

# 第五章：SQL高级处理

## 5.1 窗口函数

### 5.1.1 窗口函数概念及基本的使用方法

窗口函数也称为**OLAP函数**。OLAP 是 OnLine AnalyticalProcessing 的简称，意思是对数据库数据进行实时分析处理。

为了便于理解，称之为**窗口函数**。常规的SELECT语句都是对整张表进行查询，而窗口函数可以让我们有选择的去某一部分数据进行汇总、计算和排序。

窗口函数的通用形式：

```
<窗口函数> OVER ([PARTITION BY <列名>]  
                  ORDER BY <排序用列名>)
```

[ ]中的内容可以省略。

窗口函数最关键的是搞明白关键字 **PARTITION BY** 和 **ORDER BY** 的作用。

**PARTITION BY** 是用来分组，即选择要看哪个窗口，类似于 GROUP BY 子句的分组功能，但是 PARTITION BY 子句并不具备 GROUP BY 子句的汇总功能，并不会改变原始表中记录的行数。

**ORDER BY** 是用来排序，即决定窗口内，是按那种规则(字段)来排序的。

举个栗子：

```
SELECT product_name  
       ,product_type  
       ,sale_price  
       ,RANK() OVER (PARTITION BY product_type  
                     ORDER BY sale_price) AS ranking  
FROM product;
```

得到的结果是：

## 执行结果

product_name	product_type	sale_price	ranking
叉子	厨房用具	500	1
擦菜板	厨房用具	880	2
菜刀	厨房用具	3000	3
高压锅	厨房用具	6800	4
T恤衫	衣服	1000	1
运动T恤	衣服	4000	2
圆珠笔	办公用品	100	1
打孔器	办公用品	500	2

我们先忽略生成的新列 - [ranking], 看下原始数据在PARTITION BY 和 ORDER BY 关键字的作用下发生了什么变化。

PARTITION BY 能够设定窗口对象范围。本例中, 为了按照商品种类进行排序, 我们指定了 product\_type。即一个商品种类就是一个小的"窗口"。

ORDER BY 能够指定按照哪一列、何种顺序进行排序。为了按照销售单价的升序进行排列, 我们指定了sale\_price。此外, 窗口函数中的ORDER BY与SELECT语句末尾的ORDER BY一样, 可以通过关键字ASC/DESC来指定升序/降序。省略该关键字时会默认按照ASC, 也就是

升序进行排序。本例中就省略了上述关键字。

图8-1 PARTITION BY和ORDER BY的作用

通过PARTITION BY分组后的记录的集合可以称为窗口

ORDER BY的顺序 (销售单价由低到高的顺序)

product_id (商品编号)	product_name (商品名称)	product_type (商品种类)	sale_price (销售单价)	purchase_price (进货单价)	regist_date (登记日期)
0006	叉子	厨房用具	500		2009-09-20
0007	擦菜板	厨房用具	880	790	2008-04-28
0004	菜刀	厨房用具	3000	2800	2009-09-20
0005	高压锅	厨房用具	6800	5000	2009-01-15
0001	T恤衫	衣服	1000	500	2009-09-20
0003	运动T恤	衣服	4500	2800	
0008	圆珠笔	办公用品	100		2009-11-11
0002	打孔器	办公用品	500	320	2009-09-11

PARTITION BY的分组 (根据商品种类)

## 5.2 窗口函数种类

大致来说，窗口函数可以分为两类。

一是 将SUM、MAX、MIN等聚合函数用在窗口函数中

二是 RANK、DENSE\_RANK等排序用的专用窗口函数

### 5.2.1 专用窗口函数

#### • RANK函数

计算排序时，如果存在相同位次的记录，则会跳过之后的位次。

例) 有 3 条记录排在第 1 位时：1 位、1 位、1 位、4 位.....

#### • DENSE\_RANK函数

同样是计算排序，即使存在相同位次的记录，也不会跳过之后的位次。

例）有 3 条记录排在第 1 位时：1 位、1 位、1 位、2 位……

### • ROW\_NUMBER函数

赋予唯一的连续位次。

例）有 3 条记录排在第 1 位时：1 位、2 位、3 位、4 位

运行以下代码：

```
SELECT product_name
,product_type
,sale_price
,RANK() OVER (ORDER BY sale_price) AS ranking
,DENSE_RANK() OVER (ORDER BY sale_price) AS dense_ranking
,ROW_NUMBER() OVER (ORDER BY sale_price) AS row_num
FROM product;
```

执行结果

			RANK	DENSE_RANK	ROW_NUMBER
product_name	product_type	sale_price	ranking	dense_ranking	row_num
圆珠笔	办公用品	100	1	1	1
叉子	厨房用具	500	2	2	2
打孔器	办公用品	500	2	2	3
擦菜板	厨房用具	880	4	3	4
T恤衫	衣服	1000	5	4	5
菜刀	厨房用具	3000	6	5	6
运动T恤	衣服	4000	7	6	7
高压锅	厨房用具	6800	8	7	8

## 5.2.2 聚合函数在窗口函数上的使用

聚合函数在窗口函数中的使用方法和之前的专用窗口函数一样，只是出来的结果是一个**累计**的聚合函数值。

运行以下代码

```

SELECT product_id
,product_name
,sale_price
,SUM(sale_price) OVER (ORDER BY product_id) AS current_sum
,AVG(sale_price) OVER (ORDER BY product_id) AS current_avg
FROM product;

```

### 执行结果

product_id	product_name	sale_price	current_sum	
0001	T恤衫	1000	1000	←1000
0002	打孔器	500	1500	←1000+500
0003	运动T恤	4000	5500	←1000+500+4000
0004	菜刀	3000	8500	←1000+500+4000+3000
0005	高压锅	6800	15300	.
0006	叉子	500	15800	.
0007	擦菜板	880	16680	.
0008	圆珠笔	100	16780	.

### 执行结果

product_id	product_name	sale_price	current_avg	
0001	T恤衫	1000	1000.0000000000000000	←(1000)/1
0002	打孔器	500	750.0000000000000000	←(1000+500)/2
0003	运动T恤	4000	1833.3333333333333333	←(1000+500+4000)/3
0004	菜刀	3000	2125.0000000000000000	←(1000+500+4000+3000)/4
0005	高压锅	6800	3060.0000000000000000	←(1000+500+4000+3000+6800)/5
0006	叉子	500	2633.3333333333333333	.
0007	擦菜板	880	2382.8571428571428571	.
0008	圆珠笔	100	2097.5000000000000000	.

可以看出，聚合函数结果是，按我们指定的排序，这里是product\_id，当前所在行及之前所有的行的合计或均值。即累计到当前行的聚合。

## 5.3 窗口函数的应用 - 计算移动平均

在上面提到，聚合函数在窗口函数使用时，计算的是累积到当前行的所有的数据的聚合。实际

上，还可以指定更加详细的**汇总范围**。该汇总范围成为 **框架** (frame)。

## 语法

```
<窗口函数> OVER (ORDER BY <排序用列名>
                  ROWS n PRECEDING )
```

```
<窗口函数> OVER (ORDER BY <排序用列名>
                  ROWS BETWEEN n PRECEDING AND n FOLLOWING)
```

PRECEDING (“之前”)， 将框架指定为“截止到之前 n 行”，加上自身行

FOLLOWING (“之后”)， 将框架指定为“截止到之后 n 行”，加上自身行

BETWEEN 1 PRECEDING AND 1 FOLLOWING，将框架指定为“之前1行”+“之后1行”+“自身”

执行以下代码：

```
SELECT product_id
       ,product_name
       ,sale_price
       ,AVG(sale_price) OVER (ORDER BY product_id
                              ROWS 2 PRECEDING) AS moving_avg
       ,AVG(sale_price) OVER (ORDER BY product_id
                              ROWS BETWEEN 1 PRECEDING
                              AND 1 FOLLOWING) AS moving_avg
FROM product;
```

**执行结果：**

注意观察框架的范围。

ROWS 2 PRECEDING：



product_id	product_name	sale_price	moving_avg	
0001	T恤衫	1000	1000	$\leftarrow (1000) / 1$
0002	打孔器	500	750	$\leftarrow (1000+500) / 2$
0003	运动T恤	4000	1833	$\leftarrow (1000+500+4000) / 3$
0004	菜刀	3000	2500	$\leftarrow (500+4000+3000) / 3$
0005	高压锅	6800	4600	$\leftarrow (4000+3000+6800) / 3$
0006	叉子	500	3433	.
0007	擦菜板	880	2726	.
0008	圆珠笔	100	493	.

ROWS BETWEEN 1 PRECEDING AND 1 FOLLOWING :

product_id	product_name	sale_price	moving_avg	
0001	T恤衫	1000	750	$\leftarrow (1000+500) / 2$
0002	打孔器	500	1833	$\leftarrow (1000+500+4000) / 3$
0003	运动T恤	4000	2500	$\leftarrow (500+4000+3000) / 3$
0004	菜刀	3000	4600	$\leftarrow (4000+3000+6800) / 3$
0005	高压锅	6800	3433	.
0006	叉子	500	2726	.
0007	擦菜板	880	493	.
0008	圆珠笔	100	490	.

### 5.3.1 窗口函数适用范围和注意事项

- 原则上，窗口函数只能在SELECT子句中使用。
- 窗口函数OVER 中的ORDER BY 子句并不会影响最终结果的排序。其只是用来决定窗口函数按何种顺序计算。

## 5.4 GROUPING运算符

### 5.4.1 ROLLUP - 计算合计及小计

常规的GROUP BY 只能得到每个分类的小计，有时候还需要计算分类的合计，可以用 ROLLUP

关键字。

```
SELECT product_type
,regist_date
,SUM(sale_price) AS sum_price
FROM product
GROUP BY product_type, regist_date WITH ROLLUP;
```

得到的结果为：

product_type	regist_date	sum_price	
办公用品	2009-09-11	500	
办公用品	2009-11-11	100	
办公用品	NULL	600	← 小计(办公用品)
厨房用具	2008-04-28	880	
厨房用具	2009-01-15	6800	
厨房用具	2009-09-20	3500	
厨房用具	NULL	11180	← 小计(厨房用品)
衣服	NULL	4000	
衣服	2009-09-20	1000	
衣服	NULL	5000	← 小计(衣服)
NULL	NULL	16780	← 合计

product_type	regist_date	sum_price	
		16780	← 合计
厨房用具		11180	← 小计(厨房用具)
厨房用具	2008-04-28	880	
厨房用具	2009-01-15	6800	
厨房用具	2009-09-20	3500	
办公用品		600	← 小计(办公用品)
办公用品	2009-09-11	500	
办公用品	2009-11-11	100	
衣服		5000	← 小计(衣服)
衣服	2009-09-20	1000	
衣服		4000	

这里ROLLUP 对product\_type, regist\_date两列进行合计汇总。结果实际上有三层聚合，如下图模块3是常规的 GROUP BY 的结果，需要注意的是衣服 有个注册日期为空的，这是本来数据就存在日期为空的，不是对衣服类别的合计；模块2和1是 ROLLUP 带来的合计，模块2是对产品种类的合计，模块1是对全部数据的总计。

ROLLUP 可以对多列进行汇总求小计和合计。



product_type	regist_date	sum_price
.....		
		16780 模块①
厨房用具		11180
办公用品		600 模块②
衣服		5000
办公用品	2009-09-11	500
办公用品	2009-11-11	100
厨房用具	2008-04-28	880
厨房用具	2009-01-15	6800 模块③
厨房用具	2009-09-20	3500
衣服	2009-09-20	1000
衣服		4000

## 练习题

### 5.1

请说出针对本章中使用的 product（商品）表执行如下 SELECT 语句所能得到的结果。

```
SELECT product_id
       ,product_name
       ,sale_price
       ,MAX(sale_price) OVER (ORDER BY product_id) AS Current_max_price
FROM product;
```

### 5.2

继续使用product表，计算出按照登记日期（regist\_date）升序进行排列的各日期的销售单价（sale\_price）的总额。排序是需要将登记日期为NULL的“运动T恤”记录排在第1位（也就是将其看作比其他日期都早）

## 5.3

### 思考题

- ① 窗口函数不指定PARTITION BY的效果是什么？
- ② 为什么说窗口函数只能在SELECT子句中使用？实际上，在ORDER BY子句使用系统并不会报错。

DataWhale出品，仅供学习，请勿外传