第三节——HiveSQL窗口函数

课堂目标

1、掌握sum()、avg()用于累计计算的窗口函数

2、掌握row_number()、rank()用于排序的窗口函数

3、掌握ntile()用于分组查询的窗口函数

4、掌握lag()、lead()偏移分析窗口函数

累计计算窗口函数(25min)

1、sum(...) over(.....)

大家在做报表的时候，经常会遇到计算截止某月的累计数值，通常在EXCEL里可以通过函数来实现。

月份	交易额	累计交易额
1	10	10
2	10	20
3	10	30
4	10	40
5	10	50
6	10	60
7	10	70
8	10	80
9	10	90
10	10	100
11	10	110
12	10	120

那么在hive里，该如何实现这种累计数值的计算呢？——利用窗口函数！

- **2018年每月的支付总额和当年累积支付总额：**

user_trade列名	举例
user_name	Amy, Dennis
piece	购买数量
price	价格
pay_amount	支付金额
goods_category	food, clothes, book, computer, electronics, shoes
pay_time	1323308943，时间戳
dt	partition，‘yyyy-mm-dd’

```
--2018年每月的支付总额和当年累积支付总额--
SELECT a.month,
       a.pay_amount,
       sum(a.pay_amount) over(order by a.month)
FROM
      (SELECT month(dt) month,
              sum(pay_amount) pay_amount
      FROM user_trade
      WHERE year(dt)=2018
      GROUP BY month(dt))a;
```

```
1       317697.2        317697.2
2       2214537.1       2532234.3000000003
3       3108435.9       5640670.2
4       2717482.5999999996      8358152.8
5       2723670.1       1.10818229E7
6       3808041.3       1.48898642E7
7       5426222.3       2.03160865E7
8       2749747.0       2.30658335E7
9       891197.0        2.39570305E7
10      1510374.2999999998      2.54674048E7
11      2307257.4       2.77746622E7
12      1759487.2       2.95341494E7
Time taken: 50.176 seconds, Fetched: 12 row(s)
```

- **2017-2018年每月的支付总额和当年累积支付总额：**

```sql
--2017-2018年每月的支付总额和当年累积支付总额--
SELECT a.year,
       a.month,
       a.pay_amount,
       sum(a.pay_amount) over(partition by a.year order by a.month)
FROM
      (SELECT year(dt) year,
              month(dt) month,
              sum(pay_amount) pay_amount
      FROM user_trade
      WHERE year(dt) in (2017,2018)
      GROUP BY year(dt),
               month(dt))a;
```

```
2017    1       241755.69999999998      241755.69999999998
2017    2       2582410.5999999996      2824166.3
2017    3       1977644.7000000002      4801811.0
2017    4       1162322.7999999998      5964133.8
2017    5       3038255.2       9002389.0
2017    6       2773154.4       1.17755434E7
2017    7       1677527.2999999998      1.34530707E7
2017    8       2135214.4       1.55882851E7
2017    9       1355307.3000000003      1.69435924E7
2017    10      1380672.7       1.8324265099999998E7
2017    11      2428753.9       2.0753018999999996E7
2017    12      3580954.6       2.4333973599999998E7
2018    1       317697.2        317697.2
2018    2       2214537.1       2532234.3000000003
2018    3       3108435.9       5640670.2
2018    4       2717482.5999999996      8358152.8
2018    5       2723670.1       1.10818229E7
2018    6       3808041.3       1.48898642E7
2018    7       5426222.3       2.03160865E7
2018    8       2749747.0       2.30658335E7
2018    9       891197.0        2.39570305E7
2018    10      1510374.2999999998      2.54674048E7
2018    11      2307257.4       2.77746622E7
2018    12      1759487.2       2.95341494E7
Time taken: 38.016 seconds, Fetched: 24 row(s)
```

说明：

1、partition by起到分组的作用

2、order by 按照什么顺序进行累加，升序ASC、降序DESC，默认升序

常见错误——分组没有限制正确：

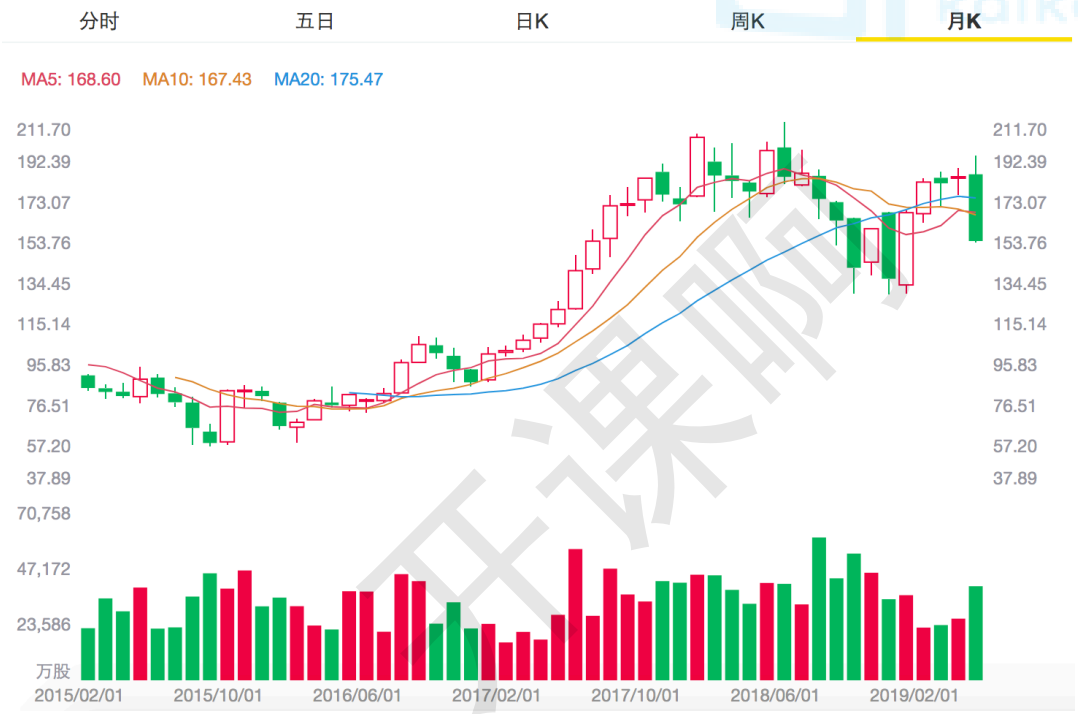
```
SELECT a.year,
       a.month,
       a.pay_amount,
       sum(a.pay_amount) over(partition by a.year,a.month order by
a.month)
FROM
      (SELECT year(dt) year,
              month(dt) month,
              sum(pay_amount) pay_amount
      FROM user_trade
      WHERE year(dt) in (2017,2018)
      GROUP BY year(dt),
               month(dt))a;
```

```
2017    1       241755.69999999998      241755.69999999998
2017    2       2582410.5999999996      2582410.5999999996
2017    3       1977644.7000000002      1977644.7000000002
2017    4       1162322.7999999998      1162322.7999999998
2017    5       3038255.2       3038255.2
2017    6       2773154.4       2773154.4
2017    7       1677527.2999999998      1677527.2999999998
2017    8       2135214.4       2135214.4
2017    9       1355307.3000000003      1355307.3000000003
2017    10      1380672.7       1380672.7
2017    11      2428753.9       2428753.9
2017    12      3580954.6       3580954.6
2018    1       317697.2        317697.2
2018    2       2214537.1       2214537.1
2018    3       3108435.9       3108435.9
2018    4       2717482.5999999996      2717482.5999999996
2018    5       2723670.1       2723670.1
2018    6       3808041.3       3808041.3
2018    7       5426222.3       5426222.3
2018    8       2749747.0       2749747.0
2018    9       891197.0        891197.0
2018    10      1510374.2999999998      1510374.2999999998
2018    11      2307257.4       2307257.4
2018    12      1759487.2       1759487.2
Time taken: 41.126 seconds, Fetched: 24 row(s)
```

最终导致每月的数据各为一组，分组累计求和后和自己的数值一样，没有达到目标要求。所以，如何正确的分组非常关键。

2、avg(...) over(.....)

大家看股票的时候，经常会看到这种K线图吧，里面经常用到的就是7日、30日移动平均的趋势图，那如何使用窗口函数来计算移动平均值呢?



- **2018年每个月的近三月移动平均支付金额:**

定义

若依次得到测定值 $(x_1, x_2, x_3, ..., x_n)$ 时，按顺序取一定个数所做的全部算术平均值。 例如 $\frac{(x_1+x_2+x_3)}{3}, \frac{(x_2+x_3+x_4)}{3}, \frac{(x_3+x_4+x_5)}{3}, \frac{(x_4+x_5+x_6)}{3}, ...$ 等是移动平均值。 [1]

```
--2018年每个月的近三月移动平均支付金额--
SELECT a.month,
       a.pay_amount,
       avg(a.pay_amount) over(order by a.month rows between 2 preceding
and current row)
FROM
      (SELECT month(dt) month,
              sum(pay_amount) pay_amount
      FROM user_trade
      WHERE year(dt)=2018
      GROUP BY month(dt))a;
```

```
1       317697.2        317697.2
2       2214537.1       1266117.1500000001
3       3108435.9       1880223.4000000001
4       2717482.5999999996      2680151.8666666667
5       2723670.1       2849862.8666666667
6       3808041.3       3083064.6666666665
7       5426222.3       3985977.9
8       2749747.0       3994670.1999999997
9       891197.0        3022388.766666667
10      1510374.2999999998      1717106.1000000003
11      2307257.4       1569609.5666666664
12      1759487.2       1859039.6333333328
Time taken: 56.042 seconds, Fetched: 12 row(s)
```

说明：

我们用rows between 2 preceding and current row来限制计算移动平均的范围，本语句含义是包含本行及前两行，这个就是我们题目中要求的近三月的写法。

3、语法总结

sum(...**A**...) over(partition by ...**B**... order by ...**C**... rows between ...**D1**... and ...**D2**...)

avg(...**A**...) over(partition by ...**B**... order by ...**C**... rows between ...**D1**... and ...**D2**...)

A：需要被加工的字段名称

B：分组的字段名称

C：排序的字段名称

D：计算的行数范围

- **rows between unbounded preceding and current row**——包括本行和之前所有的行
- **rows between current row and unbounded following**——包括本行和之后所有的行
- **rows between 3 preceding and current row**——包括本行以内和前三行
- **rows between 3 preceding and 1 following**——从前三行到下一行(5行)

拓展：

max(.....) over(partition by ..... order by ..... rows between ..... and .....)

min(.....) over(partition by ..... order by ..... rows between ..... and .....)

分区排序窗口函数(25min)

1、row_number() over(.....)、rank() over(.....)、dense_rank() over(.....)

这三个函数的作用都是返回相应规则的排序序号

row_number() over(partition by ...**A**... order by ...**B**...)

rank() over(partition by ...**A**... order by ...**B**...)

dense_rank() over(partition by ...**A**... order by ...**B**...)

A：分组的字段名称

B：排序的字段名称

注意：row_number()的这个括号内是不加任何字段名称的，rank() 和dense_rank() 同理。

- **row_number**：它会为查询出来的每一行记录生成一个序号，依次排序且不会重复。
- **rank&dense_rank**：如果使用rank函数来生成序号，over子句中排序字段值相同的序号是一样的，后面字段值不相同的序号将跳过相同的排名号排下一个，也就是相关行之前的排名数加一。dense_rank函数在生成序号时是连续的，而rank函数生成的序号有可能不连续。dense_rank函数出现相同排名时，将不跳过相同排名号，rank值紧接上一次的rank值。在各个分组内，rank()是跳跃排序，有两个第一名时接下来就是第三名，dense_rank()是连续排序，有两个第一名时仍然跟着第二名。

实例对比：

user_trade列名	举例
user_name	Amy, Dennis
piece	购买数量
price	价格
pay_amount	支付金额
goods_category	food, clothes, book, computer, electronics, shoes
pay_time	1323308943，时间戳
dt	partition，‘yyyy-mm-dd’

2019年1月，用户购买商品品类数量的排名：

```sql
--2019年1月，用户购买商品品类数量的排名--
SELECT user_name,
       count(distinct goods_category),
       row_number() over(order by count(distinct goods_category)),
       rank() over(order by count(distinct goods_category)),
       dense_rank() over(order by count(distinct goods_category))
FROM user_trade
WHERE substr(dt,1,7)='2019-01'
GROUP BY user_name;
```

```
Wheeler 1       1       1       1
Ward    1       2       1       1
Rupert  1       3       1       1
Peterson        1       4       1       1
Parker  1       5       1       1
Mitchell        1       6       1       1
Jill    1       7       1       1
Janet   1       8       1       1
Frank   1       9       1       1
Fiona   1       10      1       1
Cloris  1       11      1       1
Cherry  1       12      1       1
Cathy   1       13      1       1
Cameron 1       14      1       1
Amanda  1       15      1       1
Ingrid  2       16      16      2
Christy 2       17      16      2
Catherine       2       18      16      2
Angelia 2       19      16      2
Payne   2       20      16      2
Time taken: 49.336 seconds, Fetched: 20 row(s)
```

user_name	goods_num	row_number	rank	dense_rank
Wheeler	1	1	1	1
Ward	1	2	1	1
Rupert	1	3	1	1
Peterson	1	4	1	1
Parker	1	5	1	1
Mitchell	1	6	1	1
Jill	1	7	1	1
Janet	1	8	1	1
Frank	1	9	1	1
Fiona	1	10	1	1
Cloris	1	11	1	1
Cherry	1	12	1	1
Cathy	1	13	1	1
Cameron	1	14	1	1
Amanda	1	15	1	1
Ingrid	2	16	16	2
Christy	2	17	16	2
Catherine	2	18	16	2
Angelia	2	19	16	2
Payne	2	20	16	2

练习：

- 选出2019年支付金额排名在第10、20、30名的用户:

```sql
--选出2019年支付金额排名在第10、20、30名的用户--
SELECT a.user_name,
       a.pay_amount,
       a.rank
FROM
      (SELECT user_name,
              sum(pay_amount) pay_amount,
              rank() over(order by sum(pay_amount) desc) rank
      FROM user_trade
      WHERE year(dt)=2019
      GROUP BY user_name)a
WHERE a.rank in (10,20,30);
```

```
James   286638.0        10
Marshall        144430.0        20
Nolan   30996.0 30
Time taken: 52.166 seconds, Fetched: 3 row(s)
```

分组排序窗口函数(10min)

1、ntile(n) over(.....)

ntile(**n**) over(partition by ...**A**... order by ...**B**...)

n：切分的片数

A：分组的字段名称

B：排序的字段名称

- 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 ..... ORDER BY ..... 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

将2019年1月的支付用户，按照支付金额分成5组：

```sql
--将2019年1月的支付用户，按照支付金额分成5组--
SELECT user_name,
       sum(pay_amount) pay_amount,
       ntile(5) over(order by sum(pay_amount) desc) level
FROM user_trade
WHERE substr(dt,1,7)='2019-01'
GROUP BY user_name;
```

```
Angelia 677710.0        1
Cherry  359964.0        1
Christy 217772.5        1
Rupert  62680.8 1
Janet   56481.6 2
Ward    26174.4 2
Frank   22041.6 2
Wheeler 21600.0 2
Jill    14800.0 3
Payne   12247.3 3
Peterson        2066.4  3
Catherine       1320.9  3
Cameron 1219.3  4
Cathy   1000.0  4
Amanda  827.7   4
Parker  631.9   4
Mitchell        471.7   5
Ingrid  465.2   5
Cloris  418.3   5
Fiona   110.0   5
Time taken: 51.71 seconds, Fetched: 20 row(s)
```

练习：

- 选出**2019**年退款金额排名前**10%**的用户：

user_refund列名	举例
user_name	Amy, Dennis
refund_piece	退款件数
refund_amount	退款金额
refund_time	1323308943，时间戳
dt	partition，‘yyyy-mm-dd’

```sql
--选出2019年退款金额排名前10%的用户--
SELECT a.user_name,
       a.refund_amount,
       a.level
FROM
      (SELECT user_name,
              sum(refund_amount) refund_amount,
              ntile(10) over(order by sum(refund_amount) desc) level
      FROM user_refund
      WHERE year(dt)=2019
      GROUP BY user_name)a
WHERE a.level=1;
```

```
Phillips        626604.0        1
Angelia 586608.0        1
Keith   579942.0        1
JUNE    379962.0        1
Time taken: 52.344 seconds, Fetched: 4 row(s)
```

偏移分析窗口函数(20min)

1、lag(...) over(.....)、lead(...) over(.....)

Lag和Lead分析函数可以在同一次查询中取出同一字段的前N行的数据(Lag)和后N行的数据(Lead)作为独立的列。

在实际应用当中，若要用到取今天和昨天的某字段差值时，Lag和Lead函数的应用就显得尤为重要。当然，这种操作可以用表的自连接实现，但是LAG和LEAD与left join、right join等自连接相比，效率更高，SQL更简洁。

lag(exp_str,offset,defval) over(partion by .....order by .....)

lead(exp_str,offset,defval) over(partion by .....order by .....)

- exp_str是字段名称。
- offset是偏移量，即是上1个或上N个的值，假设当前行在表中排在第5行，则offset 为3，则表示我们所要找的数据行就是表中的第2行（即5-3=2）。offset默认值为1。
- defval默认值，当两个函数取上N/下N个值，当在表中从当前行位置向前数N行已经超出了表的范围时，lag()函数将defval这个参数值作为函数的返回值，若没有指定默认值，则返回NULL，那么在数学运算中，总要给一个默认值才不会出错。

lag()实例：

```sql
--Alice和Alexander的各种时间偏移--
SELECT user_name,
       dt,
       lag(dt,1,dt) over(partition by user_name order by dt),
       lag(dt) over(partition by user_name order by dt),
       lag(dt,2,dt) over(partition by user_name order by dt),
       lag(dt,2) over(partition by user_name order by dt)
FROM user_trade
WHERE dt>'0'
      and user_name in ('Alice','Alexander');
```

```
Alexander       2017-04-08      2017-04-08      NULL    2017-04-08      NULL
Alexander       2017-12-02      2017-04-08      2017-04-08      2017-12-02      NULL
Alexander       2017-12-02      2017-12-02      2017-12-02      2017-04-08      2017-04-08
Alexander       2018-02-03      2017-12-02      2017-12-02      2017-12-02      2017-12-02
Alice   2017-12-11      2017-12-11      NULL    2017-12-11      NULL
Alice   2018-07-01      2017-12-11      2017-12-11      2018-07-01      NULL
Alice   2018-07-01      2018-07-01      2018-07-01      2017-12-11      2017-12-11
Alice   2018-08-15      2018-07-01      2018-07-01      2018-07-01      2018-07-01
Time taken: 30.632 seconds, Fetched: 8 row(s)
```

user_name	dt	lag(dt,1,dt)	lag(dt)	lag(dt,2,dt)	lag(dt,2)
Alexander	2017/4/8	2017/4/8	NULL	2017/4/8	NULL
Alexander	2017/12/2	2017/4/8	2017/4/8	2017/12/2	NULL
Alexander	2017/12/2	2017/12/2	2017/12/2	2017/4/8	2017/4/8
Alexander	2018/2/3	2017/12/2	2017/12/2	2017/12/2	2017/12/2
Alice	2017/12/11	2017/12/11	NULL	2017/12/11	NULL
Alice	2018/7/1	2017/12/11	2017/12/11	2018/7/1	NULL
Alice	2018/7/1	2018/7/1	2018/7/1	2017/12/11	2017/12/11
Alice	2018/8/15	2018/7/1	2018/7/1	2018/7/1	2018/7/1

lead()实例：

```sql
--Alice和Alexander的各种时间偏移--
SELECT user_name,
       dt,
       lead(dt,1,dt) over(partition by user_name order by dt),
       lead(dt) over(partition by user_name order by dt),
       lead(dt,2,dt) over(partition by user_name order by dt),
       lead(dt,2) over(partition by user_name order by dt)
FROM user_trade
WHERE dt>'0'
      and user_name in ('Alice','Alexander');
```

```
Alexander       2017-04-08      2017-12-02      2017-12-02      2017-12-02      2017-12-02
Alexander       2017-12-02      2017-12-02      2017-12-02      2018-02-03      2018-02-03
Alexander       2017-12-02      2018-02-03      2018-02-03      2017-12-02      NULL
Alexander       2018-02-03      2018-02-03      NULL    2018-02-03      NULL
Alice   2017-12-11      2018-07-01      2018-07-01      2018-07-01      2018-07-01
Alice   2018-07-01      2018-07-01      2018-07-01      2018-08-15      2018-08-15
Alice   2018-07-01      2018-08-15      2018-08-15      2018-07-01      NULL
Alice   2018-08-15      2018-08-15      NULL    2018-08-15      NULL
Time taken: 28.804 seconds, Fetched: 8 row(s)
```

user_name	dt	lead(dt,1,dt)	lead(dt)	lead(dt,2,dt)	lead(dt,2)
Alexander	2017/4/8	2017/12/2	2017/12/2	2017/12/2	2017/12/2
Alexander	2017/12/2	2017/12/2	2017/12/2	2018/2/3	2018/2/3
Alexander	2017/12/2	2018/2/3	2018/2/3	2017/12/2	NULL
Alexander	2018/2/3	2018/2/3	NULL	2018/2/3	NULL
Alice	2017/12/11	2018/7/1	2018/7/1	2018/7/1	2018/7/1
Alice	2018/7/1	2018/7/1	2018/7/1	2018/8/15	2018/8/15
Alice	2018/7/1	2018/8/15	2018/8/15	2018/7/1	NULL
Alice	2018/8/15	2018/8/15	NULL	2018/8/15	NULL

练习：

- 支付时间间隔超过**100**天的用户数：

```sql
--支付时间间隔超过100天的用户数--
SELECT count(distinct user_name)
FROM
      (SELECT user_name,
            dt,
            lead(dt) over(partition by user_name order by dt) lead_dt
      FROM user_trade
      WHERE dt>'0' )a
WHERE a.lead_dt is not null
      and datediff(a.lead_dt,a.dt)>100;
```

重点练习(20min)

1、每个城市，不同性别，2018年支付金额最高的TOP3用户(使用user_trade和user_info两个表)

user_info 列名	举例
user_id	10001,10002
user_name	Amy, Dennis
sex	[male, female]
age	[13,70]
city	beijing, shanghai
firstactivetime	2019-04-19 15:40:00
level	[1,10]
extra1	string类型：{"systemtype":"ios","education":"master","marriage_status":"1","phonebrand":"iphone X"}
extra2	map<string,string>类型：{"systemtype":"ios","education":"master","marriage_status":"1","phonebrand":"iphone X"}

user_trade列名	举例
user_name	Amy, Dennis
piece	购买数量
price	价格
pay_amount	支付金额
goods_category	food, clothes, book, computer, electronics, shoes
pay_time	1323308943，时间戳
dt	partition，‘yyyy-mm-dd’

```sql
--每个城市，不同性别，2018年支付金额最高的TOP3用户--
SELECT c.user_name,
       c.city,
       c.sex,
       c.pay_amount,
       c.rank
FROM
      (SELECT a.user_name,
              b.city,
              b.sex,
              a.pay_amount,
              row_number() over(partition by b.city,b.sex order by a.pay_amount desc) rank
      FROM
            (SELECT user_name,
                    sum(pay_amount) pay_amount
            FROM user_trade
            WHERE year(dt)=2018
            GROUP BY user_name)a
            LEFT JOIN user_info b on a.user_name=b.user_name)c
WHERE c.rank<=3;
```

```
Christine       beijing female  633270.0        1
DARCY   beijing female  486704.9        2
Becky   beijing female  268957.7        3
Campbell        beijing male    653347.2        1
Ross    beijing male    546811.2        2
Mitchell        beijing male    466620.0        3
Joanna  changchun      female  1059184.0       1
Andrea  changchun      female  1058965.3       2
Cherry  changchun      female  524638.5        3
Noah    changchun      male    1033230.0       1
Perry   changchun      male    509348.4        2
Spencer changchun      male    123608.4        3
Charlene        guangzhou       female  644533.6
DAISY   guangzhou      female  533280.0        2
Brenda  guangzhou      female  365237.30000000005
Ellis   guangzhou      male    534089.9        1
Johnson guangzhou      male    281859.7        2
Rupert  guangzhou      male    271084.0        3
Connie  hangzhou       female  717706.0        1
Jennifer        hangzhou        female  639936.0
JOY     hangzhou       female  583889.2        3
Albert  hangzhou       male    782144.0        1
Oliver  hangzhou       male    335595.7        2
Reeves  hangzhou       male    206646.0        3
Ashley  shanghai       female  659797.2        1
Cindy   shanghai       female  649336.0        2
Amber   shanghai       female  540056.0        3
Martin  shanghai       male    616879.7        1
Regan   shanghai       male    373496.0        2
Franklin        shanghai        male    344410.0
DIANA   shenzhen       female  567065.9        1
Jessica shenzhen       female  406626.0        2
Amy     shenzhen       female  394088.3        3
James   shenzhen       male    351076.0        1
Porter  shenzhen       male    331563.6        2
David   shenzhen       male    324412.0        3
Time taken: 59.219 seconds, Fetched: 36 row(s)
```

常见错误——没有指定字段的表别名：

```sql
SELECT c.user_name,
       c.city,
       c.sex,
       c.pay_amount,
       c.rank
FROM
      (SELECT user_name,
              city,
              sex,
              pay_amount,
              row_number() over(partition by city,sex order by pay_amount desc) rank
      FROM
            (SELECT user_name,
                    sum(pay_amount) pay_amount
            FROM user_trade
            WHERE year(dt)=2018
            GROUP BY user_name)a
            LEFT JOIN user_info b on a.user_name=b.user_name)c
WHERE c.rank<=3;
```

```
hive> SELECT c.user_name,
    >        c.city,
    >        c.sex,
    >        c.pay_amount,
    >        c.rank
    > FROM
    >       (SELECT user_name,
    >               city,
    >               sex,
    >               pay_amount,
    >               row_number() over(partition by city,sex order by pay_amount desc) rank
    >       FROM
    >             (SELECT user_name,
    >                     sum(pay_amount) pay_amount
    >             FROM user_trade
    >             WHERE year(dt)=2018
    >             GROUP BY user_name)a
    >             LEFT JOIN user_info b on a.user_name=b.user_name)c
    > WHERE c.rank<=3;
FAILED: SemanticException Column user_name Found in more than One Tables/Subqueries
```

2、每个手机品牌退款金额前25%的用户(使用user_refund和user_info两个表)

user_refund列名	举例
user_name	Amy, Dennis
refund_piece	退款件数
refund_amount	退款金额
refund_time	1323308943，时间戳
dt	partition，‘yyyy-mm-dd’

```
--每个手机品牌退款金额前25%的用户--
SELECT *
FROM
      (SELECT a.user_name,
             extra2['phonebrand'] as phonebrand,
             a.refund_amount,
             ntile(4) over(partition by extra2['phonebrand'] order by a.refund_amount desc) level
      FROM
            (SELECT user_name,
                  sum(refund_amount) refund_amount
            FROM user_refund
            WHERE dt>'0'
            GROUP BY user_name)a
            LEFT JOIN user_info b on a.user_name=b.user_name)c
WHERE c.level=1;
```

```
Martin   CHUIZI   606953.1          1
David    CHUIZI   277750.0          1
DORIS    CHUIZI   231168.3          1
Jordan   CHUIZI   182266.7          1
Betty    CHUIZI   142450.8          1
Angelia  HUAWEI   586697.1          1
DIANA    HUAWEI   566610.0          1
Cassie   HUAWEI   234443.2          1
Christy  HUAWEI   217756.0          1
KATE     MI       826619.6          1
ELLIE    MI       664728.3          1
JUNE     MI       379962.0          1
Willis   MI       342032.4          1
Mason    MI       333300.0          1
Andy     MI       299970.0          1
Andrea   VIVO     1039896.0         1
Maynard  VIVO     477686.0          1
Katrina  VIVO     393294.0          1
Porter   VIVO     264061.2          1
Abby     VIVO     156252.0          1
Angela   YIJIA    573631.7000000001         1
Cloris   YIJIA    573276.0          1
Regan    YIJIA    373296.0          1
Klein    YIJIA    293304.0          1
Frieda   YIJIA    192670.0          1
Phillips          iphone4 626604.0          1
Amber    iphone4  539946.0          1
JOY      iphone5  582919.2          1
Ashley   iphone5  406626.0          1
DAISY    iphone5  379982.9          1
Harrison          iphone5 273776.0          1
Albert   iphone6  259974.0          1
Dunn     iphone6  243934.0          1
Johnson  iphone6  220556.5          1
Hayes    iphone6s         499884.0          1
Eva      iphone6s         379364.0          1
James    iphone6s         286638.0          1
Bennett  iphone6s         55339.799999999996        1
Noah     iphone7  601287.6          1
Keith    iphone7  584542.0          1
Arnold   iphone7  188974.5          1
```

常见错误——忽略执行顺序，先使用别名后的字段做运算：

```
SELECT *
FROM
       (SELECT a.user_name,
               extra2['phonebrand'] as phonebrand,
               a.refund_amount,
               ntile(4) over(partition by phonebrand order by
a.refund_amount desc) level
       FROM
              (SELECT user_name,
                     sum(refund_amount) refund_amount
              FROM user_refund
              WHERE dt>'0'
              GROUP BY user_name)a
              LEFT JOIN user_info b on a.user_name=b.user_name)c
WHERE c.level=1;
```

```
hive> SELECT *
    > FROM
    >       (SELECT a.user_name,
    >              extra2['phonebrand'] as phonebrand,
    >              a.refund_amount,
    >              ntile(4) over(partition by phonebrand order by a.refund_amount desc) level
    >       FROM
    >             (SELECT user_name,
    >                    sum(refund_amount) refund_amount
    >             FROM user_refund
    >             WHERE dt>'0'
    >             GROUP BY user_name)a
    >             LEFT JOIN user_info b on a.user_name=b.user_name)c
    > WHERE c.level=1;
FAILED: SemanticException Failed to breakup Windowing invocations into Groups. At least 1 group must only depend on inp
ut columns. Also check for circular dependencies.
Underlying error: org.apache.hadoop.hive.ql.parse.SemanticException: Line 6:40 Invalid table alias or column reference
'phonebrand': (possible column names are: a.user_name, a.refund_amount, b.user_id, b.user_name, b.sex, b.age, b.city, b
.firstactivetime, b.level, b.extra1, b.extra2)
```

总结

1、注意如何对sum()、avg()这类累计计算的窗口函数的行数限制

2、不要混淆row_number()、rank()、dense_rank()三种函数

3、会使用ntile()进行分组查询

4、lag()：前N行、lead()：后N行

作业

作业1：计算出每12个月的用户累计支付金额

作业2：计算出每4个月的最大退款金额

作业3：退款时间间隔最长的用户