**Case表达式**

这次的文章我们来看一看，Case表达式的一些案例，以及如何使用Pandas来实现相应的功能，这样就能对Case表达式有一个深度的认识。

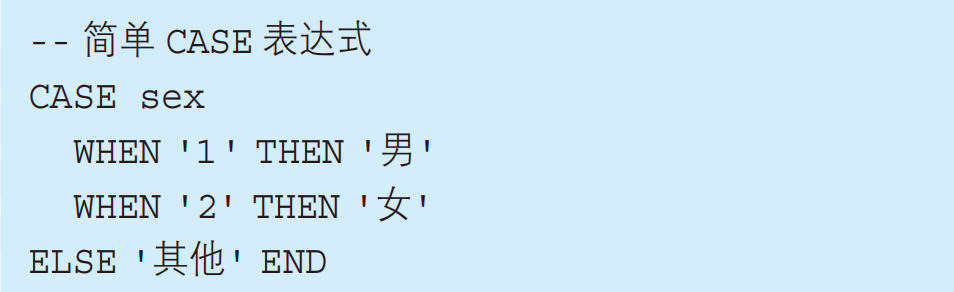
1. **Case表达式语法**

在SQL查询中，你可以使用CASE表达式为特定条件赋值。CASE 表达式允许你根据某个条件的结果，选择不同的值进行赋值。

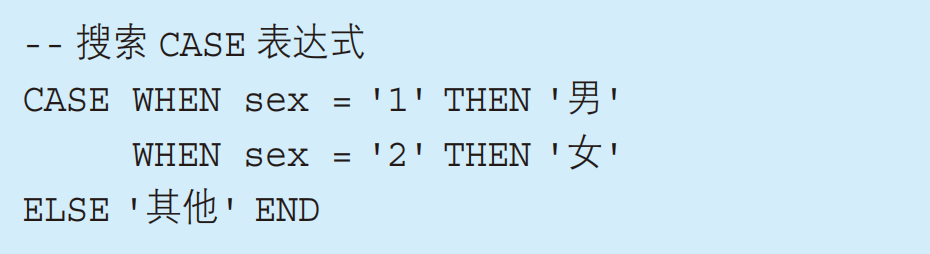
CASE 表达式的种类：

1. 有简单 CASE 表达式 （simple case expression）
2. 搜索 CASE 表达式（searched case expression）

简单CASE表达式：



搜索CASE表达式:



1. **CASE表达式书写注意点**

这里总结一下书写CASE表达式需要注意的点：

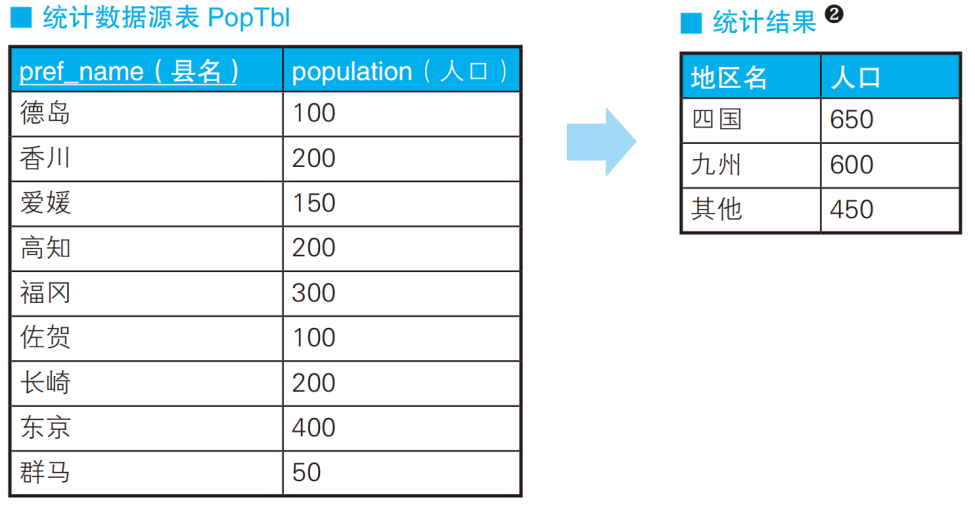
1. 在使用CASE表达式时，注意避免使用保留字作为列名或变量名。
2. CASE表达式里各个分支返回的数据类型是否一致
3. 可以使用聚合函数（如COUNT、SUM、AVG等）作为CASE表达式的结果。
4. 使用CASE表达式时，需要将条件和结果用关键字WHEN、THEN和END进行分隔。
5. 不要忘记写END
6. **案例1-将已有编号方式转换为新的方式并统计**

现在有一个需求是这样的，对PopTal表格进行查询得到相应的结果表。

其中：

四国地区包含德岛、香川、爱媛、高知。

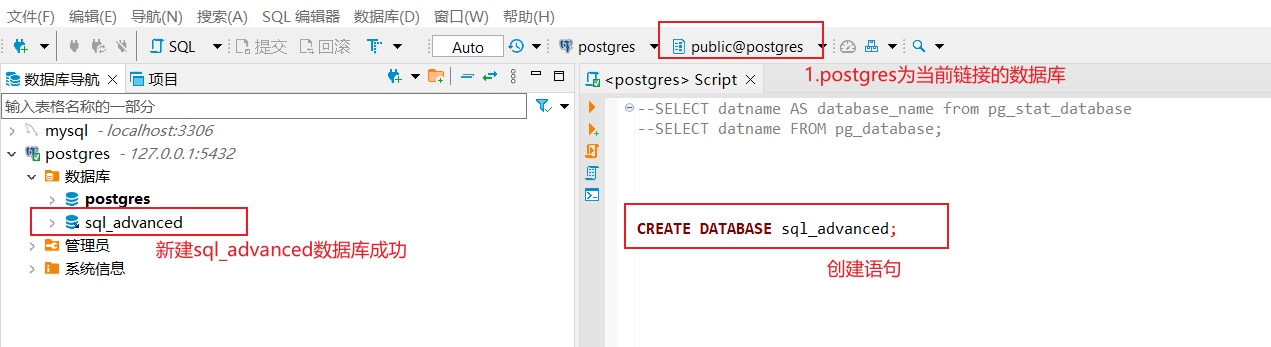
九州地区包含福冈、佐贺、长崎。



接下来我们一起来使用CASE语句来实现这个需求。

（1）创建一个数据库，我们使用下述语句来进行创建：

**CREATE** **DATABASE** sql\_advanced;



这里我们创建了一个新的数据库sql\_advanced。

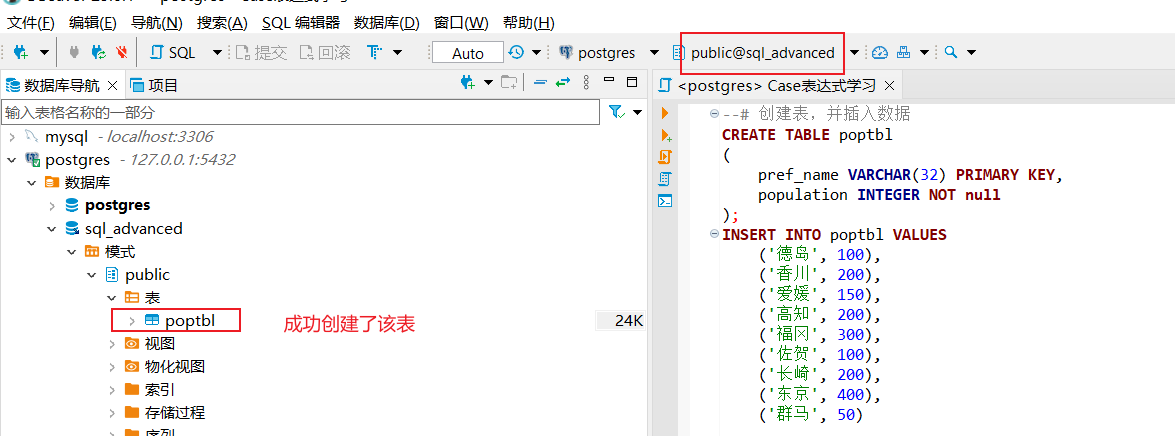
（2）创建PopTbl数据表，按照下述步骤进行：

接下里我们在这个数据库上新建一个PopTbl表并拆入数据，我们的代码如下所示：

--# 创建表，并插入数据

**CREATE** **TABLE** poptbl   
(  
pref\_name **VARCHAR**(32) **PRIMARY** **KEY**,  
population **INTEGER** **NOT** **null**);  
**INSERT** **INTO** poptbl **VALUES**('德岛', 100),  
('香川', 200),  
('爱媛', 150),  
('高知', 200),  
('福冈', 300),  
('佐贺', 100),  
('长崎', 200),  
('东京', 400),  
('群马', 50)

运行脚本创建成功之后，可以看出已经显示出新的表了：



（3）接下来我们实现这个查询，先看《SQL进阶教程》中的实现：原书的这种写法是这样的，使用一个个CASE-WHEN-THEN的方式：

**SELECT** **CASE** pref\_name

**WHEN** '德岛' **THEN** '四国'

**WHEN** '香川' **THEN** '四国'

**WHEN** '爱媛' **THEN** '四国'

**WHEN** '高知' **THEN** '四国'

**WHEN** '福冈' **THEN** '九州'

**WHEN** '佐贺' **THEN** '九州'

**WHEN** '长崎' **THEN** '九州'

**ELSE** '其他' **END** **AS** district,

**SUM**(population)

**FROM** PopTbl

**GROUP** **BY** district

**WHEN** '德岛' **THEN** '四国'

**WHEN** '香川' **THEN** '四国'

**WHEN** '爱媛' **THEN** '四国'

**WHEN** '高知' **THEN** '四国'

**WHEN** '福冈' **THEN** '九州'

**WHEN** '佐贺' **THEN** '九州'

**WHEN** '长崎' **THEN** '九州'

**ELSE** '其他' **END**;

原书中的代码对一个个地区进行举例，然后归属到不同的地区类别中，这样的代码会显得比较长，这里我们使用SQL中的in方法，这种方法比较灵活，代码量也比较少：

**SELECT** **CASE**

**WHEN** pref\_name **in** ('德岛', '香川', '爱媛', '高知') **THEN** '九州'

**WHEN** pref\_name **in** ('福冈', '佐贺', '长崎') **THEN** '四国'

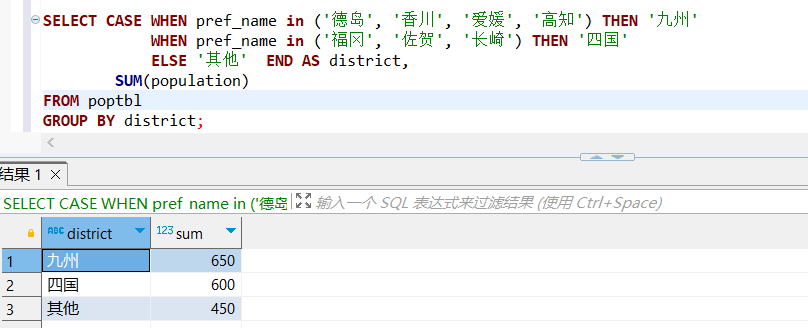
**ELSE** '其他' **END** **AS** district,

**SUM**(population)

**FROM** poptbl

**GROUP** **BY** district;

输出如下所示：



如果这一步有一些跳跃，我们可以先不GROUP BY：  
**SELECT** pref\_name,population,  
 **CASE** **WHEN** pref\_name **in** ('德岛', '香川', '爱媛', '高知') **THEN** '九州'  
 **WHEN** pref\_name **in** ('福冈', '佐贺', '长崎') **THEN** '四国'  
 **ELSE** '其他'   
 **END** **AS** district  
**FROM** poptbl

然后再加载SUM()聚合函数求解每一个分类地区的人数。原文提到先进行GROUP BY再进行SELECT的方式在一些诸如Oracle、DB2、SQL Server 等数据库里是会报错的，在MYSQL和PostgreSQL中是能顺利执行的。

（4）接下来我们使用Pandas来实现这个功能，我们直接上代码：

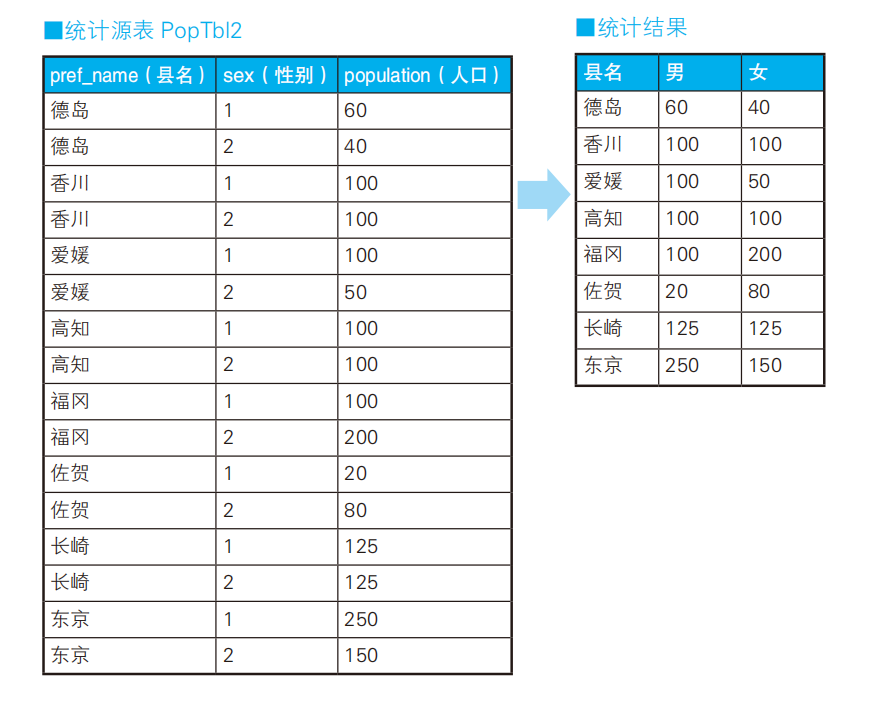
import psycopg2  
import pandas as pd  
from sqlalchemy import create\_engine, text  
*# 连接数据库取数*engine = create\_engine('postgresql+psycopg2://postgres:123456@127.0.0.1:5432/sql\_advanced')   
df = pd.read\_sql\_query('SELECT \* FROM poptbl', engine)   
  
*# 创建一个字典，用来映射pref\_name和district*district\_map = {  
 '德岛': '九州',  
 '香川': '九州',  
 '爱媛': '九州',  
 '高知': '九州',  
 '福冈': '四国',  
 '佐贺': '四国',  
 '长崎': '四国'  
}  
*# 用字典的get方法来替换pref\_name列的值，如果没有匹配到就返回'其他'*df['district'] = df['pref\_name'].apply(lambda x: district\_map.get(x, '其他'))  
*# 按照district分组，对population求和，重置索引*df = df.groupby('district')['population'].sum().reset\_index()  
df



上述使用Pandas来实现的时候，唯一要注意的是最后在分组求和的之后重置索引，也就是reset\_index()方法的使用，reset\_index()方法的主要作用是将分组后的数据框重新设置索引，以便在结果中保留原始的分组信息。如果不使用reset\_index()，则返回的结果将只包含分组后的汇总信息，而不会保留分组的层次结构。大家可以自己实验一下，至此我们就使用pandas实现了这个功能，还是挺有意思的。

1. **案例2-使用一条语句进行不同条件的统计**

此案例的输入和输出为（按照县名和性别汇总统计人口）：



一般想到的是使用子查询，然后将两者的结果进行join：

**select** A.pref\_name,A.男,B.女

**from**

(**SELECT**

pref\_name,

**SUM**(population) **as** 男

**FROM** PopTbl2

**WHERE** sex = '1'

**GROUP** **BY** pref\_name) **as** A

**left** **join**

(**SELECT**

pref\_name,

**SUM**(population) **as** 女

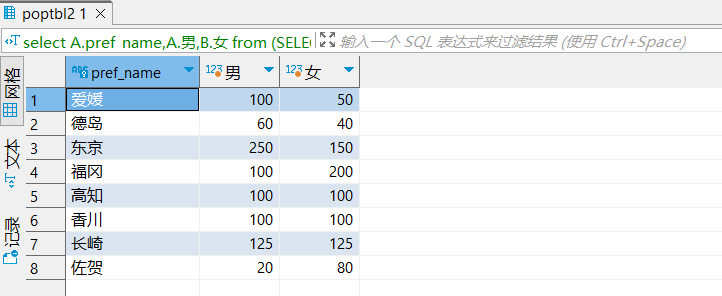
**FROM** PopTbl2

**WHERE** sex = '2'

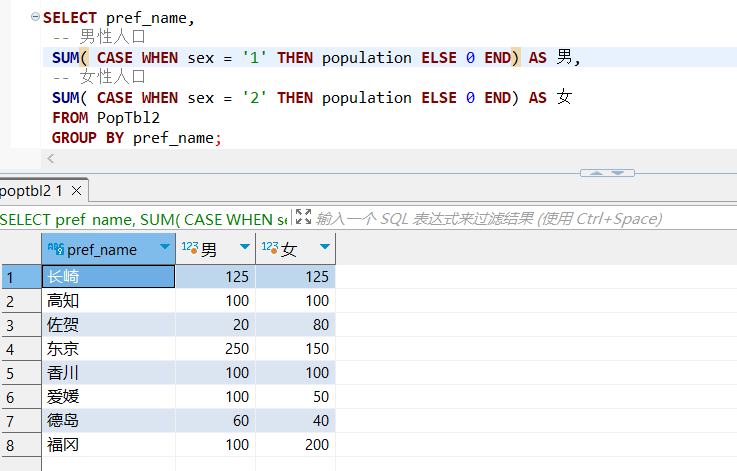
**GROUP** **BY** pref\_name) **as** B

**on** A.pref\_name =B.pref\_name

结果如下：



上述方案在一定程度上是比较清晰的，但是不够“技巧”。书中提供了另一种解答：



我们尝试用Pandas来实现这个功能，在此之前我们知道上述过程中需要对pref\_name字段进行groupby的操作，为此我们先看一下groupby的结果：



结果显示，上述一共

1. **上述**
2. **11**
3. **参考文献**

（1）MICK-《SQL进阶教程》  
（2）Joe Celko-《SQL权威指南》