spark、hive中窗口函数实现原理

一、业务背景

前些天在和业务方一起看数据时，发现了一个问题，关于窗口函数的理解和使用的。
下面做了一张测试表模拟业务，用少量的数据复现了一下：

```
create table window_test_table (
  id int,    --用户设备号
  sq string,  --标识每次搜索词
  cell_type int, --代表结果类型，阿拉丁为26
  rank int  --这次搜索下结果的位置
)ROW FORMAT DELIMITED FIELDS TERMINATED BY ',';
```

在该表中插入以下数据：

```
hive> select * from window_test_table;
OK
1       flower  10      1
1       happy   12      2
1       tree    26      3
1       hive    10      4
1       hadoop  13      5
1       spark   26      6
1       flink   14      7
1       sqoop   10      8
```

现在有个需求，想新加一列 每个用户每次搜索下非阿拉丁类型的结果位置自然排序，如果下效果：

```
20/04/04 23:50:52 INFO storage.BlockManagerInfo: Remo
1       flower  10      1       1
1       happy   12      2       2
1       tree    26      3       NULL
1       hive    10      4       3
1       hadoop  13      5       4
1       spark   26      6       NULL
1       flink   14      7       5
1       sqoop   10      8       6
Time taken: 0.801 seconds, Fetched 8 row(s)
```

```sql
--业务方的写法
select
    id,
    sq,
    cell_type,
    rank,
    if(cell_type!=26,row_number() over(partition by id  order by rank),null) naturl_rank
from window_test_table order by rank;
```

结果:

```
20/04/04 23:57:28 INFO scheduler.DAGScheduler: Job 14 finished: pro
1       flower  10      1       1
1       happy   12      2       2
1       tree    26      3       NULL
1       hive    10      4       4
1       hadoop  13      5       5
1       spark   26      6       NULL
1       flink   14      7       7
1       sqoop   10      8       8
Time taken: 0.729 seconds, Fetched 8 row(s)
```

上面这种结果显然不是我们想要的，虽然把26类型的rank去掉了，但是非阿拉丁的结果的位置排序没有向上补充

为什么呢？ 感觉写的也没错呀？ ~~~~
下面，我们来盘一盘window Funtion的实现原理

二、window 实现原理

在分析原理之前，先简单过一下window Funtion的使用范式：

```
select row_number() over( partition by col1 order by col2 ) from table
```

上面的语句主要分两部分
1. window函数部分（window_func）
2. 窗口定义部分

2.1 window函数部分

windows函数部分就是所要在窗口上执行的函数，spark支持三中类型的窗口函数：
1. 聚合函数 （aggregate functions）
2. 排序函数（Ranking functions）
3. 分析窗口函数（Analytic functions）

第一种都比较熟悉就是常用的count 、sum、avg等
第二种就是row_number、rank这样的排序函数
第三种专门为窗口而生的函数比如：cume_dist函数计算当前值在窗口中的百分位数

2.2 窗口定义部分

这部分就是over里面的内容了
里面也有**三部分**
1. partition by
2. order by
3. ROWS | RANGE BETWEEN

前两部分就是把数据**分桶**然后**桶内排序**，排好了序才能很好的定位出你需要向前或者向后取哪些数据来参与计算。
这第三部分就是确定你需要哪些数据了。

spark提供了两种方式一种是**ROWS BETWEEN**也就是按照距离来取
例如

1. *ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW*就是取从最开始到当前这一条数据，row_number()这个函数就是这样取的

2. *ROWS BETWEEN 2 PRECEDING AND 2 FOLLOWING*代表取前面两条和后面两条数据参与计算，比如计算前后五天内的移动平均就可以这样算.
3. **RANGE BETWEEN** 这种就是以当前值为锚点进行计算。比如*RANGE BETWEEN 20 PRECEDING AND 10 FOLLOWING*当前值为50的话就去前后的值在30到60之间的数据。

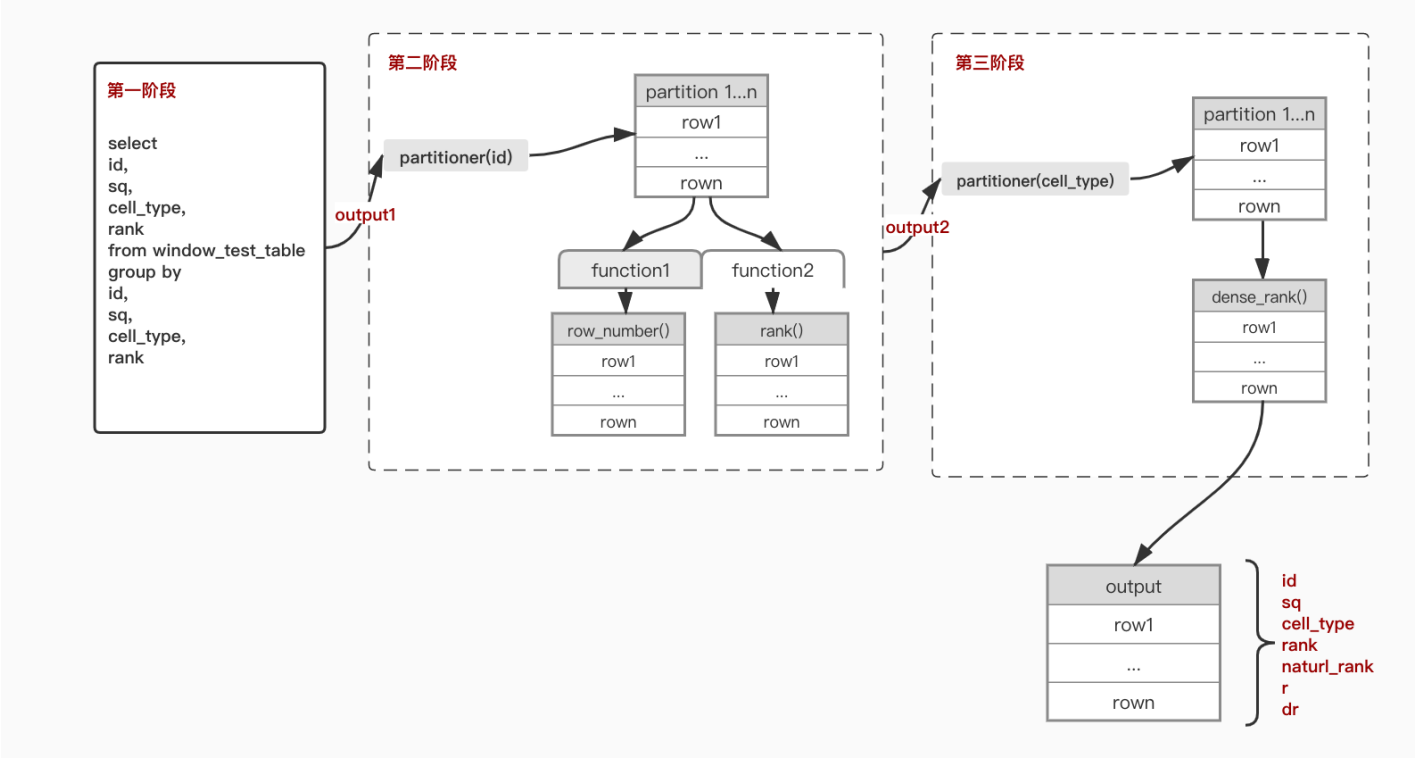
2.3 window Function 实现原理

窗口函数的实现，主要借助 Partitioned Table Function （即PTF）；
PTF的输入可以是：表、子查询或另一个PTF函数输出；
PTF输出也是一张表。

写一个相对复杂的sql，来看一下执行窗口函数时，数据的流转情况：

```
select
   id,
   sq,
   cell_type,
   rank,
   row_number() over(partition by id  order by rank ) naturl_rank,
   rank() over(partition by id order by rank) as r,
   dense_rank() over(partition by  cell_type order by id) as dr
from window_test_table
group by id,sq,cell_type,rank;
```

数据流转如下图：



以上代码实现主要有三个阶段：

- 计算除窗口函数以外所有的其他运算，如：group by，join ，having等。上面的代码的第一阶段即为：

```
select
      id,
      sq,
      cell_type,
      rank
from window_test_table
group by
      id,
      sq,
      cell_type,
      rank
```

- 将第一步的输出作为第一个 PTF 的输入，计算对应的窗口函数值。上面代码的第二阶段即为：

```
select id,sq,cell_type,rank,naturl_rank,r from
window(
<w>,--将第一阶段输出记为w
partition by id, --分区
```

```
order by rank, --窗口函数的order
[naturl_rank:row_number(),r:rank()] --窗口函数调用
)
```

由于row_number()，rank() 两个函数对应的窗口是相同的（partition by id order by rank），因此，这两个函数可以在一次shuffle中完成。

- 将第二步的输出作为 第二个PTF 的输入，计算对应的窗口函数值。上面代码的第三阶段即为：

```
select id,sq,cell_type,rank,naturl_rank,r,dr from
window(
<w1>,--将第二阶段输出记为w1
partition by cell_type, --分区
order by id, --窗口函数的order
[dr:dense_rank()] --窗口函数调用
)
```

由于dense_rank()的窗口与前两个函数不同，因此需要再partition一次，得到最终的输出结果。

以上可知，得到最终结果，需要shuffle三次，反应在 mapreduce上面，就是要经历三次map->reduce组合； 反应在spark sql上，就是要Exchange三次，再加上中间排序操作，在数据量很大的情况下，效率基本没救~~
这些可能就是窗口函数运行效率慢的原因之一了。

这里给附上spark sql的执行计划，可以仔细品一下（hive sql的执行计划实在太长，但套路基本是一样的）：

```
spark-sql> explain select id,sq,cell_type,rank,row_number() over(partition by id order by rank ) naturl_rank,rank() over(partition by id order by rank) as r,dense_rank() over(partition by cell_type order by id) as dr from window_test_table group by id,sq,cell_type,rank;

== Physical Plan ==
Window [dense_rank(id#164) windowspecdefinition(cell_type#166, id#164 ASC NULLS FIRST, specifiedwindowframe(RowFrame, unboundedpreceding$(), currentrow$())) AS dr#156], [cell_type#166], [id#164 ASC NULLS FIRST]
+- *(4) Sort [cell_type#166 ASC NULLS FIRST, id#164 ASC NULLS FIRST], false, 0
   +- Exchange hashpartitioning(cell_type#166, 200)
```

```
      +- Window [row_number() windowspecdefinition(id#164, rank#167 ASC NULLS FIRST, specifiedwindowframe(RowFrame, unboundedpreceding$(), currentrow$())) AS naturl_rank#154, rank(rank#167) windowspecdefinition(id#164, rank#167 ASC NULLS FIRST, specifiedwindowframe(RowFrame, unboundedpreceding$(), currentrow$())) AS r#155], [id#164], [rank#167 ASC NULLS FIRST]
         +- *(3) Sort [id#164 ASC NULLS FIRST, rank#167 ASC NULLS FIRST], false, 0
            +- Exchange hashpartitioning(id#164, 200)
               +- *(2) HashAggregate(keys=[id#164, sq#165, cell_type#166, rank#167], functions=[])
                  +- Exchange hashpartitioning(id#164, sq#165, cell_type#166, rank#167, 200)
                     +- *(1) HashAggregate(keys=[id#164, sq#165, cell_type#166, rank#167], functions=[])
                        +- Scan hive tmp.window_test_table [id#164, sq#165, cell_type#166, rank#167], HiveTableRelation `tmp`.`window_test_table`, org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe, [id#164, sq#165, cell_type#166, rank#167]
Time taken: 0.064 seconds, Fetched 1 row(s)
```

三、解决方案

回顾上面sql的写法：

```
select
        id,sq,cell_type,rank,
        if(cell_type!=26,row_number() over(partition by id  order by rank),null) naturl_rank
from window_test_table
```

从执行计划中，可以看到sql中 if 函数的执行位置如下：

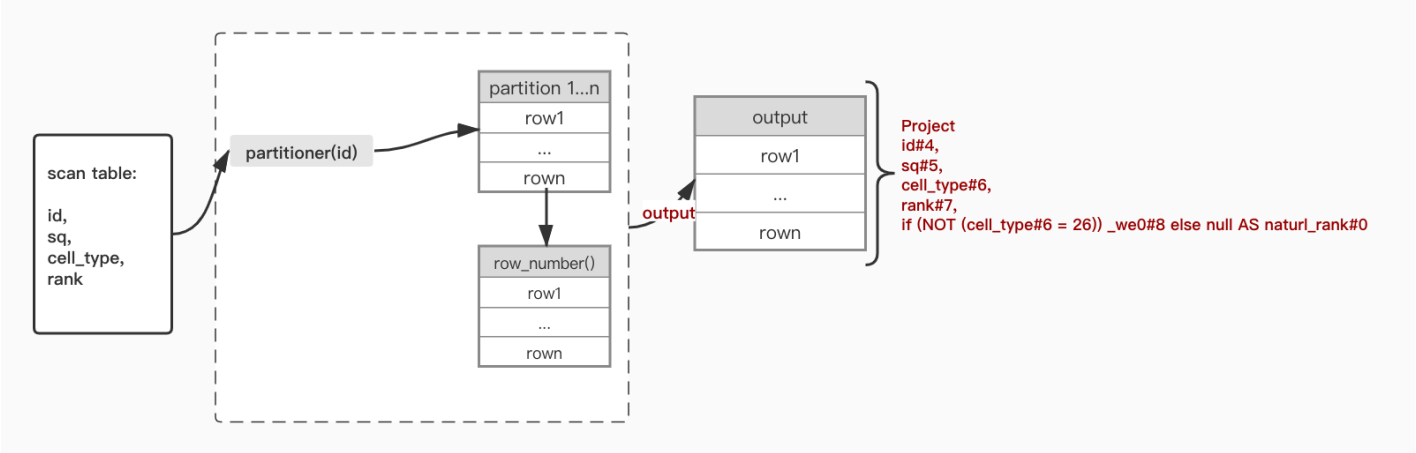
```
spark-sql> explain select id,sq,cell_type,rank,if(cell_type!=26,row_number() over(partition by id order by rank),null) naturl_rank from window_test_table;


== Physical Plan ==
*(2) Project [id#4, sq#5, cell_type#6, rank#7, if (NOT (cell_type#6 = 26)) _we0#8 else null AS naturl_rank#0]  -- partition以及row_number后，执行if
```

```
+- Window [row_number() windowspecdefinition(id#4, rank#7 ASC NULLS FIRST, specifiedwindowframe(RowFrame, unboundedpreceding$(), currentrow$())) AS _we0#8], [id#4], [rank#7 ASC NULLS FIRST]
  +- *(1) Sort [id#4 ASC NULLS FIRST, rank#7 ASC NULLS FIRST], false, 0
   +- Exchange hashpartitioning(id#4, 200)
      +- Scan hive tmp.window_test_table [id#4, sq#5, cell_type#6, rank#7], HiveTableRelation `tmp`.`window_test_table`, org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe, [id#4, sq#5, cell_type#6, rank#7]

Time taken: 0.728 seconds, Fetched 1 row(s)
```

数据流转：



if函数在partition以及row_number后执行，因此得到的位置排名不正确。改写一下：

```
select
        id,sq,cell_type,rank,
        if(cell_type!=26,row_number() over(partition by if(cell_type!=26,id,rand()) order by rank),null) naturl_rank
from window_test_table
```

这样写法要注意的地方：要保证 rand() 函数不会与id发生碰撞。
或者下面的写法也可以：

```
select
        id,sq,cell_type,rank,
        row_number() over(partition by id        order by rank) as naturl_rank
from window_test_table
where cell_type!=26
```

```
union all
select
        id,sq,cell_type,rank,
        null as naturl_rank
from window_test_table
where cell_type=26
```

缺点就是要读两遍 window_test_table 表

四、其它总结

sparksql比hivesql优化的点（窗口函数）
Hive sql窗口函数源码分析

五、参考资料

- https://issues.apache.org/jira/browse/HIVE-896
- https://cwiki.apache.org/confluence/display/Hive/LanguageManual+WindowingAndAnalytics
- https://github.com/hbutani/SQLWindowing
- https://content.pivotal.io/blog/time-series-analysis-1-introduction-to-window-functions
- https://databricks.com/blog/2015/07/15/introducing-window-functions-in-spark-sql.html