# Flink SQL 功能——流式 TopN 挑战与实现

TopN 是统计报表和大屏非常常见的功能，主要用来实时计算排行榜。流式的 TopN 不同于批处理的 TopN，它的特点是持续的在内存中按照某个统计指标（如出现次数）计算 TopN 排行榜，然后当排行榜发生变化时，发出更新后的排行榜。本文主要讲解 Flink SQL 是如何从语法和实现上设计 TopN 的。

## TopN 语法

### 全局 TopN

用户最关心的是如何用 SQL 写出 TopN 的查询。大家最熟悉的 TopN 的写法一般是这样的：

```
SELECT column_name(s) FROM table_name WHERE condition
ORDER BY order_field [DESC|ASC] LIMIT number
```

如上语法是 MySQL 的 TopN 语法，使用 `ORDER BY` 指定排序键和排序方向，使用 `LIMIT` 来指定选出前几名。不同的数据库的 TopN 语法不尽相同，比如 MS SQL Server 使用 TOP 的关键字，Oracle 使用 ROWNUM 的隐藏字段。不过几家数据库提供的 TopN 语法都是全局 TopN，也就是**数据是全局进行排序**的，查询的结果只有一组排行榜。比如希望对全网商家按销售额排序，计算出销售额排名前十的商家。这就是全局 TopN，范例如下：

```
SELECT * FROM shop_sales ORDER BY sales DESC LIMIT 10
```

### 分组 TopN

上文讲述了全局 TopN 的语法，但是很多时候用户希望根据不同的分组进行排序，计算出每个分组的一个排行榜。例如对全网商家根据行业按销售额排序，计算出每个行业销售额前十名的商家。这时候，传统的 TopN 语法就无法表达这种需求了。有些 Stream SQL 系统为了解决这个问题，会 hack 一种新的 TopN 语法允许用户指定分组字段。但是 **Flink SQL 是基于 ANSI SQL 标准语法的，不能加入任何非标准的语法**。于是我们尝试从批处理的角度去思考这个问题，在**传统批处理中常用 ROW_NUMBER 的开窗聚合函数来解决分组 TopN 的问题**。语法如下所示：

```
SELECT *
```

```
FROM (
    SELECT *,
      ROW_NUMBER() OVER ([PARTITION BY col1[, col2..]]
        ORDER BY col1 [asc|desc][, col2 [asc|desc]...]) AS rownum
    FROM table_name)
WHERE rownum <= N [AND conditions]
```

参数说明：

ROW_NUMBER(): 是一个计算行号的OVER窗口函数，行号计算从1开始。

PARTITION BY col1[, col2..]： 指定分区的列，可以不指定。

ORDER BY col1 [asc|desc][, col2 [asc|desc]...]: 指定排序的列，可以多列不同排序方向。

如上语法所示，**TopN 需要两层 query，子查询中使用ROW_NUMBER()开窗函数来为每条数据标上排名，排名的计算根据PARTITION BY和ORDER BY来指定分区列和排序列，也就是说每一条数据会计算其在所属分区中，根据排序列排序得到的排名。**在外层查询中，对排名进行过滤，只取出排名小于 N 的，如 N=10，那么就是取 Top 10 的数据。如果没有指定PARTITION BY那么就是一个全局 TopN 的计算，所以 ROW_NUMBER 在使用上更为灵活。

Flink SQL 是一个流与批统一的 API，也就是说理论上一段 SQL 要既能跑在批处理模式下，也能跑在流处理模式下，且输出的结果是一致的。那么在流处理模式下理所当然地应该支持 ROW_NUMBER 形式的 TopN 语法。例如上文说的对全网商家根据行业按销售额排序，计算出每个行业销售额前十名的商家，SQL 范例如下。

```
SELECT *
FROM (
  SELECT *,
    ROW_NUMBER() OVER (PARTITION BY category ORDER BY sales DESC) AS rownum
  FROM shop_sales)
WHERE rownum <= 10
```

## TopN 实现和优化

ROW_NUMBER 方式的 TopN 语法非常灵活，能满足全局 TopN 和分组 TopN 的需求。但是在流计算上的物理执行是一个挑战。如上文所述的每个行业销售额前十商家排行榜，经过 SQL 编译后得到的抽象语法树（AST）如下所示。

**LogicalWindow 会对所有数据进行排名，也就是说每当到达一个数据，就要对历史数据进行重排序，并输出历史数据的新的排名，然后 LogicalCalc 节点会根据排名进行过滤。**这在性能上是非常糟糕的，因为这无限放大了流量。而我们知道，**最优的流式 TopN**

**的计算只需要维护一个 N 元素大小的小根堆，每当有数据到达时，只需要与堆顶元素比较，如果比堆顶元素还小，则直接丢弃；如果比堆顶元素大，则更新小根堆，并输出更新后的排行榜**。也就是说我们不需要分为两个节点进行计算，不需要将所有数据进行排序，只需要在一个节点中就可以高效地完成计算。所以我们在查询优化器中加入了一条规则，在使用 TopN 语法时，将 LogicalWindow 和 LogicalCalc 合并成了 LogicalRank 节点。LogicalRank 在翻译成物理执行计划时，会使用一个经过特殊设计的 TopN 算子。

**TopN 算子的实现上主要有两个数据结构，一个是 TreeMap，另一个是 MapState**。TreeMap 的作用类似于上文的小根堆，有序地存放了排名前 N 的元素。但是 TreeMap 是个内存数据结构，在 failover 后会丢失，无法保证数据的一致性。因此我们还有一个 MapState 结构，MapState 是 Flink 提供的状态接口，用来存储 TopN 的数据（保证数据不丢）。当有 failover 发生后，MapState 能保证状态的恢复，而 TreeMap 会从 MapState 中重新构造出来。我们并有没有把顺序也存到状态中去，因为顺序是可以在恢复时重构的。因为每一次状态的读写操作都会涉及到序列化/反序列化，往往是性能的瓶颈，所以 TreeMap 的主要作用是降低了对 MapState 状态的读写操作。对大部分数据来说都是与 TreeMap 进行交互，不需要对 MapState 进行读写的，全是内存操作，所以 TopN 的性能是非常高的。

TopN 算子的主要处理流程是，每当有数据到达时，会与 TreeMap 的最小的元素比较，如果比它小，那么该数据就不可能是 TopN 的一员，直接丢弃即可。如果比它大，那么就会先更新 TreeMap，同时更新 MapState 中的存的数据。最后输出更新后的排行榜。为了减少冗余数据的输出，我们只会输出排名发生变化的数据。例如原先的第7名上升到了第六名，那么只需要输出新的第六名和第七名即可。

## 嵌套 TopN 解决热点问题

TopN 的计算与 GroupBy 的计算类似，如果**数据存在倾斜，则会有计算热点的现象**。比如全局 TopN，那么所有的数据只能汇集到一个节点进行 TopN 的计算，那么计算能力就会受限于单台机器，无法做到水平扩展。解决思路与 GroupBy 是类似的，就是**使用嵌套 TopN，或者说两层 TopN。在原先的 TopN 前面，再加一层 TopN，用于分散热点**。例如，计算全网排名前十的商铺，会导致单点的数据热点，那么可以先加一层分组 TopN，组的划分规则是根据店铺 ID 哈希取模后分成128组（并发的倍数）。第二层 TopN 与原先的写法一样，没有 PARTITION BY。第一层会计算出每一组的 TopN，而后在第二层中进行合并汇总，得到最终的全网前十。第二层虽然仍是单点，但是大量的计算量由第一层分担了，而第一层是可以水平扩展的。使用嵌套 TopN 的优化写法如下所示：

```
CREATE VIEW tmp_topn AS
SELECT *
FROM (
  SELECT *,
```

```
  ROW_NUMBER() OVER (PARTITION BY HASH_CODE(shop_id)%128 ORDER BY sales DESC)
   FROM shop_sales)    WHERE rownum <= 10

SELECT *
FROM (
  SELECT shop_id, shop_name, sales,
    ROW_NUMBER() OVER (ORDER BY sales DESC) AS rownum
  FROM tmp_topn)
WHERE rownum <= 10
```

## 总结

流式 TopN 不仅在语法以及算法上会遇到很多挑战，在不同场景下的优化方案也是个非常有意思的话题。目前 Flink SQL 的 TopN 功能已经大量应用于彩票业务、阿里云的CDN项目、WAF项目等等。未来，我们除了会针对更多的场景对 TopN 进行优化，还会提供除了 ROW_NUMBER 外的 RANK、RANK_DENSE 排名函数，使得 TopN 更加灵活。