

## 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

`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：因为开窗函数中不能使用窗口函数，而窗口函数本质上还是聚合函数

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

```

1 with aa as
2 (
3  SELECT 1 a,1 b, 3 c FROM dual union
4  SELECT 2 a,2 b, 3 c FROM dual union
5  SELECT 3 a,3 b, 3 c FROM dual union
6  SELECT 4 a,4 b, 3 c FROM dual union
7  SELECT 5 a,5 b, 3 c FROM dual union
8  SELECT 6 a,5 b, 3 c FROM dual union
9  SELECT 7 a,2 b, 3 c FROM dual union
10 SELECT 7 a,2 b, 8 c FROM dual union
11 SELECT 9 a,3 b, 3 c FROM dual
12 )
13 SELECT a,b,c,sum(c) over( partition by b ) partition_sum,
14 sum(c) over( partition by b order by a desc ) partition_order_sum
15 FROM aa;
```

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

```

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

显示结果：

	ID	ENAME	SAL	CODE	COUNT_OVER	COUNT_ORDER_BY
1	1	a001	100	aaaa	4	1
2	2	a001	500	aaaa	4	2
3	5	a002	300	aaaa	4	3
4	9	a003	100	aaaa	4	4
5	3	a001	100	bbbb	1	1
6	4	a002	500	cccc	3	1
7	6	a003	200	cccc	3	2
8	8	a002	400	cccc	3	3
9	7	a003	800	dddd	2	1
10	10	a003	200	dddd	2	2

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

```

1 | select id ,ename, sal ,code ,
2 |     max(sal) over(partition by code ) max_over,      --取分组内数据的最大值
3 |     max(sal) over(partition by code order by sal asc) max_order_by  --分步取最大值
4 | 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	bbbbbb	100	100
6	6	a003	200	cccccc	500	200
7	8	a002	400	cccccc	500	400
8	4	a002	500	cccccc	500	500
9	10	a003	200	dddddd	800	200
10	7	a003	800	dddddd	800	800

- sum() over () 分组求和

```

1 | select id ,ename, sal ,code ,
2 |     sum(sal) over(partition by ename ) sum_over,      --分组数据求和
3 |     sum(sal) over(partition by ename order by id) sum_order_by  --分步求和
4 | 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	bbbbbb	700	700
4	4	a002	500	cccccc	1200	500
5	5	a002	300	aaaaa	1200	800
6	8	a002	400	cccccc	1200	1200
7	6	a003	200	cccccc	1300	200
8	7	a003	800	dddddd	1300	1000
9	9	a003	100	aaaaa	1300	1100
10	10	a003	200	dddddd	1300	1300

avg() over() 分组取平均

```

1 | select id ,ename, sal ,code ,
2 |     avg(sal) over(partition by code ) avg_over,      --分组内数据平均
3 |     avg(sal) over(partition by code order by id) avg_order_by  --分步取平均
4 | 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	bbbbbb	100	100
6	4	a002	500	cccccc	366.666666666667	500
7	6	a003	200	cccccc	366.666666666667	350
8	8	a002	400	cccccc	366.666666666667	366.666666666667
9	7	a003	800	dddddd	500	800
10	10	a003	200	dddddd	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。

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

显示结果:

	ID	ENAME	SAL	CODE	LEADS	LAGS
1	1	a001	100	aaaa	100	0
2	3	a001	100	bbbb	500	100
3	2	a001	500	aaaa	0	100
4	5	a002	300	aaaa	400	0
5	8	a002	400	cccc	500	300
6	4	a002	500	cccc	0	400
7	9	a003	100	aaaa	200	0
8	6	a003	200	cccc	200	100
9	10	a003	200	jjjj	800	200
10	7	a003	800	dddd	0	200

## 开窗函数的滑动窗口

赞同 14



### 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: