

本笔记为阿里云天池龙珠计划SQL训练营的学习内容，链接为：<https://tianchi.aliyun.com/specials/promotion/aicampsql>

Task05: 窗口函数

[TOC]

5.0 MySQL 函数

本节为您提供最常用的**MySQL函数，**包括聚合函数，字符串函数，日期时间函数，控制流函数等。

MySQL聚合函数

- ✓ • **AVG** - 计算一组值或表达式的平均值。
- ✓ • **COUNT** - 计算表中的行数。
- **INSTR** - 返回字符串中第一次出现的子字符串的位置。
- **SUM** - 计算一组值或表达式的总和。
- **MIN** - 在一组值中找到最小值
- **MAX** - 在一组值中找到最大值

MySQL字符串函数

- **CONCAT** - 将两个或多个字符串组合成一个字符串。
- **LENGTH&CHAR_LENGTH** - 获取字符串的长度，以字节和字符为单位。
- **LEFT** - 获取具有指定长度的字符串的左侧部分。
- **REPLACE** - 搜索并替换字符串中的子字符串。
- **SUBSTRING** - 从具有特定长度的位置开始提取子字符串。
- **TRIM** - 从字符串中删除不需要的字符。
- **FIND_IN_SET** - 在以逗号分隔的字符串列表中查找字符串。
- **FORMAT** - 格式化具有特定区域设置的数字，四舍五入到小数位数

MySQL控制流功能

- **CASE** - THEN 如果 WHEN 满足分支中的条件，则返回分支中的相应结果，否则返回 ELSE 分支中的结果。
- **IF** - 根据给定条件返回值。
- **IFNULL** - 如果它不是NULL则返回第一个参数，否则返回第二个参数。

- **NULLIF** - 如果第一个参数等于第二个参数，**则**返回NULL，否则返回第一个参数。

MySQL日期和时间函数

- **CURDATE** - 返回当前日期。
- **DATEDIFF** - 计算两个 **DATE** 值之间的天数。
- **DAY** - 获取指定日期的月份日期。
- **DATE_ADD** - 将日期值添加到日期值。
- **DATE_SUB** - 从日期值中减去时间值。
- **DATE_FORMAT** - 根据指定的日期格式格式化日期值。
- **DAYNAME** - 获取指定日期的工作日名称。
- **DAYOFWEEK** - 返回日期的工作日索引。
- **EXTRACT** - 提取日期的一部分。
- **NOW** - 返回执行语句的当前日期和时间。
- **MONTH** - 返回表示指定日期月份的整数。
- **STR_TO_DATE** - 根据指定的格式将字符串转换为日期和时间值。
- **SYSDATE** - 返回当前日期。
- **TIMEDIFF** - 计算两个 **TIME** 或 **DATETIME** 值之间的差异。
- **TIMESTAMPDIFF** - 计算两个 **DATE** 或 **DATETIME** 值之间的差异。
- **WEEK** - 返回一个星期的日期。
- **WEEKDAY** - 返回日期的工作日索引。
- **YEAR** - 返回日期值的年份部分。

MySQL比较功能

- **COALESCE** - 返回第一个非null参数，这对于替换null非常方便。
- **GREATEST & LEAST** - 取n个参数并分别返回n个参数的最大值和最小值。
- **ISNULL** - 如果参数为null，则返回1，否则返回零。

MySQL数学函数

- **ABS** - 返回数字的绝对值。
- **CEIL** - 返回大于或等于输入数字的最小整数值。
- **FLOOR** - 返回不大于参数的最大整数值。
- **MOD** - 返回数字的余数除以另一个。
- **ROUND** - 将数字四舍五入到指定的小数位数。
- **TRUNCATE** - 将数字截断为指定的小数位数。

其他MySQL功能

- `LAST_INSERT_ID` - 获取最后生成的最后一个插入记录的序列号。
- `CAST` - 将任何类型的值转换为具有指定类型的值。

5.1 窗口函数

MySQL窗口函数列表

下表显示了MySQL中的窗口函数：

名称	描述
<code>CUME_DIST</code>	计算一组值中值的累积分布。
<code>DENSE_RANK</code>	根据ORDER BY子句为其分区中的每一行分配一个排名。它为具有相同值的行分配相同的排名。如果两行或更多行具有相同的等级，则排序值序列中将有间隙。
<code>FIRST_VALUE</code>	返回指定表达式相对于窗口框架中第一行的值。
<code>LAG</code>	返回分区中当前行之前的第N行的值。如果不存在前一行，则返回NULL。
<code>LAST_VALUE</code>	返回指定表达式相对于窗口框架中最后一行的值。
<code>LEAD</code>	返回分区中当前行之后的第N行的值。如果不存在后续行，则返回NULL。
<code>NTH_VALUE</code>	返回窗口框架第N行的参数值
<code>NTILE</code>	将每个窗口分区的行分配到指定数量的已排名组中。
<code>PERCENT_RANK</code>	计算分区或结果集中的百分位数
<code>RANK</code>	与DENSE_RANK () 函数类似，只是当两行或更多行具有相同的排名时，排序值序列中存在间隙。
<code>ROW_NUMBER</code>	为其分区中的每一行分配一个连续整数

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 Analytical Processing 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🍌：

```
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	4000	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 窗口函数分类

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 (英式排序)

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 (中式排序)

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

聚合函数在开窗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
0002	打孔器	500	1500
0003	运动T恤	4000	5500
0004	菜刀	3000	8500
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
0002	打孔器	500	750.0000000000000000
0003	运动T恤	4000	1833.3333333333333333
0004	菜刀	3000	2125.0000000000000000
0005	高压锅	6800	3060.0000000000000000
0006	叉子	500	2633.3333333333333333
0007	擦菜板	880	2382.8571428571428571
0008	圆珠笔	100	2097.5000000000000000

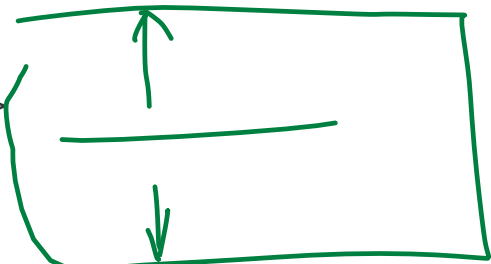
可以看出，聚合函数结果是，按我们指定的排序，这里是 product_id，当前所在行及之前所有的行的合计或均值。即累计到当前行的聚合。

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 (

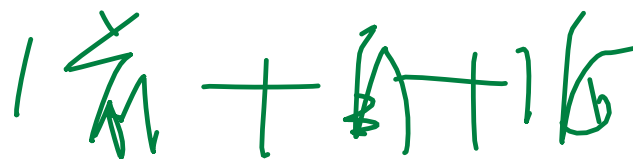
ORDER BY product_id

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:

前 2

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
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
衣服	NULL	4000
衣服	2009-09-20	1000
衣服	NULL	5000
NULL	NULL	16780

product_type	regist_date	sum_price
		16780
厨房用具		11180
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
办公用品	2009-09-11	500
办公用品	2009-11-11	100
衣服		5000
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

练习 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
```

按照 product_id 升序排列，计算出截至当前行的最高 sale_price。

练习 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）

如下两种方法都可以实现：

-- ①regist_date为NULL时，显示“1年1月1日”。

```
SELECT
    regist_date,
    product_name,
    sale_price,
    SUM(sale_price) OVER (
        ORDER BY COALESCE(regist_date, CAST('0001-01-01' AS
DATE))) AS current_sum_price

FROM Product;
```

-- ②regist_date为NULL时，将该记录放在最前显示。

```
SELECT
    regist_date,
    product_name,
    sale_price,
    SUM(sale_price) OVER (
        ORDER BY regist_date NULLS FIRST) AS current_sum_pr
ice
FROM Product;
```

练习 5.3

思考题

① 窗口函数不指定 PARTITION BY 的效果是什么？

窗口函数不指定 PARTITION BY 就是针对排序列进行全局排序。

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

本质上是因为 SQL 语句的执行顺序。

FROM → WHERE → GROUP BY → HAVING → SELECT → ORDER BY

如果在 WHERE, GROUP BY, HAVING 使用了窗口函数，就是说提前进行了一次排序，排序之后再去除 记录、汇总、汇总过滤，第一次排序结果就是错误的，没有实际意义。而 ORDER BY 语句执行顺序在 SELECT 语句之后，自然是可以使用。