Flink原理（七）——动态表（Dynamic tables）

目录

回到顶部

前言

本文是结合Flink官网，个人理解所得，**若是有误欢迎留言指出**，谢谢！文中图皆来自官网（链接[1]）。

本文将随着下面这个问题展开，针对该问题更为生动的解释可以参见金竹老师的分享（链接[2]）。

SQL适合流计算场景吗?

对于流计算，每一条数据的到来都会触发一次查询产生一个结果，并发射出去。我们发现对于相同的数据源，使用相同的SQL查询时，批、流的结果是相同的，即在不同模式下，SQL的语意是一致的（One Query One Result），最终的结果是一致。

回到顶部

1、动态表与连续查询（Dynamic Table&Continuous Query）

和动态表对应的是静态表——常规的数据库中的表或批处理中的表等，其在查询时数据不再变化。动态表是随时间变化的，即使是在查询的时候。怎么理解了？流上的数据是源源不断的，一条数据的到来会触发一次查询，这次查询在执行时还有下一条数据到来，对表本身数据是在变化的。

对动态表的查询是连续的，即连续查询（Continuous Query）。实质上， 动态表上的连续查询与定义物理视图（Materialized View）的查询很相似。物理视图定义为SQL查询，就像常规的虚拟视图一样，不同的是物理视图会缓存查询结果，这样在访问时不需要重新计算，而缓存带来的挑战是有可能提供过时的结果，Eager View Maintenance则是用于及时跟新物理视图的技术，这里就不展开了。

流、动态表、连续查询三者的关系如下图所示：

用一句话概括是：流被转换为动态表，对动态表的连续查询生成新的动态表（结果表），然后结果表被转换为流。

回到顶部

2、流上定义表

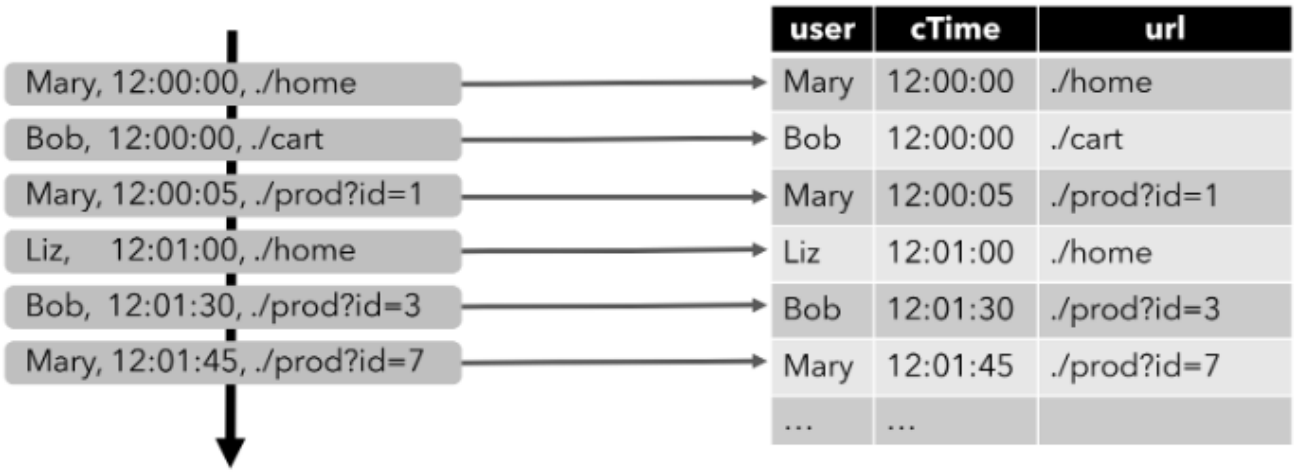
2.1 定义表

为了在流上使用关系型查询，需要将流转换成表。下面的分析过程均采用官网（Ref[2]）中的例子进行说明。

1）点击事件流的schema如下：

```
[
  user:  VARCHAR,   // the name of the user
  cTime: TIMESTAMP, // the time when the URL was accessed
  url:   VARCHAR    // the URL that was accessed by the user
]
```

2）从概念上来说，流上的每天记录都是动态表进行INSERT修改。从本质上讲，是从一个INSERT-Only（仅插入）的ChangeLog流上构建一个表。点击事件流上构建表如下图所示，且随着更多点击流记录的插入，生成的表不断增长：



Note：定义在流上的表在内部是没有实现的。

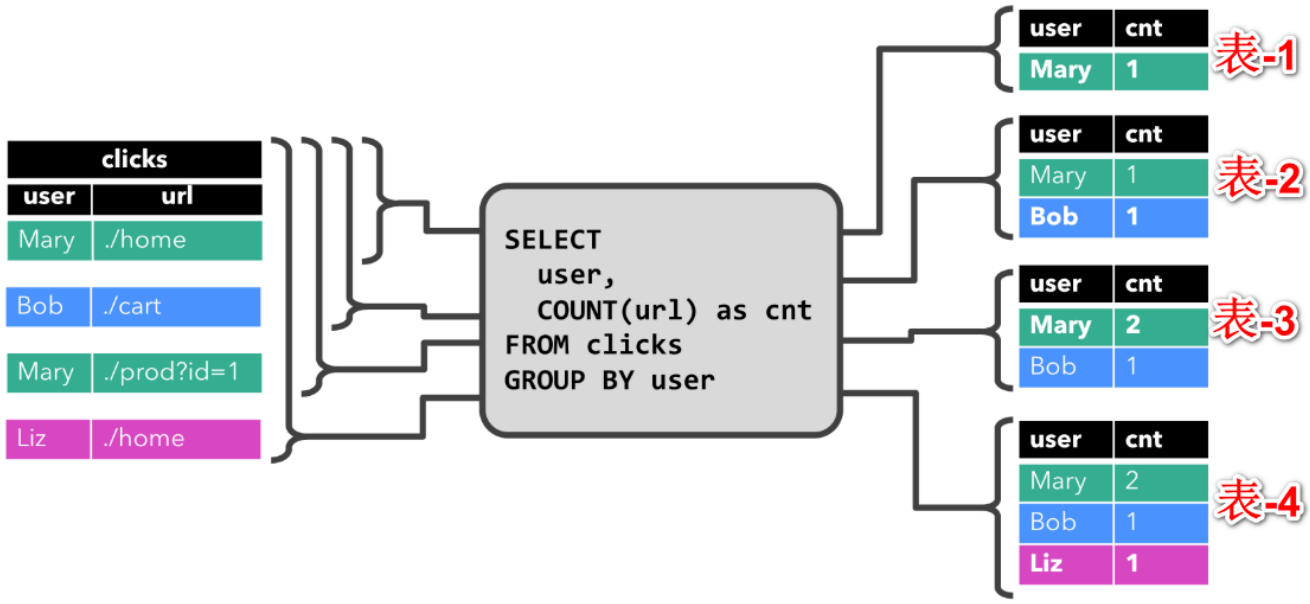
2.2 连续查询

连续查询不会中止，会根据输入表来更新结果表，下面介绍两种查询的例子。

1）简单的GROUP-BY Count聚合查询

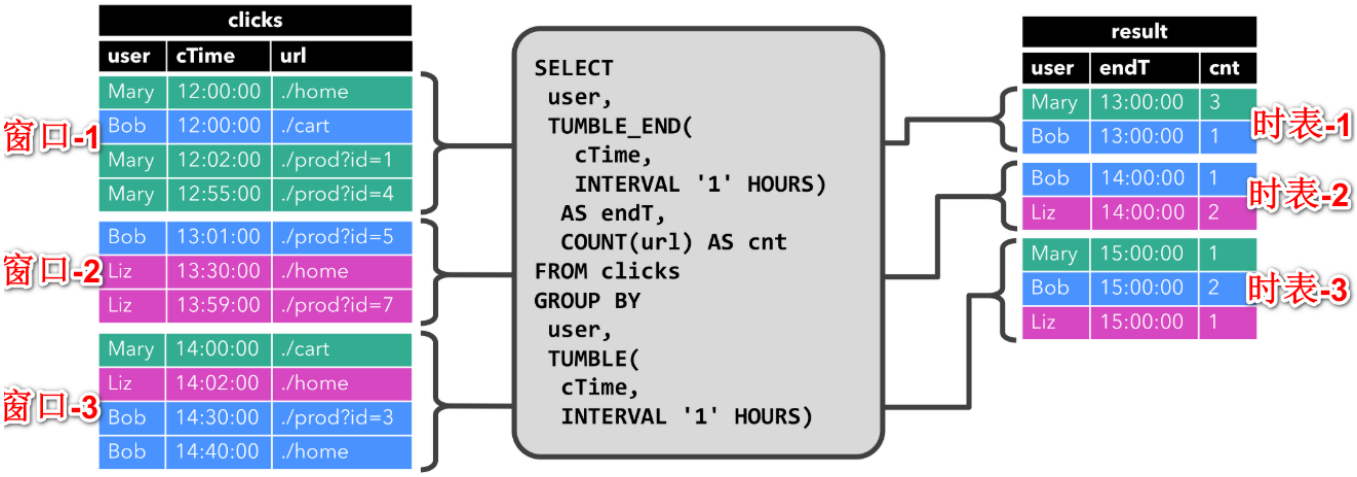
下图中，左边是输入表click，是随着时间updata增加的，右边是查询的结果表。开始clicks表中只有一条数据[Mary, ./home]时其结果表是表-1，当clicks表中新增一条数据[Bob, ./cart]时，其结果表是表-2，依次下

推。每一条新数据的到来会对之前表行进update或INSERT操作，SQL语句就会根据现有数据更新的结果表。



2）带有窗口（window）的聚合查询

窗口的时间间隔是1个小时，窗口-1对应的时表-1，窗口-2对应的时表-2，依次类推。和第一种查询不一样的是，每一张时表只是统计对应窗口的数据，之前窗口的数据对其没有影响，对不同窗口的查询结果是以追加的形式写入result表中的。



2.3 Update和Append查询

2.2中的两种例子分别对应的两种查询方式，

1）例子1对应着Update查询，这种方式需要更新之前已经发出的结果，包括INSERT和UPDATE两种改变。改变之前已经发出的结果意味着，这种查询需要维护更多的状态（state）数据；

2）例子2对应着Append查询，这种方式查询的结果都是以追加的形式加入到result表中，仅包含INSERT操作。这种方式生成的表和update生成的表转换成流的方式不一样（见下文）。

2.4 Restrictions查询

对于有些SQL查询会因需要保留的state多大或重新计算已发出的记录用来更新的代价太大而得不偿失。

1）state size：例如下面的SQL，在连续查询中，当一条新的消息到来时，为了更新之前已发出的结果（联想2.2中例1），需要保存之前的计算结果即state。当时当连续查询持续很长时间时，需要保存的state的容量会

很大，且随着时间的递增会越来越大，更糟糕的是若不断有新用户（分配不同的username）加入，其要保存的count会随着时间更加恐怖，最后有可能导致任务失败。

```
SELECT user, COUNT(url) FROM clicks GROUP BY user;
```

2）computing updates：例如下面的SQL，当clicks表新增一条记录，为计算rank，需要对之前所有的重新计算和更新已发出结果的中很大一部分，一条记录的的增加，有可能导致很多user的rank变化。

```
SELECT user, RANK() OVER (ORDER BY lastLogin)
FROM (
  SELECT user, MAX(cTime) AS lastAction FROM clicks GROUP BY user
);
```

3）查询配置（链接[3]）

在常见的场景中，对长期的运行的job做连续查询，为了防止保存的state过大超出存储而任务失败，可能会对state的大小做一定限制即删除state。但这种方式可能引发另一个问题——查询出来的结果可能不准确。Flink Table API和SQL中提供查询参数试图在准确性和资源消耗中找到一个平衡点。

Idle State Retention Time含义是state的key在被删除之前多长时间没有被更新，即没有被更新state的保存时间。使用方式如下：

```
StreamQueryConfig qConfig = ...

// set idle state retention time: min = 12 hours, max = 24 hours
qConfig.withIdleStateRetentionTime(Time.hours(12), Time.hours(24));
```

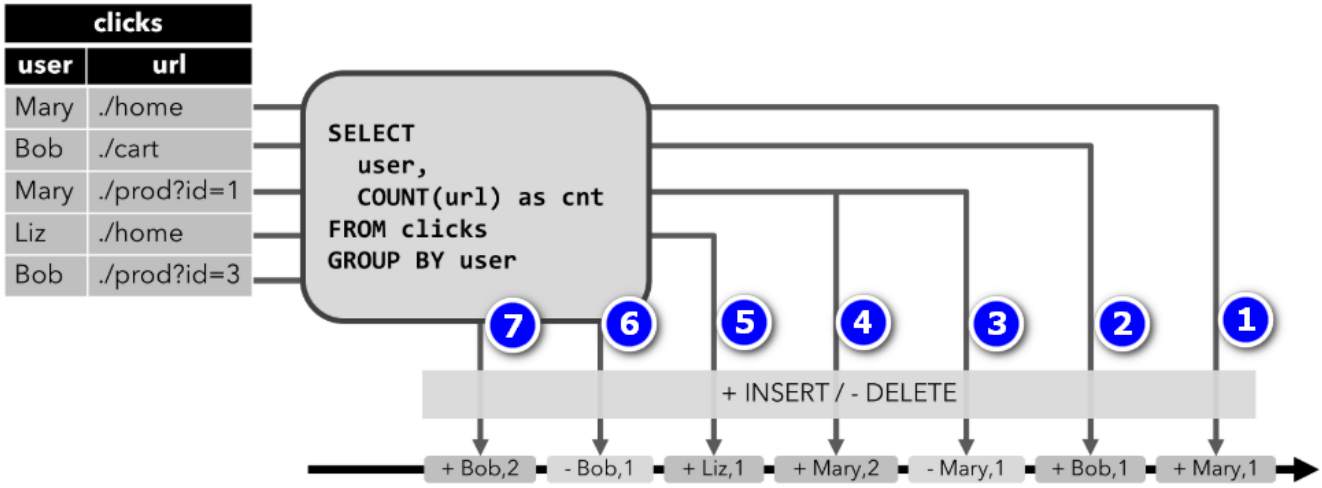
回到顶部

3、Table到流的转换

可以通过INSERT、UPDATE、DELETE像修改常规表一样去改变动态表。将动态表转换为流或将其写入外部系统时，需要对这些更改进行encode。 Flink的Table API和SQL支持三种encode改变动态表的方法：

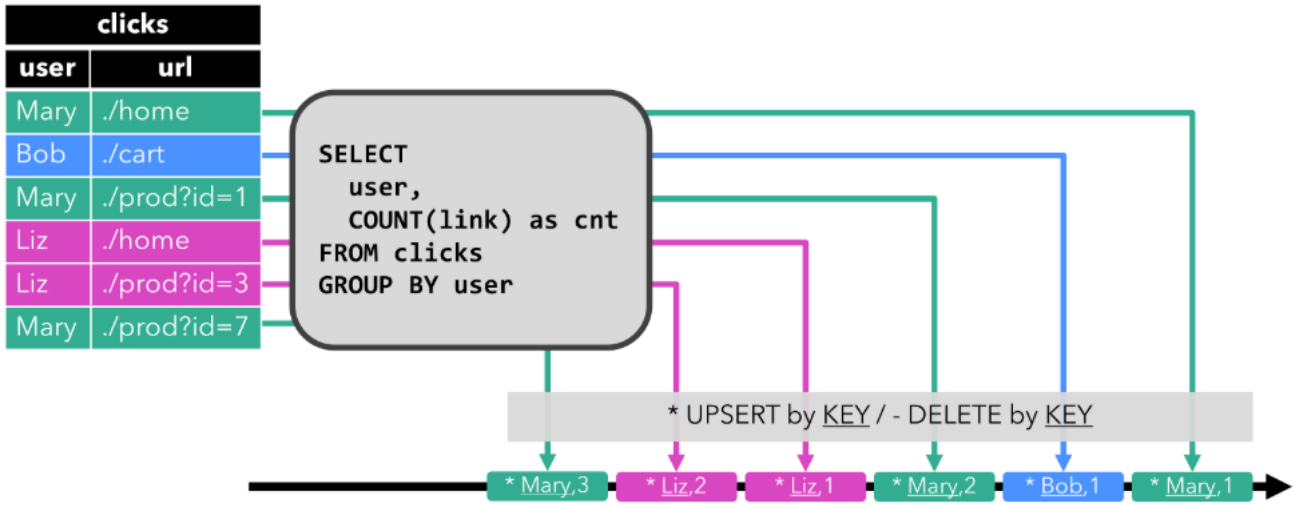
1）Append-only Stream（仅追加流）：仅通过INSERT操作得到的动态表可以发射插入行来转换为流（联想2.2中例2），这种方式转换的流中数据都是片段性的，一个片段代表一个窗口；

2）Retract Stream（回溯流）：restract stream有两种消息：添加（add）消息和回溯（retract）消息。将动态表转换为回溯（retract）流，通过将INSERT更改encode为添加消息，将DELETE更改encode为回溯消息，将UPDATE更改endcode为更新（上一个）行的回溯消息以及添加消息更新新的行 。 下图显示了动态表到回溯流的转换。



流上每条消息都有一个标识位，其中+标识INSERT操作，-标识DELETE操作。在clicks表中第一、二行消息[Mary, ./home]和[Bob, ./cat]被转换为流中1第、2条消息，当clicks表中第三行[Mary, ./prod?id=1]转换时，会先将已发出的第1条信息标记为DELETE告诉下游，然后第4条消息重新插入user为Mary的消息，依次类推，这样可以保证输出结果的正确性。

3）Upsert Stream（上插流）：Upsert流包括upsert消息和删除消息。 动态表要转换为upsert流需要（可能是复合的）唯一键。 通过将INSERT和UPDATE 操作encode为upsert消息，并将DELETE更改encode为删除消息，可以是具有唯一键的动态表转换为流。 流运算需要知道唯一键属性才能正确应用消息。 与回溯流的主要区别在于UPDATE使用单个消息（(主键)）进行编码，因此更有效。



（**个人理解待验证**）Upsert流和Retract流的区别在于数据存在第三方系统中时，前者可能存在重复数据，后者没有。

NOTE：在将动态表（dynamic table）转换为数据流（Data Stream）时，仅支持append和retract两种方式。

Ref:

[1]https://ci.apache.org/projects/flink/flink-docs-release-1.9/dev/table/streaming/dynamic_tables.html#table-to-stream-conversion

[2]http://www.itdks.com/Course/detail?id=13213&from=search

[3]https://ci.apache.org/projects/flink/flink-docs-release-1.9/dev/table/streaming/query_configuration.html