**必知必会的窗口函数一：基础知识**

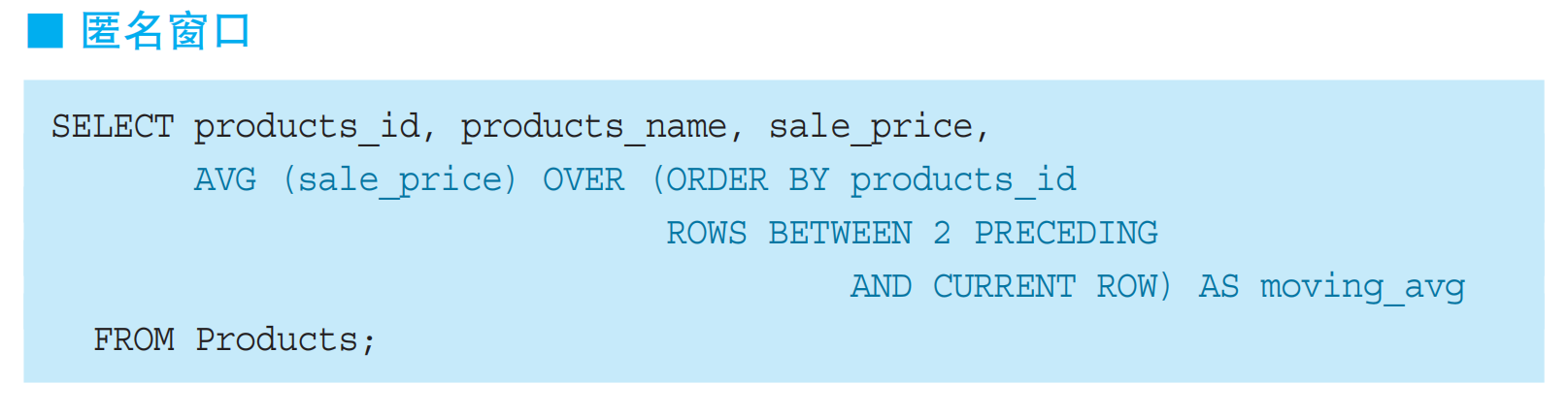
这次的文章我们来看一看，《SQL进阶教程》第二章节的知识点-窗口函数。在学习的过程中我们将对某一些案例使用Pandas来重写和复现。这样就能对窗口函数有一个更为深度的认识。

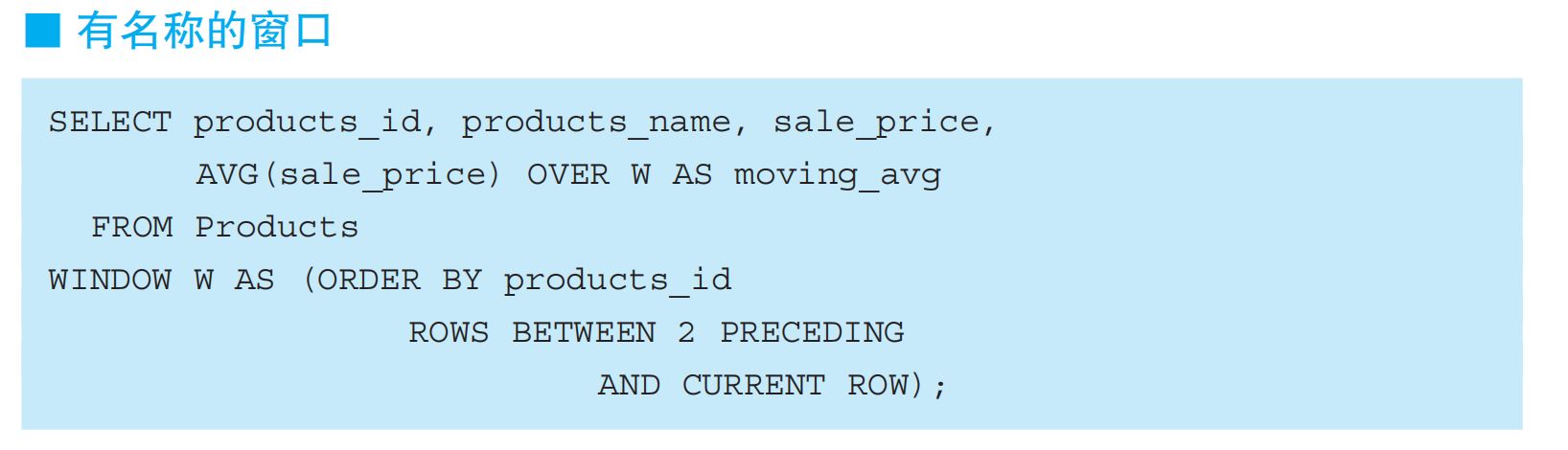
1. **什么是窗口函数**在SQL中，窗口函数（Window Function）是一种特殊的函数，用于在查询结果集的窗口（即一组相关的行）上执行计算。

从它的名字可以看出-窗口函数。其运算是在一个“窗口”上进行的。而这个窗口是具有滑动属性的。一个滑动窗口是一个移动变化的小区间，所以窗口函数可以在不断变化的小区间里实现各种复杂的统计分析，统计的数据范围也在不断地变化，是一种非常重要且常用的分析OLAP（Online Analytical Processing) 联机分析处理函数。

我们记住它的主要特点就是在每行上执行计算，并根据指定的排序规则和窗口范围来聚合数据。与聚合函数（如SUM、AVG等）不同，窗口函数可以基于每行的值执行计算，而聚合函数是在整个结果集上执行计算。

我们来看两个不同的窗口函数（下图所示），WINDOW表示窗口，其中一个是匿名窗口，一个是有名称的窗口。从结果上看两个效果一样，我们暂且不分析其中的SQL语句的含义和作用。





注：图片来源《SQL进阶教程》

匿名窗口使用比较简约，从一定程度上我们的代码量比较少；而使用有名称的窗口函数的好处是窗口函数可以重复使用。在有一些数据库管理系统中，有一些窗口函数是不支持的，这点注意。

**常用的窗口函数主要以下几类：**  
（1）排序函数：rank(), dense\_rank()、row\_number()等

1. 聚合函数：max(),min(),sum(),avg()等
2. 位移：lead()、lag()等
3. 分箱：ntile()等
4. 等等
5. **窗口函数语法**

MYSQL8.0中的窗口函数的语法如下：

**<**窗口函数**>** OVER (

[**PARTITION BY** **<**用于分组的列名**>**]

**ORDER** **BY** **<**用于排序的列名**>**

**[window\_frame]**

)

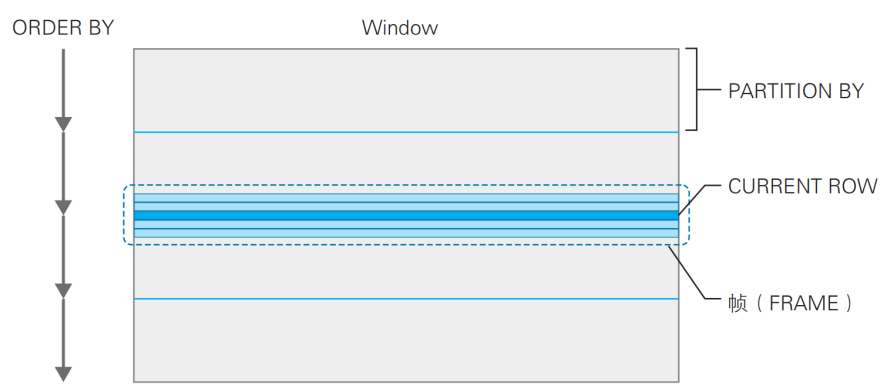
我们来解释上述几个方法的含义：

（1）partition by用于对数据分组。

（2）order by用于指定窗口范围内的数据排序方式。

（3）window\_frame用于指定在每个分区中用于窗口函数的行范围。

《SQL进阶教程》中使用下图来解释窗口函数的功能：



从图中可以看出，窗口函数使用帧子句（FRAME）定义以当前记录为中心的子集，帧子句可以定义窗口函数的输入是当前行的前后多少行，或者是当前行所在的分区的开始或结束。其一般的语法如下：

**[** RANGE | ROWS **] frame\_start**

**[** RANGE | ROWS **]** BETWEEN **frame\_start** AND **frame\_end**

其中，frame\_start 和 frame\_end 可以是以下之一：

（1）UNBOUNDED PRECEDING：分区的第一行  
（2）UNBOUNDED FOLLOWING：分区的最后一行

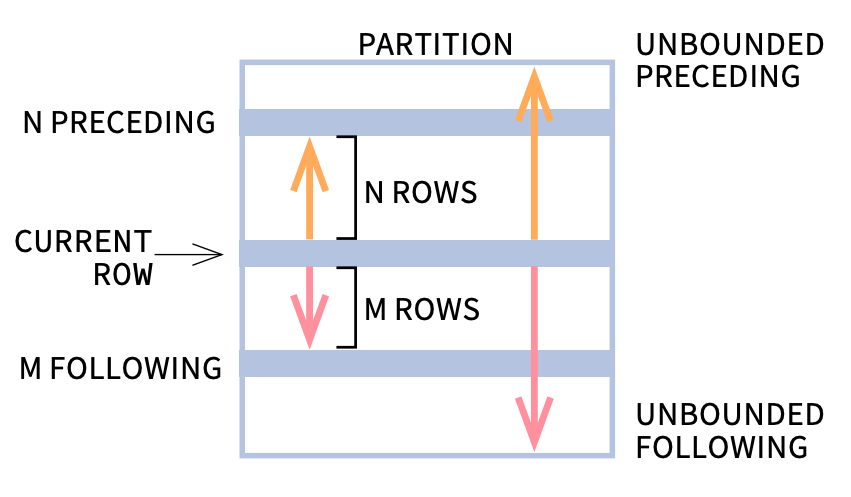
（3）CURRENT ROW：当前行

（4）n PRECEDING：仅向前行移动n行

（5）n FOLLOWING：仅向后行移动n行

（6）RANGE 和 ROWS 是用于指定帧边界的模式。从两个关键词的字面意思不难理解。RANGE 模式表示帧边界是根据排序键的值来确定的，而 ROWS 模式表示帧边界是根据行的位置来确定的。也就是说ROWS表示在当前行之前或之后的行数，而RANGE表示窗口的范围。

不管怎样，ROWS/RANGE BETWEEN表示窗口范围的定义，也就是确定当前窗口包含哪些数据。参考文献3中有一个比较好的示意图和大家分享一下：



**按照帧字句的语法规则我们有：**

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

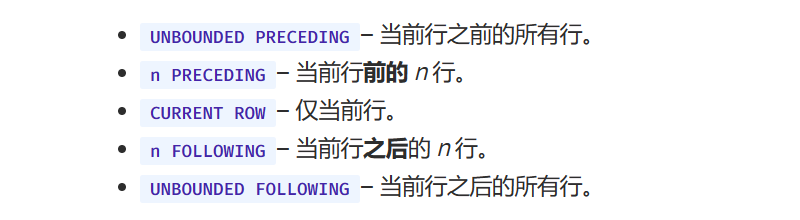
1. RANGE BETWEEN CURRENT ROW AND 3 FOLLOWING

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]

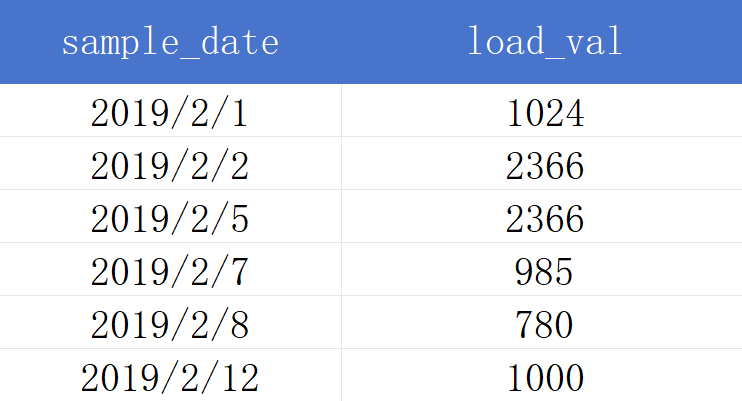
**总结一下就是：**



在之后的练习中我们会使用到这些范围来选取数据进行计算。接下来我们来看一些案例，案例来源于参考文献1。

1. **案例一：使用帧子句将其他行移至当前行**

帧子句的作用是能通过SQL简单的计算出移动平均值等以当前记录为基准计算的统计指标。此处我们可以借助帧子句将其他行移至当前行，假设现在我们有表LoadSample且数据如下：



该表的创建语句如下：

**CREATE** **TABLE** LoadSample (

sample\_date **date** **NOT** **NULL**,

loadval **int** **NOT** **NULL**

);

**INSERT** **INTO** LoadSample **VALUES**(**date**'2019-02-01',1024);

**INSERT** **INTO** LoadSample **VALUES**(**date**'2019-02-02',2366);

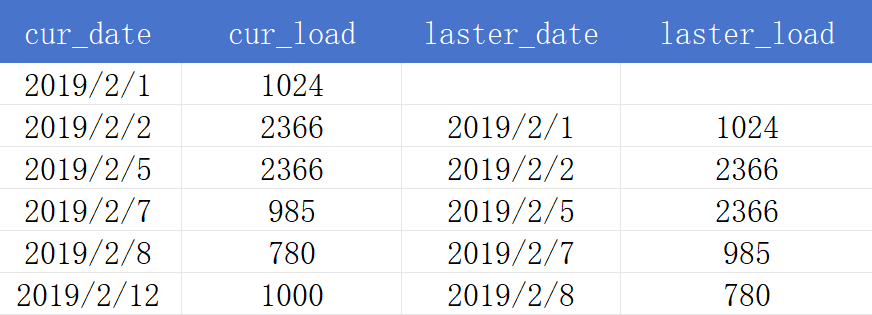
**INSERT** **INTO** LoadSample **VALUES**(**date**'2019-02-05',2366);

**INSERT** **INTO** LoadSample **VALUES**(**date**'2019-02-07',985);

**INSERT** **INTO** LoadSample **VALUES**(**date**'2019-02-08',780);

**INSERT** **INTO** LoadSample **VALUES**(**date**'2019-02-12',1000);

现在要求完成一个基本的时间序列分析问题，要求对原表的数据进行偏移，然后形成如下的一个结果：



从结果上可以看出，这个结果相当于从2019/2/1开始整体“后移”一步。

**实现**

这个实现不是很难，如果根据规则去取数肯定是可以完成的，那我们如何使用窗口函数和帧子句来完成呢？透过现象看本质，我们先按照教材上的步骤，先处理时间：

**select** sample\_date **as** cur\_date,

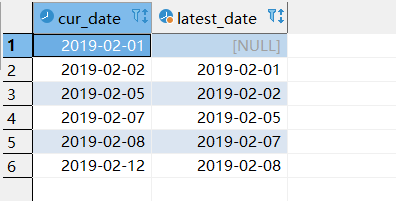
**min**(sample\_date)

**over** (**order** **by** sample\_date **asc**

**rows** **between** 1 **preceding** **and** 1 **preceding**) **as** latest\_date

**from** loadsample

输出如下：



语句中的“**rows** **between** 1 **preceding** **and** 1 **preceding**”即用于计算时间间隔或计算当前行与前一行之间的差异[。在这种情况下，1 PRECEDING 表示当前行的前一行，因此 BETWEEN 1 PRECEDING AND 1 PRECEDING 实际上是指当前行和其前一行之间的值。](https://stackoverflow.com/questions/48784722/rows-between-1-preceding-and-preceding-1" \t "https://copilot.microsoft.com/_blank)

由于该表中没有2019-02-01之前的数据，因此2019-02-01这行的日期为NULL。当时间日期为2019-02-02（当前行）的时候，其前一行的数据日期为2019-02-01，这样就形成了一个类似一个范围的数据。

对于load\_val字段的计算也是一样：

**select** load\_val **as** cur\_load,

**min**(load\_val)

**over** (**order** **by** load\_val **asc**

**rows** **between** 1 **preceding** **and** 1 **preceding**) **as** latest\_load

**from** loadsample

我们将两者的计算合在一起：

**select** sample\_date **as** cur\_date,load\_val **as** cur\_load,

**min**(sample\_date)

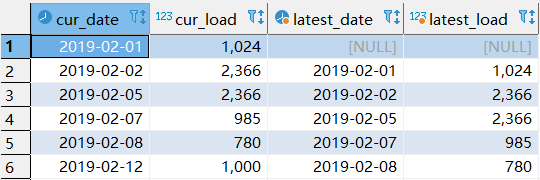
**over** (**order** **by** sample\_date **asc** **rows** **between** 1 **preceding** **and** 1 **preceding**) **as** latest\_date,

**min**(load\_val)

**over** (**order** **by** sample\_date **asc** **rows** **between** 1 **preceding** **and** 1 **preceding**) **as** latest\_load

**from** loadsample

输出如下：



如果使用有名窗口，上述SQL语句可以优化为：

**select** sample\_date **as** cur\_date,load\_val **as** cur\_load,

**min**(sample\_date) **over** w **as** laster\_date,

**min**(load\_val) **over** w **as** laster\_load

**from** loadsample

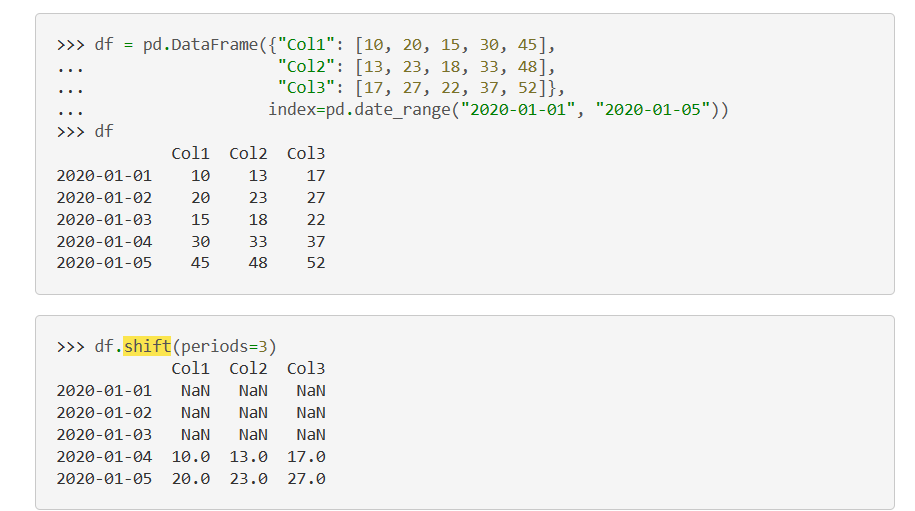
**window** w **as** (**order** **by** sample\_date **asc** **rows** **between** 1 **preceding** **and** 1 **preceding**)

**Pandas实现**

使用Pandas来实现这个案例就变得非常简单了，我们可以直接使用shift方法，关于shift的相关方法，大家可以参考下述链接进行学习：

<https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.shift.html#pandas.DataFrame.shift>

比如官方文档里面就有一个比较好的案例：



使用period参数就可以进行数据的“移动”。那么关于这个案例，使用pandas的实现就可以这样：

import pandas as pd

df = pd.read\_sql\_query('SELECT \* FROM LoadSample', engine)

# sample\_date排序

df = df.sort\_values('sample\_date')

df['latest\_date'] = df['sample\_date'].shift(periods=1)

df['latest\_load'] = df['load\_val'].shift(periods=1)

# 重命名

df = df.rename(columns={'sample\_date': 'cur\_date', 'load\_val': 'cur\_load'})

df

输出这里就不在贴上了，上述是使用shift进行实现的，感兴趣的小伙伴想想看怎么使用pandas的rolling()函数来实现这个需求。

1. **案例一拓展：设置“一天前”或“两天前”**

前面说到RANGE 模式表示帧边界是根据排序键的值来确定的，而 ROWS 模式表示帧边界是根据行的位置来确定的。使用RANGE可以完成下面的效果：

**select** sample\_date **as** cur\_date,load\_val **as** cur\_load,

**min**(sample\_date)

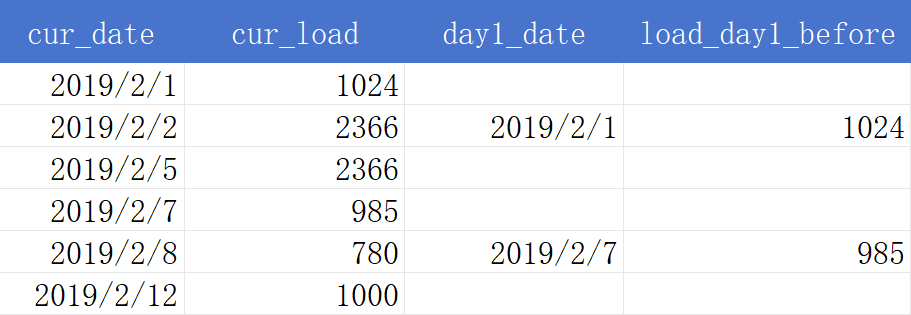
**over** (**order** **by** sample\_date **asc** **range** **between** **interval** '1' **day** **preceding** **and** **interval** '1' **preceding**) **as** day1\_before,

**min**(load\_val)

**over** (**order** **by** sample\_date **asc** **range** **between** **interval** '1' **day** **preceding** **and** **interval** '1' **preceding**) **as** load\_day1\_before

**from** loadsample

输出如下：



上述**interval** '1' **day**是根据日期类型指定日期间隔的语法。由于Loadsample中的数据不是连续的，如果没有前一天的数据，结果中day1\_date ,load\_day1\_before列中就会显示NULL。

那么在pandas中我们该如何实现这个需求呢，直接看一下代码：  
import pandas as pd

df = pd.read\_sql\_query('SELECT \* FROM LoadSample', engine)

# 使用shift函数

df['day1\_before'] = df.loc[df['sample\_date'].shift() == df['sample\_date'] - pd.Timedelta(days=1), 'sample\_date']

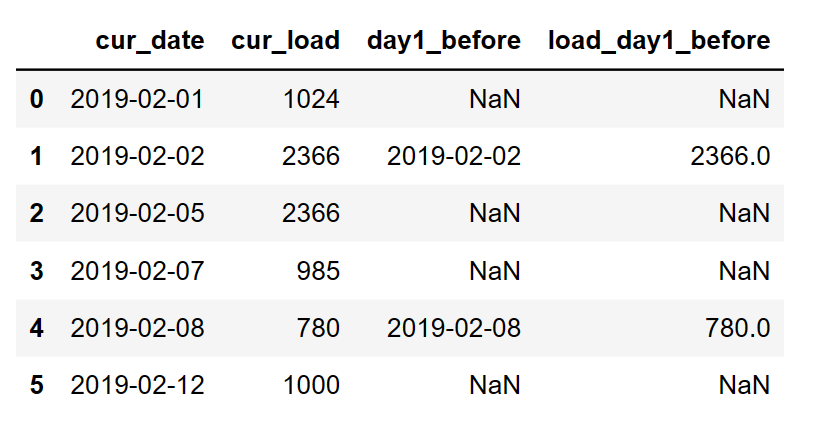
df['load\_day1\_before'] = df.loc[df['sample\_date'].shift() == df['sample\_date'] - pd.Timedelta(days=1), 'load\_val']

# 重命名列

df = df.rename(columns={'sample\_date': 'cur\_date', 'load\_val': 'cur\_load'})

df

输出如下：



使用Pandas解决这个问题的思路为首先按sample\_date对数据框进行排序，然后使用shift()函数和条件索引获取前一天的sample\_date和load\_val。思路上和SQL的解决方法是一致的，大家可以使用rolling()来实现，至此我们就完成了窗口函数的一些知识点的学习。

后面我们接续来看窗口函数相关的实战案例。

1. **参考文献**
2. MICK-《SQL进阶教程》第2版
3. Joe Celko-《SQL权威指南》
4. <https://learnsql.com/blog/sql-window-functions-rows-clause/>
5. http://pandas.pydata.org/docs/user\_guide/index.html