Flink SQL语法详解

1. ANSI-SQL标准采用

1986年，ANSI和ISO标准组正式采用了标准的"数据库语言SQL"语言定义。该标准的新版本发布于1989,1992,1996,1999,2003,2006,2008,2011，以及最近的2016。Apache Flink SQL 核心算子的语义设计也参考了1992 、2011等ANSI-SQL标准。

2. 大数据计算领域对SQL的应用

离线计算（批计算）

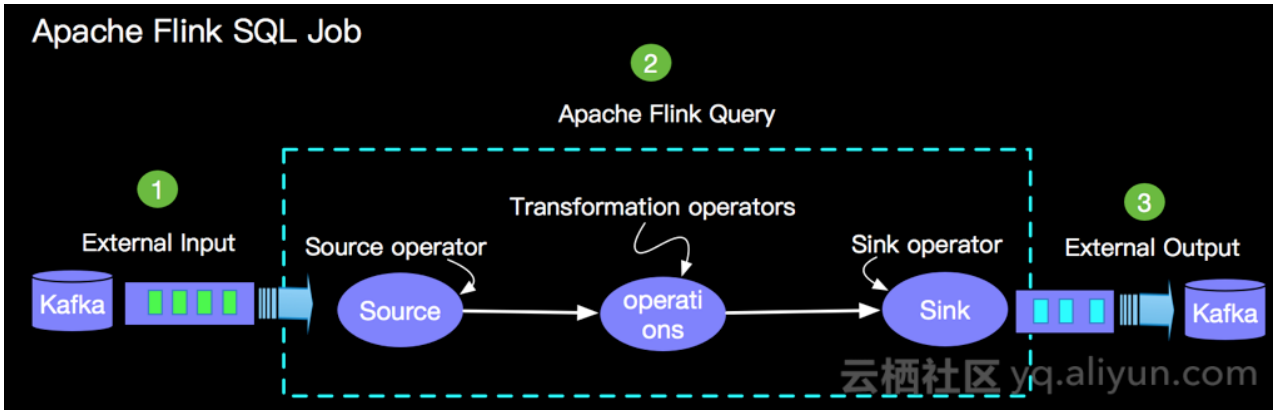
提及大数据计算领域不得不说MapReduce计算模型，MapReduce最早是由Google公司研究提出的一种面向大规模数据处理的并行计算模型和方法，并发于2004年发表了论文Simplified Data Processing on Large Clusters。
论文发表之后Apache 开源社区参考Google MapReduce，基于Java设计开发了一个称为Hadoop的开源MapReduce并行计算框架。很快得到了全球学术界和工业界的普遍关注，并得到推广和普及应用。
但利用Hadoop进行MapReduce的开发，需要开发人员精通Java语言，并了解MapReduce的运行原理，这样在一定程度上提高了MapReduce的开发门槛，所以在开源社区又不断涌现了一些为了简化MapReduce开发的开源框架，其中Hive就是典型的代表。HSQL可以让用户以类SQL的方式描述MapReduce计算，比如原本需要几十行，甚至上百行才能完成的wordCount，用户一条SQL语句就能完成了，这样极大的降低了MapReduce的开发门槛，进而也成功的将SQL应用到了大数据计算领域当中来。

实时计算（流计算）

SQL不仅仅被成功的应用到了离线计算，SQL的易用性也吸引了流计算产品，目前最热的Spark，Flink也纷纷支持了SQL，尤其是Flink支持的更加彻底，集成了Calcite，完全遵循ANSI-SQL标准。
Apache Flink在low-level API上面用DataSet支持批计算，用DataStream支持流计算，但在High-Level API上面利用SQL将流与批进行了统一，使得用户编写一次SQL既可以在流计算中使用，又可以在批计算中使用，为既有流计算业务，又有批计算业务的用户节省了大量开发成

3. Flink SQL Job的组成

Apache Flink SQL编写的计算Job由读取原始数据，计算逻辑和写入计算结果数据三部分组成。



- Source Operator - Soruce operator是对外部数据源的抽象, 目前Apache Flink内置了很多常用的数据源实现，比如上图提到的Kafka。
- Query Operators - 查询算子主要完成如图的Query Logic，目前支持了Union，Join，Projection,Difference, Intersection以及window等大多数传统数据库支持的操作。
- Sink Operator - Sink operator 是对外结果表的抽象，目前Apache Flink也内置了很多常用的结果表的抽象，比如上图提到的Kafka。

4. Flink SQL核心算子

目前Flink SQL支持Union，Join，Projection，Difference，Intersection以及Window等大多数传统数据库支持的操作。

4.1 SELECT

SELECT 用于从数据集/流中选择数据，语法遵循ANSI-SQL标准，语义是关系代数中的投影(Projection),对关系进行垂直分割，消去某些列。

4.2 WHERE

WHERE 用于从数据集/流中过滤数据，与SELECT一起使用，语法遵循ANSI-SQL标准，语义是关系代数的Selection，根据某些条件对关系做水平分割，即选择符合条件的记录。

`WHERE`是对满足一定条件的数据进行过滤， `WHERE`支持=, <, >, <>, >=, <=以及`AND`， `OR`等表达式的组合，最终满足过滤条件的数据会被选择出来。并且 `WHERE` 可以结合`IN`,`NOT IN`联合使用，

PS：IN是关系代数中的Intersection， NOT IN是关系代数的Difference。

4.3 GROUP BY

GROUP BY 是对数据进行分组的操作。

4.4 UNION ALL

UNION ALL 将两个表合并起来，要求两个表的字段完全一致，包括字段类型、字段顺序,语义对应关系代数的Union，只是关系代数是Set集合操作，会有去重复操作，**UNION ALL 不进行去重**。

4.5 UNION

UNION 将两个流给合并起来，要求两个流的字段完全一致，包括字段类型、字段顺序，并其 UNION 不同于UNION ALL，UNION会**对结果数据去重**，与关系代数的Union语义一致。

4.6 JOIN

JOIN 用于把来自两个表的行联合起来形成一个宽表，Apache Flink支持的JOIN类型：

- JOIN - INNER JOIN
- LEFT JOIN - LEFT OUTER JOIN
- RIGHT JOIN - RIGHT OUTER JOIN
- FULL JOIN - FULL OUTER JOIN

`FULL JOIN`相当于 `RIGHT JOIN` 和 `LEFT JOIN` 之后进行`UNION ALL`操作。**PS：Flink不支持Cross JOIN。**

4.7 Window

在Apache Flink中有2种类型的Window，一种是OverWindow，即传统数据库的标准开窗，每一个元素都对应一个窗口。一种是GroupWindow，目前在SQL中GroupWindow都是基于时间进行窗口划分的。

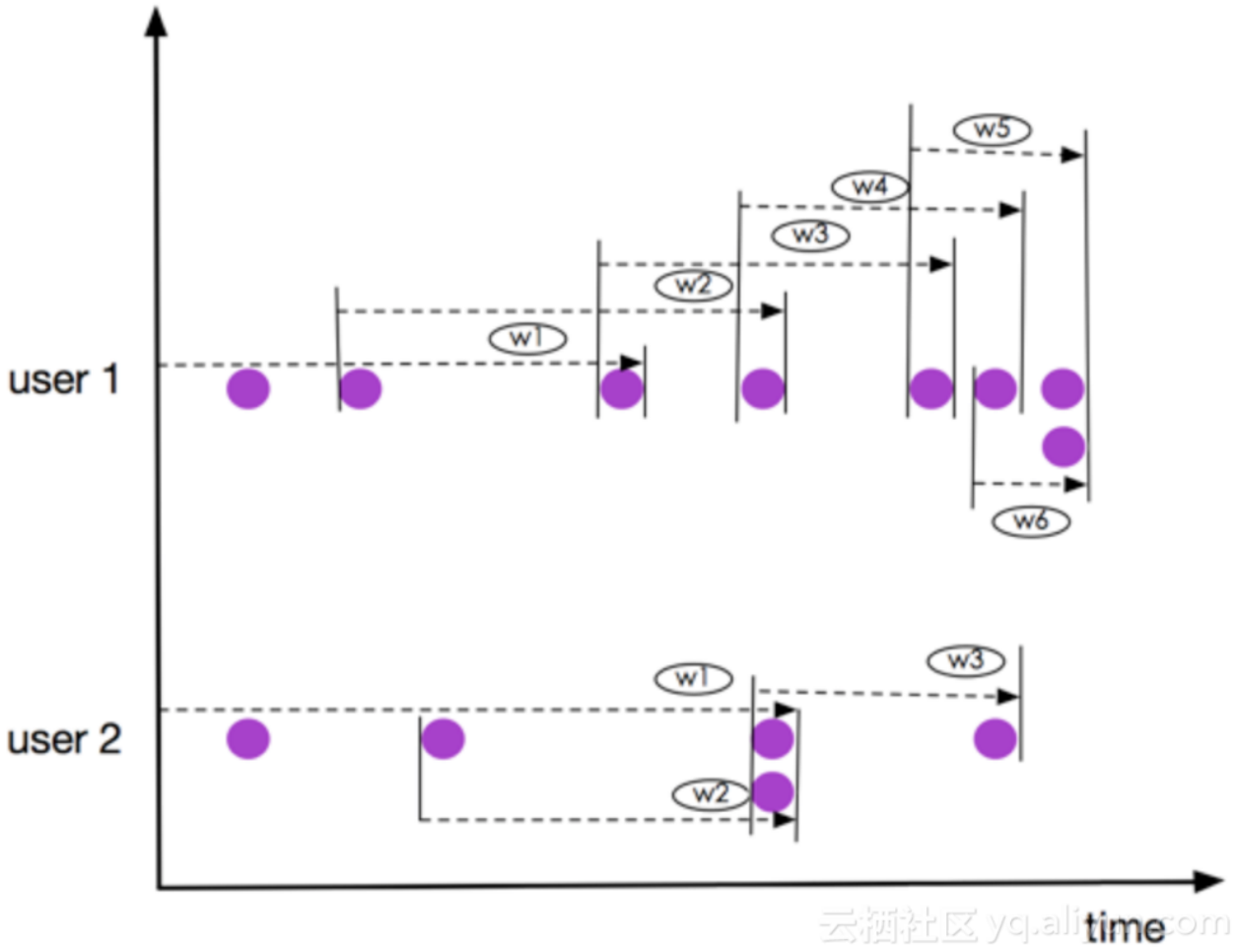
4.7.1 Over Window

Apache Flink中对OVER Window的定义遵循标准SQL的定义语法。按ROWS和RANGE分类是传统数据库的标准分类方法，在Apache Flink中还可以根据时间类型(ProcTime/EventTime)和窗口的有限和无限(Bounded/UnBounded)进行分类，共计8种类型。为了避免大家对过细分类造成困扰，我们按照确定当前行的不同方式将OVER Window分成两大类进行介绍，如下:

- ROWS OVER Window - 每一行元素都视为新的计算行，即，每一行都是一个新的窗口。
- RANGE OVER Window - 具有相同时间值的所有元素行视为同一计算行，即，具有相同时间值的所有行都是同一个窗口。

Bounded ROWS OVER Window

Bounded ROWS OVER Window 每一行元素都视为新的计算行，即，每一行都是一个新的窗口。



上图所示窗口 user 1 的 w5和w6， user 2的 窗口 w2 和 w3，虽然有元素都是同一时刻到达，但是他们仍然是在不同的窗口，这一点有别于RANGE OVER Window。

Bounded ROWS OVER Window 语法如下：

```
SELECT
    agg1(col1) OVER(
     [PARTITION BY (value_expression1,..., value_expressionN)]
     ORDER BY timeCol
     ROWS
     BETWEEN (UNBOUNDED | rowCount) PRECEDING AND CURRENT ROW) AS colName,
...
FROM Tab1
```

- value_expression - 进行分区的字表达式；
- timeCol - 用于元素排序的时间字段；
- rowCount - 是定义根据当前行开始向前追溯几行元素。

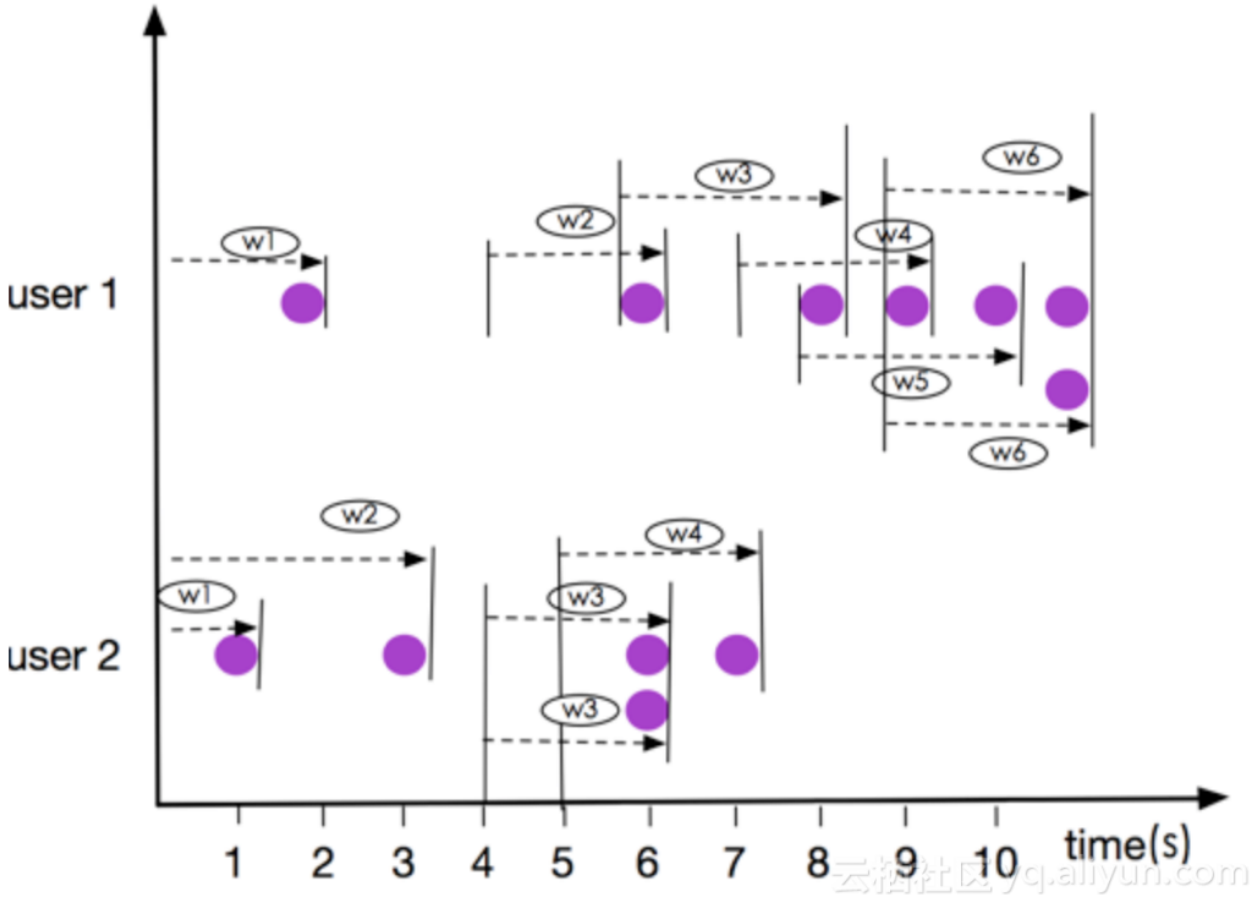
示例：

```
SELECT
    itemID,
    itemType,
    onSellTime,
    price,
    MAX(price) OVER (
        PARTITION BY itemType
        ORDER BY onSellTime
        ROWS BETWEEN 2 preceding AND CURRENT ROW) AS maxPrice -- 当前行向前追溯两
行
  FROM item_tab
```

Bounded RANGE OVER Window

Bounded RANGE OVER Window 具有相同时间值的所有元素行视为同一计算行，即，具有相同时间值的所有行都是同一个窗口。

以3秒中数据(INTERVAL '2' SECOND)的窗口为例，如下图：



注意: 上图所示窗口 user 1 的 w6， user 2的 窗口 w3，元素都是同一时刻到达,他们是在同一个窗口，这一点有别于ROWS OVER Window。

Bounded RANGE OVER Window的语法如下：

```
SELECT
    agg1(col1) OVER(
     [PARTITION BY (value_expression1,..., value_expressionN)]
     ORDER BY timeCol
     RANGE
     BETWEEN (UNBOUNDED | timeInterval) PRECEDING AND CURRENT ROW) AS colName,
...
FROM Tab1
```

- value_expression - 进行分区的字表达式；
- timeCol - 用于元素排序的时间字段；
- timeInterval - 是定义根据当前行开始向前追溯指定时间的元素行；

示例：

```
SELECT
    itemID,
    itemType,
    onSellTime,
    price,
    MAX(price) OVER (
        PARTITION BY itemType
        ORDER BY rowtime
        RANGE BETWEEN INTERVAL '2' MINUTE preceding AND CURRENT ROW) AS
maxPrice -- 两分钟前的数据
  FROM item_tab
```

4.7.2 Group Window

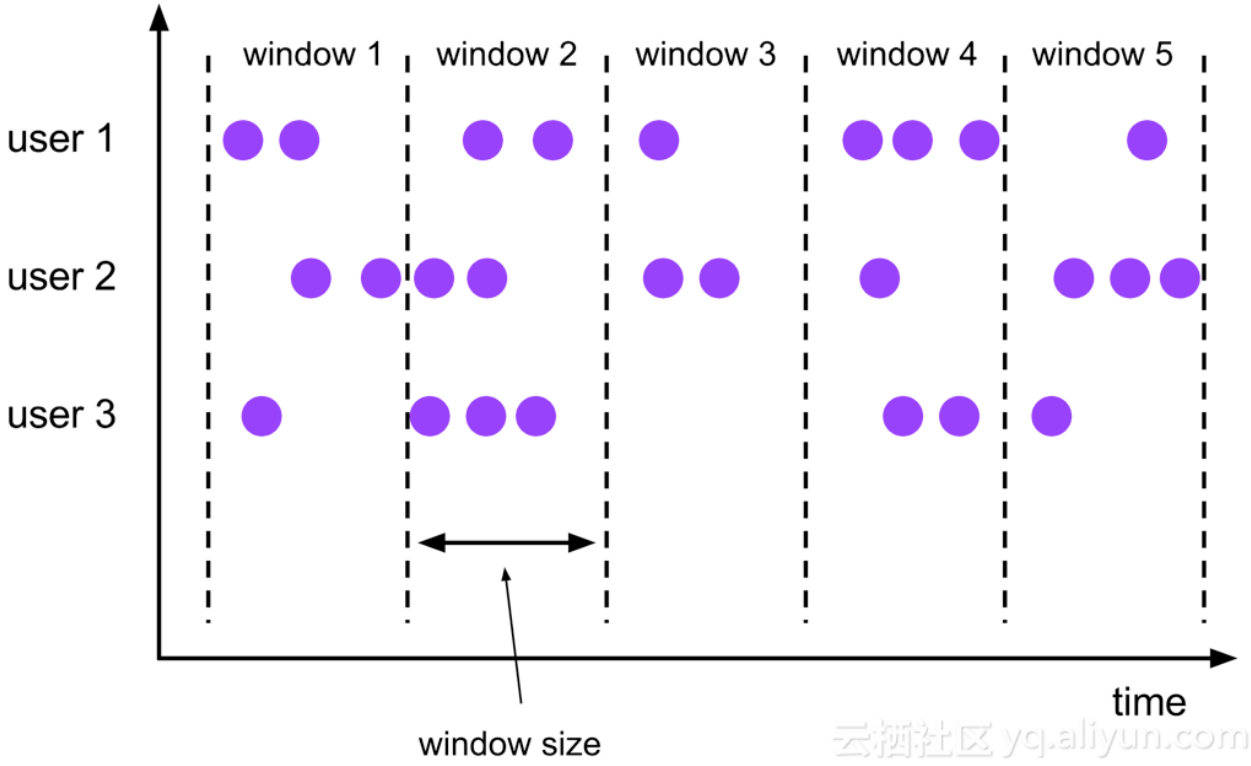
根据窗口数据划分的不同，目前Apache Flink有如下3种Bounded Winodw:

- Tumble - 滚动窗口，窗口数据有固定的大小，窗口数据无叠加；
- Hop - 滑动窗口，窗口数据有固定大小，并且有固定的窗口重建频率，窗口数据有叠加；
- Session - 会话窗口，窗口数据没有固定的大小，根据窗口数据活跃程度划分窗口，窗口数据无叠加。

说明： Aapche Flink 还支持UnBounded的 Group Window，也就是**全局Window**，流上所有数据都在一个窗口里面，语义非常简单，这里不做详细介绍了。

Tumble

Tumble 滚动窗口有固定size，窗口数据不重叠,具体语义如下：



Tumble 滚动窗口对应的语法如下：

```
SELECT
    [gk],
    [TUMBLE_START(timeCol, size)],
    [TUMBLE_END(timeCol, size)],
    agg1(col1),
    ...
    aggn(colN)
FROM Tab1
GROUP BY [gk], TUMBLE(timeCol, size)
```

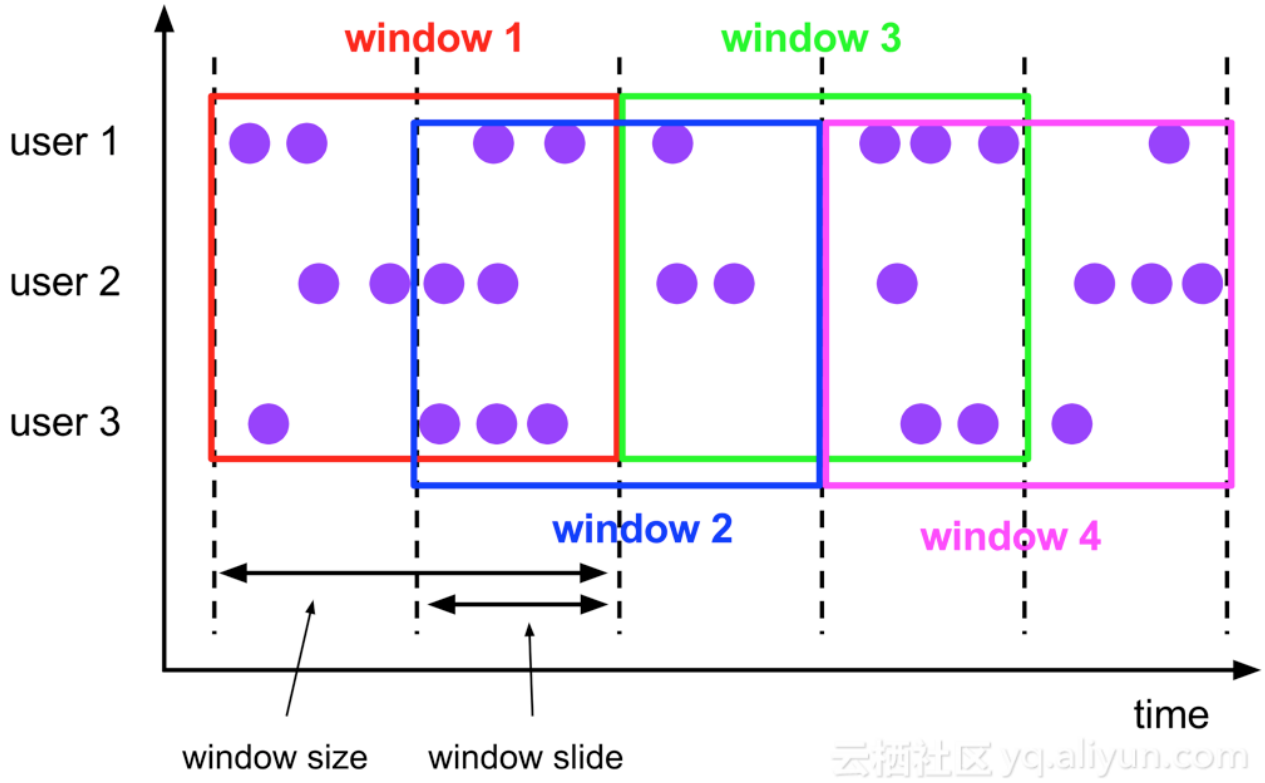
- [gk] - 决定了流是Keyed还是/Non-Keyed;
- TUMBLE_START - 窗口开始时间;
- TUMBLE_END - 窗口结束时间;
- timeCol - 是流表中表示时间字段;
- size - 表示窗口的大小，如 秒，分钟，小时，天。

示例：

```
SELECT
    region,
    TUMBLE_START(rowtime, INTERVAL '2' MINUTE) AS winStart,
    TUMBLE_END(rowtime, INTERVAL '2' MINUTE) AS winEnd,
    COUNT(region) AS pv
FROM pageAccess_tab
GROUP BY region, TUMBLE(rowtime, INTERVAL '2' MINUTE)
```

Hop

Hop 滑动窗口和滚动窗口类似，窗口有固定的size，与滚动窗口不同的是滑动窗口可以通过slide参数控制滑动窗口的新建频率。因此当**slide值小于窗口size的值的时候多个滑动窗口会重叠**。



Hop 滑动窗口语义如下所示：

```
SELECT
    [gk],
    [HOP_START(timeCol, slide, size)] ,
    [HOP_END(timeCol, slide, size)],
    agg1(col1),
    ...
    aggN(colN)
FROM Tab1
GROUP BY [gk], HOP(timeCol, slide, size)
```

- [gk] 决定了流是Keyed还是/Non-Keyed;
- HOP_START - 窗口开始时间;
- HOP_END - 窗口结束时间;
- timeCol - 是流表中表示时间字段;
- slide - 是滑动步伐的大小;
- size - 是窗口的大小，如 秒，分钟，小时，天;

```
SELECT
  HOP_START(rowtime, INTERVAL '5' MINUTE, INTERVAL '10' MINUTE) AS winStart,
  HOP_END(rowtime, INTERVAL '5' MINUTE, INTERVAL '10' MINUTE) AS winEnd,
  SUM(accessCount) AS accessCount
FROM pageAccessCount_tab
GROUP BY HOP(rowtime, INTERVAL '5' MINUTE, INTERVAL '10' MINUTE)
```

4.7.3 Session

Seeeion 会话窗口 是没有固定大小的窗口，通过session的活跃度分组元素。不同于滚动窗口和滑动窗口，会话窗口不重叠,也没有固定的起止时间。一个会话窗口在一段时间内没有接收到元素时，即当出现非活跃间隙时关闭。一个会话窗口 分配器通过配置session gap来指定非活跃周期的时长.

Seeeion 会话窗口对应语法如下：

```
SELECT
    [gk],
    SESSION_START(timeCol, gap) AS winStart,
    SESSION_END(timeCol, gap) AS winEnd,
    agg1(col1),
     ...
    aggn(colN)
FROM Tab1
GROUP BY [gk], SESSION(timeCol, gap)
```

- [gk] 决定了流是Keyed还是/Non-Keyed;
- SESSION_START - 窗口开始时间；
- SESSION_END - 窗口结束时间；
- timeCol - 是流表中表示时间字段；
- gap - 是窗口数据非活跃周期的时长；

示例：

```
SELECT
    region,
    SESSION_START(rowtime, INTERVAL '3' MINUTE) AS winStart,
    SESSION_END(rowtime, INTERVAL '3' MINUTE) AS winEnd,
    COUNT(region) AS pv
FROM pageAccessSession_tab
GROUP BY region, SESSION(rowtime, INTERVAL '3' MINUTE)
```