## 1.3 数据库与数据分析基本流程组织

当数据量开始膨胀，常规存储数据方式的简单文件形式，虽然逻辑简单，但可扩展性差，不能有效解决数据的完整性、一致性及安全性等一系列问题，由此产生了数据管理系统（Database Management System，DBMS），即数据库（database）。数据库是按照一定规则保存数据，能给予多个用户共享，具有尽可能小的冗余度，与应用程序彼此独立，并能通过查询取回程序检索所需数据的数据存储和管理方式。数据库有多种类型，例如与分布处理技术结合产生的分布式数据库，与并行处理技术结合产生的并行数据库，特定领域的地理数据库、空间数据库等。Web最常使用基于关系模型的数据库，即关系型数据库（Relational Database Management System，RDBMS），或称为SQL（Structured Query Language）数据库，使用结构化查询语言操作。与之相反的是最近流行的文档数据库和键-值对数据库，即NoSQL数据库[1]57。其中关系型数据库把数据存储在表中，表的列（colunm）为字段（field），每一字段为“样本”的一个属性，行（row）为每一“样本”数据，包含一个或多个属性。常用的关系型数据库有MySQL（其替代品包括MariaDB等），及SQLite和PostgreSQL。

在城市空间数据分析方法研究中，主要使用[SQLite](https://www.sqlite.org/index.html)①和[PostgreSQL](https://www.postgresql.org/)②两个数据库。当涉及到地理空间信息数据，需要配置投影坐标系统及在[QGIS](https://www.qgis.org/en/site/)③中读取地理信息建立地图时，使用PostgreSQL；否则，一般使用轻量型的数据库SQLite。

### 1.3.1 SQLite数据库

[SQLite](https://www.sqlite.org/index.html)是一个用C编程语言编写的SQL数据库引擎，具有小型、快速、自包含（self-contained）、高可靠性，功能齐全等特点。已有超过1万亿(1e12)SQLite数据库在活跃的使用。其文档格式稳定、跨平台，向后兼容，同时其开发人员保证到2050年一直保持这种格式。

对SQLite关系型数据库的操作，包含通过SQLite命令执行（SQL语句），通过Python等语言执行（大多数数据库引擎都有对应的Python包）。对于Python，使用两个库，一个是[sqlite3](https://docs.python.org/3/library/sqlite3.html)④操作SQLite数据库的库，另一是[SQLAlchemy（flask\_sqlalchemy）库](https://www.sqlalchemy.org/)⑤（数据库抽象层代码包，可以直接处理高等级的Python对象，而不用关注表、文档或查询语言等数据库实体）。当然pandas等库也提供了直接读写数据库对应的方法，进一步简化了对数据库的操作。

SQLite数据库应用途径，引用*《漫画数据库》*[2]中的数据，结合代码实现阐释。同时使用[DB Browser for SQLite(DB4S)](https://sqlitebrowser.org/)⑥辅助查看、管理SQLite数据库。

#### 查看版本

%%cmd  
sqlite3 version

Microsoft Windows [Version 10.0.22000.856]  
(c) Microsoft Corporation. All rights reserved.  
  
(USDAlab) C:\Users\richi\omen\_richiebao\omen\_github\USDA\_CH\_endup\USDA\notebook>sqlite3 version  
  
(USDAlab) C:\Users\richi\omen\_richiebao\omen\_github\USDA\_CH\_endup\USDA\notebook>

import sqlalchemy  
sqlalchemy.\_\_version\_\_

'1.4.32'

#### 2）根据漫画数据库中的销售数据集录入数据

该数据包括4个表，分别是销售表*sales\_table*，包含的字段（列）有报表编码*idx*，日期*date*和出口国编码*exporting\_country\_ID*；出口国表*exporting\_country\_table*，包含的字段有出口国编码*exporting\_country\_ID*和出口国名称*exporting\_country\_name*；销售明细表*sale\_details\_table*，包含的字段有报表编码*idx*，商品编码*commodity\_code*和数量*number*；商品表*commodity\_table*，包含的字段有商品编码*commodity\_code*和商品名称*commodity\_name*。

数据录入时的数据结构为字典，然后转换为pandas的DataFrame数据结构。注意时间格式采用datetime库的datetime方法格式化时间。

import pandas as pd  
from datetime import datetime  
  
# 定义字典类型的假设数据  
sales\_dic={'idx':[1101,1102,1103,1104,1105],  
 'date':[datetime(2020,3,5),datetime(2020,3,7),datetime(2020,3,8),datetime(2020,3,10),datetime(2020,3,12)],  
 "exporting\_country\_ID":[12,23,25,12,25]}  
exporting\_country\_dic={"exporting\_country\_ID":[12,23,25],  
 'exporting\_country\_name':['kenya','brazil','peru']}  
sale\_details\_dic={'idx':[1101,1101,1102,1103,1104,1105,1105],  
 'commodity\_code':[101,102,103,104,101,103,104],  
 'number':[1100,300,1700,500,2500,2000,700]}  
commodity\_dic={'commodity\_code':[101,102,103,104],  
 'commodity\_name':['muskmelon','strawberry','apple','lemon']}  
  
# 为方便数据管理，将字典格式数据转换为pandas的DataFrame格式  
sales\_table=pd.DataFrame.from\_dict(sales\_dic)  
exporting\_country\_table=pd.DataFrame.from\_dict(exporting\_country\_dic)  
sale\_details\_table=pd.DataFrame.from\_dict(sale\_details\_dic)  
commodity\_table=pd.DataFrame.from\_dict(commodity\_dic)  
  
print("-"\*50,"sales\_table","\n",sales\_table)  
print("-"\*50,"exporting\_country\_table","\n",exporting\_country\_table)  
print("-"\*50,"sale\_details\_table","\n",sale\_details\_table)  
print("-"\*50,"commodity\_table","\n",commodity\_table)

-------------------------------------------------- sales\_table   
 idx date exporting\_country\_ID  
0 1101 2020-03-05 12  
1 1102 2020-03-07 23  
2 1103 2020-03-08 25  
3 1104 2020-03-10 12  
4 1105 2020-03-12 25  
-------------------------------------------------- exporting\_country\_table   
 exporting\_country\_ID exporting\_country\_name  
0 12 kenya  
1 23 brazil  
2 25 peru  
-------------------------------------------------- sale\_details\_table   
 idx commodity\_code number  
0 1101 101 1100  
1 1101 102 300  
2 1102 103 1700  
3 1103 104 500  
4 1104 101 2500  
5 1105 103 2000  
6 1105 104 700  
-------------------------------------------------- commodity\_table   
 commodity\_code commodity\_name  
0 101 muskmelon  
1 102 strawberry  
2 103 apple  
3 104 lemon

#### 3）创建数据库（链接）

首先调用函数库sqlalchemy，使用from sqlalchemy import create\_engine调入create\_engine方法；

在当前目录下创建数据库，例如使用engine=create\_engine('sqlite:///x.sqlite')语句；

在相对或绝对路径创建数据库，例如使用engine=create\_engine('sqlite:///./data/fruits.sqlite'）或engine=create\_engine('sqlite:///absolute/data/fruits.sqlite'）语句；

如果创建内存数据库，例如使用engine=create\_engine('sqlite://')或engine=create\_engine('sqlite:///:memory:', echo=True)语句。

Unix、Max及Window系统的文件路径分隔符可能不同，如果出现异常，可以尝试在/或\切换，同时注意\也是转义符号，因此可能需要写成\\。

from sqlalchemy import create\_engine  
  
db\_fp=r'./database/fruits.sqlite'  
engine=create\_engine('sqlite:///'+'\\\\'.join(db\_fp.split('\\')),echo=True)

执行create\_engine语句，只是建立了数据库链接，只有向其写入表数据（或者对数据库执行任务，例如engine.connect()）等操作，才会在硬盘指定路径下找到该文件，如果没有则建立新数据库文件。如果存在同名数据库，重复执行此语句，只执行数据库链接操作。

connection=engine.connect()  
connection.close()

#### 4) 向数据库中写入表及数据

##### 1. 写入方法-pandas.DataFrame.to\_sql()

其中参数*if\_exists*，可以选择fail-为默认值，如果表存在，则返回异常；replace-先删除已经存在的表，再重新插入新表；append-向存在的表中追加行。

pandas方法，不需要建立模型（表结构），数据库模型的表述方法通常易于与机器学习模型，或者算法模型的说法混淆，因此为了便于区分，这里用表结构代替模型的说法。pandas可以根据DataFrame格式数据信息，尤其包含有自动生成的数据类型，直接在数据库中建立对应数据格式的表，不需要自行预先定义表结构。但是在应用程序中调入表中数据时，又往往需要调用表结构来读取数据库信息，例如Flask Web框架（参看Flask部分阐述）等。因此可以用DB4S来查看刚刚建立的SQLite数据库及写入的表，可以看到表结构，根据表结构的数据类型信息，再手工建立表结构。表结构通常以类（class）的形式定义。因为手工定义相对比较繁琐，尤其字段比较多，不容易确定数据类型时，可以使用数据库逆向工程的方法，例如使用sqlacodegen库生成数据库表结构。

def df2SQLite(db\_fp, df, table\_name, method='fail'):   
 '''  
 function - 把pandas DataFrame格式数据写入数据库（同时创建表）  
   
 Paras:  
 db\_fp - 数据库链接；string  
 df - 待写入数据库的DataFrame格式数据；DataFrame  
 table - 表名称；string  
 method - 写入方法，'fail'，'replace'或'append'；string  
   
 Returns:  
 None  
 '''  
 from sqlalchemy import create\_engine  
   
 engine=create\_engine('sqlite:///'+'\\\\'.join(db\_fp.split('\\')),echo=True)   
 try:   
 df.to\_sql(table\_name, con=engine, if\_exists="%s" % method)  
 if method=='replace':   
 print("\_"\*10,'the %s table has been overwritten...'%table\_name)   
 elif method=='append':  
 print("\_"\*10, 'the %s table has been appended...' % table\_name)  
 else:  
 print("\_"\*10, 'the %s table has been written......' % table\_name)  
 except:  
 print("\_"\*10, 'the %s table has been existed......' % table\_name)  
  
df2SQLite(db\_fp,sales\_table,'sales','fail')

2022-10-15 10:57:10,965 INFO sqlalchemy.engine.Engine PRAGMA main.table\_info("sales")  
2022-10-15 10:57:10,966 INFO sqlalchemy.engine.Engine [raw sql] ()  
2022-10-15 10:57:10,967 INFO sqlalchemy.engine.Engine PRAGMA temp.table\_info("sales")  
2022-10-15 10:57:10,968 INFO sqlalchemy.engine.Engine [raw sql] ()  
2022-10-15 10:57:10,969 INFO sqlalchemy.engine.Engine BEGIN (implicit)  
2022-10-15 10:57:10,970 INFO sqlalchemy.engine.Engine   
CREATE TABLE sales (  
 "index" BIGINT,   
 idx BIGINT,   
 date DATETIME,   
 "exporting\_country\_ID" BIGINT  
)  
  
2022-10-15 10:57:10,971 INFO sqlalchemy.engine.Engine [no key 0.00090s] ()  
2022-10-15 10:57:10,978 INFO sqlalchemy.engine.Engine CREATE INDEX ix\_sales\_index ON sales ("index")  
2022-10-15 10:57:10,979 INFO sqlalchemy.engine.Engine [no key 0.00065s] ()  
2022-10-15 10:57:10,984 INFO sqlalchemy.engine.Engine COMMIT  
2022-10-15 10:57:10,987 INFO sqlalchemy.engine.Engine BEGIN (implicit)  
2022-10-15 10:57:10,988 INFO sqlalchemy.engine.Engine INSERT INTO sales ("index", idx, date, "exporting\_country\_ID") VALUES (?, ?, ?, ?)  
2022-10-15 10:57:10,989 INFO sqlalchemy.engine.Engine [generated in 0.00070s] ((0, 1101, '2020-03-05 00:00:00.000000', 12), (1, 1102, '2020-03-07 00:00:00.000000', 23), (2, 1103, '2020-03-08 00:00:00.000000', 25), (3, 1104, '2020-03-10 00:00:00.000000', 12), (4, 1105, '2020-03-12 00:00:00.000000', 25))  
2022-10-15 10:57:10,990 INFO sqlalchemy.engine.Engine COMMIT  
\_\_\_\_\_\_\_\_\_\_ the sales table has been written......

’sales’表已经写入数据库，如果配置method='fail'，再次写入时，则返回异常，即提示表已经存在。

df2SQLite(db\_fp,sales\_table,'sales','fail')

2022-10-15 10:58:19,755 INFO sqlalchemy.engine.Engine PRAGMA main.table\_info("sales")  
2022-10-15 10:58:19,756 INFO sqlalchemy.engine.Engine [raw sql] ()  
\_\_\_\_\_\_\_\_\_\_ the sales table has been existed......

* 由sqlacodegen库生成SQLite数据库中’sales’表结构。对于sqlacodegen方法可以在命令行中输入sqlacodegen --help查看。生成的表结构写入到指定的文件中（下述代码写入到了sales\_table\_structure.py文件下）。

sqlacodegen库的安装通常使用pip install sqlacodegen 途径

%%cmd  
sqlacodegen sqlite:///./database/fruits.sqlite --tables sales --outfile sales\_table\_structure.py

Microsoft Windows [Version 10.0.22000.856]  
(c) Microsoft Corporation. All rights reserved.  
  
(USDAlab) C:\Users\richi\omen\_richiebao\omen\_github\USDA\_CH\_endup\USDA\notebook>sqlacodegen sqlite:///./database/fruits.sqlite --tables sales --outfile sales\_table\_structure.py  
  
(USDAlab) C:\Users\richi\omen\_richiebao\omen\_github\USDA\_CH\_endup\USDA\notebook>

打开存储有表结构的sales\_table\_structure.py文件，内容如下：

# coding: utf-8  
from sqlalchemy import BigInteger, Column, DateTime, MetaData, Table  
from sqlalchemy.ext.declarative import declarative\_base  
  
metadata = MetaData()  
t\_sales = Table(  
 'sales', metadata,  
 Column('index', BigInteger, index=True),  
 Column('idx', BigInteger),  
 Column('date', DateTime),  
 Column('exporting\_country\_ID', BigInteger)  
)

##### 2.写入方法-sqlalchemy创建表结构，及写入表

定义的表结构需要继承declarative\_base()映射类。完成表结构的定义后，执行BASE.metadata.create\_all(engine, checkfirst=True)写入表结构，注意此时并未写入数据。

from sqlalchemy.ext.declarative import declarative\_base  
import sqlalchemy as db  
  
BASE=declarative\_base() # 基本映射类，需要自定义的表结构继承  
  
class exporting\_country(BASE):  
 \_\_tablename\_\_='exporting\_country'   
   
 index=db.Column(db.Integer, primary\_key=True, autoincrement=True) # 自动生成的索引列  
 exporting\_country\_ID=db.Column(db.Integer)  
 exporting\_country\_name=db.Column(db.Text)  
   
 def \_\_repr\_\_(self): # 用于表结构打印时输出的字符串，亦可以不用写  
 return '<exporting\_country %r>'%self.exporting\_country\_ID   
exporting\_country.\_\_table\_\_ # 查看表结构

Table('exporting\_country', MetaData(), Column('index', Integer(), table=<exporting\_country>, primary\_key=True, nullable=False), Column('exporting\_country\_ID', Integer(), table=<exporting\_country>), Column('exporting\_country\_name', Text(), table=<exporting\_country>), schema=None)

checkfirst=True，用于检查该表是否存在，如果存在则不建立，默认为True。可以增加tables=[Base.metadata.tables['exporting\_country']]参数指定创建哪些表，或者直接使用exporting\_country.\_\_table\_\_.create(engine, checkfirst=True)方法。

BASE.metadata.create\_all(engine, checkfirst=True)

2022-10-15 11:00:02,381 INFO sqlalchemy.engine.Engine BEGIN (implicit)  
2022-10-15 11:00:02,382 INFO sqlalchemy.engine.Engine PRAGMA main.table\_info("exporting\_country")  
2022-10-15 11:00:02,382 INFO sqlalchemy.engine.Engine [raw sql] ()  
2022-10-15 11:00:02,384 INFO sqlalchemy.engine.Engine PRAGMA temp.table\_info("exporting\_country")  
2022-10-15 11:00:02,385 INFO sqlalchemy.engine.Engine [raw sql] ()  
2022-10-15 11:00:02,387 INFO sqlalchemy.engine.Engine   
CREATE TABLE exporting\_country (  
 "index" INTEGER NOT NULL,   
 "exporting\_country\_ID" INTEGER,   
 exporting\_country\_name TEXT,   
 PRIMARY KEY ("index")  
)  
  
  
2022-10-15 11:00:02,388 INFO sqlalchemy.engine.Engine [no key 0.00074s] ()  
2022-10-15 11:00:02,394 INFO sqlalchemy.engine.Engine COMMIT

将数据写入到新定义的表中。使用session.add\_all方法可以一次性写入多组数据，但是需要将其转换为对应的格式。

from sqlalchemy.orm import sessionmaker  
  
SESSION=sessionmaker(bind=engine) #建立会话链接  
session=SESSION() #实例化  
  
def zip\_dic\_tableSQLite(dic,table\_model):  
 '''  
 function - 按字典的键，成对匹配，返回用于写入SQLite数据库的列表  
   
 Paras:  
 dic - 字典格式数据；dict  
 table\_model - 表结构（模型）。数据将写入到该表中；Class  
 '''  
 keys=list(dic.keys())  
 vals=dic.values()  
 vals\_zip=list(zip(\*list(vals)))   
 return [table\_model(\*\*{k:i for k,i in zip(keys, v)}) for v in vals\_zip]  
  
exporting\_country\_table\_model=zip\_dic\_tableSQLite(exporting\_country\_dic,exporting\_country)  
session.add\_all(exporting\_country\_table\_model)  
session.commit()

2022-10-15 11:00:15,709 INFO sqlalchemy.engine.Engine BEGIN (implicit)  
2022-10-15 11:00:15,712 INFO sqlalchemy.engine.Engine INSERT INTO exporting\_country ("exporting\_country\_ID", exporting\_country\_name) VALUES (?, ?)  
2022-10-15 11:00:15,714 INFO sqlalchemy.engine.Engine [generated in 0.00127s] (12, 'kenya')  
2022-10-15 11:00:15,718 INFO sqlalchemy.engine.Engine INSERT INTO exporting\_country ("exporting\_country\_ID", exporting\_country\_name) VALUES (?, ?)  
2022-10-15 11:00:15,719 INFO sqlalchemy.engine.Engine [cached since 0.006257s ago] (23, 'brazil')  
2022-10-15 11:00:15,720 INFO sqlalchemy.engine.Engine INSERT INTO exporting\_country ("exporting\_country\_ID", exporting\_country\_name) VALUES (?, ?)  
2022-10-15 11:00:15,721 INFO sqlalchemy.engine.Engine [cached since 0.008062s ago] (25, 'peru')  
2022-10-15 11:00:15,722 INFO sqlalchemy.engine.Engine COMMIT

提取一组数据kenya（为类的形式），通过\_\_dict\_\_方式可以查看该类的属性值，包括待写入数据库中表的字段和对应值。

exporting\_country\_table\_model

2022-10-15 11:01:14,818 INFO sqlalchemy.engine.Engine BEGIN (implicit)  
2022-10-15 11:01:14,821 INFO sqlalchemy.engine.Engine SELECT exporting\_country."index" AS exporting\_country\_index, exporting\_country."exporting\_country\_ID" AS "exporting\_country\_exporting\_country\_ID", exporting\_country.exporting\_country\_name AS exporting\_country\_exporting\_country\_name   
FROM exporting\_country   
WHERE exporting\_country."index" = ?  
2022-10-15 11:01:14,822 INFO sqlalchemy.engine.Engine [generated in 0.00064s] (1,)  
2022-10-15 11:01:14,824 INFO sqlalchemy.engine.Engine SELECT exporting\_country."index" AS exporting\_country\_index, exporting\_country."exporting\_country\_ID" AS "exporting\_country\_exporting\_country\_ID", exporting\_country.exporting\_country\_name AS exporting\_country\_exporting\_country\_name   
FROM exporting\_country   
WHERE exporting\_country."index" = ?  
2022-10-15 11:01:14,825 INFO sqlalchemy.engine.Engine [cached since 0.004014s ago] (2,)  
2022-10-15 11:01:14,827 INFO sqlalchemy.engine.Engine SELECT exporting\_country."index" AS exporting\_country\_index, exporting\_country."exporting\_country\_ID" AS "exporting\_country\_exporting\_country\_ID", exporting\_country.exporting\_country\_name AS exporting\_country\_exporting\_country\_name   
FROM exporting\_country   
WHERE exporting\_country."index" = ?  
2022-10-15 11:01:14,828 INFO sqlalchemy.engine.Engine [cached since 0.006631s ago] (3,)  
  
  
  
  
  
[<exporting\_country 12>, <exporting\_country 23>, <exporting\_country 25>]

kenya=exporting\_country\_table\_model[0]  
print(kenya.\_\_dict\_\_)

{'\_sa\_instance\_state': <sqlalchemy.orm.state.InstanceState object at 0x0000021F3716BC70>, 'index': 1, 'exporting\_country\_name': 'kenya', 'exporting\_country\_ID': 12}

将剩下的两组数据用pandas写入SQLite数据库的方法写入。同时应用sqlacodegen库生成对应的数据库表结构。

df2SQLite(db\_fp,sale\_details\_table,table\_name='sale\_details')  
df2SQLite(db\_fp,commodity\_table,table\_name='commodity')

2022-10-15 11:06:05,909 INFO sqlalchemy.engine.Engine PRAGMA main.table\_info("sale\_details")  
2022-10-15 11:06:05,911 INFO sqlalchemy.engine.Engine [raw sql] ()  
2022-10-15 11:06:05,912 INFO sqlalchemy.engine.Engine PRAGMA temp.table\_info("sale\_details")  
2022-10-15 11:06:05,913 INFO sqlalchemy.engine.Engine [raw sql] ()  
2022-10-15 11:06:05,915 INFO sqlalchemy.engine.Engine BEGIN (implicit)  
2022-10-15 11:06:05,916 INFO sqlalchemy.engine.Engine   
CREATE TABLE sale\_details (  
 "index" BIGINT,   
 idx BIGINT,   
 commodity\_code BIGINT,   
 number BIGINT  
)  
  
  
2022-10-15 11:06:05,917 INFO sqlalchemy.engine.Engine [no key 0.00073s] ()  
2022-10-15 11:06:05,924 INFO sqlalchemy.engine.Engine CREATE INDEX ix\_sale\_details\_index ON sale\_details ("index")  
2022-10-15 11:06:05,925 INFO sqlalchemy.engine.Engine [no key 0.00074s] ()  
2022-10-15 11:06:05,931 INFO sqlalchemy.engine.Engine COMMIT  
2022-10-15 11:06:05,934 INFO sqlalchemy.engine.Engine BEGIN (implicit)  
2022-10-15 11:06:05,935 INFO sqlalchemy.engine.Engine INSERT INTO sale\_details ("index", idx, commodity\_code, number) VALUES (?, ?, ?, ?)  
2022-10-15 11:06:05,936 INFO sqlalchemy.engine.Engine [generated in 0.00061s] ((0, 1101, 101, 1100), (1, 1101, 102, 300), (2, 1102, 103, 1700), (3, 1103, 104, 500), (4, 1104, 101, 2500), (5, 1105, 103, 2000), (6, 1105, 104, 700))  
2022-10-15 11:06:05,938 INFO sqlalchemy.engine.Engine COMMIT  
\_\_\_\_\_\_\_\_\_\_ the sale\_details table has been written......  
2022-10-15 11:06:05,953 INFO sqlalchemy.engine.Engine PRAGMA main.table\_info("commodity")  
2022-10-15 11:06:05,954 INFO sqlalchemy.engine.Engine [raw sql] ()  
2022-10-15 11:06:05,955 INFO sqlalchemy.engine.Engine PRAGMA temp.table\_info("commodity")  
2022-10-15 11:06:05,956 INFO sqlalchemy.engine.Engine [raw sql] ()  
2022-10-15 11:06:05,958 INFO sqlalchemy.engine.Engine BEGIN (implicit)  
2022-10-15 11:06:05,959 INFO sqlalchemy.engine.Engine   
CREATE TABLE commodity (  
 "index" BIGINT,   
 commodity\_code BIGINT,   
 commodity\_name TEXT  
)  
  
  
2022-10-15 11:06:05,959 INFO sqlalchemy.engine.Engine [no key 0.00052s] ()  
2022-10-15 11:06:05,966 INFO sqlalchemy.engine.Engine CREATE INDEX ix\_commodity\_index ON commodity ("index")  
2022-10-15 11:06:05,968 INFO sqlalchemy.engine.Engine [no key 0.00108s] ()  
2022-10-15 11:06:05,974 INFO sqlalchemy.engine.Engine COMMIT  
2022-10-15 11:06:05,976 INFO sqlalchemy.engine.Engine BEGIN (implicit)  
2022-10-15 11:06:05,977 INFO sqlalchemy.engine.Engine INSERT INTO commodity ("index", commodity\_code, commodity\_name) VALUES (?, ?, ?)  
2022-10-15 11:06:05,978 INFO sqlalchemy.engine.Engine [generated in 0.00063s] ((0, 101, 'muskmelon'), (1, 102, 'strawberry'), (2, 103, 'apple'), (3, 104, 'lemon'))  
2022-10-15 11:06:05,980 INFO sqlalchemy.engine.Engine COMMIT  
\_\_\_\_\_\_\_\_\_\_ the commodity table has been written......

使用sqlacodegen库分别生成对应的数据库表结构。

%%cmd  
sqlacodegen sqlite:///./database/fruits.sqlite --tables sale\_details --outfile sale\_details\_table\_structure.py  
sqlacodegen sqlite:///./database/fruits.sqlite --tables commodity --outfile commodity\_table\_structure.py

Microsoft Windows [Version 10.0.22000.856]  
(c) Microsoft Corporation. All rights reserved.  
  
(USDAlab) C:\Users\richi\omen\_richiebao\omen\_github\USDA\_CH\_endup\USDA\notebook>sqlacodegen sqlite:///./database/fruits.sqlite --tables sale\_details --outfile sale\_details\_table\_structure.py  
  
(USDAlab) C:\Users\richi\omen\_richiebao\omen\_github\USDA\_CH\_endup\USDA\notebook>sqlacodegen sqlite:///./database/fruits.sqlite --tables commodity --outfile commodity\_table\_structure.py  
  
(USDAlab) C:\Users\richi\omen\_richiebao\omen\_github\USDA\_CH\_endup\USDA\notebook>

打开存储有表结构的sale\_details\_table\_structure.py文件，内容如下：

# coding: utf-8  
from sqlalchemy import BigInteger, Column, MetaData, Table  
from sqlalchemy.ext.declarative import declarative\_base  
  
metadata = MetaData()  
t\_sale\_details = Table(  
 'sale\_details', metadata,  
 Column('index', BigInteger, index=True),  
 Column('idx', BigInteger),  
 Column('commodity\_code', BigInteger),  
 Column('number', BigInteger)  
)

打开存储有表结构的commodity\_table\_structure.py文件，内容如下：

# coding: utf-8  
from sqlalchemy import BigInteger, Column, MetaData, Table, Text  
from sqlalchemy.ext.declarative import declarative\_base  
  
metadata = MetaData()  
t\_commodity = Table(  
 'commodity', metadata,  
 Column('index', BigInteger, index=True),  
 Column('commodity\_code', BigInteger),  
 Column('commodity\_name', Text)  
)

#### 5) 查询、增删和修改数据库

##### 1. 查询数据库

* 使用session.query()方法

使用类定义的表结构，在应用session.query()读取数据库时返回的是一个对象’<exporting\_country 12>’，需要给定字段读取具体的值。

读取的方法有多种，可以自行搜索查询。

exporting\_country\_query=session.query(exporting\_country).filter\_by(exporting\_country\_ID=12).first() #.all()将读取所有匹配，.first()仅返回首个匹配对象  
  
print("\_"\*50)  
print(exporting\_country\_query)  
print(exporting\_country\_query.exporting\_country\_name,exporting\_country\_query.exporting\_country\_ID)

2022-10-15 17:30:19,221 INFO sqlalchemy.engine.Engine SELECT exporting\_country."index" AS exporting\_country\_index, exporting\_country."exporting\_country\_ID" AS "exporting\_country\_exporting\_country\_ID", exporting\_country.exporting\_country\_name AS exporting\_country\_exporting\_country\_name   
FROM exporting\_country   
WHERE exporting\_country."exporting\_country\_ID" = ?  
 LIMIT ? OFFSET ?  
2022-10-15 17:30:19,222 INFO sqlalchemy.engine.Engine [cached since 2.283e+04s ago] (12, 1, 0)  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
<exporting\_country 12>  
kenya 12

使用sqlacodegen库生成数据库表结构，是使用sqlalchemy.Table定义。在应用session.query()读取数据库时返回的是一个元组，顺序包含所有字段的值。

# coding: utf-8  
from sqlalchemy import BigInteger, Column, DateTime, MetaData, Table  
  
metadata = MetaData()  
t\_sales=Table(  
 'sales', metadata,  
 Column('index', BigInteger, index=True),  
 Column('idx', BigInteger),  
 Column('date', DateTime),  
 Column('exporting\_country\_ID', BigInteger)  
)  
sales\_query=session.query(t\_sales).filter\_by(idx=1101).first()  
print("\_"\*50)  
print(sales\_query)

2022-10-15 11:11:10,126 INFO sqlalchemy.engine.Engine SELECT sales."index" AS sales\_index, sales.idx AS sales\_idx, sales.date AS sales\_date, sales."exporting\_country\_ID" AS "sales\_exporting\_country\_ID"   
FROM sales   
WHERE sales.idx = ?  
 LIMIT ? OFFSET ?  
2022-10-15 11:11:10,127 INFO sqlalchemy.engine.Engine [generated in 0.00057s] (1101, 1, 0)  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
(0, 1101, datetime.datetime(2020, 3, 5, 0, 0), 12)

* 使用pandas库提供的方法

应用pandas读取数据库，相对sqlite3和SQLAlchemy库而言较为简单，不需要配置表结构，能直接读取。

import sqlite3  
import pandas as pd  
  
db\_fp=r'./database/fruits.sqlite'  
conn=sqlite3.connect(db\_fp)  
df\_sqlite=pd.read\_sql('select \* from sqlite\_master',con=conn) # pd.read\_sql将读取数据库结构(database structure)信息  
df\_sqlite

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **type** | **name** | **tbl\_name** | **rootpage** | **sql** |
| **0** | table | sales | sales | 2 | CREATE TABLE sales (\n\t"index" BIGINT, \n\tid... |
| **1** | index | ix\_sales\_index | sales | 3 | CREATE INDEX ix\_sales\_index ON sales ("index") |
| **2** | table | exporting\_country | exporting\_country | 4 | CREATE TABLE exporting\_country (\n\t"index" IN... |
| **3** | table | sale\_details | sale\_details | 5 | CREATE TABLE sale\_details (\n\t"index" BIGINT,... |
| **4** | index | ix\_sale\_details\_index | sale\_details | 6 | CREATE INDEX ix\_sale\_details\_index ON sale\_det... |
| **5** | table | commodity | commodity | 7 | CREATE TABLE commodity (\n\t"index" BIGINT, \n... |
| **6** | index | ix\_commodity\_index | commodity | 8 | CREATE INDEX ix\_commodity\_index ON commodity (... |

df\_sales=pd.read\_sql\_table('sales', 'sqlite:///./database/fruits.sqlite') # pd.read\_sql\_table()从数据库中读取指定的表  
df\_sales

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **index** | **idx** | **date** | **exporting\_country\_ID** |
| **0** | 0 | 1101 | 2020-03-05 | 12 |
| **1** | 1 | 1102 | 2020-03-07 | 23 |
| **2** | 2 | 1103 | 2020-03-08 | 25 |
| **3** | 3 | 1104 | 2020-03-10 | 12 |
| **4** | 4 | 1105 | 2020-03-12 | 25 |

df\_sales\_query=pd.read\_sql\_query('select idx,exporting\_country\_ID from sales', con=conn) # pd.read\_sql\_query()将根据SQL query 或 SQLAlchemy Selectable查询语句读取特定的值   
df\_sales\_query

|  |  |  |
| --- | --- | --- |
|  | **idx** | **exporting\_country\_ID** |
| **0** | 1101 | 12 |
| **1** | 1102 | 23 |
| **2** | 1103 | 25 |
| **3** | 1104 | 12 |
| **4** | 1105 | 25 |

##### 2.增-删数据库

* 向已有表中增加数据

sqlacodegen库生成数据库表结构并运行，’sales’表则被存储于metadata元数据中。如果再定义一个类，同样指向这个表，则需要配置'extend\_existing': True，表示在已有列基础上进行扩展，即sqlalchemy允许类是表的子集（一个表可以指向多个表结构的类）。

metadata.tables

FacadeDict({'sales': Table('sales', MetaData(), Column('index', BigInteger(), table=<sales>), Column('idx', BigInteger(), table=<sales>), Column('date', DateTime(), table=<sales>), Column('exporting\_country\_ID', BigInteger(), table=<sales>), schema=None)})

class sales(BASE):  
 \_\_tablename\_\_='sales'   
 \_\_table\_args\_\_={'extend\_existing': True}   
   
 index=db.Column(db.Integer, primary\_key=True, autoincrement=True) # 因为该sales类是在执行t\_sales之后定义，只能是在原有表上扩展，无法修改原表结构属性，因此index字段并不会实现自动增加的属性。需要手动增加index字段值  
 idx=db.Column(db.Integer)  
 date=db.Column(db.DateTime)  
 exporting\_country\_ID=db.Column(db.Integer)  
   
from sqlalchemy.orm import sessionmaker  
SESSION=sessionmaker(bind=engine) # 建立会话链接  
session=SESSION() # 实例化   
   
new\_sale=sales(index=5,idx=1106,date=datetime(2020,12,18),exporting\_country\_ID=25)  
session.add(new\_sale)  
session.commit()

2022-10-15 17:39:40,423 INFO sqlalchemy.engine.Engine BEGIN (implicit)  
2022-10-15 17:39:40,426 INFO sqlalchemy.engine.Engine INSERT INTO sales ("index", idx, date, "exporting\_country\_ID") VALUES (?, ?, ?, ?)  
2022-10-15 17:39:40,427 INFO sqlalchemy.engine.Engine [generated in 0.00094s] (5, 1106, '2020-12-18 00:00:00.000000', 25)  
2022-10-15 17:39:40,430 INFO sqlalchemy.engine.Engine COMMIT

从表中读取新增加的数据

del\_sale=session.query(sales).filter\_by(idx=1106).first() # 如果该行中有值为空，例如在增加该行数据时未定义写入index=5字段，该语句返回值会为空。如允许出现空值，在定义表结构时需要配置nullabley=True  
print("\_"\*50)  
print(del\_sale.exporting\_country\_ID,del\_sale.date)

2022-10-15 17:41:57,556 INFO sqlalchemy.engine.Engine BEGIN (implicit)  
2022-10-15 17:41:57,558 INFO sqlalchemy.engine.Engine SELECT sales."index" AS sales\_index, sales.idx AS sales\_idx, sales.date AS sales\_date, sales."exporting\_country\_ID" AS "sales\_exporting\_country\_ID"   
FROM sales   
WHERE sales.idx = ?  
 LIMIT ? OFFSET ?  
2022-10-15 17:41:57,559 INFO sqlalchemy.engine.Engine [generated in 0.00078s] (1106, 1, 0)  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
25 2020-12-18 00:00:00

* 从表中删除已有的数据

session.delete(del\_sale)  
session.commit()

2022-10-15 17:41:59,500 INFO sqlalchemy.engine.Engine DELETE FROM sales WHERE sales."index" = ?  
2022-10-15 17:41:59,501 INFO sqlalchemy.engine.Engine [generated in 0.00152s] (5,)  
2022-10-15 17:41:59,504 INFO sqlalchemy.engine.Engine COMMIT

##### 3.修改数据库

mod\_sale=session.query(sales).filter\_by(idx=1105).first()  
mod\_sale.exporting\_country\_ID=23 # 修改字段值  
session.commit()

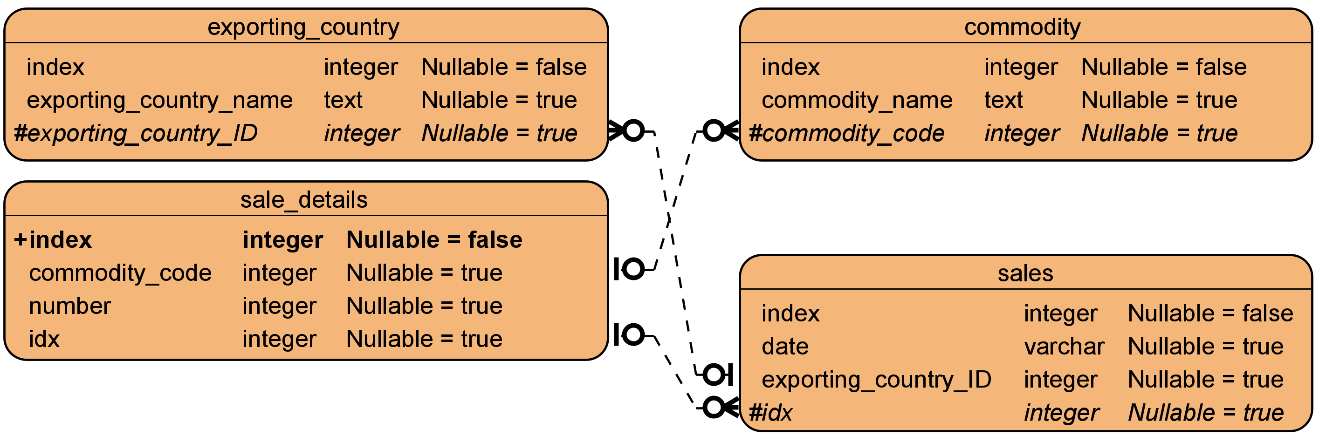
2022-10-15 17:42:23,038 INFO sqlalchemy.engine.Engine BEGIN (implicit)  
2022-10-15 17:42:23,039 INFO sqlalchemy.engine.Engine SELECT sales."index" AS sales\_index, sales.idx AS sales\_idx, sales.date AS sales\_date, sales."exporting\_country\_ID" AS "sales\_exporting\_country\_ID"   
FROM sales   
WHERE sales.idx = ?  
 LIMIT ? OFFSET ?  
2022-10-15 17:42:23,040 INFO sqlalchemy.engine.Engine [cached since 25.48s ago] (1105, 1, 0)  
2022-10-15 17:42:23,043 INFO sqlalchemy.engine.Engine UPDATE sales SET "exporting\_country\_ID"=? WHERE sales."index" = ?  
2022-10-15 17:42:23,044 INFO sqlalchemy.engine.Engine [generated in 0.00066s] (23, 4)  
2022-10-15 17:42:23,046 INFO sqlalchemy.engine.Engine COMMIT

### 1.3.2 SQLite数据库的表间关系（多表关联)

上文建立表结构时，并未配置表间关联，各个表是独立的，如果想通过一个表的数据字段查询另一个表的字段内容就比较困难，例如想根据销售数量，查询对应的商品名称时，是无法直接在’commodity’商品表中直接查询商品名称的，需要先根据待查询的销售数量例如300，在’sale\_details’销售明细表里找到对应的commodity\_code商品编码为102，根据这个商品编码，再在’commodity’商品表找到对应的商品名称为’strawberry’。因为这个过程很繁琐，尤其数据库结构和表结构进一步复杂时，这个问题会更凸显，因此需要建立表间的联系。

SQLite的表间关系配置，可以包括1对多，多对1，1对1和多对多，SQLAlchemy给出表结构配置的方法[Relationship Configuration](https://docs.sqlalchemy.org/en/14/orm/relationships.html)⑦，可以根据其阐述进行配置。在配置时，参数’back\_populates’定义反向引用，用于建立双向关系，例如销售明细表->商品表，均包括relationship()语句，显示定义关系属性。如果使用参数’backref’添加反向引用，会自动在另一侧建立关系属性，为’back\_populates’的简化形式。参数uselist=True（默认值）时，为1对多关系，如果配置1对1时，需要将其配置为uselist=False。在配置表关系时，为了能够清晰易读，通常以表名作为变量名，例如销售明细表->商品表，销售明细表为父表(parent)，商品表为子表(child)，父表中语句commodity=relationship('commodity',uselist=False, back\_populates="sale\_details")以子表为变量名，而子表中语句sale\_details=relationship('sale\_details',back\_populates="commodity")以父表为变量名，这样可以更清晰的表述表之间的关系。

可以使用sqlacodegen库生成数据库表结构，往往应用于类似pandas写入数据库而没有定义表结构的情况下，这是一种逆向工程。下述已经定义4个表结构，那么则可以使用逆向工程反馈表的内容和表之间的关系，例如使用[Visual Paradigm](https://www.visual-paradigm.com/)⑧反馈有下表关系，即统一建模语言（Unified Modeling Language,UML）。可以清晰直观的读出表结构和表间关系，其中’sales’是’exporting\_country’的父表，连接的关键字段（ForeignKey）是’exporting\_country\_ID’；同时’sales’是’sale\_details’的子表，联系的关键字段是’idx’，其它的关系以此类推，一目了然。



from sqlalchemy import create\_engine  
  
db\_fp=r'./database/fruits\_relational.sqlite'  
engine=create\_engine('sqlite:///'+'\\\\'.join(db\_fp.split('\\')),echo=True)

from sqlalchemy.ext.declarative import declarative\_base  
import sqlalchemy as db  
from sqlalchemy.orm import relationship  
from sqlalchemy.schema import ForeignKey  
  
BASE=declarative\_base() # 基本映射类，需要自定义的表结构继承  
  
# 销售明细表  
class sale\_details(BASE):  
 \_\_tablename\_\_='sale\_details'  
 index=db.Column(db.Integer, primary\_key=True, autoincrement=True)   
 commodity\_code=db.Column(db.Integer)  
 number=db.Column(db.Integer)   
 idx=db.Column(db.Integer)  
   
 sales=relationship('sales',uselist=False, back\_populates="sale\_details")  
   
 commodity=relationship('commodity',uselist=False, back\_populates="sale\_details")   
  
# 销售表  
class sales(BASE):  
 \_\_tablename\_\_='sales'   
   
 index=db.Column(db.Integer, primary\_key=True, autoincrement=True) # 自动生成的索引列   
 date=db.Column(db.DateTime)  
 exporting\_country\_ID=db.Column(db.Integer)  
   
 exporting\_country=relationship('exporting\_country',uselist=False,back\_populates="sales")  
   
 idx=db.Column(db.Integer,ForeignKey('sale\_details.idx'))  
 sale\_details=relationship('sale\_details',back\_populates="sales")   
  
# 出口国表   
class exporting\_country(BASE):  
 \_\_tablename\_\_='exporting\_country'   
   
 index=db.Column(db.Integer, primary\_key=True, autoincrement=True)   
 exporting\_country\_name=db.Column(db.Text)   
   
 exporting\_country\_ID=db.Column(db.Integer,db.ForeignKey('sales.exporting\_country\_ID'))  
 sales=relationship('sales',back\_populates="exporting\_country")   
  
# 商品表  
class commodity(BASE):  
 \_\_tablename\_\_='commodity'  
 index=db.Column(db.Integer, primary\_key=True, autoincrement=True)   
 commodity\_name=db.Column(db.Text)  
   
 commodity\_code=db.Column(db.Integer,ForeignKey('sale\_details.commodity\_code'))  
 sale\_details=relationship('sale\_details',back\_populates="commodity")  
   
BASE.metadata.create\_all(engine, checkfirst=True) # 将所有表结构写入数据库

2022-10-15 18:08:14,506 INFO sqlalchemy.engine.Engine BEGIN (implicit)  
2022-10-15 18:08:14,507 INFO sqlalchemy.engine.Engine PRAGMA main.table\_info("sale\_details")  
2022-10-15 18:08:14,508 INFO sqlalchemy.engine.Engine [raw sql] ()  
2022-10-15 18:08:14,510 INFO sqlalchemy.engine.Engine PRAGMA temp.table\_info("sale\_details")  
2022-10-15 18:08:14,510 INFO sqlalchemy.engine.Engine [raw sql] ()  
2022-10-15 18:08:14,511 INFO sqlalchemy.engine.Engine PRAGMA main.table\_info("sales")  
2022-10-15 18:08:14,512 INFO sqlalchemy.engine.Engine [raw sql] ()  
2022-10-15 18:08:14,513 INFO sqlalchemy.engine.Engine PRAGMA temp.table\_info("sales")  
2022-10-15 18:08:14,514 INFO sqlalchemy.engine.Engine [raw sql] ()  
2022-10-15 18:08:14,514 INFO sqlalchemy.engine.Engine PRAGMA main.table\_info("exporting\_country")  
2022-10-15 18:08:14,515 INFO sqlalchemy.engine.Engine [raw sql] ()  
2022-10-15 18:08:14,516 INFO sqlalchemy.engine.Engine PRAGMA temp.table\_info("exporting\_country")  
2022-10-15 18:08:14,517 INFO sqlalchemy.engine.Engine [raw sql] ()  
2022-10-15 18:08:14,517 INFO sqlalchemy.engine.Engine PRAGMA main.table\_info("commodity")  
2022-10-15 18:08:14,518 INFO sqlalchemy.engine.Engine [raw sql] ()  
2022-10-15 18:08:14,518 INFO sqlalchemy.engine.Engine PRAGMA temp.table\_info("commodity")  
2022-10-15 18:08:14,519 INFO sqlalchemy.engine.Engine [raw sql] ()  
2022-10-15 18:08:14,520 INFO sqlalchemy.engine.Engine   
CREATE TABLE sale\_details (  
 "index" INTEGER NOT NULL,   
 commodity\_code INTEGER,   
 number INTEGER,   
 idx INTEGER,   
 PRIMARY KEY ("index")  
)  
  
  
2022-10-15 18:08:14,520 INFO sqlalchemy.engine.Engine [no key 0.00047s] ()  
2022-10-15 18:08:14,528 INFO sqlalchemy.engine.Engine   
CREATE TABLE sales (  
 "index" INTEGER NOT NULL,   
 date DATETIME,   
 "exporting\_country\_ID" INTEGER,   
 idx INTEGER,   
 PRIMARY KEY ("index"),   
 FOREIGN KEY(idx) REFERENCES sale\_details (idx)  
)  
  
  
2022-10-15 18:08:14,529 INFO sqlalchemy.engine.Engine [no key 0.00062s] ()  
2022-10-15 18:08:14,535 INFO sqlalchemy.engine.Engine   
CREATE TABLE commodity (  
 "index" INTEGER NOT NULL,   
 commodity\_name TEXT,   
 commodity\_code INTEGER,   
 PRIMARY KEY ("index"),   
 FOREIGN KEY(commodity\_code) REFERENCES sale\_details (commodity\_code)  
)  
  
  
2022-10-15 18:08:14,537 INFO sqlalchemy.engine.Engine [no key 0.00141s] ()  
2022-10-15 18:08:14,544 INFO sqlalchemy.engine.Engine   
CREATE TABLE exporting\_country (  
 "index" INTEGER NOT NULL,   
 exporting\_country\_name TEXT,   
 "exporting\_country\_ID" INTEGER,   
 PRIMARY KEY ("index"),   
 FOREIGN KEY("exporting\_country\_ID") REFERENCES sales ("exporting\_country\_ID")  
)  
  
  
2022-10-15 18:08:14,545 INFO sqlalchemy.engine.Engine [no key 0.00102s] ()  
2022-10-15 18:08:14,553 INFO sqlalchemy.engine.Engine COMMIT

将数据写入各个表。

from sqlalchemy.orm import sessionmaker  
  
SESSION=sessionmaker(bind=engine) # 建立会话链接  
session=SESSION() # 实例化  
  
sales\_=zip\_dic\_tableSQLite(sales\_dic,sales)  
exporting\_country\_=zip\_dic\_tableSQLite(exporting\_country\_dic,exporting\_country)  
sale\_details\_=zip\_dic\_tableSQLite(sale\_details\_dic,sale\_details)  
commodity\_=zip\_dic\_tableSQLite(commodity\_dic,commodity)  
  
session.add\_all(sales\_)  
session.add\_all(exporting\_country\_)  
session.add\_all(sale\_details\_)  
session.add\_all(commodity\_)  
session.commit()

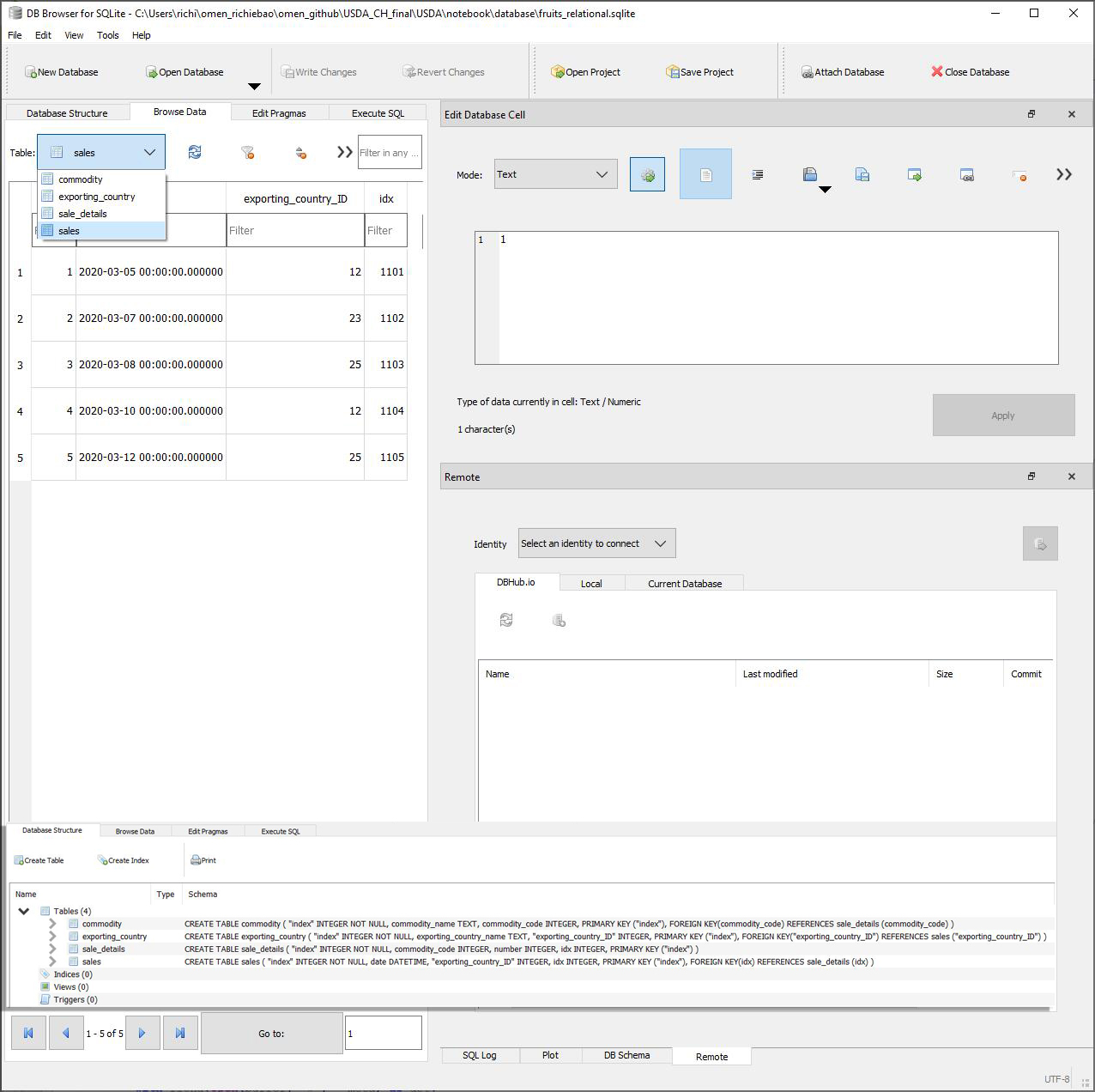
2022-10-15 18:08:38,919 INFO sqlalchemy.engine.Engine BEGIN (implicit)  
2022-10-15 18:08:38,920 INFO sqlalchemy.engine.Engine INSERT INTO sale\_details (commodity\_code, number, idx) VALUES (?, ?, ?)  
2022-10-15 18:08:38,920 INFO sqlalchemy.engine.Engine [generated in 0.00060s] (101, 1100, 1101)  
2022-10-15 18:08:38,923 INFO sqlalchemy.engine.Engine INSERT INTO sale\_details (commodity\_code, number, idx) VALUES (?, ?, ?)  
2022-10-15 18:08:38,924 INFO sqlalchemy.engine.Engine [cached since 0.004276s ago] (102, 300, 1101)  
2022-10-15 18:08:38,926 INFO sqlalchemy.engine.Engine INSERT INTO sale\_details (commodity\_code, number, idx) VALUES (?, ?, ?)  
2022-10-15 18:08:38,927 INFO sqlalchemy.engine.Engine [cached since 0.007129s ago] (103, 1700, 1102)  
2022-10-15 18:08:38,928 INFO sqlalchemy.engine.Engine INSERT INTO sale\_details (commodity\_code, number, idx) VALUES (?, ?, ?)  
2022-10-15 18:08:38,929 INFO sqlalchemy.engine.Engine [cached since 0.009754s ago] (104, 500, 1103)  
2022-10-15 18:08:38,931 INFO sqlalchemy.engine.Engine INSERT INTO sale\_details (commodity\_code, number, idx) VALUES (?, ?, ?)  
2022-10-15 18:08:38,931 INFO sqlalchemy.engine.Engine [cached since 0.01176s ago] (101, 2500, 1104)  
2022-10-15 18:08:38,932 INFO sqlalchemy.engine.Engine INSERT INTO sale\_details (commodity\_code, number, idx) VALUES (?, ?, ?)  
2022-10-15 18:08:38,933 INFO sqlalchemy.engine.Engine [cached since 0.01373s ago] (103, 2000, 1105)  
2022-10-15 18:08:38,935 INFO sqlalchemy.engine.Engine INSERT INTO sale\_details (commodity\_code, number, idx) VALUES (?, ?, ?)  
2022-10-15 18:08:38,936 INFO sqlalchemy.engine.Engine [cached since 0.01609s ago] (104, 700, 1105)  
2022-10-15 18:08:38,937 INFO sqlalchemy.engine.Engine INSERT INTO commodity (commodity\_name, commodity\_code) VALUES (?, ?)  
2022-10-15 18:08:38,938 INFO sqlalchemy.engine.Engine [generated in 0.00087s] ('muskmelon', 101)  
2022-10-15 18:08:38,940 INFO sqlalchemy.engine.Engine INSERT INTO commodity (commodity\_name, commodity\_code) VALUES (?, ?)  
2022-10-15 18:08:38,940 INFO sqlalchemy.engine.Engine [cached since 0.003051s ago] ('strawberry', 102)  
2022-10-15 18:08:38,941 INFO sqlalchemy.engine.Engine INSERT INTO commodity (commodity\_name, commodity\_code) VALUES (?, ?)  
2022-10-15 18:08:38,942 INFO sqlalchemy.engine.Engine [cached since 0.004218s ago] ('apple', 103)  
2022-10-15 18:08:38,942 INFO sqlalchemy.engine.Engine INSERT INTO commodity (commodity\_name, commodity\_code) VALUES (?, ?)  
2022-10-15 18:08:38,943 INFO sqlalchemy.engine.Engine [cached since 0.005403s ago] ('lemon', 104)  
2022-10-15 18:08:38,991 INFO sqlalchemy.engine.Engine INSERT INTO sales (date, "exporting\_country\_ID", idx) VALUES (?, ?, ?)  
2022-10-15 18:08:38,992 INFO sqlalchemy.engine.Engine [generated in 0.00057s] ('2020-03-05 00:00:00.000000', 12, 1101)  
2022-10-15 18:08:38,992 INFO sqlalchemy.engine.Engine INSERT INTO sales (date, "exporting\_country\_ID", idx) VALUES (?, ?, ?)  
2022-10-15 18:08:38,993 INFO sqlalchemy.engine.Engine [cached since 0.001759s ago] ('2020-03-07 00:00:00.000000', 23, 1102)  
2022-10-15 18:08:38,993 INFO sqlalchemy.engine.Engine INSERT INTO sales (date, "exporting\_country\_ID", idx) VALUES (?, ?, ?)  
2022-10-15 18:08:38,993 INFO sqlalchemy.engine.Engine [cached since 0.002511s ago] ('2020-03-08 00:00:00.000000', 25, 1103)  
2022-10-15 18:08:38,994 INFO sqlalchemy.engine.Engine INSERT INTO sales (date, "exporting\_country\_ID", idx) VALUES (?, ?, ?)  
2022-10-15 18:08:38,994 INFO sqlalchemy.engine.Engine [cached since 0.003507s ago] ('2020-03-10 00:00:00.000000', 12, 1104)  
2022-10-15 18:08:38,995 INFO sqlalchemy.engine.Engine INSERT INTO sales (date, "exporting\_country\_ID", idx) VALUES (?, ?, ?)  
2022-10-15 18:08:38,995 INFO sqlalchemy.engine.Engine [cached since 0.004366s ago] ('2020-03-12 00:00:00.000000', 25, 1105)  
2022-10-15 18:08:38,997 INFO sqlalchemy.engine.Engine INSERT INTO exporting\_country (exporting\_country\_name, "exporting\_country\_ID") VALUES (?, ?)  
2022-10-15 18:08:38,997 INFO sqlalchemy.engine.Engine [generated in 0.00046s] ('kenya', 12)  
2022-10-15 18:08:38,998 INFO sqlalchemy.engine.Engine INSERT INTO exporting\_country (exporting\_country\_name, "exporting\_country\_ID") VALUES (?, ?)  
2022-10-15 18:08:38,998 INFO sqlalchemy.engine.Engine [cached since 0.001737s ago] ('brazil', 23)  
2022-10-15 18:08:38,999 INFO sqlalchemy.engine.Engine INSERT INTO exporting\_country (exporting\_country\_name, "exporting\_country\_ID") VALUES (?, ?)  
2022-10-15 18:08:38,999 INFO sqlalchemy.engine.Engine [cached since 0.002792s ago] ('peru', 25)  
2022-10-15 18:08:39,001 INFO sqlalchemy.engine.Engine COMMIT

通过正向引用或者反向引用轻松的获取关联表中对应的数据。例如由商品销售数量找到对应的商品名称。

sale\_details\_info=session.query(sale\_details).filter\_by(number=300).first()  
commodity\_info=sale\_details\_info.commodity  
commodity\_name=commodity\_info.commodity\_name  
print("\_"\*50)  
print("销量number=300的商品名为:%s"%commodity\_name)

2022-10-15 18:08:55,687 INFO sqlalchemy.engine.Engine BEGIN (implicit)  
2022-10-15 18:08:55,690 INFO sqlalchemy.engine.Engine SELECT sale\_details."index" AS sale\_details\_index, sale\_details.commodity\_code AS sale\_details\_commodity\_code, sale\_details.number AS sale\_details\_number, sale\_details.idx AS sale\_details\_idx   
FROM sale\_details   
WHERE sale\_details.number = ?  
 LIMIT ? OFFSET ?  
2022-10-15 18:08:55,691 INFO sqlalchemy.engine.Engine [generated in 0.00076s] (300, 1, 0)  
2022-10-15 18:08:55,695 INFO sqlalchemy.engine.Engine SELECT commodity."index" AS commodity\_index, commodity.commodity\_name AS commodity\_commodity\_name, commodity.commodity\_code AS commodity\_commodity\_code   
FROM commodity   
WHERE ? = commodity.commodity\_code  
2022-10-15 18:08:55,696 INFO sqlalchemy.engine.Engine [generated in 0.00101s] (102,)  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
销量number=300的商品名为:strawberry

* 使用[DB Browser for SQLite(DB4S)](https://sqlitebrowser.org/)查看数据库



### 1.3.3 PostgreSQL数据库

[PostgreSQL](https://www.postgresql.org/)⑨是一个强大开源的对象关系数据库系统（open source object-relational database system）。经过30多年的发展，其在可靠性、特征的健壮性和性能方面赢得了很高的声誉。同时，因为PostgreSQL可以存储具有投影坐标系统的地理空间数据，在[QGIS](https://qgis.org/en/site/)⑩等地理信息系统工具平台下可以直接从PostgreSQL（PostGIS）中读入与显示数据，建立地图，弥补了Python在地图表达上的不足，而又可以充分利用Python的数据处理能力。

通常使用开源的[pgAdmin](https://www.pgadmin.org/)⑪工具查看管理PostgreSQL数据库。

用[Array of Things(AoT) 城市环境传感器](https://arrayofthings.github.io/)⑫数据演示Python数据处理、写入和读取PostgreSQL数据库，及使用[QGIS](https://www.qgis.org/en/site/)调入数据库中的数据，建立地图的方法。[数据下载地址](https://www.mcs.anl.gov/research/projects/waggle/downloads/datasets/index.php)：https://www.mcs.anl.gov/research/projects/waggle/downloads/datasets/index.php

2019年10月，AoT团队基于已有研究成功申请了国家科学基金（美）的资助，创建新的软硬件基础设施构建城市环境传感器网络，并开源了获取的实时传感器数据。关于数据的详细说明可以查看数据包中的说明文档，详细解释了各字段的含义，同时给出了所使用传感器的型号和详细说明链接，这对于研究城市环境下的局地小气候具有重要价值。

#### 1）读取nodes数据，并转换为GeoDataFrame格式

nodes.csv数据文件，包括所有布置于城市的传感器位置节点编号、坐标（wgs84）、地址等信息。使用经纬度，通过[shapely库](https://shapely.readthedocs.io/en/stable/manual.html)建立地理空间点后，用[geopandas库](https://geopandas.org/en/stable/)，给定坐标系统wgs84的epsg编号4326，转换为GeoDataFrame格式数据，方便地理信息数据的存储、分析和写入PostgreSQL数据库。

import pandas as pd  
import geopandas as gpd  
from shapely.geometry import Point  
  
AoT\_nodes\_fp='./data/AoT\_Chicago.complete.2021-12-20/nodes.csv'  
AoT\_nodes\_df=pd.read\_csv(AoT\_nodes\_fp,sep=",",header=0)  
  
epsg\_wgs84=4326  
AoT\_nodes\_df["geometry"]=AoT\_nodes\_df.apply(lambda row:Point(row.lon,row.lat),axis=1) # 使用shapely库建立几何点数据  
AoT\_nodes\_gdf=gpd.GeoDataFrame(AoT\_nodes\_df,crs=epsg\_wgs84)  
print("crs{}{}".format("-"\*10,AoT\_nodes\_gdf.crs))  
AoT\_nodes\_gdf

crs----------epsg:4326

|  | **node\_id** | **project\_id** | **vsn** | **address** | **lat** | **lon** | **description** | **start\_timestamp** | **end\_timestamp** | **geometry** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 001e0610ba46 | AoT\_Chicago | 004 | State St & Jackson Blvd Chicago IL | 41.878377 | -87.627678 | AoT Chicago (S) [C] | 2017/10/09 00:00:00 | NaN | POINT (-87.62768 41.87838) |
| **1** | 001e0610ba3b | AoT\_Chicago | 006 | 18th St & Lake Shore Dr Chicago IL | 41.858136 | -87.616055 | AoT Chicago (S) | 2017/08/08 00:00:00 | NaN | POINT (-87.61606 41.85814) |
| **2** | 001e0610f02f | AoT\_Chicago | 00A | Lake Shore Drive & Fullerton Ave Chicago IL | 41.926261 | -87.630758 | AoT Chicago (S) [CA] | 2018/05/07 00:00:00 | NaN | POINT (-87.63076 41.92626) |
| **3** | 001e0610ba8f | AoT\_Chicago | 00D | Cornell & 47th St Chicago IL | 41.810342 | -87.590228 | AoT Chicago (S) | 2017/08/08 00:00:00 | NaN | POINT (-87.59023 41.81034) |
| **4** | 001e0610ba16 | AoT\_Chicago | 010 | Homan Ave & Roosevelt Rd Chicago IL | 41.866349 | -87.710543 | AoT Chicago (S) [C] | 2018/07/18 00:00:00 | NaN | POINT (-87.71054 41.86635) |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **121** | 001e06118433 | AoT\_Chicago | 10E | ComEd Training Center | 41.829806 | -87.659467 | AoT Chicago (S) [CP] {ComEd} | 2019/04/25 00:00:00 | NaN | POINT (-87.65947 41.82981) |
| **122** | 001e061183bf | AoT\_Chicago | 11A | ComEd Training Center | 41.829806 | -87.659467 | AoT Chicago (S) [CP] {ComEd} | 2019/04/25 00:00:00 | NaN | POINT (-87.65947 41.82981) |
| **123** | 001e0611804d | AoT\_Chicago | 11E | ComEd Training Center | 41.829806 | -87.659467 | AoT Chicago (S) [CP] {ComEd} | 2019/04/25 00:00:00 | NaN | POINT (-87.65947 41.82981) |
| **124** | 001e061182a2 | AoT\_Chicago | 13B | ComEd Training Center | 41.829806 | -87.659467 | AoT Chicago (S) [CP] {ComEd} | 2019/04/25 00:00:00 | NaN | POINT (-87.65947 41.82981) |
| **125** | 001e061144be | AoT\_Chicago | 890 | UChicago, Regenstine Chicago IL | 41.792543 | -87.600008 | AoT Chicago (S) [C] {UChicago} | 2018/03/15 00:00:00 | NaN | POINT (-87.60001 41.79254) |

126 rows × 10 columns

从[Chicago Data Portal](https://data.cityofchicago.org/)⑬中搜索下载行政区划数据，读取后与nodes数据叠合显示，方便定位传感器在城市中的位置。

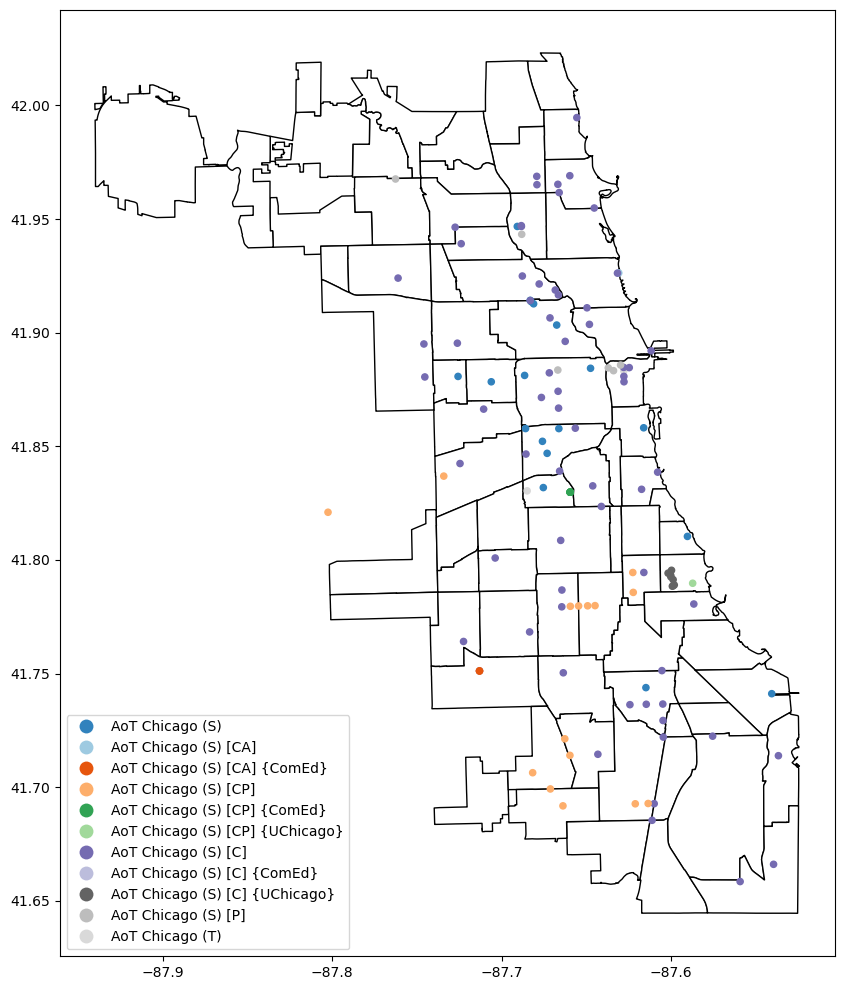
Chicago\_community\_areas\_fp='./data/ChicagoCommunityAreas/ChicagoCommunityAreas.shp'  
Chicago\_community\_areas=gpd.read\_file(Chicago\_community\_areas\_fp)  
print("crs<Chicago\_community\_areas>:{}".format(Chicago\_community\_areas.crs))  
Chicago\_community\_areas

crs<Chicago\_community\_areas>:GEOGCS["WGS84(DD)",DATUM["WGS84",SPHEROID["WGS84",6378137,298.257223563]],PRIMEM["Greenwich",0],UNIT["degree",0.0174532925199433],AXIS["Longitude",EAST],AXIS["Latitude",NORTH]]

|  | **area** | **area\_num\_1** | **area\_numbe** | **comarea** | **comarea\_id** | **community** | **perimeter** | **shape\_area** | **shape\_len** | **geometry** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 0.0 | 35 | 35 | 0.0 | 0.0 | DOUGLAS | 0.0 | 4.600462e+07 | 31027.054510 | POLYGON ((-87.60914 41.84469, -87.60915 41.844... |
| **1** | 0.0 | 36 | 36 | 0.0 | 0.0 | OAKLAND | 0.0 | 1.691396e+07 | 19565.506153 | POLYGON ((-87.59215 41.81693, -87.59231 41.816... |
| **2** | 0.0 | 37 | 37 | 0.0 | 0.0 | FULLER PARK | 0.0 | 1.991670e+07 | 25339.089750 | POLYGON ((-87.62880 41.80189, -87.62879 41.801... |
| **3** | 0.0 | 38 | 38 | 0.0 | 0.0 | GRAND BOULEVARD | 0.0 | 4.849250e+07 | 28196.837157 | POLYGON ((-87.60671 41.81681, -87.60670 41.816... |
| **4** | 0.0 | 39 | 39 | 0.0 | 0.0 | KENWOOD | 0.0 | 2.907174e+07 | 23325.167906 | POLYGON ((-87.59215 41.81693, -87.59215 41.816... |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **72** | 0.0 | 74 | 74 | 0.0 | 0.0 | MOUNT GREENWOOD | 0.0 | 7.558429e+07 | 48665.130539 | POLYGON ((-87.69646 41.70714, -87.69644 41.706... |
| **73** | 0.0 | 75 | 75 | 0.0 | 0.0 | MORGAN PARK | 0.0 | 9.187734e+07 | 46396.419362 | POLYGON ((-87.64215 41.68508, -87.64249 41.685... |
| **74** | 0.0 | 76 | 76 | 0.0 | 0.0 | OHARE | 0.0 | 3.718356e+08 | 173625.984660 | MULTIPOLYGON (((-87.83658 41.98640, -87.83658 ... |
| **75** | 0.0 | 77 | 77 | 0.0 | 0.0 | EDGEWATER | 0.0 | 4.844999e+07 | 31004.830946 | POLYGON ((-87.65456 41.99817, -87.65456 41.998... |
| **76** | 0.0 | 9 | 9 | 0.0 | 0.0 | EDISON PARK | 0.0 | 3.163631e+07 | 25937.226841 | POLYGON ((-87.80676 42.00084, -87.80676 42.000... |

77 rows × 10 columns

import matplotlib.pyplot as plt  
  
fig, ax=plt.subplots(figsize=(10,15))  
Chicago\_community\_areas.plot(ax=ax,color='white', edgecolor='black')  
AoT\_nodes\_gdf.plot(ax=ax,markersize=20,column='description',legend=True,cmap='tab20c',legend\_kwds={'loc': 'lower left'})  
#ax.axis('off')  
plt.show()



#### 2）GeoDataFrame数据读写PostgreSQL数据库

读写GeoDataFrame数据于PostgreSQL数据库，分别定义gpd2postSQL和postSQL2gpd函数，可以放置于自定义的.py文件下，例如本书定义的util\_database.py文件，方便日后调用。需要注意，建立数据库时，首先本地安装PostgreSQL，再使用安装的pgAdmin工具建立数据库，例如本例建立数据库名为’AoT’,用户名为’postgres’，密码为’123456’。同时，要在pgAdmin的Query Tool下执行CREATE EXTENSION postgis;命令，从而可以存储具有坐标系统的地理几何对象，否则将GeoDataFrame数据写入PostgreSQL数据库时，会提示错误。

def gpd2postSQL(gdf,table\_name,\*\*kwargs):   
 '''  
 function - 将GeoDataFrame格式数据写入PostgreSQL数据库  
   
 Paras:  
 gdf - GeoDataFrame格式数据，含geometry字段（几何对象，点、线和面，数据值对应定义的坐标系统）；GeoDataFrame  
 table\_name - 写入数据库中的表名；string  
 \*\*kwargs - 连接数据库相关信息，包括myusername（数据库的用户名），mypassword（用户密钥），mydatabase（数据库名）；string  
   
 Returns:  
 None  
 '''   
 from sqlalchemy import create\_engine  
   
 engine=create\_engine("postgresql://{myusername}:{mypassword}@localhost:5432/{mydatabase}".format(myusername=kwargs['myusername'],mypassword=kwargs['mypassword'],mydatabase=kwargs['mydatabase']))   
 gdf.to\_postgis(table\_name, con=engine, if\_exists='replace', index=False,)   
 print("\_"\*50)  
 print('The GeoDataFrame has been written to the PostgreSQL database.The table name is {}.'.format(table\_name))  
  
def postSQL2gpd(table\_name,geom\_col='geometry',\*\*kwargs):   
 '''  
 function - 读取PostgreSQL数据库中的表为GeoDataFrame格式数据  
   
 Paras:  
 table\_name - 待读取数据库中的表名；string  
 geom\_col='geometry' - 几何对象，常规默认字段为'geometry'；string  
 \*\*kwargs - 连接数据库相关信息，包括myusername（数据库的用户名），mypassword（用户密钥），mydatabase（数据库名）；string  
 Returns:  
 读取的表数据；GeoDataFrame  
 '''  
 from sqlalchemy import create\_engine  
 import geopandas as gpd   
   
 engine=create\_engine("postgresql://{myusername}:{mypassword}@localhost:5432/{mydatabase}".format(myusername=kwargs['myusername'],mypassword=kwargs['mypassword'],mydatabase=kwargs['mydatabase']))   
 gdf=gpd.read\_postgis(table\_name, con=engine,geom\_col=geom\_col)  
 print("\_"\*50)  
 print('The data has been read from PostgreSQL database. The table name is {}.'.format(table\_name))   
 return gdf   
   
gpd2postSQL(AoT\_nodes\_gdf,table\_name='AoT\_nodes',myusername='postgres',mypassword='123456',mydatabase='AoT')   
gpd2postSQL(Chicago\_community\_areas,table\_name='Chicago\_community\_areas',myusername='postgres',mypassword='123456',mydatabase='AoT')

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
The GeoDataFrame has been written to the PostgreSQL database.The table name is AoT\_nodes.  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
The GeoDataFrame has been written to the PostgreSQL database.The table name is Chicago\_community\_areas.

为方便后期QIGS调用建立地图，可以直接定义投影后再写入数据库。

epsg\_Chicago=32616  
gpd2postSQL(AoT\_nodes\_gdf.to\_crs(epsg\_Chicago),table\_name='AoT\_nodes\_prj',myusername='postgres',mypassword='123456',mydatabase='AoT')   
gpd2postSQL(Chicago\_community\_areas.to\_crs(epsg\_Chicago),table\_name='Chicago\_community\_areas\_prj',myusername='postgres',mypassword='123456',mydatabase='AoT')

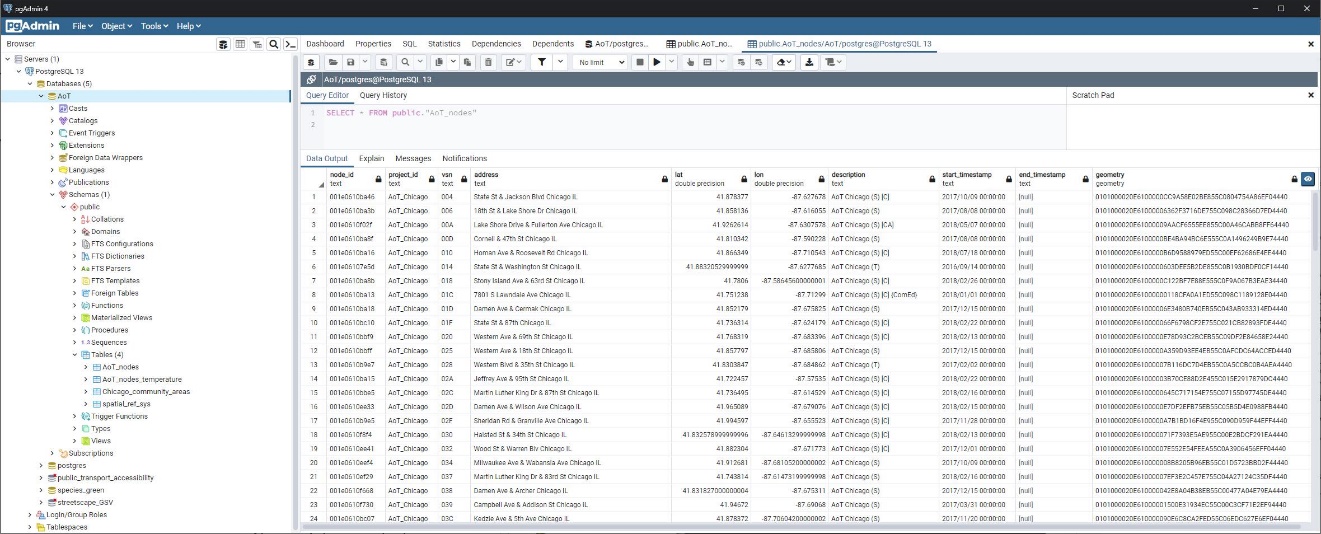
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
The GeoDataFrame has been written to the PostgreSQL database.The table name is AoT\_nodes\_prj.  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
The GeoDataFrame has been written to the PostgreSQL database.The table name is Chicago\_community\_areas\_prj.

在建立一个研究项目时，通常将基本的数据写入数据库后，再从数据库中调用对应的表读入数据，进行后续的分析。

AoT\_nodes\_gdf=postSQL2gpd(table\_name='AoT\_nodes',geom\_col='geometry',myusername='postgres',mypassword='123456',mydatabase='AoT')  
Chicago\_community\_areas=postSQL2gpd(table\_name='Chicago\_community\_areas',geom\_col='geometry',myusername='postgres',mypassword='123456',mydatabase='AoT')

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
The data has been read from PostgreSQL database. The table name is AoT\_nodes.  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
The data has been read from PostgreSQL database. The table name is Chicago\_community\_areas.

打开pgAdmin工具，可以查看写入的数据。



#### 3）读取data传感器记录的数据与初步处理

截止December 20 2021 19:18:35 CS时，data数据约有39.5GB，并且为单独一个csv格式文件。如果内存量较小，则需要分批读入处理再写入数据库。这里仅示范读取2018/01/01一天所记录的数据，并计算每小时的温度均值（对应处理’value\_hrf’字段），再将其对应’node\_id’字段，与*AoT\_nodes\_gdf*变量合并后，写入数据库。

AoT\_data\_fp=r"F:\data\AoT\_Chicago.complete.2021-12-20\data"  
chunksize=10\*\*6  
for chunk in pd.read\_csv(AoT\_data\_fp,chunksize=chunksize) :  
 AoT\_data\_part=chunk  
 break  
AoT\_data\_20180101=AoT\_data\_part[(AoT\_data\_part['timestamp'] >= '2018/01/01 00:00:00') & (AoT\_data\_part['timestamp'] <= '2018/01/01 23:59:59')]  
print("parameter-Sensor parameter that was measured:{}\n{}".format(AoT\_data\_20180101.parameter.unique(),"\_"\*50))  
print("sensor-Sensor that was measured:{}\n{}".format(AoT\_data\_20180101.sensor.unique(),"\_"\*50))  
AoT\_data\_20180101

parameter-Sensor parameter that was measured:['temperature' 'id' 'concentration' 'pressure' 'humidity' 'ir\_intensity'  
 'uv\_intensity' 'visible\_light\_intensity' 'intensity']  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
sensor-Sensor that was measured:['at0' 'at1' 'at2' 'at3' 'chemsense' 'co' 'h2s' 'lps25h' 'no2' 'o3'  
 'oxidizing\_gases' 'reducing\_gases' 'sht25' 'si1145' 'so2' 'apds\_9006\_020'  
 'hih6130' 'ml8511' 'mlx75305' 'tmp421' 'tsl250rd' 'tsl260rd' 'bmp180'  
 'hih4030' 'htu21d' 'metsense' 'pr103j2' 'spv1840lr5h\_b' 'tmp112' 'tsys01']  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

|  | **timestamp** | **node\_id** | **subsystem** | **sensor** | **parameter** | **value\_raw** | **value\_hrf** |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 2018/01/01 00:00:06 | 001e0610e532 | chemsense | at0 | temperature | -1106 | -11.06 |
| **1** | 2018/01/01 00:00:06 | 001e0610e532 | chemsense | at1 | temperature | -1077 | -10.77 |
| **2** | 2018/01/01 00:00:06 | 001e0610e532 | chemsense | at2 | temperature | -1009 | -10.09 |
| **3** | 2018/01/01 00:00:06 | 001e0610e532 | chemsense | at3 | temperature | -972 | -9.72 |
| **4** | 2018/01/01 00:00:06 | 001e0610e532 | chemsense | chemsense | id | NaN | 541eec3ebfa6 |
| **...** | ... | ... | ... | ... | ... | ... | ... |
| **769628** | 2018/01/01 23:59:59 | 001e0610e540 | metsense | pr103j2 | temperature | 372 | -17.15 |
| **769629** | 2018/01/01 23:59:59 | 001e0610e540 | metsense | spv1840lr5h\_b | intensity | 811 | NaN |
| **769630** | 2018/01/01 23:59:59 | 001e0610e540 | metsense | tmp112 | temperature | NaN | -17.81 |
| **769631** | 2018/01/01 23:59:59 | 001e0610e540 | metsense | tsl250rd | intensity | 2 | 0.101 |
| **769632** | 2018/01/01 23:59:59 | 001e0610e540 | metsense | tsys01 | temperature | NaN | -18.47 |

769633 rows × 7 columns

读取的data数据，各个字段数据为字符串类型，应用pd.to\_numeric方法将’value\_hrf’数值字段（已转换的各传感器测量值）转换为数值类型。

from tqdm import tqdm  
  
AoT\_data\_20180101\_temperature=AoT\_data\_20180101[(AoT\_data\_20180101.parameter=='temperature')] # &(AoT\_data\_20180101.sensor=='at2')  
print("列数据类型：\n{}".format(AoT\_data\_20180101\_temperature.dtypes))  
print("\_"\*50)  
print("列名（数据类型为字符串-str-object）:{}".format(AoT\_data\_20180101\_temperature.columns[AoT\_data\_20180101\_temperature.dtypes.eq('object')]))  
columns\_dtypeEQstr=['value\_raw', 'value\_hrf']   
AoT\_data\_20180101\_temperature[columns\_dtypeEQstr]=AoT\_data\_20180101\_temperature[columns\_dtypeEQstr].apply(pd.to\_numeric,errors='coerce', axis=1)  
  
node\_id\_unqiue=AoT\_data\_20180101\_temperature.node\_id.unique()  
print("node\_id-ID of node which did the measurement:{}\n{}".format(node\_id\_unqiue,"\_"\*50))  
  
AoT\_data\_20180101\_temperature\_grouped=AoT\_data\_20180101\_temperature.groupby(AoT\_data\_20180101\_temperature.node\_id)  
  
value\_raw\_dic={}  
for n\_id in tqdm(node\_id\_unqiue):  
 sub\_df=AoT\_data\_20180101\_temperature\_grouped.get\_group(n\_id)  
 sub\_df.set\_index(pd.to\_datetime(sub\_df["timestamp"]),inplace=True)   
 value\_raw\_dic[n\_id]=sub\_df.groupby(sub\_df.index.hour)[['value\_hrf']].mean()['value\_hrf']  
  
value\_raw\_df=pd.DataFrame.from\_dict(value\_raw\_dic,orient='columns')  
value\_raw\_df

列数据类型：  
timestamp object  
node\_id object  
subsystem object  
sensor object  
parameter object  
value\_raw object  
value\_hrf object  
dtype: object  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
列名（数据类型为字符串-str-object）:Index(['timestamp', 'node\_id', 'subsystem', 'sensor', 'parameter', 'value\_raw',  
 'value\_hrf'],  
 dtype='object')  
  
node\_id-ID of node which did the measurement:['001e0610e532' '001e0610bc07' '001e0610ef27' '001e0610e540'  
 '001e0610ee61' '001e0610fb4c' '001e0610ba18' '001e0610ba3b'  
 '001e0610ba57' '001e0610eef4']  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
  
  
100%|██████████| 10/10 [00:00<00:00, 61.43it/s]

|  | **001e0610e532** | **001e0610bc07** | **001e0610ef27** | **001e0610e540** | **001e0610ee61** | **001e0610fb4c** | **001e0610ba18** | **001e0610ba3b** | **001e0610ba57** | **001e0610eef4** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **timestamp** |  |  |  |  |  |  |  |  |  |  |
| **0** | -9.495390 | -9.917552 | -10.226980 | -8.861876 | -9.366752 | -8.686524 | -9.450483 | -7.710883 | -9.393620 | NaN |
| **1** | -10.813103 | -10.995152 | -11.326974 | -10.034667 | -10.435219 | -9.652314 | -10.546560 | -9.042917 | -10.692798 | NaN |
| **2** | -11.573774 | -11.687457 | -12.102297 | -11.064686 | -11.291581 | -10.516943 | -11.084158 | -9.999010 | -11.409276 | NaN |
| **3** | -12.369585 | -12.489413 | -12.751727 | -11.809482 | -11.942571 | -10.883657 | -11.918073 | -10.383381 | -12.361975 | NaN |
| **4** | -13.095918 | -13.507696 | -13.738810 | -12.568714 | -12.657502 | -11.868581 | -12.701017 | -11.531895 | -12.953586 | NaN |
| **5** | -13.551785 | -14.439482 | -14.571918 | -13.237114 | -13.476429 | -12.675438 | -13.376158 | -12.122713 | -13.156931 | NaN |
| **6** | -14.569456 | -15.302564 | -15.470538 | -14.197971 | -14.364105 | -13.381010 | -14.159520 | -12.947673 | -13.956623 | NaN |
| **7** | -15.245185 | -15.877248 | -16.133722 | -14.912029 | -15.140795 | -14.087324 | -14.920011 | -13.660048 | -14.691400 | NaN |
| **8** | -15.873405 | -16.625105 | -16.862518 | -15.496039 | -15.749362 | -14.682562 | -15.698409 | -14.333693 | -15.350267 | NaN |
| **9** | -16.401733 | -17.160867 | -17.465528 | -16.081622 | -16.401067 | -15.227514 | -16.457512 | -14.739315 | -15.775695 | NaN |
| **10** | -16.785810 | -17.597057 | -17.824492 | -16.495790 | -16.710400 | -15.573029 | -16.724333 | -15.053427 | -16.113267 | NaN |
| **11** | -17.216205 | -17.946057 | -18.192976 | -16.960333 | -16.979152 | -15.872752 | -17.060516 | -15.608010 | -16.528219 | NaN |
| **12** | -17.611728 | -18.381014 | -18.359292 | -17.252210 | -17.358428 | -16.358029 | -17.462519 | -16.109831 | -16.914276 | NaN |
| **13** | -17.806830 | -18.529229 | -18.495518 | -17.293212 | -17.529314 | -16.459971 | -17.504382 | -16.060376 | -17.081043 | -17.358937 |
| **14** | -17.651686 | -18.058190 | -16.940301 | -16.946705 | -16.176010 | -15.323431 | -17.179048 | -14.384124 | -15.849938 | -16.302238 |
| **15** | -17.015144 | -15.972505 | -15.721166 | -16.296347 | -14.262914 | -13.819606 | -16.529705 | -12.838782 | -14.290691 | -15.339267 |
| **16** | -16.036328 | -15.292181 | -13.998253 | -14.868127 | -12.801219 | -12.685005 | -14.946886 | -12.192076 | -13.180194 | -13.971981 |
| **17** | -14.950979 | -13.869676 | -12.780328 | -13.304919 | -12.256590 | -10.676219 | -13.021371 | -10.194095 | -12.244765 | -12.332029 |
| **18** | -13.876210 | -12.862067 | -12.558569 | -12.037333 | -12.352848 | -9.764724 | -11.578667 | -8.927877 | -10.622000 | -11.304867 |
| **19** | -12.404682 | -12.737584 | -12.022304 | -11.406105 | -11.506419 | -9.759933 | -10.299181 | -8.096674 | -10.116876 | -10.750838 |
| **20** | -12.065452 | -12.428440 | -11.959846 | -10.974516 | -10.648790 | -9.756133 | -9.839695 | -8.805656 | -10.319571 | -10.314295 |
| **21** | -12.485923 | -13.010326 | -12.590530 | -11.916676 | -10.988782 | -10.503259 | -11.377819 | -10.707610 | -11.273475 | -10.693224 |
| **22** | -13.456077 | -14.234840 | -14.133451 | -13.413810 | -12.211169 | -11.952600 | -12.797833 | -12.155292 | -12.818308 | -12.757085 |
| **23** | -14.141487 | -15.405905 | -15.061102 | -14.382365 | -13.325823 | -12.918876 | -13.744962 | -13.581819 | -13.886686 | -13.877945 |

AoT\_nodes\_temperature\_gdf=pd.merge(AoT\_nodes\_gdf,value\_raw\_df.T.reset\_index().rename(columns={'index':'node\_id'}),on="node\_id")  
AoT\_nodes\_temperature\_gdf

|  | **node\_id** | **project\_id** | **vsn** | **address** | **lat** | **lon** | **description** | **start\_timestamp** | **end\_timestamp** | **geometry** | **...** | **14** | **15** | **16** | **17** | **18** | **19** | **20** | **21** | **22** | **23** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 001e0610ba3b | AoT\_Chicago | 006 | 18th St & Lake Shore Dr Chicago IL | 41.858136 | -87.616055 | AoT Chicago (S) | 2017/08/08 00:00:00 | NaN | POINT (-87.61606 41.85814) | ... | -14.384124 | -12.838782 | -12.192076 | -10.194095 | -8.927877 | -8.096674 | -8.805656 | -10.707610 | -12.155292 | -13.581819 |
| **1** | 001e0610ba18 | AoT\_Chicago | 01D | Damen Ave & Cermak Chicago IL | 41.852179 | -87.675825 | AoT Chicago (S) | 2017/12/15 00:00:00 | NaN | POINT (-87.67583 41.85218) | ... | -17.179048 | -16.529705 | -14.946886 | -13.021371 | -11.578667 | -10.299181 | -9.839695 | -11.377819 | -12.797833 | -13.744962 |
| **2** | 001e0610eef4 | AoT\_Chicago | 034 | Milwaukee Ave & Wabansia Ave Chicago IL | 41.912681 | -87.681052 | AoT Chicago (S) | 2017/10/09 00:00:00 | NaN | POINT (-87.68105 41.91268) | ... | -16.302238 | -15.339267 | -13.971981 | -12.332029 | -11.304867 | -10.750838 | -10.314295 | -10.693224 | -12.757085 | -13.877945 |
| **3** | 001e0610bc07 | AoT\_Chicago | 03C | Kedzie Ave & 5th Ave Chicago IL | 41.878372 | -87.706042 | AoT Chicago (S) | 2017/11/20 00:00:00 | NaN | POINT (-87.70604 41.87837) | ... | -18.058190 | -15.972505 | -15.292181 | -13.869676 | -12.862067 | -12.737584 | -12.428440 | -13.010326 | -14.234840 | -15.405905 |
| **4** | 001e0610ee61 | AoT\_Chicago | 03F | Pulaski Rd & Madison St Chicago IL | 41.880732 | -87.725660 | AoT Chicago (S) | 2017/11/20 00:00:00 | NaN | POINT (-87.72566 41.88073) | ... | -16.176010 | -14.262914 | -12.801219 | -12.256590 | -12.352848 | -11.506419 | -10.648790 | -10.988782 | -12.211169 | -13.325823 |
| **5** | 001e0610ba57 | AoT\_Chicago | 041 | Western Ave & Madison St Chicago IL | 41.881172 | -87.686359 | AoT Chicago (S) | 2017/11/20 00:00:00 | NaN | POINT (-87.68636 41.88117) | ... | -15.849938 | -14.290691 | -13.180194 | -12.244765 | -10.622000 | -10.116876 | -10.319571 | -11.273475 | -12.818308 | -13.886686 |
| **6** | 001e0610ef27 | AoT\_Chicago | 04C | Western Ave & 25th St Chicago IL | 41.846579 | -87.685557 | AoT Chicago (S) [C] | 2017/12/15 00:00:00 | NaN | POINT (-87.68556 41.84658) | ... | -16.940301 | -15.721166 | -13.998253 | -12.780328 | -12.558569 | -12.022304 | -11.959846 | -12.590530 | -14.133451 | -15.061102 |
| **7** | 001e0610fb4c | AoT\_Chicago | 04D | Leavitt St & Milwaukee Ave Chicago IL | 41.913583 | -87.682414 | AoT Chicago (S) | 2017/10/09 00:00:00 | NaN | POINT (-87.68241 41.91358) | ... | -15.323431 | -13.819606 | -12.685005 | -10.676219 | -9.764724 | -9.759933 | -9.756133 | -10.503259 | -11.952600 | -12.918876 |
| **8** | 001e0610e532 | AoT\_Chicago | 053 | Racine Ave & 18th St Chicago IL | 41.857959 | -87.656427 | AoT Chicago (S) [C] | 2017/12/15 00:00:00 | NaN | POINT (-87.65643 41.85796) | ... | -17.651686 | -17.015144 | -16.036328 | -14.950979 | -13.876210 | -12.404682 | -12.065452 | -12.485923 | -13.456077 | -14.141487 |
| **9** | 001e0610e540 | AoT\_Chicago | 05A | Fort Dearborn Dr & 31st St Chicago IL | 41.838618 | -87.607817 | AoT Chicago (S) [C] | 2017/11/29 00:00:00 | NaN | POINT (-87.60782 41.83862) | ... | -16.946705 | -16.296347 | -14.868127 | -13.304919 | -12.037333 | -11.406105 | -10.974516 | -11.916676 | -13.413810 | -14.382365 |

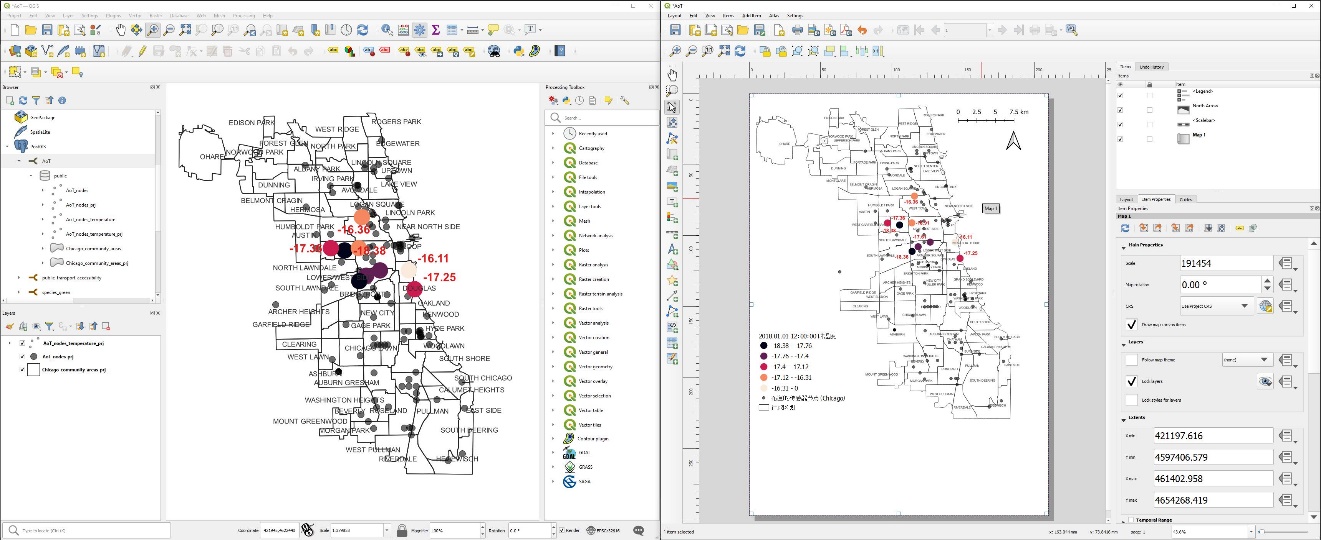
10 rows × 34 columns

gpd2postSQL(AoT\_nodes\_temperature\_gdf,table\_name='AoT\_nodes\_temperature',myusername='postgres',mypassword='123456',mydatabase='AoT')   
gpd2postSQL(AoT\_nodes\_temperature\_gdf.to\_crs(epsg\_Chicago),table\_name='AoT\_nodes\_temperature\_prj',myusername='postgres',mypassword='123456',mydatabase='AoT')

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
has been written to into the PostSQL database...  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
has been written to into the PostSQL database...

#### 4）QGIS读取PostgreSQL数据库

虽然Python中的GeoPandas，及其它图表库可以直接打印地图，但是，不是很方便处理地图的细节表达，尤其用于论文发表，专著或者其它传播用途时。应用QGIS来构建地图，可以直接从PostgreSQL数据库中读取表（在PostGIS下建立数据库连接），和Python基本无缝结合，这可以充分利用Python的数据处理能力，和QGIS的地图表达能力。



#### 5）计算样本总长度（附）

data数据为单独的一个csv格式文件，可以通过下述代码来计算总样本数，即行数。也可以读取指定范围的部分行。但是读取部分行时，仍旧需要耗费一定时间来略过需要忽略的行，通常可以分批处理后，分别写入数据库或存储为单独的文件再读取处理。

from tqdm.auto import tqdm  
  
count=0  
for chunk in tqdm(pd.read\_csv(AoT\_data\_fp,chunksize=chunksize)):  
 count+= 1 # 样本分组数  
 last\_len=len(chunk) # 最后一组的样本数量  
data\_length=(count\*chunksize+last\_len-chunksize) # 数据行（样本）总长度  
print("数据行（样本）总长度={}".format(data\_length))

0it [00:00, ?it/s]  
  
  
数据行（样本）总长度=573074785

rows\_diff=data\_length-chunksize  
AoT\_data\_lastChunck=pd.read\_csv(AoT\_data\_fp,skiprows=range(1,rows\_diff),nrows=chunksize-1)

AoT\_data\_lastChunck

|  | **timestamp** | **node\_id** | **subsystem** | **sensor** | **parameter** | **value\_raw** | **value\_hrf** |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 2018/05/15 08:00:25 | 001e06113a24 | lightsense | hmc5883l | magnetic\_field\_y | 564 | 512.727 |
| **1** | 2018/05/15 08:00:25 | 001e06113a24 | lightsense | hmc5883l | magnetic\_field\_z | 52 | 53.061 |
| **2** | 2018/05/15 08:00:25 | 001e06113a24 | lightsense | ml8511 | intensity | 9312 | 38.133 |
| **3** | 2018/05/15 08:00:25 | 001e06113a24 | lightsense | mlx75305 | intensity | 405 | -4.151 |
| **4** | 2018/05/15 08:00:25 | 001e06113a24 | lightsense | tmp421 | temperature | 12464 | 48.69 |
| **...** | ... | ... | ... | ... | ... | ... | ... |
| **999994** | 2018/05/15 11:12:05 | 001e0610ee41 | metsense | tsl250rd | intensity | 157 | 7.906 |
| **999995** | 2018/05/15 11:12:05 | 001e0610ee41 | metsense | tsys01 | temperature | NaN | 20.76 |
| **999996** | 2018/05/15 11:12:05 | 001e0610f703 | alphasense | opc\_n2 | fw | ffff | NaN |
| **999997** | 2018/05/15 11:12:05 | 001e0610f703 | lightsense | apds\_9006\_020 | intensity | 34 | 2.733 |
| **999998** | 2018/05/15 11:12:05 | 001e0610f703 | lightsense | hih6130 | humidity | 23629 | 44.22 |

999999 rows × 7 columns

### 1.3.4 数据分析基本流程组织

开始一个以数据分析为主的研究课题（项目），通常建立一个单独的文件夹，在该文件夹下放置该项目的所有代码、及相关的数据、图表等内容。一般定义的子文件夹包括： 1. data - 存放原始的数据 2. data\_processed - 存放处理的过程数据 3. database - 放置数据库 4. graph - 存放图表（一般由Python代码直接输出） 5. imgs - 存放一般的图像 6. map - 放置地图文件（例如QGIS） 7. model - 存储训练好的模型（例如机器/深度学习，网络模型等）

文件夹命名可以参考上述，亦可自行灵活调整。因为代码工程量会随着分析内容的深入不断增加，为了防止代码丢失，非常必要将其推送（push）到[GitHub](https://github.com/)代码托管平台，或国内相关的代码托管平台上。当代码更新时，可以推送更新云端仓库（repository）。可以使用GitHub的[桌面版（GitHub Desktop）](https://desktop.github.com/)操作，具体方法可以查看官网。

通过子文件夹的结构，可以明了数据分析基本流程组织。从data下读取原始数据（如果数据文件较大，也会存储于外置硬盘中）；处理的过程数据则放置于data\_processed中；为了方便数据的管理和读写，优先选择将数据写入数据库，SQLite为单独的文件，可以直接放置于database中，而postgreSQL是直接写入默认的安装路径下，在单独一个项目完结后，可以备份导出数据库；分析过程图表（用于说明分析结果，或过程数据分析描述，往往为书写科研论文的重要部分或用以报告）放置于graph下；其它非Python直接输出的图表或图像，存放于imgs下；QGIS读取数据库构建的地图放在map下；训练的模型则存储于model子文件中；.py的代码则直接位于根目录下，与子文件夹并列，这是为了方便直接读写数据。例如db\_fp=r'./database/fruits.sqlite'和AoT\_nodes\_fp='./data/AoT\_Chicago.complete.2021-12-20/nodes.csv'等，使用相对路径比较简单明了。如果有特殊需要，再建立存储代码的子文件夹。

常用自定义或者迁移的代码工具，为了方便调用，通常存储于单独的.py文件下，例如本书的util\_database.py（用于数据库操作的代码函数），util\_misc.py（包括显示文件的结构等杂项代码类或函数）等。

下述迁移的代码可以查看文件夹的结构，打印了截止当前，本书代码工程项目的文件夹内容结构。

class DisplayablePath(object):  
 '''  
 class - 返回指定路径下所有文件夹及其下文件的结构。代码未改动，迁移于'https://stackoverflow.com/questions/9727673/list-directory-tree-structure-in-python'  
 '''  
   
 display\_filename\_prefix\_middle = '├──'  
 display\_filename\_prefix\_last = '└──'  
 display\_parent\_prefix\_middle = ' '  
 display\_parent\_prefix\_last = '│ '  
  
 def \_\_init\_\_(self, path, parent\_path, is\_last):  
 from pathlib import Path  
   
 self.path = Path(str(path))  
 self.parent = parent\_path  
 self.is\_last = is\_last  
 if self.parent:  
 self.depth = self.parent.depth + 1  
 else:  
 self.depth = 0  
  
 @property  
 def displayname(self):  
 if self.path.is\_dir():  
 return self.path.name + '/'  
 return self.path.name  
  
 @classmethod  
 def make\_tree(cls, root, parent=None, is\_last=False, criteria=None):  
 from pathlib import Path  
   
 root = Path(str(root))  
 criteria = criteria or cls.\_default\_criteria  
  
 displayable\_root = cls(root, parent, is\_last)  
 yield displayable\_root  
  
 children = sorted(list(path  
 for path in root.iterdir()  
 if criteria(path)),  
 key=lambda s: str(s).lower())  
 count = 1  
 for path in children:  
 is\_last = count == len(children)  
 if path.is\_dir():  
 yield from cls.make\_tree(path,  
 parent=displayable\_root,  
 is\_last=is\_last,  
 criteria=criteria)  
 else:  
 yield cls(path, displayable\_root, is\_last)  
 count += 1  
  
 @classmethod  
 def \_default\_criteria(cls, path):  
 return True  
  
 @property  
 def displayname(self):  
 if self.path.is\_dir():  
 return self.path.name + '/'  
 return self.path.name  
  
 def displayable(self):  
 if self.parent is None:  
 return self.displayname  
  
 \_filename\_prefix = (self.display\_filename\_prefix\_last  
 if self.is\_last  
 else self.display\_filename\_prefix\_middle)  
  
 parts = ['{!s} {!s}'.format(\_filename\_prefix,  
 self.displayname)]  
  
 parent = self.parent  
 while parent and parent.parent is not None:  
 parts.append(self.display\_parent\_prefix\_middle  
 if parent.is\_last  
 else self.display\_parent\_prefix\_last)  
 parent = parent.parent  
  
 return ''.join(reversed(parts))  
   
app\_root=r'C:\Users\richi\omen\_richiebao\omen\_github\USDA\_CH\_final\USDA\notebook'  
  
from pathlib import Path  
paths = DisplayablePath.make\_tree(Path(app\_root))  
for path in paths:  
 print(path.displayable())

notebook/  
├── .ipynb\_checkpoints/  
│ ├── 1.3\_数据库与数据分析基本流程组织-checkpoint.ipynb  
│ └── 2.1.1 数据POI与描述性统计和正态分布-checkpoint.ipynb  
├── 1.3\_数据库与数据分析基本流程组织.ipynb  
├── 2.1.1 数据POI与描述性统计和正态分布.ipynb  
├── \_\_pycache\_\_/  
│ ├── coordinate\_transformation.cpython-38.pyc  
│ └── util\_database.cpython-38.pyc  
├── commodity\_table\_structure.py  
├── coordinate\_transformation.py  
├── data/  
│ ├── AoT\_Chicago.complete.2021-12-20/  
│ │ ├── nodes.csv  
│ │ ├── offsets.csv  
│ │ ├── provenance.csv  
│ │ ├── README.md  
│ │ └── sensors.csv  
│ ├── Chicago Community Areas/  
│ │ ├── Chicago Community Areas.dbf  
│ │ ├── Chicago Community Areas.prj  
│ │ ├── Chicago Community Areas.shp  
│ │ └── Chicago Community Areas.shx  
│ ├── poi\_csv.csv  
│ └── poi\_json.json  
├── data\_processed/  
├── database/  
│ ├── AoT.1.3.sql  
│ ├── fruits.sqlite  
│ └── fruits\_relational.sqlite  
├── graph/  
├── imgs/  
│ ├── 1\_3\_01.png  
│ ├── 1\_3\_02.jpg  
│ ├── 1\_3\_03.jpg  
│ ├── 1\_3\_04.jpg  
│ ├── 1\_3\_04.psd  
│ ├── 1\_3\_05.jpg  
│ └── 1\_3\_06.jpg  
├── map/  
│ └── AoT.qgz  
├── model/  
├── sale\_details\_table\_structure.py  
├── sales\_table\_structure.py  
├── util\_database.py  
└── util\_misc.py

注释（Notes）：

① SQLite，不是一个独立的应用程序，而是用C编程语言编写的数据库引擎，可以将其嵌入到APP中，因此属于嵌入式数据库系列（<https://www.sqlite.org/index.html>）。

② PostgreSQL，也称为Postgres，一个自由开源的关系型数据库管理系统，强调可扩展性和SQL兼容性（<https://www.postgresql.org/>）。

③ QGIS，为一个免费开源的跨平台桌面地理信息系统（geographic information system，GIS）应用程序，支持查看、编辑、打印和分析地理空间数据（<https://www.qgis.org/en/site/>）。

④ sqlite3，是一个C语言库，提供了一个基于磁盘的轻量级数据库，不需要单独的服务器进程，并允许使用SQL查询语言的非标准变体访问数据库。一些应用程序可以使用SQLite进行内部数据存储或建立一个应用程序的原型，然后将代码移植到一个更大的数据库，例如PostgreSQL或Oracle等 （<https://docs.python.org/3/library/sqlite3.html>）。

⑤ SQLAlchemy，是Python SQL工具包和对象关系映射器，为应用程序开发人员提供了SQL的全部功能和灵活性（<https://www.sqlalchemy.org/>）。

⑥ DB Browser for SQLite (DB4S) ，是一个高质量、可视化、开源的工具，用于创建、设计和编辑与SQLite兼容的数据库文件（<https://sqlitebrowser.org/>）。

⑦ Relationship Configuration，SQLAlchemy中relationship()函数的用法（<https://docs.sqlalchemy.org/en/14/orm/relationships.html>）。

⑧ Visual Paradigm，为统一建模语言（Unified Modeling Language，UML），是设计、分析和管理工具套件，可推动IT项目开发。除了支持建模，还可以生成报告和生成代码，并可以依据代码生成逆向工程图表（<https://www.visual-paradigm.com/>）。

⑨ PostgreSQL，也称为Postgres，自由开源的关系型数据库管理系统（relational database management system，RDBMS）（<https://www.postgresql.org/>）。

⑩ QGIS，在Windows、Mac、Linux、BSD和移动设备上创建、编辑、可视化、分析和发布地理空间信息（<https://qgis.org/en/site/>）。

⑪ pgAdmin，为最流行和功能丰富的PostgreSQL开源管理和开发平台（<https://www.pgadmin.org/>）。

⑫ Array of Things(AoT) 城市环境传感器，以收集城市环境、基础设施和活动的实时数据供研究和公共使用（<https://arrayofthings.github.io/>）。

⑬ Chicago Data Portal，为芝加哥城开放数据门户，可免费下载数据用于相关分析，其中许多数据集每天至少更新一次或数次（<https://data.cityofchicago.org/>）。

参考文献（References）:

[1] Miguel Grinberg.Flask Web Development: Developing Web Applications with Python[M]. O’Reilly Media; 2nd edition.April 3, 2018. （中文版：Miguel Grinberg.安道译.Flask Web开发：基于Python的Web应用开发实战[M].人民邮电出版社,2018.）

[2] 高桥麻奈著,崔建锋译.株式会社TREND-PRO漫画制作.漫画数据库[M].科学出版社.北京,2010.5.