

39 | 从SQL到Streaming SQL：突破静态数据查询的次元

2019-07-24 蔡元楠

大规模数据处理实战

[进入课程 >](#)



讲述：蔡元楠

时长 09:47 大小 8.97M



你好，我是蔡元楠。

今天我要与你分享的主题是“从 SQL 到 Streaming SQL：突破静态数据查询的次元”。

在前面的章节中，我们介绍了一些流数据处理相关的知识和技术，比如 Apache Spark 的流处理模块——Spark Streaming 和 Structured Streaming，以及 Apache Beam 中的窗口处理。相信你对流处理的重要性和一些基本手段都有所了解了。

流处理之所以重要，是因为现在是个数据爆炸的时代，大部分数据源是每时每刻都在更新的，数据处理系统对时效性的要求都很高。作为当代和未来的数据处理架构师，我们势必要深刻掌握流数据处理的技能。

“批” “流” 两手抓，两手都要硬。

你还记得，我在[第 15 讲](#)中介绍过的 Spark SQL 吗？它最大的优点就是 DataFrame/DataSet 是高级 API，提供类似于 SQL 的 query 接口，方便熟悉关系型数据库的开发人员使用。

当说到批处理的时候，我们第一个想到的工具就是 SQL，因为基本上每个数据从业者都懂，而且它的语法简单易懂，方便使用。那么，你也能很自然地联想到，如果在流处理的世界中也可以用 SQL，或者相似的语言，那真是太棒了。

这样的思想在[第 17 讲](#)中我们曾经提到过。

Spark 的 Structured Streaming 就是用支持类 SQL 的 DataFrame API 去做流处理的。支持用类似于 SQL 处理流数据的平台还有很多，比如 Flink、Storm 等，但它们是把 SQL 做成 API 和其他的处理逻辑结合在一起，并没有把它单独分离成一种语言，为它定义语法。

那么，到底有没有类似 SQL 语法来对流数据进行查询的语言呢？答案是肯定的。我们把这种语言统称为 Streaming SQL。Siddhi Streaming SQL 和 Kafka KSQL 就是两个典型的 Streaming SQL 语言，下文的例子我们主要用这两种语言来描述。


不同于 SQL，Streaming SQL 并没有统一的语法规则，不同平台提供的 Streaming SQL 语法都有所不同。而且 Streaming SQL 作用的数据对象也不是有界的数据表，而是无边界的数据流，你可以把它设想为一个底部一直在增加新数据的表。

SQL 是一个强大的、对有结构数据进行查询的语言，它提供几个独立的操作，如数据映射（SELECT）、数据过滤（WHERE）、数据聚合（GROUP BY）和数据联结（JOIN）。将这些基本操作组合起来，可以实现很多复杂的查询。

在 Streaming SQL 中，数据映射和数据过滤显然都是必备而且很容易理解的。数据映射就是从流中取出数据的一部分属性，并作为一个新的流输出，它定义了输出流的格式。数据过滤就是根据数据的某些属性，挑选出符合条件的。

让我们来看一个简单的例子吧。假设，有一个锅炉温度的数据流 BoilerStream，它包含的每个数据都有一个 ID 和一个摄氏温度（t），我们要拿出所有高于 350 摄氏度的数据，并

且把温度转换为华氏度。

 复制代码

```
1 Select id, t*7/5 + 32 as tF from BoilerStream[t > 350]; //Siddhi Streaming SQL
2
3 Select id, t*7/5 + 32 as tF from BoilerStream Where t > 350; //Kafka KSQL
```

你可以看出，这两种语言与 SQL 都极为类似，相信你都可以直接看懂它的意思。

Streaming SQL 允许我们用类似于 SQL 的命令形式去处理无边界的流数据，它有如下几个优点：

简单易学，使用方便：SQL 可以说是流传最广泛的数据处理语言，对大部分人来说，Streaming SQL 的学习成本很低。

效率高，速度快：SQL 问世这么久，它的执行引擎已经被优化过很多次，很多 SQL 的优化准则被直接借鉴到 Streaming SQL 的执行引擎上。

代码简洁，而且涵盖了大部分常用的数据操作。

除了上面提到过的数据映射和数据过滤，Streaming SQL 的 GROUP BY 也和 SQL 中的用法类似。接下来，让我们一起了解 Streaming SQL 的其他重要操作：窗口（Window）、联结（Join）和模式（Pattern）。


窗口

在之前 Spark 和 Beam 的流处理章节中，我们都学习过窗口的概念。所谓窗口，就是把流中的数据按照时间戳划分成一个个有限的集合。在此之上，我们可以统计各种聚合属性如平均值等。

在现实世界中，大多数场景下我们只需要关心特定窗口，而不需要研究全局窗口内的所有数据，这是由数据的时效性决定的。

应用最广的窗口类型是以当前时间为结束的滑动窗口，比如“最近 5 小时内的车流量”，或“最近 50 个成交的商品”。

所有的 Streaming SQL 语法都支持声明窗口，让我们看下面的例子：

 复制代码

```
1 Select bid, avg(t) as T From BoilerStream#window.length(10) insert into BoilerStreamMov:
2
3 Select bid, avg(t) as T From BoilerStream WINDOW HOPPING (SIZE 10, ADVANCE BY 1); // Ka-
```

这个例子中，我们每接收到一个数据，就生成最近 10 个温度的平均值，插入到一个新的流中。

在 Beam Window 中，我们介绍过固定窗口和滑动窗口的区别，而每种窗口都可以是基于时间或数量的，所以就有 4 种组合：

滑动时间窗口：统计最近时间段 T 内的所有数据，每当经过某个时间段都会被触发一次。

固定时间窗口：统计最近时间段 T 内的所有数据，每当经过 T 都会被触发一次

滑动长度窗口：统计最近 N 个数据，每当接收到一个（或多个）数据都会被触发一次。

固定长度窗口：统计最近 N 个数据，每当接收到 N 个数据都会被触发一次。

再度细化，基于时间的窗口都可以选择不同的时间特性，例如处理时间和事件时间等。此外，还有会话（Session）窗口等针对其他场景的窗口。

联结

当我们要把两个不同流中的数据通过某个属性连接起来时，就要用到 Join。

由于在任一个时刻，流数据都不是完整的，第一个流中后面还没到的数据有可能要和第二个流中已经有的数据 Join 起来再输出。所以，对流的 Join 一般要对至少一个流附加窗口，这也和[第 20 讲](#)中提到的数据水印类似。

让我们来看一个例子，流 TempStream 里的数据代表传感器测量的每个房间的温度，每分钟更新一次；流 RegulatorStream 里的数据代表每个房间空调的开关状态。我们想要得到所有温度高于 30 度但是空调没打开的房间，从而把它们的空调打开降温：

```
1 from TempStream[temp > 30.0]#window.time(1 min) as T
2   join RegulatorStream[isOn == false]#window.length(1) as R
3   on T.roomNo == R.roomNo
4 select T.roomNo, R.deviceID, 'start' as action
5 insert into RegulatorActionStream; // Siddhi Streaming SQL
```

在上面的代码中，我们首先对 TempStream 流施加了一个长度为 1 分钟的时间窗口，这是因为温度每分钟会更新一次，上一分钟的数据已然失效；然后把它和流 RegulatorStream 中的数据根据房间 ID 连接起来，并且只选出了大于 30 度的房间和关闭的空调，插入新的流 RegulatorActionStream 中，告诉你这些空调需要被打开。

模式

通过上面的介绍我们可以看出，Streaming SQL 的数据模型继承自 SQL 的关系数据模型，唯一的不同就是每个数据都有一个时间戳，并且每个数据都是假设在这个特定的时间戳才被接收。

那么我们很自然地就会想研究这些数据的顺序，比如，事件 A 是不是发生在事件 B 之后？

这类先后顺序的问题在日常生活中很普遍。比如，炒股时，我们会计算某只股票有没有在过去 20 分钟内涨 / 跌超过 20%；规划路线时，我们会看过去 1 小时内某段路的车流量有没有在下降。

这里你不难看出，我们其实是在检测某个**模式**有没有在特定的时间段内发生。

股票价格涨 20% 是一个模式，车流量下降也是一个模式。在流数据处理中，检测模式是一类重要的问题，我们会经常需要通过对最近数据的研究去总结发展的趋势，从而“预测”未来。

在 Siddhi Streaming SQL 中，“->”这个操作符用于声明发生的先后关系。一起来看下面这个简单例子：

```
1 from every( e1=TempStream ) -> e2=TempStream[ e1.roomNo == roomNo and (e1.temp + 5) <= 1
2   within 10 min
```

```
3 select e1.roomNo, e1.temp as initialTemp, e2.temp as finalTemp
4 insert into AlertStream;
```

这个 query 检测的模式是 10 分钟内房间温度上升超过 5 度。对于每一个接收到的温度信号，把它和之前 10 分钟内收到的温度信号进行匹配。如果房间号码相同，并且温度上升超过 5 度，就插入到输出流。

很多流处理平台同样支持模式匹配，比如 Apache Flink 就有专门的 Pattern API，我建议你了解一下。

小结

今天我们初步了解了 Streaming SQL 语言的基本功能。

虽然没有统一的语法规则，但是各个 Streaming SQL 语言都支持相似的操作符，如数据映射、数据过滤、联结、窗口和模式等，大部分操作符都是继承自 SQL，只有模式是独有的，这是由于流数据天生具有的时间性所导致。

Streaming SQL 大大降低了开发人员实现流处理的难度，让流处理变得就像写 SQL 查询语句一样简单。它现在还在高速发展，相信未来会变得越来越普遍。

思考题

你觉得 Streaming SQL 的发展前景如何？欢迎留言与我一起讨论。

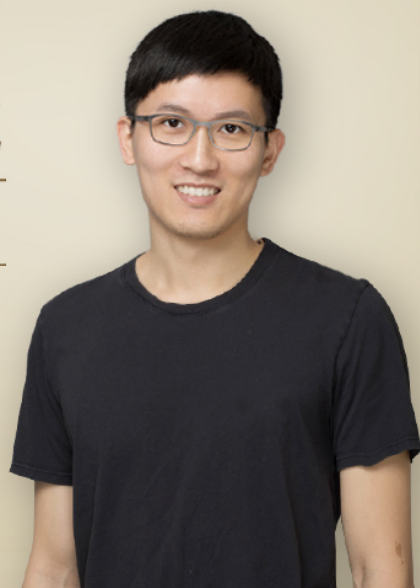
欢迎你把自己的学习体会写在留言区，与我和其他同学一起讨论。如果你觉得有所收获，也欢迎把文章分享给你的朋友。

大规模数据处理实战

Google 一线工程师的大数据架构实战经验

蔡元楠

Google Brain 资深工程师



新版升级：点击「👤 请朋友读」，20位好友免费读，邀请订阅更有**现金**奖励。

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 38 | 大规模数据处理在深度学习中如何应用？

下一篇 40 | 大规模数据处理未来之路

精选留言 (1)

写留言



Hobbin

2019-07-24

当前复杂的逻辑，Streaming SQL的支持还是比较有限的。请教一下，Streaming SQL有没有可能完全替代API开发方式呢？

展开



2