:ss`,如果不是这种形式的数据类型，需要通过定义计算列进行转换。

`watermark_strategy_expression`定义了水位线生成的策略，该表达式的返回数据类型必须是TIMESTAMP(3)类型。

Flink提供了许多常用的水位线生成策略:

- 严格单调递增的水位线：语法为

```
1 | WATERMARK FOR rowtime_column AS rowtime_column
```

即直接使用时间时间戳作为水位线

- 递增水位线:语法为

```
WATERMARK FOR rowtime_column AS rowtime_column - INTERVAL '0.001' SEC
OND
```

- 乱序水位线：语法为

```
WATERMARK FOR rowtime_column AS rowtime_column - INTERVAL 'string' ti
meUnit
-- 比如，允许5秒的乱序
WATERMARK FOR rowtime_column AS rowtime_column - INTERVAL '5' SECOND
```

分区

根据具体的字段创建分区表，每一个分区会对应一个文件路径

WITH 选项

创建Table source或者Table sink需要指定表的属性，属性是以key/value的形式配置的，具体参考其相对应的connector

> **尖叫提示**：
> Note：创建表时指定的表名有三种形式：
> （1）catalog_name.db_name.table_name
> （2）db_name.table_name
> （3）table_name
> 对于第一种形式：会将表注册到一个名为‘catalog_name’的catalog以及一个名为’db_name’d的数据库的元数据中；
> 对于第二种形式：会将表注册到当前执行环境的catalog以及名为‘db_name’的数据库的元数据中；
> 对于第三种形式：会将表注册到当前执行环境的catalog与数据库的元数据中

定义Kafka数据表

kafka是构建实时数仓常用的数据存储设备，使用Flink SQL创建kafka数据源表的语法如下：

```
CREATE TABLE MyKafkaTable (
  ...
) WITH (
  'connector.type' = 'kafka', -- 连接类型
  'connector.version' = '0.11',-- 必选: 可选的kafka版本有: 0.8/0.9/0.10/0.1
1/universal
  'connector.topic' = 'topic_name', -- 必选: 主题名称

```

```
  'connector.properties.zookeeper.connect' = 'localhost:2181', -- 必选: zk连接地址
  'connector.properties.bootstrap.servers' = 'localhost:9092', -- 必选: Kafka连接地址
  'connector.properties.group.id' = 'testGroup', --可选: 消费者组
   -- 可选:偏移量, earliest-offset/latest-offset/group-offsets/specific-offsets
  'connector.startup-mode' = 'earliest-offset',

  -- 可选: 当偏移量指定为specific offsets, 为指定每个分区指定具体位置
  'connector.specific-offsets' = 'partition:0,offset:42;partition:1,offset:300',
  'connector.sink-partitioner' = '...',  -- 可选: sink分区器, fixed/round-robin/custom
  -- 可选: 当自定义分区器时, 指定分区器的类名
  'connector.sink-partitioner-class' = 'org.mycompany.MyPartitioner',
  'format.type' = '...',                  -- 必选: 指定格式, 支持csv/json/avro
   -- 指定update-mode, 支持append/retract/upsert
  'update-mode' = 'append',

)
```

尖叫提示：

- 指定具体的偏移量位置：默认是从当前消费者组提交的偏移量开始消费
- sink分区：默认是尽可能向更多的分区写数据(每一个sink并行度实例只向一个分区写数据)，也可以自己分区策略。当使用 round-robin分区器时，可以避免分区不均衡，但是会造成Flink实例与kafka broker之间大量的网络连接
- 一致性保证：默认sink语义是at-least-once
- **Kafka 0.10+** 是时间戳：从kafka0.10开始，kafka消息附带一个时间戳作为消息的元数据，表示记录被写入kafka主题的时间，这个时间戳可以作为事件时间属性(rowtime attribute)
- **Kafka 0.11+**版本：Flink从1.7开始，支持使用universal版本作为kafka的连接器 ，可以兼容kafka0.11之后版本

定义MySQL数据表

```
CREATE TABLE MySQLTable (
  ...
) WITH (
  'connector.type' = 'jdbc', -- 必选: jdbc方式
  'connector.url' = 'jdbc:mysql://localhost:3306/flink-test', -- 必选: JDBC url
  'connector.table' = 'jdbc_table_name',  -- 必选: 表名
   -- 可选: JDBC driver, 如果不配置, 会自动通过url提取
```

```
    'connector.driver' = 'com.mysql.jdbc.Driver',

    'connector.username' = 'name', -- 可选: 数据库用户名
    'connector.password' = 'password',-- 可选: 数据库密码
      -- 可选, 将输入进行分区的字段名.
    'connector.read.partition.column' = 'column_name',
      -- 可选, 分区数量.
    'connector.read.partition.num' = '50',
      -- 可选, 第一个分区的最小值.
    'connector.read.partition.lower-bound' = '500',
      -- 可选, 最后一个分区的最大值
    'connector.read.partition.upper-bound' = '1000',
      -- 可选, 一次提取数据的行数, 默认为0, 表示忽略此配置
    'connector.read.fetch-size' = '100',
     -- 可选, lookup缓存数据的最大行数, 如果超过改配置, 老的数据会被清除
    'connector.lookup.cache.max-rows' = '5000',
     -- 可选, lookup缓存存活的最大时间, 超过该时间旧数据会过时, 注意cache.max-rows与
cache.ttl必须同时配置
    'connector.lookup.cache.ttl' = '10s',
     -- 可选, 查询数据最大重试次数
    'connector.lookup.max-retries' = '3',
     -- 可选,写数据最大的flush行数, 默认5000, 超过改配置, 会触发刷数据
    'connector.write.flush.max-rows' = '5000',
     --可选, flush数据的间隔时间, 超过该时间, 会通过一个异步线程flush数据, 默认是0s
    'connector.write.flush.interval' = '2s',
    -- 可选, 写数据失败最大重试次数
    'connector.write.max-retries' = '3'
)
```

Temporal Table Join

使用语法

```
SELECT column-names
FROM table1  [AS <alias1>]
[LEFT] JOIN table2 FOR SYSTEM_TIME AS OF table1.proctime [AS <alias2>]
ON table1.column-name1 = table2.key-name1
```

注意：目前，仅支持INNER JOIN与LEFT JOIN。在join的时候需要使用 `FOR SYSTEM_TIME AS OF` ，其中table1.proctime表示table1的proctime处理时间属性(计算列)。使用 `FOR SYSTEM_TIME AS OF table1.proctime` 表示当左边表的记录与右边的维表join时，只匹配当前处理时间维表所对应的的快照数据。

样例

```
SELECT
  o.amout, o.currency, r.rate, o.amount * r.rate
FROM
  Orders AS o
  JOIN LatestRates FOR SYSTEM_TIME AS OF o.proctime AS r
  ON r.currency = o.currency
```

使用说明

- 仅支持Blink planner
- 仅支持SQL，目前不支持Table API
- 目前不支持基于事件时间(event time)的temporal table join
- 维表可能会不断变化，JOIN行为发生后，维表中的数据发生了变化（新增、更新或删除），则已关联的维表数据不会被同步变化
- 维表和维表不能进行JOIN
- 维表必须指定主键。维表JOIN时，ON的条件必须包含所有主键的等值条件

维表Join案例

背景

Kafka中有一份用户行为数据，包括pv，buy，cart，fav行为；MySQL中有一份省份区域的维表数据。现将两种表进行JOIN，统计每个区域的购买行为数量。

步骤

维表存储在MySQL中，如下创建维表数据源：

```
CREATE TABLE dim_province (
    province_id BIGINT,  -- 省份id
    province_name  VARCHAR, -- 省份名称
        region_name VARCHAR -- 区域名称
) WITH (
    'connector.type' = 'jdbc',
    'connector.url' = 'jdbc:mysql://192.168.10.203:3306/mydw',
    'connector.table' = 'dim_province',
    'connector.driver' = 'com.mysql.jdbc.Driver',
    'connector.username' = 'root',
    'connector.password' = '123qwe',
    'connector.lookup.cache.max-rows' = '5000',
    'connector.lookup.cache.ttl' = '10min'
```

```
);
```

事实表存储在kafka中，数据为用户点击行为，格式为JSON，具体数据样例如下：

```
{"user_id":63401,"item_id":6244,"cat_id":143,"action":"pv","province":3,
"ts":1573445919}
{"user_id":9164,"item_id":2817,"cat_id":611,"action":"fav","province":28
,"ts":1573420486}
```

创建kafka数据源表，如下：

```
CREATE TABLE user_behavior (
    user_id BIGINT, -- 用户id
    item_id BIGINT, -- 商品id
    cat_id BIGINT,  -- 品类id
    action STRING,  -- 用户行为
        province INT,   -- 用户所在的省份
        ts     BIGINT,  -- 用户行为发生的时间戳
    proctime as PROCTIME(),   -- 通过计算列产生一个处理时间列
        eventTime AS TO_TIMESTAMP(FROM_UNIXTIME(ts, 'yyyy-MM-dd HH:mm:ss
')), -- 事件时间
    WATERMARK FOR eventTime as eventTime - INTERVAL '5' SECOND  -- 在even
tTime上定义watermark
) WITH (
    'connector.type' = 'kafka',  -- 使用 kafka connector
    'connector.version' = 'universal',  -- kafka 版本，universal 支持 0.11
 以上的版本
    'connector.topic' = 'user_behavior',  -- kafka主题
    'connector.startup-mode' = 'earliest-offset',  -- 偏移量，从起始 offset
 开始读取
        'connector.properties.group.id' = 'group1', -- 消费者组
    'connector.properties.zookeeper.connect' = 'kms-2:2181,kms-3:2181,km
s-4:2181',  -- zookeeper 地址
    'connector.properties.bootstrap.servers' = 'kms-2:9092,kms-3:9092,km
s-4:9092',  -- kafka broker 地址
    'format.type' = 'json'  -- 数据源格式为 json
);
```

创建MySQL的结果表，表示区域销量

```
CREATE TABLE region_sales_sink (
    region_name STRING,  -- 区域名称
```

```
    buy_cnt BIGINT  -- 销量
) WITH (

    'connector.type' = 'jdbc',
    'connector.url' = 'jdbc:mysql://192.168.10.203:3306/mydw',
    'connector.table' = 'top_region', -- MySQL中的待插入数据的表
    'connector.driver' = 'com.mysql.jdbc.Driver',
    'connector.username' = 'root',
    'connector.password' = '123qwe',
    'connector.write.flush.interval' = '1s'
);

```

用户行为数据与省份维表数据join

```
CREATE VIEW user_behavior_detail AS
SELECT
  u.user_id,
  u.item_id,
  u.cat_id,
  u.action,
  p.province_name,
  p.region_name
FROM user_behavior AS u LEFT JOIN dim_province FOR SYSTEM_TIME AS OF u.p
roctime AS p
ON u.province = p.province_id;
```

计算区域的销量，并将计算结果写入MySQL

```
INSERT INTO region_sales_sink
SELECT
  region_name,
  COUNT(*) buy_cnt
FROM user_behavior_detail
WHERE action = 'buy'
GROUP BY region_name;

```

结果查看：

```
Flink SQL> select * from  region_sales_sink; -- 在Flink SQL cli中查看

```

[外链图片转存失败,源站可能有防盗链机制,建议将图片保存下来直接上传(img-hQPzLkz3-1596813141668)(实时数仓-Flink-SQL之维表join/结果.png)]

```
mysql> select * from top_region; -- 查看MySQL的数据

```

[外链图片转存失败,源站可能有防盗链机制,建议将图片保存下来直接上传(img-Wvg28hmd-1596813141674)(实时数仓-Flink-SQL之维表join/结果1.png)]

总结

本文主要介绍了FlinkSQL的维表join，使用的方式为Temporal Table Join。首先介绍了Flink SQL创建表的基本语法，并对其中的细节进行了描述。接着介绍了如何创建Kafka以及MySQL的数据源表。然后介绍了Temporal Table Join的基本概念与使用语法。最后，给出了一个完整的维表JOIN案例。