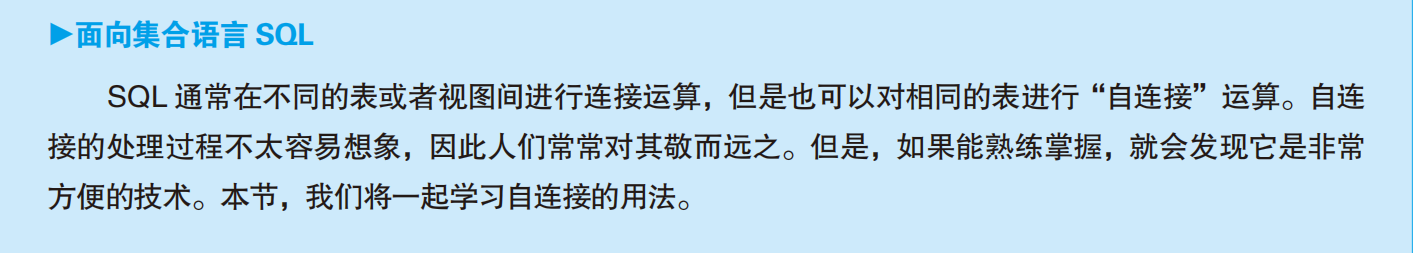
**必知必会的窗口函数四\_自连接的用法**

在上次的文章中，我们了解了窗口函数的基本知识，以及一些常见的窗口函数的使用方法，具体文章可以从以下连接中进行学习

**必知必会的窗口函数三：常见的窗口函数**

本次文章我们来学习一下SQL中的自连接技术，自连接（self join）是SQL语句中经常要用的连接方式，使用自连接可以将自身表的一个镜像当作另一个表来对待，从而能够得到一些特殊的数据。



1. **可重排列**

假设现在我们有一张Products表如下所示：



语句创建语句如下所示：

**CREATE** **TABLE** products

(**name** **VARCHAR**(16) **PRIMARY** **KEY**,

price **INTEGER** **NOT** **NULL**);

**INSERT** **INTO** products **VALUES**('苹果', 50);

**INSERT** **INTO** products **VALUES**('橘子', 100);

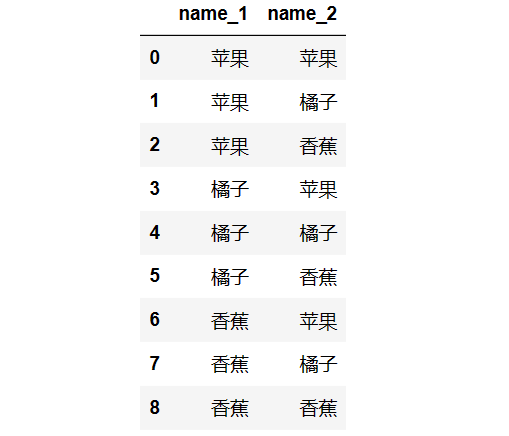
**INSERT** **INTO** products **VALUES**('香蕉', 80)

那么基于此表进行有序对的生成代码我们可以这样实现：  
-- 用于获取可重排列的 SQL 语句

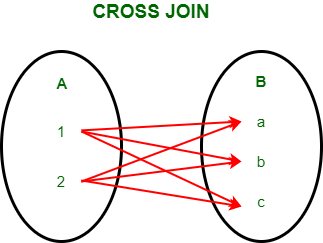
**SELECT** *P1*.name **AS** *name\_1*, *P2*.name **AS** *name\_2*

**FROM** Products *P1*, Products *P2*;

实现结果如下所示：



从结果上来看数量正好是3的平方，从结果上也有诸如（苹果,苹果）这样的数据对，这样来看正好是一个笛卡尔集。也是一种交叉连接（假设有两个表A和B）：



为了过滤掉相同的数据对，我们可以加上过滤条件：

**SELECT** *p1*.name **AS** *name\_1*, *p2*.name **AS** *name\_2*

**FROM** products **AS** *p1*, products **AS** *p2*

**WHERE** *p1*.name <> *p2*.name

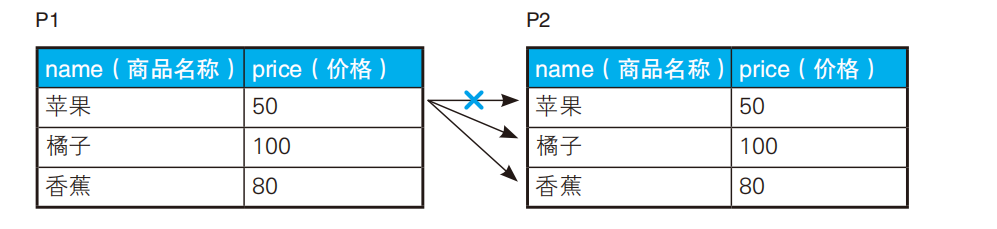
或者显示的使用连接方式：

**SELECT** *p1*.name **AS** *name\_1*, *p2*.name **AS** *name\_2*

**FROM** products **AS** *p1* **inner** **join** products **AS** *p2*

**ON** *p1*.name <> *p2*.name

也就是说强制使得下述关系包含在我们的数据处理过程中：



那么使用Pandas来实现上述的自连接，我们的代码可以编写如下所示：

query = """select \* from Products"""  
Products = pd.read\_sql\_query(query, engine)   
*# 将name列转换为一个DataFrame*name\_df = Products["name"].to\_frame()  
*# 创建一个公共的键列，值都为0*name\_df["key"] = 0  
*# 用merge函数进行交叉连接，on参数指定公共的键列，how参数指定连接方式为outer*cross\_join = name\_df.merge(name\_df, on="key", how="outer").rename(columns={"name\_x": "name\_1", "name\_y": "name\_2"})  
*# 删除结果表中的键列*cross\_join = cross\_join[['name\_1','name\_2']]  
*# 查看结果表*cross\_join

上述Python代码输出的也是9个值，如果想进行过滤可以使用以下代码：

query = """select \* from Products"""  
Products = pd.read\_sql\_query(query, engine)   
*# 将name列转换为一个DataFrame*name\_df = Products["name"].to\_frame()  
*# 创建一个公共的键列，值都为0*name\_df["key"] = 0  
*# 用merge函数进行交叉连接，on参数指定公共的键列，how参数指定连接方式为outer*cross\_join = name\_df.merge(name\_df, on="key", how="outer").rename(columns={"name\_x": "name\_1", "name\_y": "name\_2"})  
*# 用query函数筛选出结果表中name\_1和name\_2不相等的行*inner\_join = cross\_join.query("name\_1 != name\_2")  
*# 查看结果表*inner\_join

与之不同的是我们对结果集加上了**name\_1 != name\_2**进行过滤实现了上述要求，但是我们也可以更换成以下方法：

**cross\_join.drop\_duplicates(inplace=True)**

大家有什么其他好的方法也可以说出来以下学习一下。

相同的表的自连接和不同表间的普通连接并没有什么区别，自连接里的“自”这个词也没有太大的意义。

当然了我们还可以使用其他符号进行过滤：

**SELECT** *P1*.name **AS** *name\_1*, *P2*.name **AS** *name\_2*

**FROM** Products *P1*, Products *P2*

**WHERE** *P1*.name > *P2*.name;

这里补充一下，PostgreSQL 会逐个字符地比较它们的 Unicode值。

1. **删除重复行**

在关系数据库的世界里，重复行和NULL值一样，都很让人烦恼，我们经常会想很多办法来排除掉重复行。

假设我们创建表的语句如下：

**CREATE** **TABLE** products\_other

(**name** **VARCHAR**(16),

price **INTEGER** **NOT** **NULL**);

**INSERT** **INTO** products\_other **VALUES**('苹果', 50);

**INSERT** **INTO** products\_other **VALUES**('橘子', 100);

**INSERT** **INTO** products\_other **VALUES**('橘子', 100);

**INSERT** **INTO** products\_other **VALUES**('橘子', 100);

**INSERT** **INTO** products\_other **VALUES**('香蕉', 80)

可以看出（橘子,100）是一个重复的数据，我们打算将它删除。

《SQL进阶指南》中使用的是Oracle的rowid进行删除实验的，这里我们使用之前学习到的窗口函数来进行实验：

**WITH** **temp** **AS**

(**SELECT** \*, **ROW\_NUMBER**() **OVER** (**PARTITION** **BY** **name**, price **ORDER** **BY** ctid) **AS** rn

**FROM** products\_other)

**DELETE** **FROM** **temp** **WHERE** rn > 1;

[rowid 是 Oracle 中的一个特殊列，表示行的物理位置1](https://www.postgresql.org/docs/current/sql-expressions.html" \t "https://copilot.microsoft.com/_blank)[。PostgreSQL 中的类似概念是 ctid ，它表示行在表中的块号和偏移量](https://blog.csdn.net/weixin_39540651/article/details/106097373" \t "https://copilot.microsoft.com/_blank)。

这里需要注意的是在pg中索引扫描是先查询到数据的ctid，然后根据ctid去得到相应的数据。

但是ctid和oracle中的rowid并不完全相同，ctid是PG表中的系统字段，表示数据行在它所在表内的物理位置。ctid的字段类型是oid。但是VACUUM FULL操作之后，经过回收数据块内的空闲空间，数据行在块内的物理位置会发生移动，即ctid会发生变化。**大家可以实验一下是怎样变化的。**

当然了我们可以使用group by结合子查询来去重：

**DELETE** **FROM** products\_other

**WHERE** ctid **NOT** **IN** (

**SELECT** **min**(ctid)

**FROM** products\_other

**GROUP** **BY** name, price

);

这样我们就完成了重复数据的删除操作，那么在Pandas中我们怎么删除重复数据呢？

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。

**三、参考文献**

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

<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>

