**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？

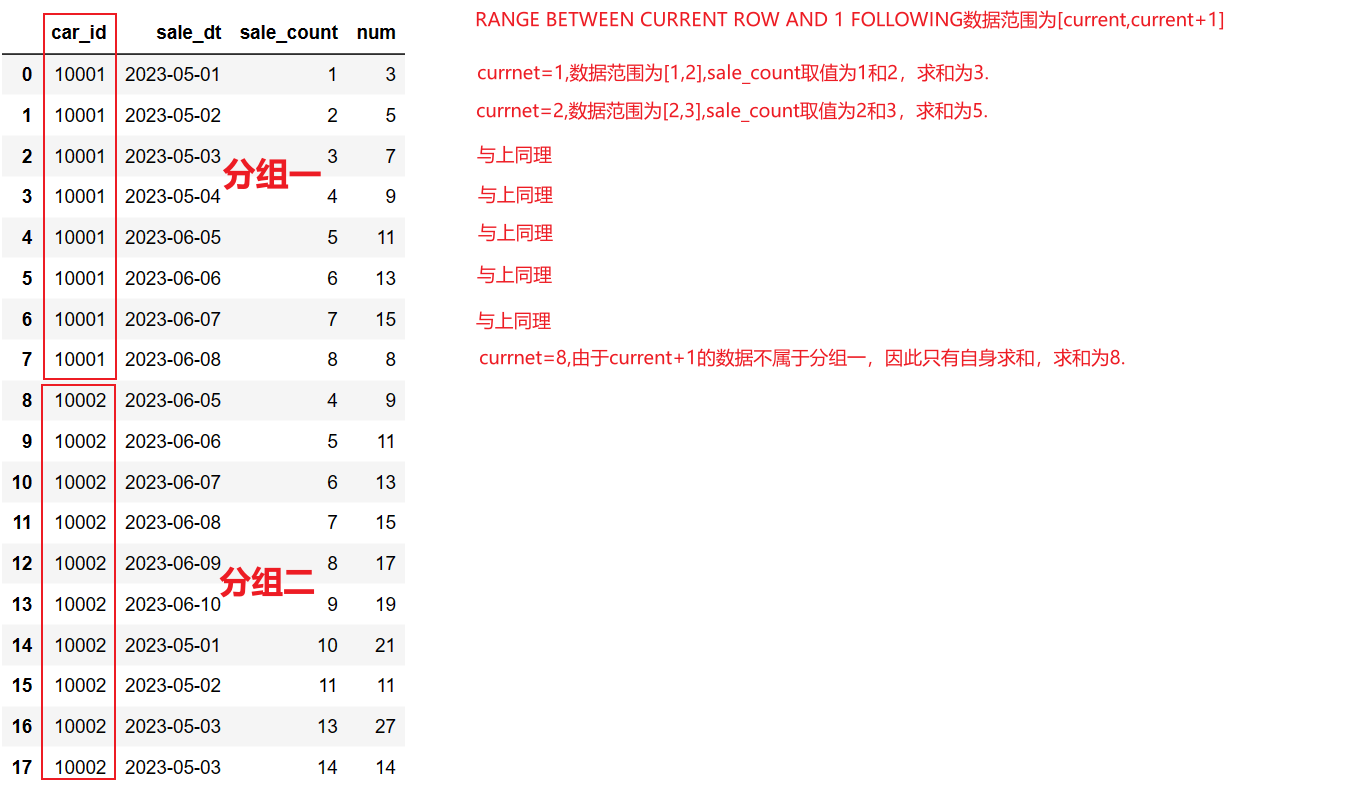
**select** car\_id,sale\_dt,sale\_count,

**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**,

salay **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：



注意的结果中是按照country\_name进行分组的，因此会出现一些“想法上”的错误，比如鲁班的min\_salary\_country是9856，这与他实际的min salary为9999，是有一些错觉的。

为了得到每个人，在2020年02023年期间salary的最小值，我们可以使用以下语句进行查询：

**select** person\_name,A.min\_salary **from**

(**select** \*,

**MIN**(salary) **OVER**(**partition** **by** person\_name) **as** min\_salary

**from** salarywz

) **as** A

**group** **by** 1,2

这里使用了子查询，大家可以换其他的查询也是可以的。这里就不在展示结果了。

**对国家降序和升序进行累加金额求和**

如果我们对每个人生活的年代进行分组，多工资进行降序和升序的排序，我们可以写出这样的代码：

**select** \*,

**sum**(salary) **OVER**() **as** total\_salary,

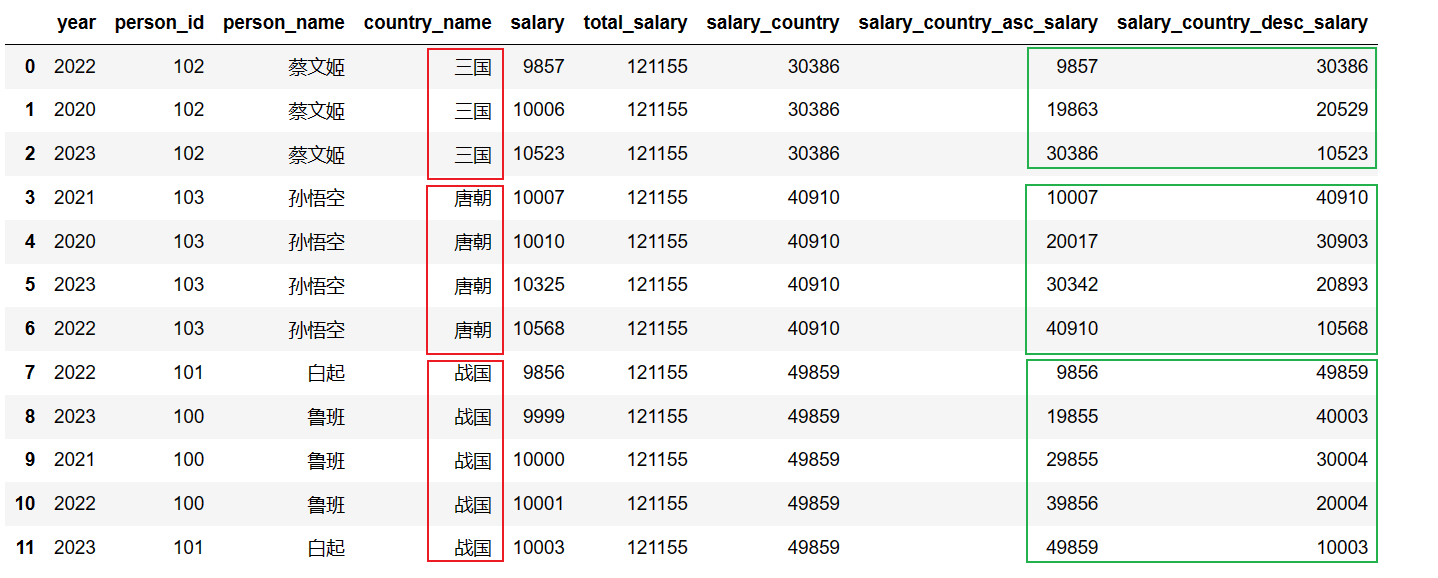
**sum**(salary) **OVER**(**partition** **by** country\_name) **as** salary\_country,

**sum**(salary) **OVER**(**partition** **by** country\_name **order** **by** salary **ASC**) **as** salary\_country\_asc\_salary,

**sum**(salary) **OVER**(**partition** **by** country\_name **order** **by** salary **DESC**) **as** salary\_country\_desc\_salary

**from** salarywz;

输出如下：



大家注意，由于我们没有使用ROWS/RANGE关键字，因此我们的窗口是所有行的数据，对于使用了分组关键词则窗口是分组内的所有行。

对于其他的聚合函数，如count、avg大家可以自己试一试了。

1. **排序函数**

排序也是SQL或者说在窗口行数中使用到的最为频繁的一个操作，在使用窗口函数排序的时候，我们比较常用的是以下几个函数：

（1）ROW\_NUMBER()：为窗口中的每一行分配一个唯一的数字。经常与OVER子句结合使用来定义排序和窗口。

ROW\_NUMBER() OVER (ORDER BY some\_column)

1. RANK()：为窗口中的每一行分配一个唯一的数字，但如果值相同则会跳过某些数字。例如，1, 1, 3, 4。

RANK() OVER (ORDER BY some\_column)

1. DENSE\_RANK()：与RANK()相似，但不会跳过任何数字。例如，1, 1, 2, 3。

DENSE\_RANK() OVER (ORDER BY some\_column

1. NTILE(n)：将窗口中的行分为n个大致相等的部分，并为每一行分配一个桶号，从1到n。

NTILE(4) OVER (ORDER BY some\_column)

（5）LEAD() 和 LAG()：这两个函数允许你访问窗口中相对于当前行的前一行或后一行的值。

LEAD(column, n, default) 返回当前行后第n行的值。

LAG(column, n, default) 返回当前行前第n行的值。

例如，如果你想比较当前行的值与下一行的值，可以使用这样的代码：

SELECT column, LEAD(column) OVER (ORDER BY some\_column) AS next\_value FROM table

（6）FIRST\_VALUE() 和 LAST\_VALUE()：返回窗口中第一行或最后一行的值。

FIRST\_VALUE(column) OVER (ORDER BY some\_column)

LAST\_VALUE(column) OVER (ORDER BY some\_column)

（7）CUME\_DIST()：返回当前行的值在窗口中的相对位置，结果是一个0到1之间的数。

**3.1 ROW\_NUMBER/RANK/DENSE\_RANK**

为了演示上述窗口函数的差异，这里我们先**UPDATE**一下数据，将同组下的salary

设置一组一样的数据：

**UPDATE** salarywz

**SET** salary = 10325

**WHERE** **year** = '2022'

**and** person\_name = '孙悟空';

之后，我们来演示三这个排序函数的差异性：

**select** \*,

**row\_number**() **over** (**partition** **by** person\_id **order** **by** salary **DESC**) **as** odery\_row\_number,

**rank**() **over** (**partition** **by** person\_id **order** **by** salary **DESC**) **as** odery\_rank,

**dense\_rank**() **over** (**partition** **by** person\_id **order** **by** salary **DESC**) **as** odery\_dense\_rank

**from** salarywz;

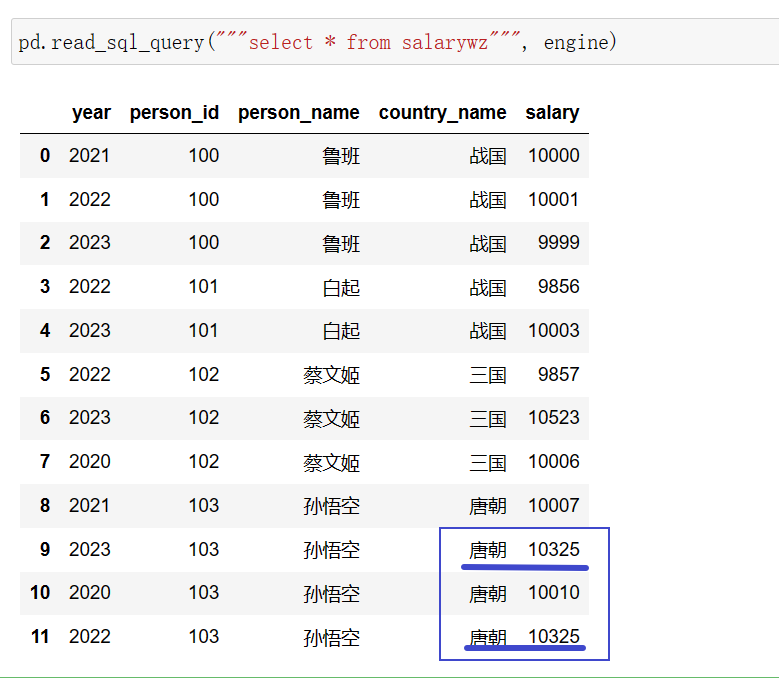
输出如下：



可以看出，row\_number函数从1开始顺序编号；rank函数也是顺序编号只不过遇到相同的排序时会留下空位，比如说1,1,3,4，空位为2；dense\_rank也是顺序编号只不过遇到相同的排序时不会会留下空位，比如说1,1,2,3。理解起来还是比较简单的。

**3.2 FIRST\_VALUE/LAST\_VALUE**

我们再来看一下FIRST\_VALUE()和LAST\_VALUE()这两个函数，首先first\_value这个函数获取的数据是分组中当前行的第一个值，而last\_value是分组中当前行的最后一个值。现在我们的数据如下所示：



那么对于这样的窗口的函数的值是怎样的呢？

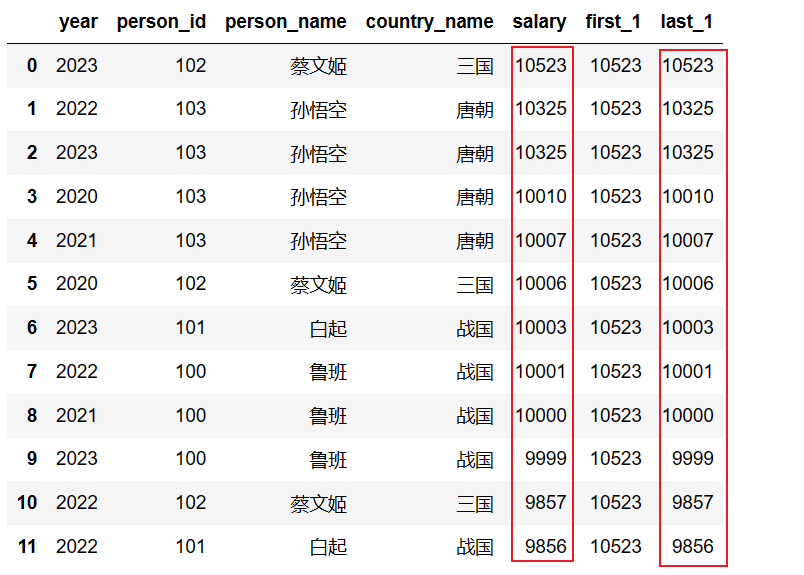
**select** \*,

**FIRST\_VALUE**(salary) **over** (**order** **by** salary **DESC**) **as** first\_1,

**LAST\_VALUE**(salary) **over** (**order** **by** salary **DESC**) **as** last\_1

**from** salarywz;

输出：



我们来解释一下上述结果：

1. 对于first的值，由于我们这里没有使用partition by 关键字，因此数据范围可以认为是整个表数据，使用**order** **by** salary **DESC**之后，第一个数据是(蔡文姬,10523)，最后一个数据是(白起,9856)。因此first\_1的结果均为10523。
2. 对于last\_1的不是相同的值。这是因为窗口语句有一个默认的规则，使用last\_value函数生效的范围是第一行至当前行，我们分步骤进行检验：
3. 查到到第1行，salary=10532。数据范围为[10532]，结果为10532。
4. 查询到第2行，salary=10532，10325。数据范围为[10532,10325]，最后一个值是10325，结果为10325；
5. 查询到第3行，salary=10532，10325，10325。数据范围为[10532,10325,10325]，最后一个值是10325，结果为10325。
6. 依次类推

我们我们使用分组：

**select** \*,

**FIRST\_VALUE**(salary) **over** (**partition** **by** country\_name **order** **by** salary **DESC**) **as** first\_11,

**LAST\_VALUE**(salary) **over** (**partition** **by** country\_name **order** **by** salary **DESC**) **as** last\_11

**from** salarywz;



上述中我们对country\_name进行了分组，之后对分组排序后的数据进行计算FIRST\_VALUE和LAST\_VALUE。

我们也可以进行这样的自定义分组：

**select** \*,

**FIRST\_VALUE**(salary) **over** (**partition** **by** country\_name **order** **by** salary **desc** **RANGE** **BETWEEN** **CURRENT** **ROW** **AND** 3 **FOLLOWING** ) **as** first\_111,

**LAST\_VALUE**(salary) **over** (**partition** **by** country\_name **order** **by** salary **desc** **RANGE** **BETWEEN** **CURRENT** **ROW** **AND** 3 **FOLLOWING** ) **as** last\_111

**from** salarywz;

RANGE BETWEEN CURRENT ROW AND 3 FOLLOWING表示我们的窗口的数据范围为[CURRENT ,CURRENT +3]的范围。

因此我们得到结果：



大家可以自己进行详细的分析和研究。

**3.3 ntile**

ntile(n)函数用于将数据按照顺序进行分桶，参数n表示分桶的数量，同时返回每一行数据所属的编号，如果是在使用**partition** **by**关键词之后进行该函数操作，即表示对当前组内的数据进行分桶。

**select** \*,

**NTILE**(3) **over** (**order** **by** salary **DESC**) **as** ntil\_1

**from** salarywz;



**select** \*,

**NTILE**(3) **over** (**partition** **by** country\_name **order** **by** salary **DESC**) **as** ntil\_2

**from** salarywz;



利用ntile函数，我们可以将某一字段（假设为金额）降序将其分为5组，则桶号为1，2的则是前80%的财富（好像分析的没毛病）。

**3.4 LEAD/LAG**

这两个函数也是比较常用的，这两个函数是两个偏移量函数，主要用于查找当前行字段的上一个值或者下一个值。lead函数是向下取值，lag函数是向上取值，如果向上取值或向下取值没有数据的时候则使用NULL替代。

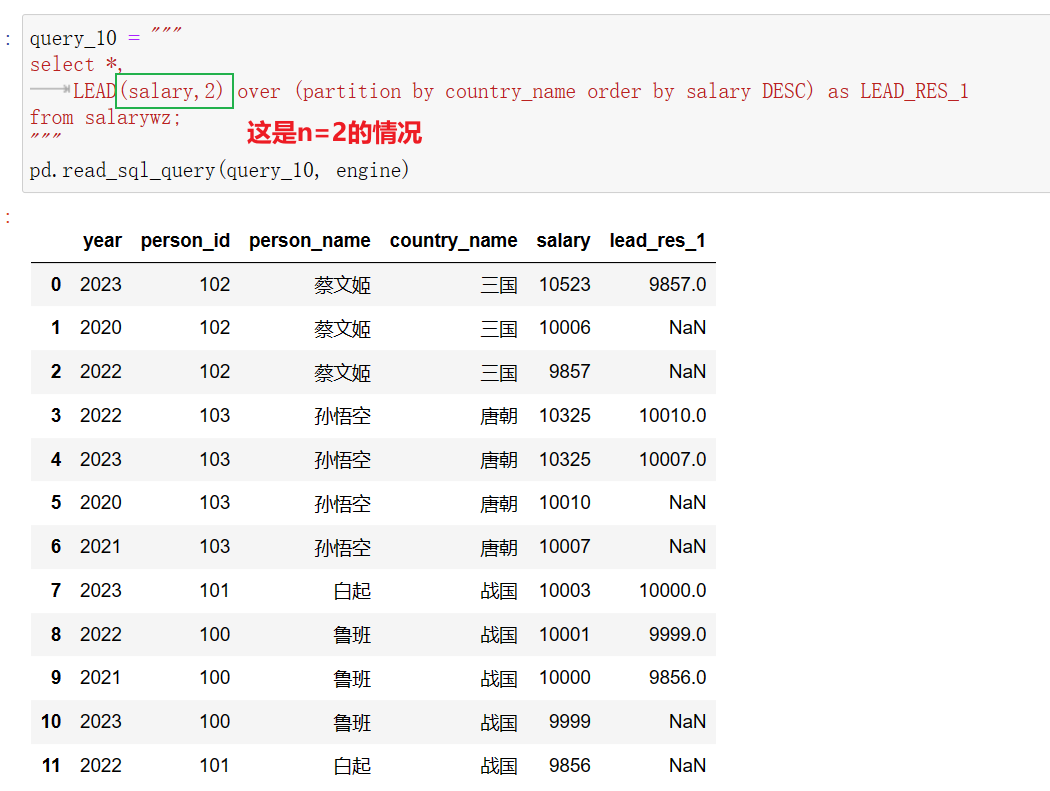
**select** \*,

**LEAD**(salary) **over** (**partition** **by** country\_name **order** **by** salary **DESC**) **as** LEAD\_RES\_1

**from** salarywz;



我们分析了country\_name为三国的分组，这是参数n=2的情况：



对于LAG函数也是一样的分析：

**select** \*,

**LAG**(salary) **over** (**partition** **by** country\_name **order** **by** salary **DESC**) **as** LEAD\_RES\_1

**from** salarywz;



1. **总结**

窗口函数在实际的SQL中还是经常使用到的，比如常见的Top N问题。因此掌握好窗口函数的用法对于提升SQL能力还是非常有帮助的，对于这些窗口函数笔者也就对其中几个比较熟悉，主要是使用的频率，希望在后面的学习中自己也加强一下，这次的学习先到这。

<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>

