本文通过几个实际的查询例子，为大家介绍 Hive SQL 面试中最常问到的窗口函数。

假设有如下表格（loan）。表中包含贷款人的唯一标识，贷款日期，以及贷款金额。

name	orderdate	amount
jack	2019/1/2	8000
tony	2019/8/8	6000
mart	2017/1/1	8000
neil	2018/4/11	12000
...	...	...

1.SUM(), MIN(),MAX(),AVG()等聚合函数，可以直接使用 over() 进行分区计算。

```
SELECT *,
/*前三次贷款的金额之和*/
SUM(amount) OVER (PARTITION BY name ORDER BY orderdate ROWS
BETWEEN 3 PRECEDING AND CURRENT ROW) AS pv1,
/*历史所有贷款 累加到下一次贷款 的金额之和*/
SUM(amount) OVER (PARTITION BY name ORDER BY orderdate ROWS
BETWEEN UNBOUNDED PRECEDING AND 1 FOLLOWING) AS pv2
FROM loan ;
```

其中，窗口函数 over()使得聚合函数 sum()可以在限定的窗口中进行聚合。本例子中，第一条语句计算每个人当前记录的前三条贷款金额之和。第二条语句计算截至到下一次贷款，客户贷款的总额。

窗口的限定语法为：ROWS BETWEEN 一个时间点 AND 一个时间点。时间节点可以使用：

n PRECEDING：前 n 行

n FOLLOWING：后 n 行

CURRENT ROW ： 当前行

如果不想限制具体的行数，可以将 n 替换为 UNBOUNDED.比如从起始到当前，可以写为:

ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW.

窗口函数 over()和 group by 的最大区别，在于 group by 之后其余列也必须按照此分区进行计算，而 over()函数使得单个特征可以进行分区。

2.NTILE(), ROW_NUMBER(), RANK(), DENSE_RANK()，可以为数据集新增加序列号。

```
SELECT *,
#将数据按 name 切分成 10 区，并返回属于第几个分区
NTILE(10) OVER (PARTITION BY name ORDER BY orderdate) AS f1,

#将数据按照 name 分区，并按照 orderdate 排序，返回排序序号
ROW_NUMBER() OVER (PARTITION BY name ORDER BY orderdate) AS
f2,
```

```
#将数据按照 name 分区，并按照 orderdate 排序，返回排序序号
RANK() OVER (PARTITION BY name ORDER BY orderdate) AS f3,

#将数据按照 name 分区，并按照 orderdate 排序，返回排序序号
DENSE_RANK() OVER (PARTITION BY name ORDER BY orderdate) AS f4

FROM loan;
```

其中第一个函数是将数据按 name 切分成 10 区，并返回属于第几个分区。后面的三个函数的功能看起来很相似。区别在于当数据中出现相同值得时候，如何编号。

ROW_NUMBER()返回的是一列连续的序号。

1
2
3
4
5
6

RANK()对于数值相同的这一项会标记为相同的序号，而下一个序号跳过。比如{4，5，6}变成了{4，4，6}.

1
2
3
4
4
6

DENSE_RANK()对于数值相同的这一项，也会标记为相同的序号，但下一个序号并不会跳过。比如{4，5，6}变成了{4，4，5}.

1
2
3
4
4
5

3.LAG(), LEAD(), FIRST_VALUE(), LAST_VALUE()函数返回一系列指定的点

```
SELECT *,
#取上一笔贷款的日期,缺失默认填 NULL
LAG(orderdate, 1) OVER(PARTITION BY name ORDER BY orderdate)
AS last_dt,

#取下一笔贷款的日期,缺失指定填'1970-1-1'
LEAD(orderdate, 1,'1970-1-1') OVER(PARTITION BY name ORDER BY
orderdate) AS next_dt,

#取最早一笔贷款的日期
FIRST_VALUE(orderdate) OVER(PARTITION BY name ORDER BY
orderdate) AS first_dt,

#取新一笔贷款的日期
LAST_VALUE(orderdate) OVER(PARTITION BY name ORDER BY
orderdate) AS latest_dt

FROM loan;
```

LAG(n)将数据向前错位 n 行。LEAD()将数据向后错位 n 行。FIRST_VALUE()取当前分区中的第一个值。 LAST_VALUE()取当前分区最后一个值。

4.GROUPING SET(),with CUBE, with ROLLUP 对 group by 进行限制

```
SELECT
A,B,C
FROM loan

#分别按照月份和日进行分区
GROUP BY substring(orderdate,1,7),orderdate

GROUPING SETS(substring(orderdate,1,7), orderdate)
ORDER BY GROUPING__ID;
```

GROUPING__ID 是 GROUPING_SET()的操作之后自动生成的。生成 GROUPING__ID 是为了区分每条输出结果是属于哪一个 group by 的数据。它是根据 group by 后面声明的顺序字段，是否存在于当前 group by 中的一个二进制位组合数据。GROUPING SETS()必须先做 GROUP BY 操作。

比如（A,C）的 group_id： group_id(A,C) = grouping(A)+grouping(B)+grouping (C) 的结果就是：二进制：101 也就是 5.

如果解释器发现 group by A,C 但是 select A,B,C 那么运行时会将所有 from 表取出的结果复制一份，B 都置为 null，也就是在结果中，B 都为 null.

```
SELECT
A,B,C
FROM loan

#分别按照月份和日进行分区
GROUP BY substring(orderdate,1,7),orderdate

with CUBE
ORDER BY GROUPING__ID;
```

with CUBE 和 GROUPING_SET()的区别就是，with CUBE 返回的是 group by 字段的笛卡尔积。

```
SELECT
A,B,C
FROM loan

#分别按照月份和日进行分区
GROUP BY substring(orderdate,1,7),orderdate

with ROLLUP
ORDER BY GROUPING__ID;
```

with ROLLUP 则不会产生第二列为键的聚合结果，在本例子中，只按照 substring(orderdate,1,7)进行展示。所以使用 with ROLLUP 时，要注意 group by 后面字段的顺序。