Hive窗口函数

简介

本文主要介绍hive中的窗口函数.hive中的窗口函数和sql中的窗口函数相类似,都是用来做一些数据分析类的工作,一般用于olap分析

概念

我们都知道在sql中有一类函数叫做聚合函数,例如sum()、avg()、max()等等,这类函数可以将多行数据按照规则聚集为一行,一般来讲聚集后的行数是要少于聚集前的行数的.但是有时我们想要既显示聚集前的数据,又要显示聚集后的数据,这时我们便引入了窗口函数.

在深入研究Over字句之前，一定要注意：在SQL处理中，窗口函数都是最后一步执行，而且仅位于Order by字句之前.

数据准备

我们准备一张order表,字段分别为name,orderdate,cost.数据内容如下:

```
jack,2015-01-01,10
tony,2015-01-02,15
jack,2015-02-03,23
tony,2015-01-04,29
jack,2015-01-05,46
jack,2015-04-06,42
tony,2015-01-07,50
jack,2015-01-08,55
mart,2015-04-08,62
mart,2015-04-09,68
neil,2015-05-10,12
mart,2015-04-11,75
neil,2015-06-12,80
mart,2015-04-13,94
```

在hive中建立一张表t_window,将数据插入进去.

实例

聚合函数+over

假如说我们想要查询在2015年4月份购买过的顾客及总人数,我们便可以使用窗口函数去去实现

```
select name,count(*) over ()
from t_window
where substring(orderdate,1,7) = '2015-04'
```

得到的结果如下:

```
name    count_window_0
mart    5
mart    5
mart    5
mart    5
jack    5
```

可见其实在2015年4月一共有5次购买记录,mart购买了4次,jack购买了1次.事实上,大多数情况下,我们是只看去重后的结果的.针对于这种情况,我们有两种实现方式

第一种：distinct

```
select distinct name,count(*) over ()
from t_window
where substring(orderdate,1,7) = '2015-04'
```

第二种:group by

```
select name,count(*) over ()
from t_window
where substring(orderdate,1,7) = '2015-04'
group by name
```

执行后的结果如下:

name count_window_0

mart 2

jack 2

partition by子句

Over子句之后第一个提到的就是Partition By.Partition By子句也可以称为查询分区子句，非常类似于Group By，都是将数据按照边界值分组，而Over之前的函数在每一个分组之内进行，如果超出了分组，则函数会重新计算.

实例

我们想要去看顾客的购买明细及月购买总额,可以执行如下的sql

```
select name,orderdate,cost,sum(cost) over(partition by month(orderdate))
from t_window
```

执行结果如下:

```
name    orderdate   cost    sum_window_0
jack    2015-01-01  10  205
jack    2015-01-08  55  205
tony    2015-01-07  50  205
jack    2015-01-05  46  205
tony    2015-01-04  29  205
tony    2015-01-02  15  205
jack    2015-02-03  23  23
mart    2015-04-13  94  341
jack    2015-04-06  42  341
mart    2015-04-11  75  341
mart    2015-04-09  68  341
mart    2015-04-08  62  341
neil    2015-05-10  12  12
neil    2015-06-12  80  80
```

可以看出数据已经按照月进行汇总了.

order by子句

上述的场景,假如我们想要将cost按照月进行累加.这时我们引入order by子句.

order by子句会让输入的数据强制排序（文章前面提到过，窗口函数是SQL语句最后执行的函数，因此可以把SQL结果集想象成输入数据）。Order By子句对于诸如Row_Number()，Lead()，LAG()等函数是必须的，因为如果数据无序，这些函数的结果就没有任何意义。因此如果有了Order By子句，则Count()，Min()等计算出来的结果就没有任何意义。

我们在上面的代码中加入order by

```
select name,orderdate,cost,sum(cost) over(partition by month(orderdate) order by orderdate )
from t_window
```

得到的结果如下：(order by默认情况下聚合从起始行当当前行的数据)

```
name    orderdate   cost    sum_window_0
jack    2015-01-01  10  10
tony    2015-01-02  15  25
tony    2015-01-04  29  54
jack    2015-01-05  46  100
tony    2015-01-07  50  150
jack    2015-01-08  55  205
jack    2015-02-03  23  23
jack    2015-04-06  42  42
mart    2015-04-08  62  104
mart    2015-04-09  68  172
mart    2015-04-11  75  247
mart    2015-04-13  94  341
neil    2015-05-10  12  12
neil    2015-06-12  80  80
```

window子句

我们在上面已经通过使用partition by子句将数据进行了分组的处理.如果我们想要更细粒度的划分，我们就要引入window子句了.

我们首先要理解两个概念:

- 如果只使用partition by子句,未指定order by的话,我们的聚合是分组内的聚合.

- 使用了order by子句,未使用window子句的情况下,默认从起点到当前行.

当同一个select查询中存在多个窗口函数时,他们相互之间是没有影响的.每个窗口函数应用自己的规则.

window子句：

- PRECEDING：往前

- FOLLOWING：往后

- CURRENT ROW：当前行

- UNBOUNDED：起点，UNBOUNDED PRECEDING 表示从前面的起点， UNBOUNDED FOLLOWING：表示到后面的终点

我们按照name进行分区,按照购物时间进行排序,做cost的累加.

如下我们结合使用window子句进行查询

```
select name,orderdate,cost,
sum(cost) over() as sample1,--所有行相加
sum(cost) over(partition by name) as sample2,--按name分组，组内数据相加
sum(cost) over(partition by name order by orderdate) as sample3,--按name分组，组内数据累加
sum(cost) over(partition by name order by orderdate rows between UNBOUNDED PRECEDING and current row )
as sample4 ,--和sample3一样,由起点到当前行的聚合
sum(cost) over(partition by name order by orderdate rows between 1 PRECEDING   and current row) as
sample5, --当前行和前面一行做聚合
sum(cost) over(partition by name order by orderdate rows between 1 PRECEDING   AND 1 FOLLOWING  ) as
sample6,--当前行和前边一行及后面一行
sum(cost) over(partition by name order by orderdate rows between current row and UNBOUNDED FOLLOWING ) as
sample7 --当前行及后面所有行
from t_window;
```

得到查询结果如下：

```
name    orderdate   cost    sample1 sample2 sample3 sample4 sample5 sample6 sample7
jack    2015-01-01  10  661 176 10  10  10  56  176
jack    2015-01-05  46  661 176 56  56  56  111 166
jack    2015-01-08  55  661 176 111 111 101 124 120
jack    2015-02-03  23  661 176 134 134 78  120 65
jack    2015-04-06  42  661 176 176 176 65  65  42
mart    2015-04-08  62  661 299 62  62  62  130 299
mart    2015-04-09  68  661 299 130 130 130 205 237
mart    2015-04-11  75  661 299 205 205 143 237 169
mart    2015-04-13  94  661 299 299 299 169 169 94
neil    2015-05-10  12  661 92  12  12  12  92  92
neil    2015-06-12  80  661 92  92  92  92  92  80
tony    2015-01-02  15  661 94  15  15  15  44  94
tony    2015-01-04  29  661 94  44  44  44  94  79
tony    2015-01-07  50  661 94  94  94  79  79  50
```

窗口函数中的序列函数

主要序列函数是不支持window子句的.

hive中常用的序列函数有下面几个:

NTILE

- NTILE(n)，用于将分组数据按照顺序切分成n片，返回当前切片值
- NTILE不支持ROWS BETWEEN，

比如 NTILE(2) OVER(PARTITION BY cookieid ORDER BY createtime ROWS BETWEEN 3 PRECEDING AND CURRENT ROW)

- 如果切片不均匀，默认增加第一个切片的分布

这个函数用什么应用场景呢?假如我们想要每位顾客购买金额前1/3的交易记录,我们便可以使用这个函数.

```
select name,orderdate,cost,
       ntile(3) over() as sample1 , --全局数据切片
       ntile(3) over(partition by name), -- 按照name进行分组,在分组内将数据切成3份
       ntile(3) over(order by cost),--全局按照cost升序排列,数据切成3份
       ntile(3) over(partition by name order by cost ) --按照name分组，在分组内按照cost升序排列,数据切成3份
from t_window
```

得到的数据如下：

```
name    orderdate   cost    sample1 sample2 sample3 sample4
jack    2015-01-01  10  3   1   1   1
jack    2015-02-03  23  3   1   1   1
jack    2015-04-06  42  2   2   2   2
jack    2015-01-05  46  2   2   2   2
jack    2015-01-08  55  2   3   2   3
mart    2015-04-08  62  2   1   2   1
mart    2015-04-09  68  1   2   3   1
mart    2015-04-11  75  1   3   3   2
mart    2015-04-13  94  1   1   3   3
neil    2015-05-10  12  1   2   1   1
neil    2015-06-12  80  1   1   3   2
tony    2015-01-02  15  3   2   1   1
tony    2015-01-04  29  3   3   1   2
tony    2015-01-07  50  2   1   2   3
```

如上述数据，我们去sample4 = 1的那部分数据就是我们要的结果

row_number、rank、dense_rank

这三个窗口函数的使用场景非常多

- row_number()从1开始，按照顺序，生成分组内记录的序列,row_number()的值不会存在重复,当排序的值相同时,按照表中记录的顺序进行排列

- RANK() 生成数据项在分组中的排名，排名相等会在名次中留下空位

- DENSE_RANK() 生成数据项在分组中的排名，排名相等会在名次中不会留下空位

**注意：

rank和dense_rank的区别在于排名相等时会不会留下空位.**

举例如下:

```
SELECT
cookieid,
createtime,
pv,
RANK() OVER(PARTITION BY cookieid ORDER BY pv desc) AS rn1,
DENSE_RANK() OVER(PARTITION BY cookieid ORDER BY pv desc) AS rn2,
ROW_NUMBER() OVER(PARTITION BY cookieid ORDER BY pv DESC) AS rn3
FROM lxw1234
WHERE cookieid = 'cookie1';

cookieid day           pv       rn1     rn2     rn3

cookie1 2015-04-12      7       1       1       1
cookie1 2015-04-11      5       2       2       2
cookie1 2015-04-15      4       3       3       3
cookie1 2015-04-16      4       3       3       4
cookie1 2015-04-13      3       5       4       5
cookie1 2015-04-14      2       6       5       6
cookie1 2015-04-10      1       7       6       7
```

rn1: 15号和16号并列第3, 13号排第5
rn2: 15号和16号并列第3, 13号排第4
rn3: 如果相等，则按记录值排序，生成唯一的次序，如果所有记录值都相等，或许会随机排吧。

LAG和LEAD函数

这两个函数为常用的窗口函数,可以返回上下数据行的数据.

以我们的订单表为例,假如我们想要查看顾客上次的购买时间可以这样去查询

```
select name,orderdate,cost,
lag(orderdate,1,'1900-01-01') over(partition by name order by orderdate ) as time1,
lag(orderdate,2) over (partition by name order by orderdate) as time2
from t_window;
```

查询后的数据为:

```
name    orderdate   cost    time1   time2
jack    2015-01-01  10  1900-01-01  NULL
jack    2015-01-05  46  2015-01-01  NULL
jack    2015-01-08  55  2015-01-05  2015-01-01
jack    2015-02-03  23  2015-01-08  2015-01-05
jack    2015-04-06  42  2015-02-03  2015-01-08
mart    2015-04-08  62  1900-01-01  NULL
mart    2015-04-09  68  2015-04-08  NULL
mart    2015-04-11  75  2015-04-09  2015-04-08
mart    2015-04-13  94  2015-04-11  2015-04-09
neil    2015-05-10  12  1900-01-01  NULL
neil    2015-06-12  80  2015-05-10  NULL
tony    2015-01-02  15  1900-01-01  NULL
tony    2015-01-04  29  2015-01-02  NULL
tony    2015-01-07  50  2015-01-04  2015-01-02
```

time1取的为按照name进行分组,分组内升序排列,取上一行数据的值.

time2取的为按照name进行分组，分组内升序排列,取上面2行的数据的值,注意当lag函数为设置行数值时,默认为1行.未设定取不到时的默认值时,取null值.

lead函数与lag函数方向相反,取向下的数据.

first_value和last_value

first_value取分组内排序后，截止到当前行，第一个值

last_value取分组内排序后，截止到当前行，最后一个值

```
select name,orderdate,cost,
first_value(orderdate) over(partition by name order by orderdate) as time1,
last_value(orderdate) over(partition by name order by orderdate) as time2
from t_window
```

查询结果如下:

```
name    orderdate   cost    time1   time2
jack    2015-01-01  10  2015-01-01  2015-01-01
```

```
jack    2015-01-05  46  2015-01-01  2015-01-05
jack    2015-01-08  55  2015-01-01  2015-01-08
jack    2015-02-03  23  2015-01-01  2015-02-03
jack    2015-04-06  42  2015-01-01  2015-04-06
mart    2015-04-08  62  2015-04-08  2015-04-08
mart    2015-04-09  68  2015-04-08  2015-04-09
mart    2015-04-11  75  2015-04-08  2015-04-11
mart    2015-04-13  94  2015-04-08  2015-04-13
neil    2015-05-10  12  2015-05-10  2015-05-10
neil    2015-06-12  80  2015-05-10  2015-06-12
tony    2015-01-02  15  2015-01-02  2015-01-02
tony    2015-01-04  29  2015-01-02  2015-01-04
tony    2015-01-07  50  2015-01-02  2015-01-07
```

参考内容:

- SQL Server中的窗口函数

- 分析函数——排序排列（rank、dense_rank、row_number）

- SQL中的窗口函数 OVER窗口函数