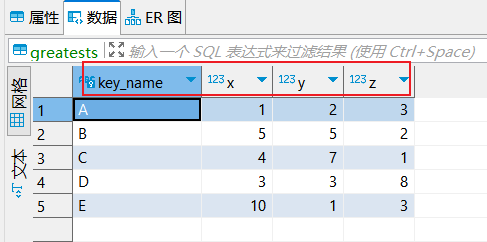
**SQL Case表达式习题**

这次的文章我们来看一看，《SQL进阶教程》第一章节Case表达式后面的习题，在学习该习题的同时，我将演示如何使用Pandas来实现相应的功能，这样就能对Case表达式有一个更为深度的认识。

1. **案例一-多列数据的最大值**

在某一些场景下，我们需要对一个表的数据进行大小判断。比较常见的就是从读个字段中去取出最大的一个数值。我们知道对一个字段进行最大值最小值的选择使用MAX()和MIN()函数就可以完成。

现在我们有这样的一个greatests表，它有3列数据：



该表的创建语句如下：

**CREATE** **TABLE** Greatests

(key\_name **VARCHAR**(16) **PRIMARY** **KEY**,

x **INTEGER**,

y **INTEGER**,

z **INTEGER**

);

**INSERT** **INTO** Greatests **VALUES**('A', 1, 2, 3);

**INSERT** **INTO** Greatests **VALUES**('B', 5, 5, 2);

**INSERT** **INTO** Greatests **VALUES**('C', 4, 7, 1);

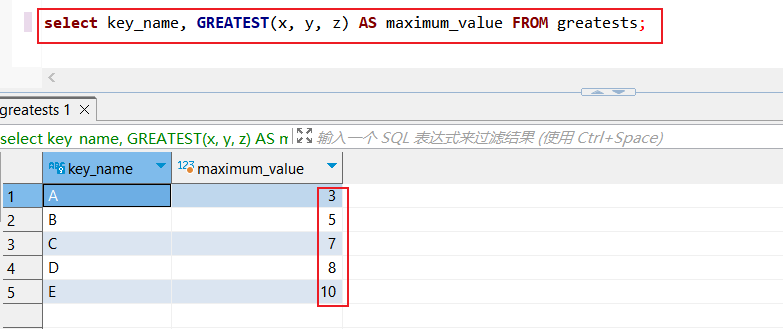
**INSERT** **INTO** Greatests **VALUES**('D', 3, 3, 8);

**INSERT** **INTO** Greatests **VALUES**('E', 10, 1, 3);

现在要对x,y,z三列数据进行判断，找出其中的最大值。

在PostgreSQL中有一个函数可以直接得出结果：

**select** key\_name, **GREATEST**(x, y, z) **AS** maximum\_value **FROM** greatests;



那么如果不使用自带的函数方法，该怎么是现在这个需求呢？我们使用两两比较的方式进行判断，我们可以写出这样的语句：

**select** key\_name,

**CASE**

**WHEN** x >= y **AND** x >= z **THEN** x

**WHEN** y >= x **AND** y >= z **THEN** y

**else** z

**END** **AS** **greatest**

**FROM** greatests;

另外一种方法可以从数学的思想中得出，学过矩阵的小伙伴可能就知道了，我们可以借用转置来进行求解，也就是说先进行行列转换，然后使用MAX函数来求解，我们想讲三行数据装换为列：

**select** key\_name,x **as** value

**from** Greatests

**union** **all**

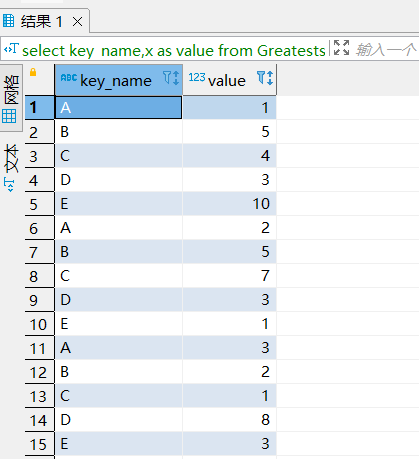
**select** key\_name,y **as** value

**from** Greatests

**union** **all**

**select** key\_name,z **as** value

**from** Greatests



然后再对其中key\_name进行group by求解最大值：

**SELECT** key\_name, **MAX**(value) **AS** **greatest**

**FROM** (**SELECT** key\_name, x **AS** value **FROM** Greatests

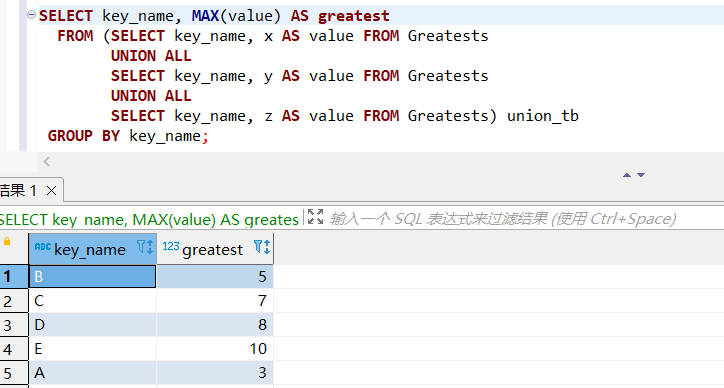
**UNION** **ALL**

**SELECT** key\_name, y **AS** value **FROM** Greatests

**UNION** **ALL**

**SELECT** key\_name, z **AS** value **FROM** Greatests) union\_tb

**GROUP** **BY** key\_name;

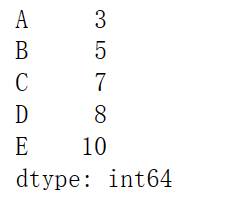


1. **案例一的pandas实现**

上述问题，使用pandas实现就变得非常简单了，因为我们的问题可以转换为找出每一行的最大值即可，这里我们可以使用dataframe.max()来实现，具体如下：

import pandas as pd  
# 创建一个dataframe，数据样式如下  
df = pd.DataFrame({  
    'x': [1, 5, 4, 3, 10],  
    'y': [2, 5, 7, 3, 1],  
    'z': [3, 2, 1, 8, 3]  
}, index=['A', 'B', 'C', 'D', 'E'])  
# 使用max()方法，指定axis=1参数，找到每一行的最大值  
max\_values = df.max(axis=1)  
max\_values

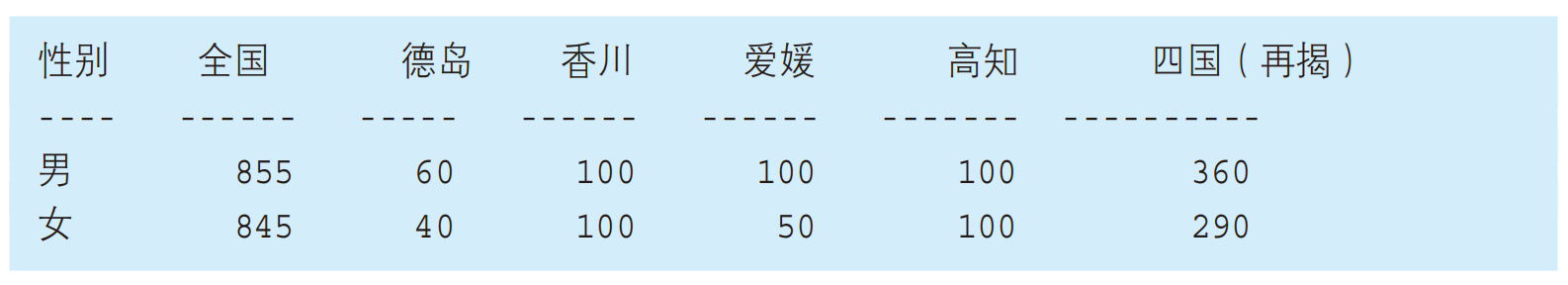
返回的是一个Series：



如果将上述代码中的df.max(axis=1)更改为df.max(axis=0)即可求解出每一列的最大值，大家可以试试。这里使用Pandas进行时间十分简单，这里就不在进行赘述了，我们来看第二个习题。

1. **案例二：转换行列-在表头里加入汇总和再揭**

将表PopTbl2作为样本数据，把行结构的数据转换为列结构的数据，结果如下：



注意，四国地区为德岛、香川、爱媛、高知。

我们先看看PopTbl2的数据样式：



稍微分析一下，结果表是两行数据，且都是性别的汇总数据，因此我们需要对sex进行group by，SQL的实现代码如下：

**SELECT** **CASE** **WHEN** sex = '1'

**THEN** '男' **ELSE** '女' **END** **AS** sex,

**SUM**(population) **AS** 全国,

**SUM**(**CASE** **WHEN** pref\_name = '德岛'

**THEN** population **ELSE** 0 **END**) **AS** 德岛,

**SUM**(**CASE** **WHEN** pref\_name = '香川'

**THEN** population **ELSE** 0 **END**) **AS** 香川,

**SUM**(**CASE** **WHEN** pref\_name = '爱媛'

**THEN** population **ELSE** 0 **END**) **AS** 爱媛,

**SUM**(**CASE** **WHEN** pref\_name = '高知'

**THEN** population **ELSE** 0 **END**) **AS** 高知,

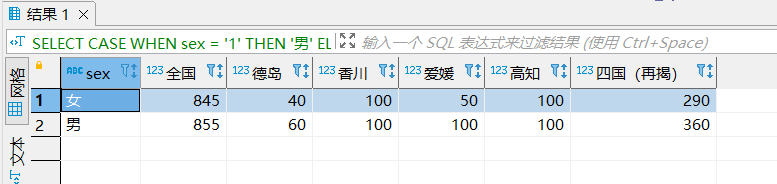
**SUM**(**CASE** **WHEN** pref\_name **IN** ('德岛','香川','爱媛','高知' )

**THEN** population **ELSE** 0 **END**) **AS** 四国（再揭）

**FROM** PopTbl2

**GROUP** **BY** sex;

输出如下：



1. **案例二的pandas实现**

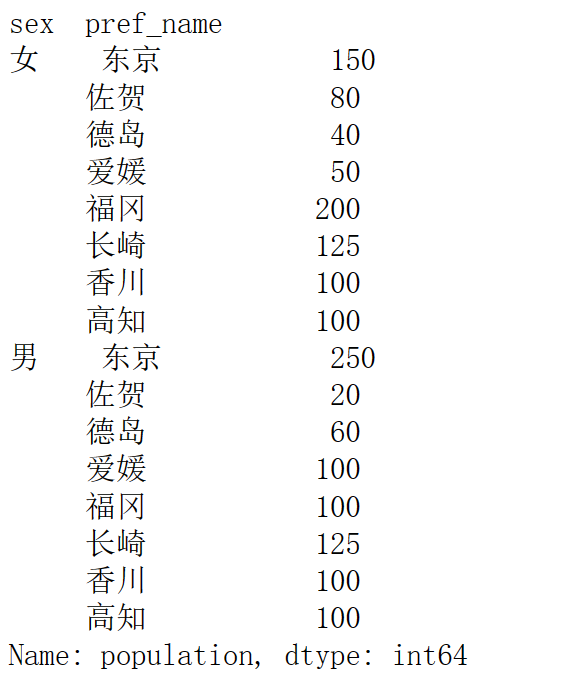
接下来我们尝试使用pandas来实现这个代码，先看代码：

import pandas as pd   
# 列映射   
PopTbl2 = pd.read\_sql\_query('SELECT \* FROM PopTbl2', engine)   
PopTbl2['sex'] = PopTbl2['sex'].map({'1': '男', '2': '女'})   
  
savecol = ['德岛', '香川', '爱媛', '高知']  
# 按tsex列进行分组，并计算总人口   
grouped = PopTbl2.groupby('sex')   
nationwide = grouped.agg({'population': 'sum'})   
  
# 各地区人数  
region\_population = PopTbl2.groupby(['sex', 'pref\_name'])['population'].sum()  
# 将pref\_name列转换为列索引，得到一个透视表  
region\_population = region\_population.unstack()  
region\_population = region\_population[savecol]  
# 四国人数  
region\_population['四国（再揭）'] = region\_population.loc[:, savecol].sum(axis=1)  
  
# 将result和region\_population  
nationwide = nationwide.join(region\_population)  
nationwide = nationwide.rename(columns={'population': '全国'})  
nationwide.reset\_index()

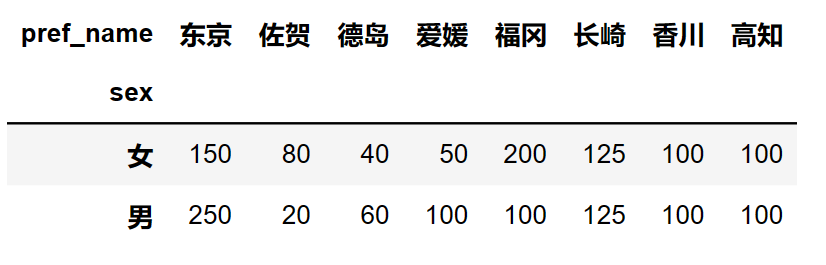
输出如下：



pandas的实现是这样的思路，首先对分组后的数据进行聚合操作，计算全国的总人口。然后按sex和pref\_name列对数据进行分组，并计算各组内population列的总和，这会得到一个以sex和pref\_name为索引，population为值的DataFrame：



之后借助unstack()方法将索引列转换为列，得到一个以 pref\_name 为列，tsex 为行，population 为值的透视表：



在透视表的基础之上进行数据的统计计算即可完成需求。其实pandas这种计算还是比较巧妙的，通常需要将数据进行转换（转置）的感觉。

对于此题，我们还可以使用Pandas中的where方法来进行实现。我们一起来看看代码：

import pandas as pd

df = pd.read\_sql\_query('SELECT \* FROM PopTbl2', engine)

df['sex'] = df['sex'].apply(lambda x: '男' if x == '1' else '女')

# 使用where进行查找和计算

df['全国'] = df['population']

df['德岛'] = df['population'].where(df['pref\_name'] == '德岛', 0)

df['香川'] = df['population'].where(df['pref\_name'] == '香川', 0)

df['爱媛'] = df['population'].where(df['pref\_name'] == '爱媛', 0)

df['高知'] = df['population'].where(df['pref\_name'] == '高知', 0)

df['四国（再揭）'] = df['population'].where(df['pref\_name'].isin(['德岛', '香川', '爱媛', '高知']), 0)

savecol = ['全国','德岛','香川','爱媛','高知','四国（再揭）']

df = df.rename(columns={'population': '全国'})

# 分组求和

result = df.groupby('sex').sum()

result = result[savecol].reset\_index()

result

这里就不在过多的解释了，大家了解一下where的用法就好。我们来看最后一个案例。

1. **案例三：用ORDER BY生成“排序”列**

对案例一种的Greatests执行查询语句 SELECT key\_name FROM Greatests ORDER BY key\_name之后，结果会按照key\_name这一列值的字母表顺序显示出来，现在要求按照 B-A-D-C-E这样的指定顺序进行排列。

这个问题比较好解决，直接新建一个列，按照列进行排序即可，SQL语句如下：

**SELECT** key\_name,

**CASE** **WHEN** key\_name = 'B' **THEN** 1

**WHEN** key\_name = 'A' **THEN** 2

**WHEN** key\_name = 'D' **THEN** 3

**WHEN** key\_name = 'C' **THEN** 4

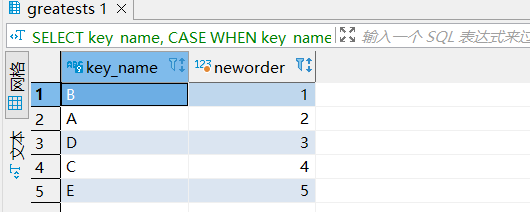
**WHEN** key\_name = 'E' **THEN** 5

**ELSE** **NULL** **END** **AS** neworder

**FROM** GREATESTS

**ORDER** **BY** neworder;

输出如下：



这题比较简单我们使用Pandas实现的代码如下：

import pandas as pd

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

# 字典映射

sort\_dict = {'B': 1,

'A': 2,

'D': 3,

'C': 4,

'E': 5}

df['neworder'] = df['key\_name'].map(sort\_dict)

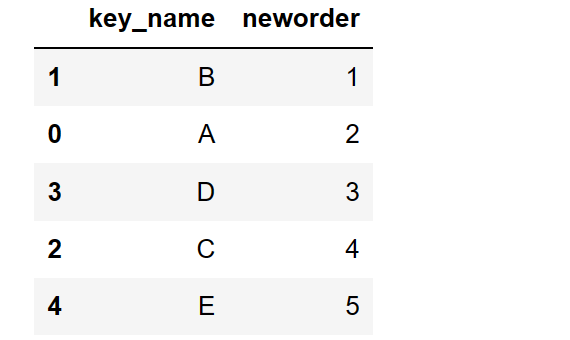
# 按照neworder排序

df = df.sort\_values('neworder')

df = df[['key\_name', 'neworder']]

Df

输出如下：



至此我们就完成了关于《SQL进阶教程》中相关的知识的学习，我们下章节来学习SQL中的自连接相关的知识。

1. **参考文献**
2. MICK-《SQL进阶教程》
3. Joe Celko-《SQL权威指南》
4. http://pandas.pydata.org/docs/user\_guide/index.html