**必知必会的窗口函数三：窗口函数案例**

这次的文章我们来看一看，其他相关的窗口函数的典型案例，基于案例我们再次巩固一下窗口函数。

在上次的文章中，我们了解了窗口函数的基本知识，以及如何进行数据“移动”。具体文章可以点击以下链接：

**必知必会的窗口函数一\_基础知识**

1. **ROWS与RANGE范围**

在之前的文章中，我们列举了一些ROWS/RANGE求解数据的范围：

1. ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW

指从起点到当前行范围内的数据

1. ROWS BETWEEN 3 PRECEDING AND 2 FOLLOWING

指当前行开始往前3行到往后2行的数据

1. ROWS BETWEEN 3 PRECEDING AND CURRENT ROW

指当前行的上3行([ROWNUM](https://so.csdn.net/so/search?q=ROWNUM&spm=1001.2101.3001.7020" \t "https://blog.csdn.net/m0_52606060/article/details/_blank)-3)到当前行的数据

1. ROWS BETWEEN CURRENT ROW AND UNBOUNDED FOLLOWING

指当前行以及之后的所有行的数据

1. RANGE BETWEEN CURRENT ROW AND 3 FOLLOWING

指当前行到当前行数据+3的范围内的数据即[now,now+3]

1. RANGE BETWEEN 3 PRECEDING AND 3 FOLLOWING

指当前行数据幅度减3加3后的范围内的数据j即[now-3,now+3]

接下来我们来看一下演示，看看实际上使用了ROWS/RANGE之后，实际上的输出是怎样的。我们的创表语句如下：

**CREATE** **TABLE** CarSale (

car\_id **int**,

sale\_dt **date**,

sale\_count **int**

) ;

**INSERT** **INTO** CarSale **VALUES** (10001, '2023-05-01', 1);

**INSERT** **INTO** CarSale **VALUES** (10001, '2023-05-02', 2);

**INSERT** **INTO** CarSale **VALUES** (10001, '2023-05-03', 3);

**INSERT** **INTO** CarSale **VALUES** (10001, '2023-05-04', 4);

**INSERT** **INTO** CarSale **VALUES** (10001, '2023-06-05', 5);

**INSERT** **INTO** CarSale **VALUES** (10001, '2023-06-06', 6);

**INSERT** **INTO** CarSale **VALUES** (10001, '2023-06-07', 7);

**INSERT** **INTO** CarSale **VALUES** (10001, '2023-06-08', 8);

**INSERT** **INTO** CarSale **VALUES** (10002, '2023-05-01', 10);

**INSERT** **INTO** CarSale **VALUES** (10002, '2023-05-02', 11);

**INSERT** **INTO** CarSale **VALUES** (10002, '2023-05-03', 13);

**INSERT** **INTO** CarSale **VALUES** (10002, '2023-05-03', 14);

**INSERT** **INTO** CarSale **VALUES** (10002, '2023-06-05', 4);

**INSERT** **INTO** CarSale **VALUES** (10002, '2023-06-06', 5);

**INSERT** **INTO** CarSale **VALUES** (10002, '2023-06-07', 6);

**INSERT** **INTO** CarSale **VALUES** (10002, '2023-06-08', 7);

**INSERT** **INTO** CarSale **VALUES** (10002, '2023-06-09', 8);

**INSERT** **INTO** CarSale **VALUES** (10002, '2023-06-10', 9);

那么对于这个输出的结果是多少呢？

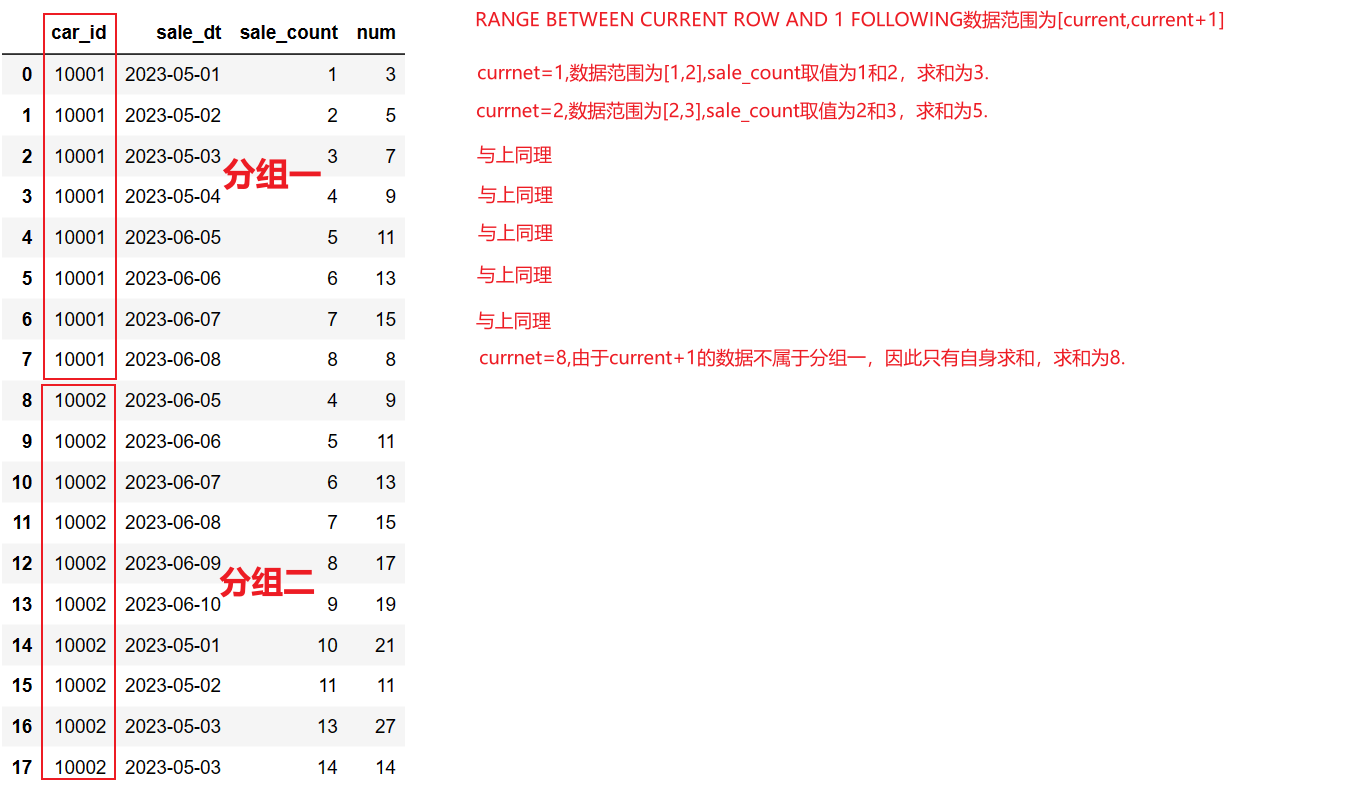
**select** car\_id,sale\_dt,sale\_count,

**sum**(sale\_count) **over**(**partition** **by** car\_id **order** **by** sale\_count **RANGE** **BETWEEN** **CURRENT** **ROW** **AND** 1 **FOLLOWING**) **as** num

**from** CarSale ;

主要的输出效果在于后面的窗口函数，即先对car\_id进行分组之后对sale\_count进行排序，排序之后进行sum(sale\_count) 求和，求和的范围由 RANGE BETWEEN CURRENT ROW AND 1 FOLLOWING来控制，数据范围为[current,current+1]

,即当前行和下一行的数据范围进行求和。我们来看一下输出：



数据输出的结果我在旁边做了注解，这里就不在进行赘述了，大家可以试着使用其他的范围来进行观测。

1. **聚合**

接下来我们来看一下怎么在窗口函数中使用聚合方法，我们先准备数据如下：

**CREATE** **TABLE** **IF** **NOT** **EXISTS** salarywz(

**year** **VARCHAR**,

person\_id **int**,

person\_name **VARCHAR**,

country\_name **VARCHAR**,

salary **int**

);

**INSERT** **INTO** salarywz **VALUES** ('2021',100,'鲁班','战国',10000);

**INSERT** **INTO** salarywz **values** ('2022',100,'鲁班','战国',10001);

**INSERT** **INTO** salarywz **values** ('2023',100,'鲁班','战国',9999);

**INSERT** **INTO** salarywz **values** ('2022',101,'白起','战国',9856);

**INSERT** **INTO** salarywz **values** ('2023',101,'白起','战国',10003);

**INSERT** **INTO** salarywz **values** ('2022',102,'蔡文姬','三国',9857);

**INSERT** **INTO** salarywz **values** ('2023',102,'蔡文姬','三国',10523);

**INSERT** **INTO** salarywz **values** ('2020',102,'蔡文姬','三国',10006);

**INSERT** **INTO** salarywz **values** ('2021',103,'孙悟空','唐朝',10007);

**INSERT** **INTO** salarywz **values** ('2022',103,'孙悟空','唐朝',10568);

**INSERT** **INTO** salarywz **values** ('2023',103,'孙悟空','唐朝',10325);

**INSERT** **INTO** salarywz **values** ('2020',103,'孙悟空','唐朝',10010);

展示如下：



首先，我们看一个简单聚合案例：

**求解各个country\_name下金额的最小值**

很简单，我们可以写出如下的语句：

select \*,

MAX(salary) OVER() as max\_salay,

MIN(salary) OVER(partition by country\_name) as min\_salary\_country

from salarywz

输出如下：



在使用窗口函数的过程中，如果over()中不添加参数，则默认对所有数据进行求和。注意上述的结果是按照country\_name进行分组的，因此才会出现person\_name为鲁班所在行出现了min\_salary\_country为9856的值。

那么我们如何取出每一个country\_name分组下，各个人名对应的最小的min\_salary\_country值呢？我们可以使用子查询进行实现：

**SELECT** person\_name,country\_name, **MIN**(min\_salary\_country) **AS** min\_salary\_country

**FROM** (

**SELECT** person\_name,country\_name, **MIN**(salary) **OVER** (**PARTITION** **BY** country\_name) **AS** min\_salary\_country

**FROM** salarywz

) **AS** subquery

**GROUP** **BY** person\_name,country\_name;

输出如下:



这样我就获取了基于country\_name进行分组后的每一个person对应的min\_salary\_country的值。实际上鲁班在这三年的最低min\_salary\_country为9999。聪明的你思考一下这是为什么。

怎么获取每一个人的最小是呢，答案如下：

**SELECT** **distinct** \*

**FROM** (

**SELECT** person\_name, **MIN**(salary) **OVER** (**PARTITION** **BY** person\_name) **AS** min\_salary

**FROM** salarywz

) **AS** subquery

<https://www.runoob.com/postgresql/postgresql-datetime.html>

<https://learnsql.com/blog/sql-window-functions-rows-clause/>

<https://www.zhihu.com/question/596882478?utm_id=0>

<https://zhuanlan.zhihu.com/p/351822793>

<https://zhuanlan.zhihu.com/p/622993495>

<https://zhuanlan.zhihu.com/p/260182391>

