order by 参与与否对结果的影响

区分开窗函数和普通聚合函数的关键字是OVER。
当普通聚合函数sum等后面加上over()时，就表示对结果集中的所有行进行聚合运算，并把结果返回到每一行中。
因此，开窗函数的运算是不改变原始数据的行数.
- over关键字用来指定函数执行的窗口范围，如果后面括号中什么都不写，则意味着窗口包含满足条件的所有行，开窗函数基于所有行进行计算

2.同时使用partition by和order by
当over中指定了分区和排序时，会按照指定字段进行排序，然后计算当前分区内的第一行到当前行取值范围内的记录。

```
-- 计算每个用户的累计订单金额
select *,sum(amount) over(partition by user_no order by order_id asc) as 累计订单金
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

order_id	user_no	amount	create_date	累计订单金额
101	001	100	2019-01-01	100
211	001	300	2019-01-02	400
308	001	500	2019-01-02	900
415	001	800	2019-01-03	1700
523	001	900	2019-01-04	2600
612	002	500	2019-01-03	500
718	002	600	2019-01-04	1100
804	002	300	2019-01-10	1400
911	002	800	2019-01-16	2200
1109	002	800	2019-01-22	3000

以第三行为例：第三行作为当前行，开窗函数是基于第一行到第三行的amout值进行了聚合运算

over中有三个参数可以用来设置窗口的范围：

（1）partition by：按照指定字段进行分区，开窗函数在不同的分区内分别执行。分区所依据的字段可以是多个，不同字段间使用逗号隔开。

（2）order by：按照指定字段进行排序，开窗函数将按照排序后的记录顺序进行编号和计算。可以和partition by子句配合使用，也可以单独使用。

（3）range|rows：当前分区的一个子集，用来定义子集的规则，通常用来作为滑动窗口使用。

开窗函数本质上一种特殊的聚合函数。我们知道聚合函数是对满足条件的行 执行计算并返回一个值。而开窗函数是针对满足条件的行进行运算，结果会返回在所有参与计算的行上。

注意：因为where子句中不能使用窗口函数，而窗口函数本质上还是个聚合函数

sum(…) over(partition by…)，同组内所行求和

sum(…) over(partition by… order by …)，同第1点中的排序求和原理，只是范围限制在组内

```
with aa as
(
SELECT 1 a,1 b, 3 c FROM dual union
SELECT 2 a,2 b, 3 c FROM dual union
SELECT 3 a,3 b, 3 c FROM dual union
SELECT 4 a,4 b, 3 c FROM dual union
SELECT 5 a,5 b, 3 c FROM dual union
SELECT 6 a,5 b, 3 c FROM dual union
SELECT 7 a,2 b, 3 c FROM dual union
SELECT 7 a,2 b, 8 c FROM dual union
SELECT 9 a,3 b, 3 c FROM dual
)
SELECT a,b,c,sum(c) over( partition by b ) partition_sum,
sum(c) over( partition by b order by a desc) partition_order_sum
  FROM aa;
```

SQL查询结果					算法解析			
A	B	C	partition_sum	partition_order_sum	组号	排序号	partition_sum 求和算法	partition_order_sum 求和算法
1	1	3	3	3	1	1	3	3
7	2	3	14	11	2	7	3+8+3=14	3+8=11
7	2	8	14	11				3+8=11
2	2	3	14	14		2		3+8+3=14
9	3	3	6	3	3	9	3+3=6	3
3	3	3	6	6		3		3+3=6
4	4	3	3	3	4	4	3	3
6	5	3	6	3	5	6	3+3=6	3
5	5	3	6	6		5		3+3=6

注意：**order by** 参与 over时候，开窗的范围是**order by** 已排好的序号进行计算，
即，是一种累加计算，如果排序中存在重复的序号，则该序号标识的行同时参与聚合函数计算.

2，分组运算

- count() over() 分组计数

```
select id ,ename, sal ,code ,
    count(1) over(partition by code ) count_over,    --计数汇总
    count(1) over(partition by code order by id) count_order_by      --分步计数
from testdata;
```

显示结果：

	ID	ENAME	SAL	CODE	COUNT_OVER	COUNT_ORDER_BY
1	1	a001	100	aaaaa	4	1
2	2	a001	500	aaaaa	4	2
3	5	a002	300	aaaaa	4	3
4	9	a003	100	aaaaa	4	4
5	3	a001	100	bbbbb	1	1
6	4	a002	500	ccccc	3	1
7	6	a003	200	ccccc	3	2
8	8	a002	400	ccccc	3	3
9	7	a003	800	ddddd	2	1
10	10	a003	200	ddddd	2	2

- **max() over() 、min() over() 分组取最大值、最小值**

```
select id ,ename, sal ,code ,
    max(sal) over(partition by code ) max_over,      --取分组内数据的最大值
    max(sal) over(partition by code order by sal asc) max_order_by   --分步取最大值
from testdata;
```

显示结果：

	ID	ENAME	SAL	CODE	MAX_OVER	MAX_ORDER_BY
1	1	a001	100	aaaaa	500	100
2	9	a003	100	aaaaa	500	100
3	5	a002	300	aaaaa	500	300
4	2	a001	500	aaaaa	500	500
5	3	a001	100	bbbbb	100	100
6	6	a003	200	ccccc	500	200
7	8	a002	400	ccccc	500	400
8	4	a002	500	ccccc	500	500
9	10	a003	200	ddddd	800	200
10	7	a003	800	ddddd	800	800

- **sum() over （） 分组求和**

```
select id ,ename, sal ,code ,
    sum(sal) over(partition by ename ) sum_over,     --分组数据求和
    sum(sal) over(partition by ename order by id) sum_order_by   --分步求和
from testdata;
```

显示结果：

	ID	ENAME	SAL	CODE	SUM_OVER	SUM_ORDER_BY
1	1	a001	100	aaaaa	700	100
2	2	a001	500	aaaaa	700	600
3	3	a001	100	bbbbb	700	700
4	4	a002	500	ccccc	1200	500
5	5	a002	300	aaaaa	1200	800
6	8	a002	400	ccccc	1200	1200
7	6	a003	200	ccccc	1300	200
8	7	a003	800	ddddd	1300	1000
9	9	a003	100	aaaaa	1300	1100
10	10	a003	200	ddddd	1300	1300

avg() over() 分组取平均

```
select id ,ename, sal ,code ,
    avg(sal) over(partition by code ) avg_over,      --分组内数据平均
    avg(sal) over(partition by code order by id) avg_order_by    --分步取平均
from testdata;
```

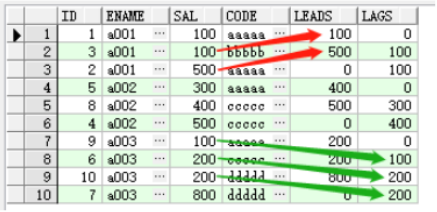
显示结果：

	ID	ENAME	SAL	CODE	AVG_OVER	AVG_ORDER_BY
1	1	a001	100	aaaaa	250	100
2	2	a001	500	aaaaa	250	300
3	5	a002	300	aaaaa	250	300
4	9	a003	100	aaaaa	250	250
5	3	a001	100	bbbbb	100	100
6	4	a002	500	ccccc	366.666666666667	500
7	6	a003	200	ccccc	366.666666666667	350
8	8	a002	400	ccccc	366.666666666667	366.666666666667
9	7	a003	800	ddddd	500	800
10	10	a003	200	ddddd	500	500

3，lag()和lead()统计函数可以在一次查询中取出同一字段的前N行的数据和后N行的值。这种操作可以使用对相同表的表连接来实现，不过使用LAG和 LEAD有更高的效率。Lag函数为Lag(exp,N,defval)，defval是当该函数无值可用的情况下返回的值。Lead函数的用法类似。

```
select id ,ename, sal ,code ,
    lead(sal,1,0) over(partition by ename order by sal asc) leads,
    lag(sal,1,0) over(partition by ename order by sal asc) lags
from testdata
```

显示结果：



	ID	ENAME	SAL	CODE	LEADS	LAGS
1	1	a001	100	aaaaa	100	0
2	3	a001	100	bbbbb	500	100
3	2	a001	500	aaaaa	0	100
4	5	a002	300	aaaaa	400	0
5	8	a002	400	ccccc	500	300
6	4	a002	500	ccccc	0	400
7	9	a003	100	aaaaa	200	0
8	6	a003	200	ccccc	200	100
9	10	a003	200	lllll	800	200
10	7	a003	800	ddddd	0	200

开窗函数的滑动窗口

Sliding Window 滑动窗口

In addition to calculating aggregate and rank information, window functions can also be used to calculate information that changes with each subsequent row in a data set. These types of window functions are called sliding windows.

除了计算汇总、聚合和排序等，窗口函数还可以用于计算随数据集中的每个后续行而变化的信息。这类窗口功能称为滑动窗口。

Sliding windows are functions that perform calculations relative to the current row of a data set. 滑动窗口是执行相对于数据集当前行的计算的功能。

You can use sliding windows to calculate a wide variety of information that aggregates one row at a time down your data set -- running totals, sums, counts, and averages in any order you need.

A sliding window calculation can also be partitioned by one or more column just like a non-sliding window.

滑动窗口 关键字（加在OVER从句中）

```
ROWS BETWEEN <start> AND <finish>
```

可用于start 和finish 的关键字有：

3.指定滑动窗口的范围

当over中指定了滑动窗口范围时，就会按照指定的范围计算。

对于滑动窗口的范围指定有两种方式：基于行（rows）和基于值（range）。

（1）使用rows来基于行指定滑动窗口范围

```
-- 按订单编号顺序查找用户当前订单的前一笔到后一笔平均订单金额
select *,
       avg(amount) over(partition by user_no order by order_id rows between 1 prece
from order_tab;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Conte

order_id	user_no	amount	create_date	平均订单金额
101	001	100	2019-01-01	200.0000
211	001	300	2019-01-02	300.0000
308	001	500	2019-01-02	533.3333
415	001	800	2019-01-03	733.3333
523	001	900	2019-01-04	850.0000
612	002	500	2019-01-03	550.0000
718	002	600	2019-01-04	466.6667
804	002	300	2019-01-10	566.6667
911	002	800	2019-01-16	633.3333
1109	002	800	2019-01-22	800.0000

(2) 使用range来基于值指定滑动窗口范围

```
1  -- 查询每个用户按下单时间顺序后前一天到后一天的平均订单金额
2  select *,avg(amount) over(partition by user_no order by create_date range between
```

	order_id	user_no	amount	create_date	平均订单金额
▶	101	001	100	2019-01-01	300.0000
	211	001	300	2019-01-02	425.0000
	308	001	500	2019-01-02	425.0000
	415	001	800	2019-01-03	625.0000
	523	001	900	2019-01-04	850.0000
	612	002	500	2019-01-03	550.0000
	718	002	600	2019-01-04	550.0000
	804	002	300	2019-01-10	300.0000
	911	002	800	2019-01-16	800.0000
	1109	002	800	2019-01-22	800.0000