Flink Table API & SQL编程指南之动态表(2)

在Flink Table API & SQL编程指南(1)一文中介绍了Flink Table API &SQL的一些基本的概念和通用的API，在本文将会更加深入地讲解Flink Table API &SQL的流处理的基本概念。Flink Table API &SQL是实现了批流处理的统一，这也意味着无论是有界的批处理输入还是无界的流处理输入，使用Flink Table API &SQL进行查询操作，都具有相同的语义。此外，由于SQL最初是为批处理而设计的，所有在无界流上使用关系查询与在有界流上使用关系查询是有所不同的，本文将着重介绍一下动态表。

动态表

SQL与关系代数最初的设计不是为了流处理，所以SQL与流处理之间存在一定的差异，Flink实现了在无界的数据流上使用SQL操作。

数据流上的关系查询

传统的关系代数(SQL)与流处理在数据的输入、执行以及结果的输出都有所差异，具体如下：

区别	关系代数/SQL	
数据输入	表、有界的元祖的集合	无界
执行	批处理，在整个输入数据上执行查询等操作	不能在所有的数据上执行
结果输出	查询处理结束之后，输出固定大小的结果	需要连续不断地更新

尽管存在这些差异，但并不意味着SQL与流处理不能融合。一些高级的关系型数据库都提供了物化视图的功能，一个物化视图由一个查询语句进行定义，与普通的视图相比，物化视图缓存了查询的结果，所以当访问物化视图时，不需要重复执行SQL查询操作。缓存的一个常见挑战是如何防止提供过时的结果，当定义物化视图的查询基表发生变化时，物化视图的结果就会过时。**Eager View Maintenance **是一种更新物化视图的技术，只要物化视图的查询基表被更新，那么物化视图就会被更新。

如果考虑以下因素，那么**Eager View Maintenance **与对流进行SQL查询之间的联系将变得显而易见：

- 数据库表是在流上执行`INSERT`，`UPDATE`和`DELETE`DML操作语句的结果，通常被称为changelog stream(更新日志流)。

- 一个物化视图由一个查询语句进行定义。为了更新雾化视图，查询会连续处理雾化视图的变更日志流。
- 雾化视图是流式SQL查询的结果。

动态表与持续的查询

动态表是Flink TableAPI &SQL支持流处理的核心概念，与批处理的静态表相比，动态表会随着时间的变化而变化。动态表查询会产生一个*Continuous Query*，Continuous Query不会终止并且会产生动态表 的结果。

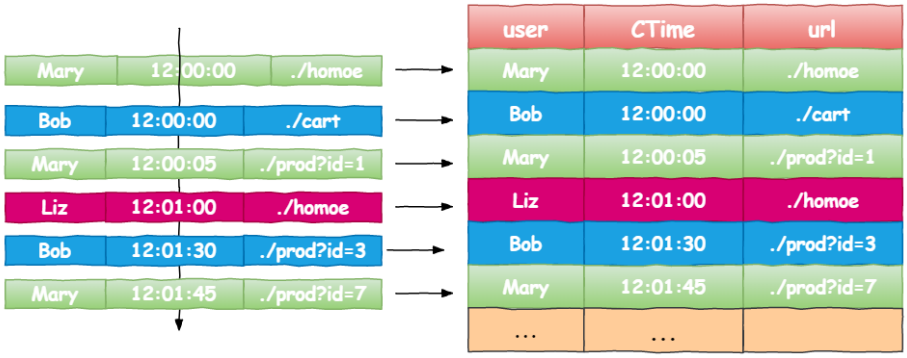
关于流、动态表与Continuous Query之间的关系，如下图所示：

- 1.流被转换为动态表
- 2.Continuous Query在动态表上不停的执行，产生一个新的动态表
- 3.动态表被转换成流

尖叫提示：动态表是一个逻辑概念，在执行查询期间动态表不会被雾化。

在流上定义表

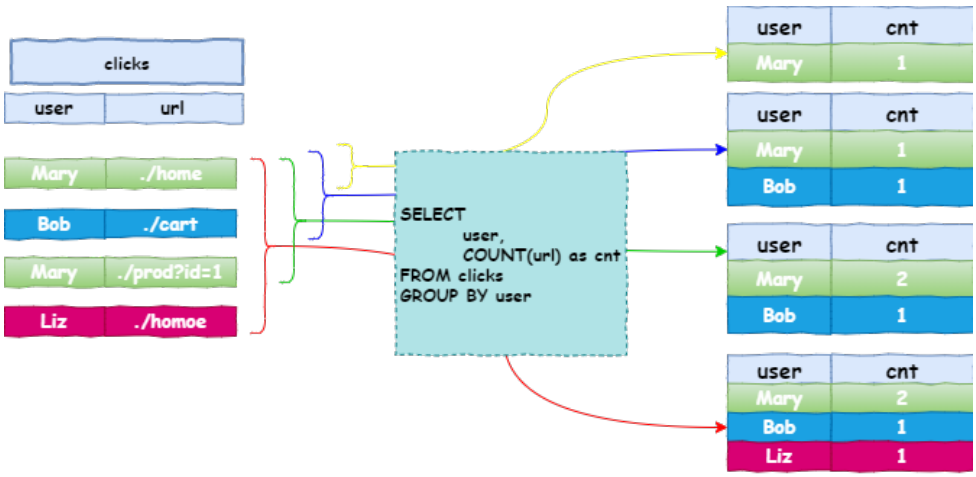
在数据流上使用SQL查询，需要将流转换成表。流中的记录会被解析并插入到表中(对于一个只有插入操作的流)，如下图所示：



持续查询

- 分组聚合(group aggregation)

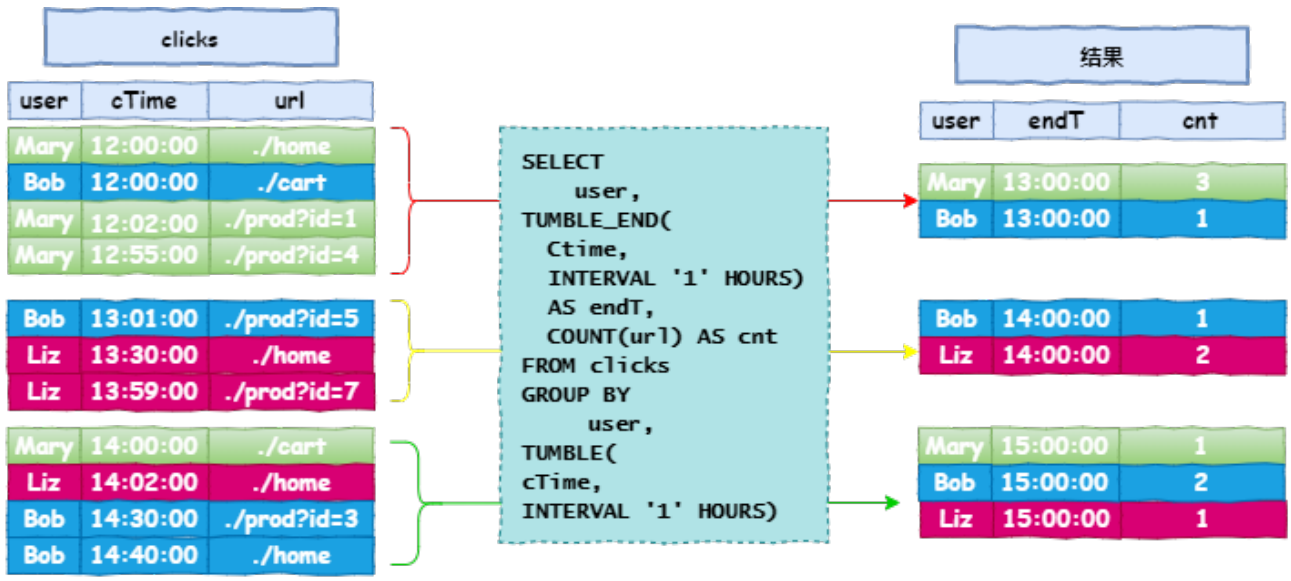
在一个动态表上的持续查询，会产生一个新的动态表结果。与批处理的查询相比，持续的查询从不会结束并且会根据输入的数据更新之前的结果。下面的示例中，展示了点击事件流，并使用分组聚合计算，如下图：



上图中展示了一个用户点击行为的数据，计算操作使用的是分组聚合(group aggregation)。当第一条数据[Mary,./home]进来时，会立即进行计算操作，并输出计算结果：[Mary，1]。当[Bob,./cart]进来时，也会立即进行计算，并输出计算结果：[Mary，1]，[Bob，1]。当[Mary,./prod?id=1]进来时，会立即进行计算，并输出结果：[Mary，2]，[Bob，1]。由此可以看出：分组聚合会作用在所有的数据上，并且会更新之前输出的结果。

- 窗口聚合(window aggregation)

上面演示的是一个分组聚合的案例，下面再来看一个窗口聚合的案例。按照一个小时的滚动窗口(Tumble Window)计算该一小时内的用户点击情况，具体如下图所示：



如上图所示：Ctime表示事件发生的时间，可以看出在[12:00:00,12:59:59]的一小时内总共有四行数据，在[13:00:00,13:59:59]的一小时内有三行数据，在[14:00:00,14:59:59]一小时内总共有四行数据。

可以看出：在[12:00:00,12:59:59]时间段内，计算的结果为[Marry,13:00:00,3],[Bob,13:00:00,1],该结果会追加(Append)到该结果表中，在[13:00:00,13:59:59]时间段内，计算结果为[Bob,14:00:00,1],[Liz,14:00:00,2]，该结果同样是追加到结果表中，之前窗口的数据并不会更新。所以窗口聚合的特点就是只计算属于该窗口的数据，并以追加的方式将结果插入结果表中。

- 分组聚合与窗口聚合的异同

比较	分组聚合	
输出模式	提前输出，每来一条数据计算一次	按窗
输出量	一个窗口输出一次结果	每个
输出流	追加流(Append Stream)	更新流
状态清理	及时清理掉过时的数据	状
	不要求输出端支持update操作	支持更新操作

更新与追加查询

上面的两个例子分别演示了更新的查询与追加查询，第一个分组聚合的案例输出的结果会更新之前的结果，即结果表包含**INSERT**与**UPDATE**操作。

第二个窗口聚合的案例仅仅是追加计算结果的结果表中，即结果表仅包含**INSERT**操作。

当一个查询产生一个仅追加(append-only)的表或者更新表(updated table)时，区别如下：

- 当查询产生的是一个更新表时(即会更新之前输出的结果)，需要维护一个更大的状态
- append-only表与更新表(updated table)转为流(Stream)的方式有所不同

表到流的转换

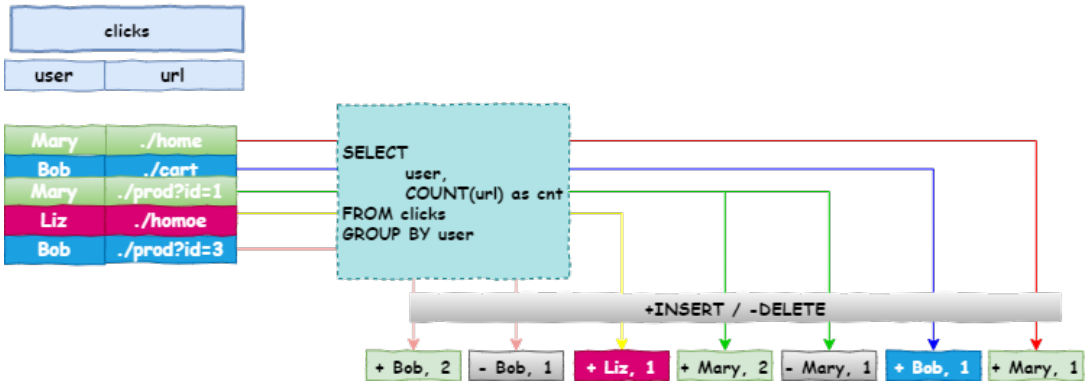
动态表会被INSER、UPDATE、DELETE操作持续地被更改。当将一个动态表转为流或者写出到外部的存储系统时，这些改变的值需要被编码，Flink Table API和SQL支持三种方式对这些改变的数据进行编码：

- **Append-only stream**

动态表只会被INSERT操作进行修改，改变的数据(新增的数据)会被插入到动态表的行中

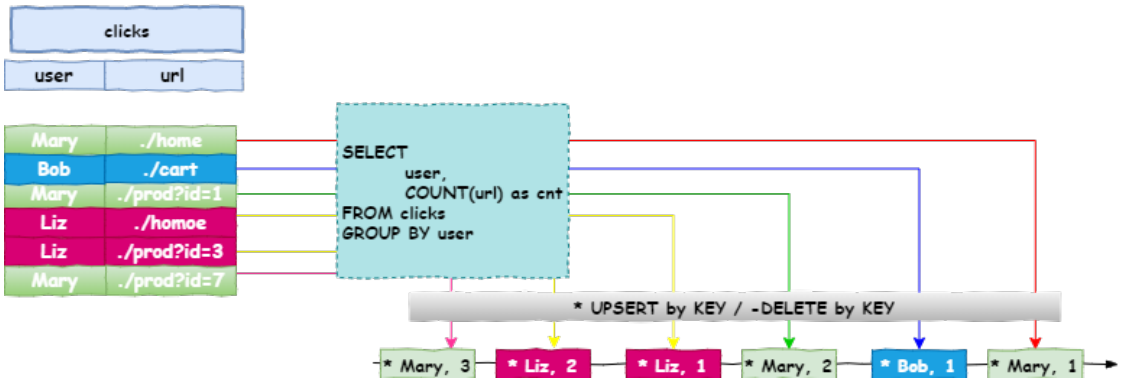
- **Retract stream**

一个retract stream包含两种类型的消息，分别为添加消息(add messages)与撤回消息(retract message)。动态表被转为retract stream(撤回流)时，将INSERT操作的变化数据编码为添加消息(add messages)，将DELETE操作引起的变化数据编码为撤回消息(retract message)，将UPDATE操作引起的变化数据而言，会先将旧数据(需要被更新的)编码为retract message，将新的更新的数据编码为添加消息(add messages)，具体如下图所示：



- **Upsert stream**

upsert 流有两种类型的消息：*upsert messages*与*delete messages*。动态表被转换为upsert流需要一个唯一主键(可能是复合)key，附带唯一主键key的动态表在被转化为流的时候，会把INSERT与UPDATE操作引起的变化数据编码为upsert messages，把DELETE操作引起的变化数据编码为delete message。与retract 流相比，upsert 流对于UPADTE操作引起的变化数据的编码，使用的是单个消息，即upsert message。对于retract 流，需要先将旧数据编码为retract message，然后再将新数据编码为add message，即需要编码Delete与Insert两条消息，因此使用upsert流效率更高。具体如下图所示：



尖叫提示：将动态表转为datastream时，仅支持append 流与retract流。将动态表输出到外部系统时，支持Append、Retract以及Upsert模式。

小结

本文主要介绍了Flink TableAPI&SQL中动态表的概念。首先介绍了动态表的基本概念，然后介绍了在流上定义表的方式，并指出了分组聚合与窗口聚合的异同，最后介绍了表到流的转换并输出到外部系统的三种模式。