

# 一场Pandas与SQL的巅峰大战七

一场Pandas与SQL的巅峰大战七

pandasql的使用

简介

安装

使用

pandas操作MySQL数据库

read\_sql

to\_sql

巅峰系列总结十条：

reference:

在之前的六篇系列文章中，我们对比了pandas和SQL在数据处理方面的多项操作。

具体来讲，第一篇文章涉及数据查看，去重计数，条件选择，合并连接，分组排序等操作。

第二篇文章涉及字符串处理，窗口函数，行列转换，类型转换等操作。

第三篇文章围绕日期操作展开，主要讨论了日期获取，日期转换，日期计算等内容。

第四篇文章学习了在MySQL，Hive SQL和pandas中用多种方式计算日环比，周同比的方法。

第五篇文章我们用多种方案实现了分组和不分组情况下累计百分比的计算。

第六篇我们主要总结学习了SQL和pandas中计算日活和多日留存率的方法。

以上的几篇我们都是从“对立”的角度讨论pandas与SQL。今天我们主要来看下二者“和谐相处”的一面。具体来讲，本篇文章我们先讨论pandas中如何使用SQL，用到了pandasql，再讨论pandas对于数据库的读写。文中代码更多以python为主。同时本文也对整个pandas 大战 SQL系列文章进行了一些回顾。文末有惊喜！

## pandasql的使用

### 简介

pandasql是由Yhat编写的模拟R包sqldf的python第三方库，能够让我们用SQL的方式操作pandas的数据结构。

### 安装

在命令行中使用 `pip install pandasql` 即可实现安装。

### 使用

从pandasql包中可以导入sqldf，这是我们核心要使用的接口。它接收两个参数，第一个是合法的SQL语句。SQL具有的功能，例如聚合，条件查询，联结，where条件，子查询等等，它都支持。第二个是locals()或者globals()表示环境变量，它会识别目前已有的dataframe作为第一个参数中的表名。我们简单举两个例子，更详细的可以看文末链接1或者官方文档(文末链接2)。

基本使用：

```
import pandas as pd
from pandasql import sqldf#导入sqldf
```

data = pd.read\_excel('orderamt.xlsx')#读取文件获得dataframe,也可以用其他方式取得

```
sql = "select * from data limit 10"#SQL语句, 表名就是dataframe的名字
result = sqldf(sql, locals())#result也是dataframe类型的,
result.head()
```

```
import pandas as pd
from pandasql import sqldf#导入sqldf
```

data = pd.read\_excel('orderamt.xlsx')#读取文件获得dataframe,也可以用其他方式取得

```
sql = "select * from data limit 10"#SQL语句, 表名就是dataframe的名字
result = sqldf(sql, locals())#result也是dataframe类型的,
result.head()
```

	id	dt	amt
0	1	2019-11-01 00:00:00.000000	1043
1	2	2019-11-02 00:00:00.000000	1119
2	3	2019-11-03 00:00:00.000000	878
3	4	2019-11-04 00:00:00.000000	1197
4	5	2019-11-05 00:00:00.000000	1156

按月聚合

```
sql2 = "select strftime('%Y-%m', dt) as mon,\
sum(amt) as amt from data group by strftime('%Y-%m', dt)"
result2 = sqldf(sql2, locals())
result2
```

```
: sql2 = "select strftime('%Y-%m', dt) as mon,\
sum(amt) as amt from data group by strftime('%Y-%m', dt)"
result2 = sqldf(sql2, locals())
result2
```

	mon	amt
0	2019-11	30545
1	2019-12	30471

官方文档中说为了避免冗余的调用可以对sqldf进行一层封装,用pysqldf代替,只需对其传入一个SQL语句参数即可,如下面代码所示。但我试了试不封装也是可以的。前面的代码可以省略locals()或者globals()。

```
pysqldf = lambda q: sqldf(q, globals())
result = pysqldf(sql)#传入合法的SQL语句
```

用自定义的SQL获取dataframe之后,可以继续用pandas处理,也可以直接进行保存。

## pandas操作MySQL数据库

这一部分我们来看下pandas直接操作数据库的例子，主要学习read\_sql和to\_sql的用法。

## read\_sql

这个函数的作用是，对数据库中的表运行SQL语句，将查询结果以dataframe的格式返回。另外还有两个read\_sql\_table，read\_sql\_query，通常使用read\_sql就够了。主要的两个参数是合法的SQL语句和数据库连接。数据链接可以使用SQLAlchemy或者字符串。其他可选参数可以参考官方文档。

## to\_sql

这个函数的作用是，将dataframe的结果写入数据库。提供表名和连接名即可，不需要新建MySQL表。

用操作MySQL举例如下，需提前安装好sqlalchemy，pymysql，直接pip安装即可。

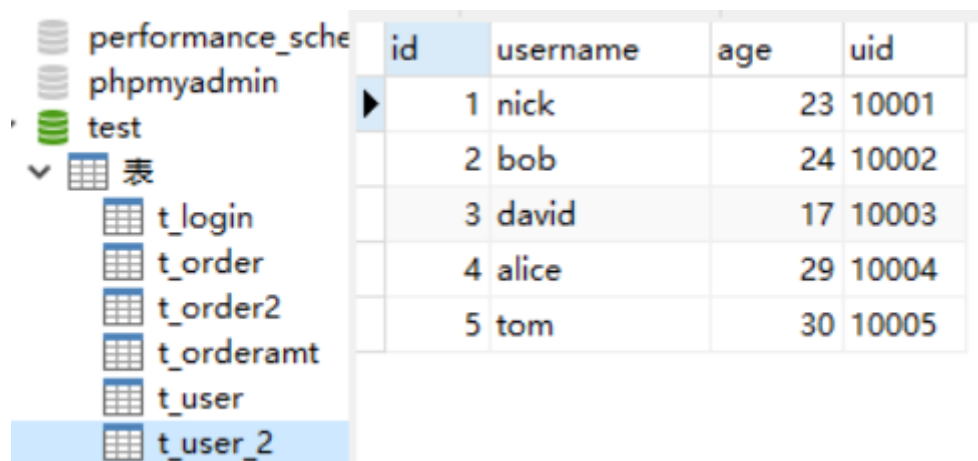
```
#read_sql举例
import pandas as pd
from sqlalchemy import create_engine

engine = create_engine("mysql+pymysql://root:@127.0.0.1:3306/test")
#数据库的用户名: root, 密码: 空, host:127.0.0.1, 端口: 3306, 数据库名: test
#统一格式, 可参考文末链接3
#engine = create_engine("dialect+driver://username:password@host:port/database")
sql = "select * from t_user"
df = pd.read_sql(sql, engine)
df
```

```
import pandas as pd
from sqlalchemy import create_engine
engine = create_engine("mysql+pymysql://root:@127.0.0.1:3306/test")
sql = "select * from t_user"
df = pd.read_sql(sql, engine)
df
```

	id	username	age	uid
0	1	nick	23	10001
1	2	bob	24	10002
2	3	david	17	10003
3	4	alice	29	10004
4	5	tom	30	10005
5	6	jim	43	10006
6	7	jimmy	41	10007
7	8	john	28	10008
8	9	frank	20	10009

```
#to_sql举例
df2 = df.head()
df2.to_sql('t_user_2', engine, index=None)
```



id	username	age	uid
1	nick	23	10001
2	bob	24	10002
3	david	17	10003
4	alice	29	10004
5	tom	30	10005

t\_user\_2是结果表名，不用事先在数据库中建立，否则会报错，表的字段名就是dataframe的列名。engine是上文创建的连接。df2就是期望写入的数据，这里只选取了上文df的前五行。需要注意如果不加 index=None 参数，会把索引也写进去，多一列index。

pandas操作SQL我就抛砖引玉先写这么多，MySQL之外的其他数据库，也大同小异，用到的时候可以查一下相关资料。

以上我们学习了pandas和SQL交互使用的方法，可以看到二者还是能够融洽相处的，对不熟悉pandas的朋友，也可以用SQL来操作dataframe，而SQL和pandas中的数据也能方便进行转换。

## 巅峰系列总结十条：

一转眼，pandas与SQL系列已经更新七篇了，也到了一个暂时性结束的阶段，但pandas和SQL本身的学习远没有结束。后续如果有机会，也有可能继续更。从开始写第一篇的时候，我压根也没想到能写成一个系列。从效果反馈来看第一篇阅读最高，被转载次数最多，在知乎被点赞，评论次数也最多，可以说无论是在形式风格上还是在内容启发上都奠定了后续几篇文章的基础。后续文章在内容上更加紧凑，更加深入，也获得了不少的关注。在这里非常感谢给我提供灵感的某位小可爱，嘿嘿。这段时间添加我好友的伙伴很多从这个系列过来的，都说很赞很实用。最近公众号后台收到的关键词回复几乎都来源于对比系列，并且很多都是从对比到对比六回复六连。也有很多读者朋友私信与我讨论文章中很多的细节问题，非常感谢大家的支持，看到你们在认真学习，我也非常开心！在这里我总结以下下几点：

- 1.提示：第一篇文章的关键字是“**对比**”，没有“一”。后面二到六都为对比跟相应数字，不要回复错了。
- 2.虽然名为对比，但本系列的目的并不是比较孰优孰劣。最开始是我在需要从SQL迁移到pandas的过程中，发现很多SQL的操作不太会实现，但我知道一定可以实现。于是进行了一些总结，便于使用的时候查阅。实际中，大家可以根据需要选择最适合的工具。
- 3.数据存储 in 数据库中的情况下，优先用SQL(MySQL 或Hive)，数据量比较大时，pandas性能会有瓶颈。而如果是文件形式的数据，可以尝试pandas，当然你也可以先导入数据库再做处理。总之当由于客观限制不能使用SQL时，就可以考虑用pandas了。另外当需要对处理好的数据调用模型时(如sklearn包)，pandas可能要有优势一些，也可以把前期工作用SQL做好，再导入到pandas。
- 4.知乎上有朋友问过为什么没有速度对比。其实上面已经提了，本系列主要聚焦于操作的熟悉，所以我们用到的数据集都是自己编的小数据集。速度对比一方面需要标准的大数据集(这个没有找)，另一方面和业务本身的相关性不大，我也缺少相关经验，所以没有做。需要这方面对比的朋友，还是要自己费心了。
- 5.第六篇文章纠正了第一篇文章的一个细节问题，是关于count(distinct)，用np.size不太严谨。这里再补充两个没有提及的：

sql中join可以有多个字段，pandas中的merge操作，如果想实现同样的效果，可以在on参数中用列表的形式。这一点在系列第六篇文章中也用到了。

系列第三篇，read\_csv读取数据时，如果有两个需要解析的时间列，parse\_dates参数可以写成一维列表的形式，但不能写成二维形式。二维情况适用于需要把两个或多个列合起来的情况。可以自己尝试一下。

6.实践出真知。在和很多朋友交流过程中，发现了很多之前没有遇到的问题。这个时候一方面要查资料看文档，另一方面也要多动手多实践，与人多交流，这样才能真正形成自己的认识。

7.系列文章多次写到了Hive的代码，很多朋友可能没有相应的环境，我们主要用到的是row\_number(), lag(), lead()函数等函数。有条件的可以自己搭建一下Hive玩一下。没有条件的可以用MySQL 8.0或者postgreSQL代替，我们用的Hive 函数他们基本都支持。安装使用教程请自行查阅，相应的导入数据的方式也要视情况而变。

8.MySQL的安装方式，最简便的是安装xampp+navicate，几乎是傻瓜式安装。

9.由于本篇代码不多，就不提供了。但仍然可以在公众号后台回复**“对比七”**获取本文的pdf版本，方便阅读与保存。

10.之前说好的红包，这里兑现一下，88元66个红包，不说人人有份，起码略表心意(现在粉丝太少了，发太多了没人领岂不尴尬哈哈)。后台回复**“红包”**获取支付宝红包口令，先到先得，拼手气。祝所有朋友新的一年多多发财，平安康健。顺便说一下，后序公众号的文章，我依然会尽力写原创，可能会更多采用独立单篇的方式，条件成熟时也会写系列文章。有时工作忙不过来也会有优质文章转载，也可能会有推广活动(为表诚意，每发一次推广我都会在群里发红包，可以加我微信hitchenghengchao拉你入群)，还希望大家多多理解，多多支持！

## reference:

- 1.<https://www.jianshu.com/p/84c17026017e>
- 2.<https://github.com/yhat/pandasql/blob/master/examples/demo.py>
- 3.[https://blog.csdn.net/walking\\_visitor/article/details/84023393](https://blog.csdn.net/walking_visitor/article/details/84023393)
- 4.<https://www.jianshu.com/p/238a13995b2b>