Hive分析窗口函数系列文章

分析窗口函数应用场景：

（1）用于分区排序

（2）动态Group By

（3）Top N

（4）累计计算

（5）层次查询

Hive分析窗口函数(一) SUM,AVG,MIN,MAX

Hive中提供了越来越多的分析函数，用于完成负责的统计分析。抽时间将所有的分析窗口函数理一遍，将陆续发布。

今天先看几个基础的，SUM、AVG、MIN、MAX。

用于实现分组内所有和连续累积的统计。

数据准备：

```
CREATE EXTERNAL TABLE lxw1234 (
cookieid string,
createtime string,   --day
pv INT
) ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
stored as textfile location '/tmp/lxw11/';

DESC lxw1234;
cookieid                STRING
createtime              STRING
pv INT

hive> select * from lxw1234;
OK
cookie1 2015-04-10      1
cookie1 2015-04-11      5
cookie1 2015-04-12      7
cookie1 2015-04-13      3
cookie1 2015-04-14      2
cookie1 2015-04-15      4
cookie1 2015-04-16      4
```

SUM — 注意,结果和ORDER BY相关,默认为升序

```
SELECT cookieid,
createtime,
pv,
SUM(pv) OVER(PARTITION BY cookieid ORDER BY createtime) AS pv1, -- 默认为从起点到当前行
```

SUM(pv) OVER(PARTITION BY cookieid ORDER BY createtime ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) AS pv2, --从起点到当前行，结果同pv1
SUM(pv) OVER(PARTITION BY cookieid) AS pv3, --分组内所有行
SUM(pv) OVER(PARTITION BY cookieid ORDER BY createtime ROWS BETWEEN 3 PRECEDING AND CURRENT ROW) AS pv4, --当前行+往前3行
SUM(pv) OVER(PARTITION BY cookieid ORDER BY createtime ROWS BETWEEN 3 PRECEDING AND 1 FOLLOWING) AS pv5, --当前行+往前3行+往后1行
SUM(pv) OVER(PARTITION BY cookieid ORDER BY createtime ROWS BETWEEN CURRENT ROW AND UNBOUNDED FOLLOWING) AS pv6 ---当前行+往后所有行
FROM lxw1234;

```
cookieid createtime    pv    pv1   pv2   pv3   pv4   pv5   pv6
---------------------------------------------------------------------------
cookie1  2015-04-10    1     1     1     26    1     6     26
cookie1  2015-04-11    5     6     6     26    6     13    25
cookie1  2015-04-12    7     13    13    26    13    16    20
cookie1  2015-04-13    3     16    16    26    16    18    13
cookie1  2015-04-14    2     18    18    26    17    21    10
cookie1  2015-04-15    4     22    22    26    16    20    8
cookie1  2015-04-16    4     26    26    26    13    13    4
```

pv1: 分组内从起点到当前行的pv累积，如，11号的pv1=10号的pv+11号的pv, 12号=10号+11号+12号

pv2: 同pv1

pv3: 分组内(cookie1)所有的pv累加

pv4: 分组内当前行+往前3行，如，11号=10号+11号， 12号=10号+11号+12号， 13号=10号+11号+12号+13号， 14号=11号+12号+13号+14号

pv5: 分组内当前行+往前3行+往后1行，如，14号=11号+12号+13号+14号+15号=5+7+3+2+4=21

pv6: 分组内当前行+往后所有行，如，13号=13号+14号+15号+16号=3+2+4+4=13，14号=14号+15号+16号=2+4+4=10

如果不指定ROWS BETWEEN,默认为从起点到当前行;

如果不指定ORDER BY，则将分组内所有值累加;

关键是理解ROWS BETWEEN含义,也叫做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

–其他AVG，MIN，MAX，和SUM用法一样。

```
--AVG
SELECT cookieid,
createtime,
pv,
```

```
AVG(pv) OVER(PARTITION BY cookieid ORDER BY createtime) AS pv1, -- 默认为从起点到当前行
AVG(pv) OVER(PARTITION BY cookieid ORDER BY createtime ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) AS pv2, --从起点到当前行，结果同pv1
AVG(pv) OVER(PARTITION BY cookieid) AS pv3, --分组内所有行
AVG(pv) OVER(PARTITION BY cookieid ORDER BY createtime ROWS BETWEEN 3 PRECEDING AND CURRENT ROW) AS pv4, --当前行+往前3行
AVG(pv) OVER(PARTITION BY cookieid ORDER BY createtime ROWS BETWEEN 3 PRECEDING AND 1 FOLLOWING) AS pv5, --当前行+往前3行+往后1行
AVG(pv) OVER(PARTITION BY cookieid ORDER BY createtime ROWS BETWEEN CURRENT ROW AND UNBOUNDED FOLLOWING) AS pv6  ---当前行+往后所有行
FROM lxw1234;
cookieid createtime    pv    pv1    pv2    pv3    pv4    pv5    pv6
---------------------------------------------------------------------------
cookie1 2015-04-10    1    1.0    1.0    3.7142857142857144    1.0    3.0    3.7142857142857144
cookie1 2015-04-11    5    3.0    3.0    3.7142857142857144    3.0    4.333333333333333    4.166666666666667
cookie1 2015-04-12    7    4.333333333333333    4.333333333333333    3.7142857142857144    4.333333333333333    4.0    4.0
cookie1 2015-04-13    3    4.0    4.0    3.7142857142857144    4.0    3.6    3.25
cookie1 2015-04-14    2    3.6    3.6    3.7142857142857144    4.25    4.2    3.3333333333333335
cookie1 2015-04-15    4    3.6666666666666665    3.6666666666666665    3.7142857142857144    4.0    4.0    4.0
cookie1 2015-04-16    4    3.7142857142857144    3.7142857142857144    3.7142857142857144    3.25    3.25    4.0

--MIN
SELECT cookieid,
createtime,
pv,
MIN(pv) OVER(PARTITION BY cookieid ORDER BY createtime) AS pv1, -- 默认为从起点到当前行
MIN(pv) OVER(PARTITION BY cookieid ORDER BY createtime ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) AS pv2, --从起点到当前行，结果同pv1
MIN(pv) OVER(PARTITION BY cookieid) AS pv3,  --分组内所有行
MIN(pv) OVER(PARTITION BY cookieid ORDER BY createtime ROWS BETWEEN 3 PRECEDING AND CURRENT ROW) AS pv4, --当前行+往前3行
MIN(pv) OVER(PARTITION BY cookieid ORDER BY createtime ROWS BETWEEN 3 PRECEDING AND 1 FOLLOWING) AS pv5,  --当前行+往前3行+往后1行
MIN(pv) OVER(PARTITION BY cookieid ORDER BY createtime ROWS BETWEEN CURRENT ROW AND UNBOUNDED FOLLOWING) AS pv6  ---当前行+往后所有行
FROM lxw1234;

cookieid createtime    pv    pv1    pv2    pv3    pv4    pv5    pv6
---------------------------------------------------------------------------
cookie1 2015-04-10    1    1    1    1    1    1    1
cookie1 2015-04-11    5    1    1    1    1    1    2
cookie1 2015-04-12    7    1    1    1    1    1    2
cookie1 2015-04-13    3    1    1    1    1    1    2
cookie1 2015-04-14    2    1    1    1    2    2    2
cookie1 2015-04-15    4    1    1    1    2    2    4
cookie1 2015-04-16    4    1    1    1    2    2    4

--MAX
SELECT cookieid,
createtime,
pv,
MAX(pv) OVER(PARTITION BY cookieid ORDER BY createtime) AS pv1, -- 默认为从起点到当前行
MAX(pv) OVER(PARTITION BY cookieid ORDER BY createtime ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) AS pv2, --从起点到当前行，结果同pv1
MAX(pv) OVER(PARTITION BY cookieid) AS pv3, --分组内所有行
```

MAX(pv) OVER(PARTITION BY cookieid ORDER BY createtime ROWS BETWEEN 3 PRECEDING AND CURRENT ROW) AS pv4, --当前行+往前3行
MAX(pv) OVER(PARTITION BY cookieid ORDER BY createtime ROWS BETWEEN 3 PRECEDING AND 1 FOLLOWING) AS pv5, --当前行+往前3行+往后1行
MAX(pv) OVER(PARTITION BY cookieid ORDER BY createtime ROWS BETWEEN CURRENT ROW AND UNBOUNDED FOLLOWING) AS pv6 ---当前行+往后所有行
FROM lxw1234;

cookieid	createtime	pv	pv1	pv2	pv3	pv4	pv5	pv6
cookie1	2015-04-10	1	1	1	7	1	5	7
cookie1	2015-04-11	5	5	5	7	5	7	7
cookie1	2015-04-12	7	7	7	7	7	7	7
cookie1	2015-04-13	3	7	7	7	7	7	4
cookie1	2015-04-14	2	7	7	7	7	7	4
cookie1	2015-04-15	4	7	7	7	7	7	4
cookie1	2015-04-16	4	7	7	7	4	4	4

Hive分析窗口函数(二)
NTILE,ROW_NUMBER,RANK,DENSE_RANK

本文中介绍前几个序列函数，NTILE,ROW_NUMBER,RANK,DENSE_RANK，下面会一一解释各自的用途。

注意： 序列函数不支持WINDOW子句。（什么是WINDOW子句，点此查看前面的文章）

数据准备：

```
CREATE EXTERNAL TABLE lxw1234 (
cookieid string,
createtime string,   --day
pv INT
) ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
stored as textfile location '/tmp/lxw11/';

DESC lxw1234;
cookieid                STRING
createtime              STRING
pv INT

hive> select * from lxw1234;
OK
cookie1 2015-04-10      1
cookie1 2015-04-11      5
cookie1 2015-04-12      7
cookie1 2015-04-13      3
cookie1 2015-04-14      2
cookie1 2015-04-15      4
cookie1 2015-04-16      4
cookie2 2015-04-10      2
cookie2 2015-04-11      3
cookie2 2015-04-12      5
cookie2 2015-04-13      6
cookie2 2015-04-14      3
```

cookie2 2015-04-15 9
cookie2 2015-04-16 7

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

```
SELECT
cookieid,
createtime,
pv,
NTILE(2) OVER(PARTITION BY cookieid ORDER BY createtime) AS rn1, --分组内将数据分成2片
NTILE(3) OVER(PARTITION BY cookieid ORDER BY createtime) AS rn2, --分组内将数据分成3片
NTILE(4) OVER(ORDER BY createtime) AS rn3 --将所有数据分成4片
FROM lxw1234
ORDER BY cookieid,createtime;
```

```
cookieid day pv rn1 rn2 rn3
---------------------------------------------------
cookie1 2015-04-10 1 1 1 1
cookie1 2015-04-11 5 1 1 1
cookie1 2015-04-12 7 1 1 2
cookie1 2015-04-13 3 1 2 2
cookie1 2015-04-14 2 2 2 3
cookie1 2015-04-15 4 2 3 3
cookie1 2015-04-16 4 2 3 4
cookie2 2015-04-10 2 1 1 1
cookie2 2015-04-11 3 1 1 1
cookie2 2015-04-12 5 1 1 2
cookie2 2015-04-13 6 1 2 2
cookie2 2015-04-14 3 2 2 3
cookie2 2015-04-15 9 2 3 4
cookie2 2015-04-16 7 2 3 4
```

–比如，统计一个cookie，pv数最多的前1/3的天

```
SELECT
cookieid,
createtime,
pv,
NTILE(3) OVER(PARTITION BY cookieid ORDER BY pv DESC) AS rn
FROM lxw1234;

--rn = 1 的记录，就是我们想要的结果
```

```
cookieid day pv rn
-----------------------------------
cookie1 2015-04-12 7 1
cookie1 2015-04-11 5 1
cookie1 2015-04-15 4 1
cookie1 2015-04-16 4 2
```

cookie1 2015-04-13 3 2
cookie1 2015-04-14 2 3
cookie1 2015-04-10 1 3
cookie2 2015-04-15 9 1
cookie2 2015-04-16 7 1
cookie2 2015-04-13 6 1
cookie2 2015-04-12 5 2
cookie2 2015-04-14 3 2
cookie2 2015-04-11 3 3
cookie2 2015-04-10 2 3

ROW_NUMBER

ROW_NUMBER() –从1开始，按照顺序，生成分组内记录的序列

–比如，按照pv降序排列，生成分组内每天的pv名次

ROW_NUMBER() 的应用场景非常多，再比如，获取分组内排序第一的记录;获取一个session中的第一条refer等。

```
SELECT
cookieid,
createtime,
pv,
ROW_NUMBER() OVER(PARTITION BY cookieid ORDER BY pv desc) AS rn
FROM lxw1234;
```

cookieid day pv rn
---
cookie1 2015-04-12 7 1
cookie1 2015-04-11 5 2
cookie1 2015-04-15 4 3
cookie1 2015-04-16 4 4
cookie1 2015-04-13 3 5
cookie1 2015-04-14 2 6
cookie1 2015-04-10 1 7
cookie2 2015-04-15 9 1
cookie2 2015-04-16 7 2
cookie2 2015-04-13 6 3
cookie2 2015-04-12 5 4
cookie2 2015-04-14 3 5
cookie2 2015-04-11 3 6
cookie2 2015-04-10 2 7

RANK 和 DENSE_RANK

—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

```
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。
```

Hive分析窗口函数(三) CUME_DIST,PERCENT_RANK

这两个序列分析函数不是很常用，这里也介绍一下。

注意： 序列函数不支持WINDOW子句。（什么是WINDOW子句，点此查看前面的文章）

数据准备：

```
CREATE EXTERNAL TABLE lxw1234 (
dept STRING,
userid string,
sal INT
) ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
stored as textfile location '/tmp/lxw11/';



hive> select * from lxw1234;
OK
d1      user1   1000
d1      user2   2000
d1      user3   3000
d2      user4   4000
d2      user5   5000

CUME_DIST

--CUME_DIST 小于等于当前值的行数/分组内总行数

--比如，统计小于等于当前薪水的人数，所占总人数的比例

    SELECT
    dept,
    userid,
    sal,
    CUME_DIST() OVER(ORDER BY sal) AS rn1,
    CUME_DIST() OVER(PARTITION BY dept ORDER BY sal) AS rn2
    FROM lxw1234;
```

```
dept    userid   sal   rn1      rn2
-------------------------------------------
d1      user1   1000    0.2     0.3333333333333333
d1      user2   2000    0.4     0.6666666666666666
d1      user3   3000    0.6     1.0
d2      user4   4000    0.8     0.5
d2      user5   5000    1.0     1.0
```

rn1: 没有partition,所有数据均为1组，总行数为5，
第一行：小于等于1000的行数为1，因此，1/5=0.2
第三行：小于等于3000的行数为3，因此，3/5=0.6
rn2: 按照部门分组，dpet=d1的行数为3,
第二行：小于等于2000的行数为2，因此，2/3=0.6666666666666666

PERCENT_RANK

–PERCENT_RANK 分组内当前行的RANK值-1/分组内总行数-1

应用场景不了解，可能在一些特殊算法的实现中可以用到吧。

```
SELECT
dept,
userid,
sal,
PERCENT_RANK() OVER(ORDER BY sal) AS rn1,   --分组内
RANK() OVER(ORDER BY sal) AS rn11,          --分组内RANK值
SUM(1) OVER(PARTITION BY NULL) AS rn12,     --分组内总行数
PERCENT_RANK() OVER(PARTITION BY dept ORDER BY sal) AS rn2
FROM lxw1234;

dept    userid  sal     rn1    rn11     rn12    rn2
-----------------------------------------------------
d1      user1   1000    0.0     1       5       0.0
d1      user2   2000    0.25    2       5       0.5
d1      user3   3000    0.5     3       5       1.0
d2      user4   4000    0.75    4       5       0.0
d2      user5   5000    1.0     5       5       1.0
```

rn1: rn1 = (rn11-1) / (rn12-1)
第一行,(1-1)/(5-1)=0/4=0
第二行,(2-1)/(5-1)=1/4=0.25
第四行,(4-1)/(5-1)=3/4=0.75
rn2: 按照dept分组，
dept=d1的总行数为3
第一行，(1-1)/(3-1)=0
第三行，(3-1)/(3-1)=1

Hive分析窗口函数(四)
LAG,LEAD,FIRST_VALUE,LAST_VALUE

继续学习这四个分析函数。

注意： 这几个函数不支持WINDOW子句。（什么是WINDOW子句，点此查看前面的文章）

数据准备：

```
CREATE EXTERNAL TABLE lxw1234 (
cookieid string,
createtime string,  --页面访问时间
url STRING      --被访问页面
) ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
stored as textfile location '/tmp/lxw11/';
```

```
hive> select * from lxw1234;
OK
cookie1 2015-04-10 10:00:02     url2
cookie1 2015-04-10 10:00:00     url1
cookie1 2015-04-10 10:03:04     1url3
cookie1 2015-04-10 10:50:05     url6
cookie1 2015-04-10 11:00:00     url7
cookie1 2015-04-10 10:10:00     url4
cookie1 2015-04-10 10:50:01     url5
cookie2 2015-04-10 10:00:02     url22
cookie2 2015-04-10 10:00:00     url11
cookie2 2015-04-10 10:03:04     1url33
cookie2 2015-04-10 10:50:05     url66
cookie2 2015-04-10 11:00:00     url77
cookie2 2015-04-10 10:10:00     url44
cookie2 2015-04-10 10:50:01     url55
```

LAG

LAG(col,n,DEFAULT) 用于统计窗口内往上第n行值

第一个参数为列名，第二个参数为往上第n行（可选，默认为1），第三个参数为默认值（当往上第n行为NULL时候，取默认值，如不指定，则为NULL）

```
SELECT cookieid,
createtime,
url,
ROW_NUMBER() OVER(PARTITION BY cookieid ORDER BY createtime) AS rn,
LAG(createtime,1,'1970-01-01 00:00:00') OVER(PARTITION BY cookieid ORDER BY createtime) AS last_1_time,
LAG(createtime,2) OVER(PARTITION BY cookieid ORDER BY createtime) AS last_2_time
FROM lxw1234;
```

```
cookieid createtime          url    rn      last_1_time          last_2_time
---------------------------------------------------------------------------------------------
cookie1 2015-04-10 10:00:00    url1    1       1970-01-01 00:00:00     NULL
cookie1 2015-04-10 10:00:02    url2    2       2015-04-10 10:00:00     NULL
cookie1 2015-04-10 10:03:04    1url3   3       2015-04-10 10:00:02     2015-04-10 10:00:00
cookie1 2015-04-10 10:10:00    url4    4       2015-04-10 10:03:04     2015-04-10 10:00:02
cookie1 2015-04-10 10:50:01    url5    5       2015-04-10 10:10:00     2015-04-10 10:03:04
cookie1 2015-04-10 10:50:05    url6    6       2015-04-10 10:50:01     2015-04-10 10:10:00
cookie1 2015-04-10 11:00:00    url7    7       2015-04-10 10:50:05     2015-04-10 10:50:01
cookie2 2015-04-10 10:00:00    url11   1       1970-01-01 00:00:00     NULL
cookie2 2015-04-10 10:00:02    url22   2       2015-04-10 10:00:00     NULL
```

cookie2 2015-04-10 10:03:04 1url33 3 2015-04-10 10:00:02 2015-04-10 10:00:00
cookie2 2015-04-10 10:10:00 url44 4 2015-04-10 10:03:04 2015-04-10 10:00:02
cookie2 2015-04-10 10:50:01 url55 5 2015-04-10 10:10:00 2015-04-10 10:03:04
cookie2 2015-04-10 10:50:05 url66 6 2015-04-10 10:50:01 2015-04-10 10:10:00
cookie2 2015-04-10 11:00:00 url77 7 2015-04-10 10:50:05 2015-04-10 10:50:01

last_1_time: 指定了往上第1行的值，default为'1970-01-01 00:00:00'
cookie1第一行，往上1行为NULL,因此取默认值 1970-01-01 00:00:00
cookie1第三行，往上1行值为第二行值，2015-04-10 10:00:02
cookie1第六行，往上1行值为第五行值，2015-04-10 10:50:01
last_2_time: 指定了往上第2行的值，为指定默认值
cookie1第一行，往上2行为NULL
cookie1第二行，往上2行为NULL
cookie1第四行，往上2行为第二行值，2015-04-10 10:00:02
cookie1第七行，往上2行为第五行值，2015-04-10 10:50:01

LEAD

与LAG相反

LEAD(col,n,DEFAULT) 用于统计窗口内往下第n行值

第一个参数为列名，第二个参数为往下第n行（可选，默认为1），第三个参数为默认值（当往下第n行为NULL时候，取默认值，如不指定，则为NULL）

```
SELECT cookieid,
createtime,
url,
ROW_NUMBER() OVER(PARTITION BY cookieid ORDER BY createtime) AS rn,
LEAD(createtime,1,'1970-01-01 00:00:00') OVER(PARTITION BY cookieid ORDER BY createtime) AS next_1_time,
LEAD(createtime,2) OVER(PARTITION BY cookieid ORDER BY createtime) AS next_2_time
FROM lxw1234;
```

cookieid createtime url rn next_1_time next_2_time
---
cookie1 2015-04-10 10:00:00 url1 1 2015-04-10 10:00:02 2015-04-10 10:03:04
cookie1 2015-04-10 10:00:02 url2 2 2015-04-10 10:03:04 2015-04-10 10:10:00
cookie1 2015-04-10 10:03:04 1url3 3 2015-04-10 10:10:00 2015-04-10 10:50:01
cookie1 2015-04-10 10:10:00 url4 4 2015-04-10 10:50:01 2015-04-10 10:50:05
cookie1 2015-04-10 10:50:01 url5 5 2015-04-10 10:50:05 2015-04-10 11:00:00
cookie1 2015-04-10 10:50:05 url6 6 2015-04-10 11:00:00 NULL
cookie1 2015-04-10 11:00:00 url7 7 1970-01-01 00:00:00 NULL
cookie2 2015-04-10 10:00:00 url11 1 2015-04-10 10:00:02 2015-04-10 10:03:04
cookie2 2015-04-10 10:00:02 url22 2 2015-04-10 10:03:04 2015-04-10 10:10:00
cookie2 2015-04-10 10:03:04 1url33 3 2015-04-10 10:10:00 2015-04-10 10:50:01
cookie2 2015-04-10 10:10:00 url44 4 2015-04-10 10:50:01 2015-04-10 10:50:05
cookie2 2015-04-10 10:50:01 url55 5 2015-04-10 10:50:05 2015-04-10 11:00:00
cookie2 2015-04-10 10:50:05 url66 6 2015-04-10 11:00:00 NULL
cookie2 2015-04-10 11:00:00 url77 7 1970-01-01 00:00:00 NULL

--逻辑与LAG一样，只不过LAG是往上，LEAD是往下。

FIRST_VALUE

取分组内排序后，截止到当前行，第一个值

```
SELECT cookieid,
createtime,
url,
ROW_NUMBER() OVER(PARTITION BY cookieid ORDER BY createtime) AS rn,
FIRST_VALUE(url) OVER(PARTITION BY cookieid ORDER BY createtime) AS first1
FROM lxw1234;
```

```
cookieid  createtime          url     rn      first1
---------------------------------------------------------
cookie1 2015-04-10 10:00:00     url1    1       url1
cookie1 2015-04-10 10:00:02     url2    2       url1
cookie1 2015-04-10 10:03:04     1url3   3       url1
cookie1 2015-04-10 10:10:00     url4    4       url1
cookie1 2015-04-10 10:50:01     url5    5       url1
cookie1 2015-04-10 10:50:05     url6    6       url1
cookie1 2015-04-10 11:00:00     url7    7       url1
cookie2 2015-04-10 10:00:00     url11   1       url11
cookie2 2015-04-10 10:00:02     url22   2       url11
cookie2 2015-04-10 10:03:04     1url33  3       url11
cookie2 2015-04-10 10:10:00     url44   4       url11
cookie2 2015-04-10 10:50:01     url55   5       url11
cookie2 2015-04-10 10:50:05     url66   6       url11
cookie2 2015-04-10 11:00:00     url77   7       url11
```

LAST_VALUE

取分组内排序后，截止到当前行，最后一个值

```
SELECT cookieid,
createtime,
url,
ROW_NUMBER() OVER(PARTITION BY cookieid ORDER BY createtime) AS rn,
LAST_VALUE(url) OVER(PARTITION BY cookieid ORDER BY createtime) AS last1
FROM lxw1234;
```

```
cookieid  createtime          url     rn      last1
---------------------------------------------------------------
cookie1 2015-04-10 10:00:00     url1    1       url1
cookie1 2015-04-10 10:00:02     url2    2       url2
cookie1 2015-04-10 10:03:04     1url3   3       1url3
cookie1 2015-04-10 10:10:00     url4    4       url4
cookie1 2015-04-10 10:50:01     url5    5       url5
cookie1 2015-04-10 10:50:05     url6    6       url6
cookie1 2015-04-10 11:00:00     url7    7       url7
cookie2 2015-04-10 10:00:00     url11   1       url11
cookie2 2015-04-10 10:00:02     url22   2       url22
cookie2 2015-04-10 10:03:04     1url33  3       1url33
cookie2 2015-04-10 10:10:00     url44   4       url44
cookie2 2015-04-10 10:50:01     url55   5       url55
cookie2 2015-04-10 10:50:05     url66   6       url66
cookie2 2015-04-10 11:00:00     url77   7       url77
```

如果不指定ORDER BY，则默认按照记录在文件中的偏移量进行排序，会出现错误的结果

```
SELECT cookieid,
createtime,
url,
FIRST_VALUE(url) OVER(PARTITION BY cookieid) AS first2
FROM lxw1234;

cookieid  createtime          url     first2
---------------------------------------------
cookie1 2015-04-10 10:00:02     url2    url2
cookie1 2015-04-10 10:00:00     url1    url2
cookie1 2015-04-10 10:03:04     1url3   url2
cookie1 2015-04-10 10:50:05     url6    url2
cookie1 2015-04-10 11:00:00     url7    url2
cookie1 2015-04-10 10:10:00     url4    url2
cookie1 2015-04-10 10:50:01     url5    url2
cookie2 2015-04-10 10:00:02     url22   url22
cookie2 2015-04-10 10:00:00     url11   url22
cookie2 2015-04-10 10:03:04     1url33  url22
cookie2 2015-04-10 10:50:05     url66   url22
cookie2 2015-04-10 11:00:00     url77   url22
cookie2 2015-04-10 10:10:00     url44   url22
cookie2 2015-04-10 10:50:01     url55   url22

SELECT cookieid,
createtime,
url,
LAST_VALUE(url) OVER(PARTITION BY cookieid) AS last2
FROM lxw1234;

cookieid  createtime          url     last2
---------------------------------------------
cookie1 2015-04-10 10:00:02     url2    url5
cookie1 2015-04-10 10:00:00     url1    url5
cookie1 2015-04-10 10:03:04     1url3   url5
cookie1 2015-04-10 10:50:05     url6    url5
cookie1 2015-04-10 11:00:00     url7    url5
cookie1 2015-04-10 10:10:00     url4    url5
cookie1 2015-04-10 10:50:01     url5    url5
cookie2 2015-04-10 10:00:02     url22   url55
cookie2 2015-04-10 10:00:00     url11   url55
cookie2 2015-04-10 10:03:04     1url33  url55
cookie2 2015-04-10 10:50:05     url66   url55
cookie2 2015-04-10 11:00:00     url77   url55
cookie2 2015-04-10 10:10:00     url44   url55
cookie2 2015-04-10 10:50:01     url55   url55
```

如果想要取分组内排序后最后一个值，则需要变通一下：

```
SELECT cookieid,
createtime,
url,
ROW_NUMBER() OVER(PARTITION BY cookieid ORDER BY createtime) AS rn,
LAST_VALUE(url) OVER(PARTITION BY cookieid ORDER BY createtime) AS last1,
FIRST_VALUE(url) OVER(PARTITION BY cookieid ORDER BY createtime DESC) AS last2
FROM lxw1234
```

```
ORDER BY cookieid,createtime;

cookieid  createtime          url    rn    last1   last2
---------------------------------------------------------------
cookie1 2015-04-10 10:00:00    url1   1     url1    url7
cookie1 2015-04-10 10:00:02    url2   2     url2    url7
cookie1 2015-04-10 10:03:04    1url3  3     1url3   url7
cookie1 2015-04-10 10:10:00    url4   4     url4    url7
cookie1 2015-04-10 10:50:01    url5   5     url5    url7
cookie1 2015-04-10 10:50:05    url6   6     url6    url7
cookie1 2015-04-10 11:00:00    url7   7     url7    url7
cookie2 2015-04-10 10:00:00    url11  1     url11   url77
cookie2 2015-04-10 10:00:02    url22  2     url22   url77
cookie2 2015-04-10 10:03:04    1url33 3     1url33  url77
cookie2 2015-04-10 10:10:00    url44  4     url44   url77
cookie2 2015-04-10 10:50:01    url55  5     url55   url77
cookie2 2015-04-10 10:50:05    url66  6     url66   url77
cookie2 2015-04-10 11:00:00    url77  7     url77   url77
```

提示：在使用分析函数的过程中，要特别注意ORDER BY子句，用的不恰当，统计出的结果就不是你所期望的。

Hive分析窗口函数(五) GROUPING SETS,GROUPING__ID,CUBE,ROLLUP

GROUPING SETS,GROUPING__ID,CUBE,ROLLUP

这几个分析函数通常用于OLAP中，不能累加，而且需要根据不同维度上钻和下钻的指标统计，比如，分小时、天、月的UV数。

数据准备：

```
CREATE EXTERNAL TABLE lxw1234 (
month STRING,
day STRING,
cookieid STRING
) ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
stored as textfile location '/tmp/lxw11/';


hive> select * from lxw1234;
OK
2015-03 2015-03-10      cookie1
2015-03 2015-03-10      cookie5
2015-03 2015-03-12      cookie7
2015-04 2015-04-12      cookie3
2015-04 2015-04-13      cookie2
2015-04 2015-04-13      cookie4
2015-04 2015-04-16      cookie4
2015-03 2015-03-10      cookie2
2015-03 2015-03-10      cookie3
2015-04 2015-04-12      cookie5
2015-04 2015-04-13      cookie6
2015-04 2015-04-15      cookie3
```

2015-04 2015-04-15 cookie2
2015-04 2015-04-16 cookie1

GROUPING SETS

在一个GROUP BY查询中，根据不同的维度组合进行聚合，等价于将不同维度的GROUP BY结果集进行UNION ALL

```
SELECT
month,
day,
COUNT(DISTINCT cookieid) AS uv,
GROUPING__ID
FROM lxw1234
GROUP BY month,day
GROUPING SETS (month,day)
ORDER BY GROUPING__ID;

month     day         uv     GROUPING__ID
-------------------------------------------------
2015-03   NULL          5      1
2015-04   NULL          6      1
NULL      2015-03-10    4      2
NULL      2015-03-12    1      2
NULL      2015-04-12    2      2
NULL      2015-04-13    3      2
NULL      2015-04-15    2      2
NULL      2015-04-16    2      2


等价于
SELECT month,NULL,COUNT(DISTINCT cookieid) AS uv,1 AS GROUPING__ID FROM lxw1234 GROUP BY month
UNION ALL
SELECT NULL,day,COUNT(DISTINCT cookieid) AS uv,2 AS GROUPING__ID FROM lxw1234 GROUP BY day
```

再如：

```
SELECT
month,
day,
COUNT(DISTINCT cookieid) AS uv,
GROUPING__ID
FROM lxw1234
GROUP BY month,day
GROUPING SETS (month,day,(month,day))
ORDER BY GROUPING__ID;

month       day         uv     GROUPING__ID
-------------------------------------------------
2015-03     NULL          5      1
2015-04     NULL          6      1
NULL        2015-03-10    4      2
NULL        2015-03-12    1      2
NULL        2015-04-12    2      2
NULL        2015-04-13    3      2
NULL        2015-04-15    2      2
NULL        2015-04-16    2      2
```

```
2015-03     2015-03-10    4     3
2015-03     2015-03-12    1     3
2015-04     2015-04-12    2     3
2015-04     2015-04-13    3     3
2015-04     2015-04-15    2     3
2015-04     2015-04-16    2     3
```

```
等价于
SELECT month,NULL,COUNT(DISTINCT cookieid) AS uv,1 AS GROUPING__ID FROM lxw1234 GROUP BY month
UNION ALL
SELECT NULL,day,COUNT(DISTINCT cookieid) AS uv,2 AS GROUPING__ID FROM lxw1234 GROUP BY day
UNION ALL
SELECT month,day,COUNT(DISTINCT cookieid) AS uv,3 AS GROUPING__ID FROM lxw1234 GROUP BY month,day
```

其中的 **GROUPING__ID**，表示结果属于哪一个分组集合。

CUBE

根据GROUP BY的维度的所有组合进行聚合。

```
SELECT
month,
day,
COUNT(DISTINCT cookieid) AS uv,
GROUPING__ID
FROM lxw1234
GROUP BY month,day
WITH CUBE
ORDER BY GROUPING__ID;
```

```
month         day           uv    GROUPING__ID
-----------------------------------------------
NULL          NULL          7     0
2015-03       NULL          5     1
2015-04       NULL          6     1
NULL          2015-04-12    2     2
NULL          2015-04-13    3     2
NULL          2015-04-15    2     2
NULL          2015-04-16    2     2
NULL          2015-03-10    4     2
NULL          2015-03-12    1     2
2015-03       2015-03-10    4     3
2015-03       2015-03-12    1     3
2015-04       2015-04-16    2     3
2015-04       2015-04-12    2     3
2015-04       2015-04-13    3     3
2015-04       2015-04-15    2     3
```

```
等价于
SELECT NULL,NULL,COUNT(DISTINCT cookieid) AS uv,0 AS GROUPING__ID FROM lxw1234
UNION ALL
SELECT month,NULL,COUNT(DISTINCT cookieid) AS uv,1 AS GROUPING__ID FROM lxw1234 GROUP BY month
```

```
UNION ALL
SELECT NULL,day,COUNT(DISTINCT cookieid) AS uv,2 AS GROUPING__ID FROM lxw1234 GROUP BY day
UNION ALL
SELECT month,day,COUNT(DISTINCT cookieid) AS uv,3 AS GROUPING__ID FROM lxw1234 GROUP BY month,day
```

ROLLUP

是CUBE的子集，以最左侧的维度为主，从该维度进行层级聚合。

```
比如，以month维度进行层级聚合：
SELECT
month,
day,
COUNT(DISTINCT cookieid) AS uv,
GROUPING__ID
FROM lxw1234
GROUP BY month,day
WITH ROLLUP
ORDER BY GROUPING__ID;

month       day          uv    GROUPING__ID
-----------------------------------------------------
NULL          NULL         7     0
2015-03       NULL         5     1
2015-04       NULL         6     1
2015-03       2015-03-10   4     3
2015-03       2015-03-12   1     3
2015-04       2015-04-12   2     3
2015-04       2015-04-13   3     3
2015-04       2015-04-15   2     3
2015-04       2015-04-16   2     3

可以实现这样的上钻过程：
月天的UV->月的UV->总UV

--把month和day调换顺序，则以day维度进行层级聚合：

SELECT
day,
month,
COUNT(DISTINCT cookieid) AS uv,
GROUPING__ID
FROM lxw1234
GROUP BY day,month
WITH ROLLUP
ORDER BY GROUPING__ID;


day      month          uv    GROUPING__ID
-----------------------------------------------------------
NULL         NULL           7     0
2015-04-13   NULL           3     1
2015-03-12   NULL           1     1
2015-04-15   NULL           2     1
2015-03-10   NULL           4     1
2015-04-16   NULL           2     1
```

2015-04-12	NULL	2	1
2015-04-12	2015-04	2	3
2015-03-10	2015-03	4	3
2015-03-12	2015-03	1	3
2015-04-13	2015-04	3	3
2015-04-15	2015-04	2	3
2015-04-16	2015-04	2	3

可以实现这样的上钻过程：
天月的UV->天的UV->总UV
（这里，根据天和月进行聚合，和根据天聚合结果一样，因为有父子关系，如果是其他维度组合的话，就会不一样）

这种函数，需要结合实际场景和数据去使用和研究，只看说明的话，很难理解。

官网的介绍：
https://cwiki.apache.org/confluence/display/Hive/Enhanced+Aggregation%2C+Cube%2C+Grouping+and+Rollup