Microsoft® Most Valuable Professional

Agus Kurniawan

Python and SQLite Development

Python and SQLite Development

Python and SQLite Development

Agus Kurniawan

1st Edition, 2021

ISBN: 978-1-716-24602-9

Table of Contents

Preface

This book provides alternative approach to build Python application with SQLite database. This book describes how to work with Python and SQLite and illustrates their use with code examples.

Agus Kurniawan

Depok, January 2021

1. Setting up Development Environment

1.1 Python

Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. In this book, we focus on learning how to develop Python programs to access SQLite database.

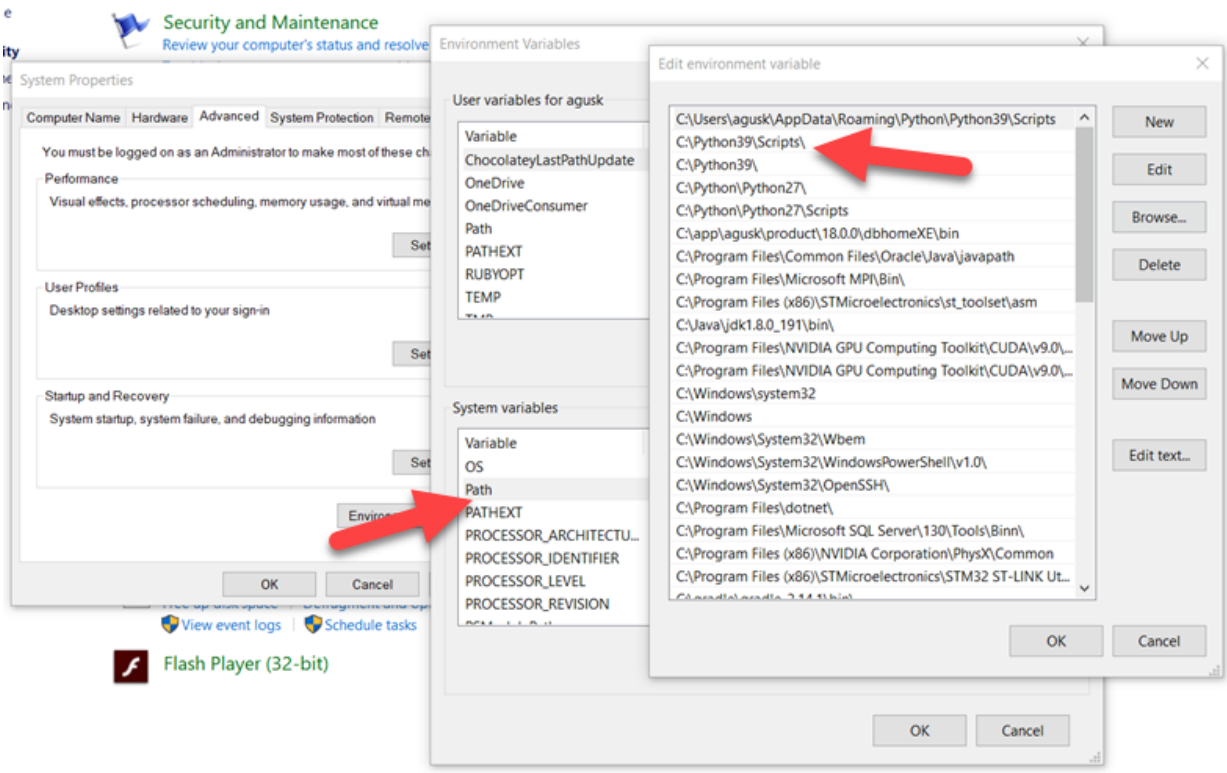
To install Python, you can download it for your platform This book covers for Windows, Mac and Linux.

After installed Python, you can verify it by opening Terminal or Command Prompt for Windows. Type this command.

```
$ python --version
$ pip --version
```

You should see python and pip versions on Terminal.

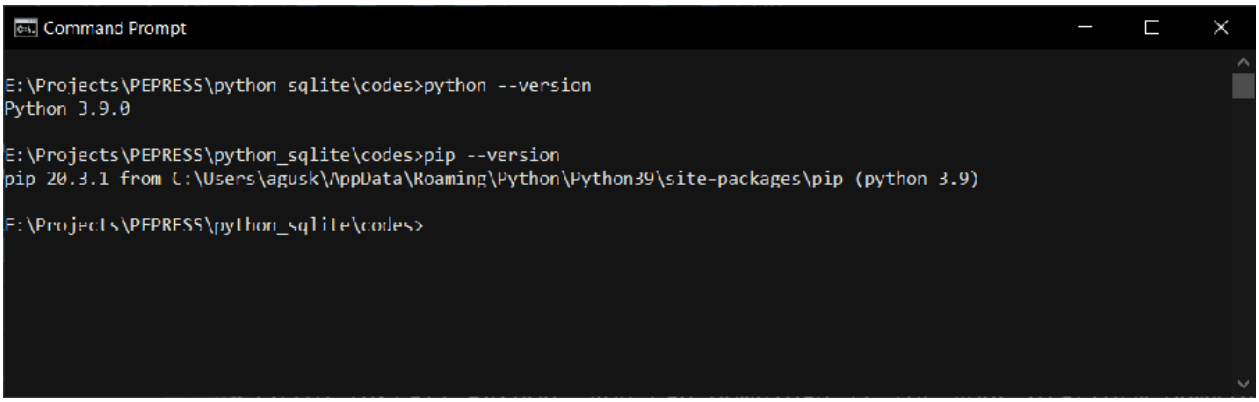
For Windows users, you probably get errors while you are running python command. You can configure Python path into Path environment from Control Panel. You can see my Python path configuration in Figure below.



You probably have Python 3.x so your pip may be pip3. You can type this command

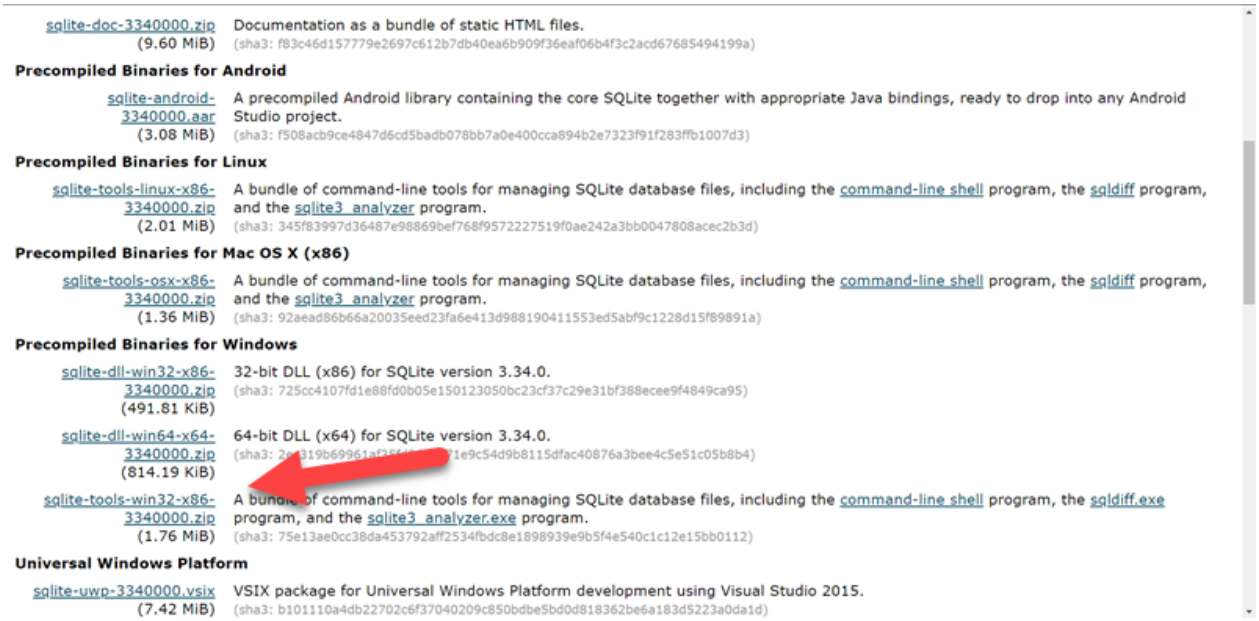
```
$ python3 --version

$ pip3 --version
```

You should get a Python version in Terminal. You can see my Python version in Windows.

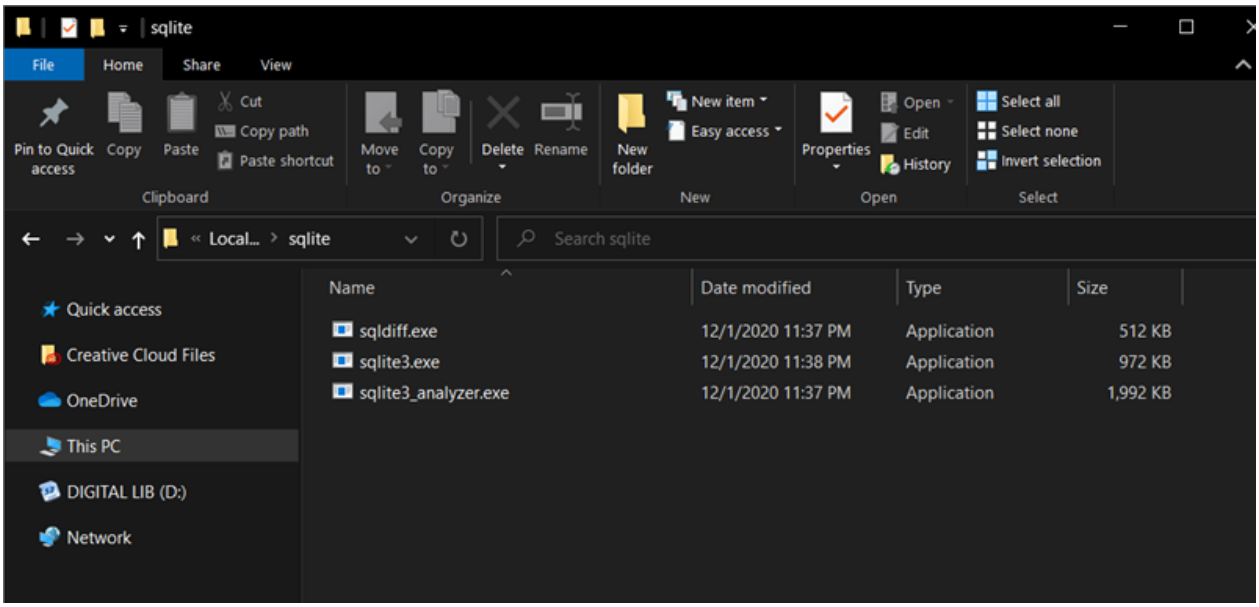

Command Prompt
E:\Projects\PEPRESS\python sqlite\codes>python --version
Python 3.9.0
E:\Projects\PEPRESS\python_sqlite\codes>pip --version
pip 20.3.1 from C:\Users\agusk\AppData\Roaming\Python\Python39\site-packages\pip (python 3.9)
E:\Projects\PEPRESS\python_sqlite\codes>

1.2 SQLite Database

I use Sqlite 3 for demo. SQLite provides a shell so you can perform data manipulation. You can download Sqlite 3 shell on this site on Download SQLite shell based your platform. For Windows, you can download sqlite-tools as shown in Figure below.



Download and extract ZIP file. Then, you can see 3 files as shown in Figure below.



For Linux, for instance Debian/Ubuntu, we can download and install using these commands on Terminal.

```
$ sudo apt-get update
$ sudo apt-get install sqlite3 libsqlite3-dev
```

If you are working with macOS, you can install SQLite3 via brew. You can type this command on Terminal.

```
$ brew install sqlite
```

Now your platform is ready for SQLite database included SQLite shell.

1.3 SQLite Driver for Python

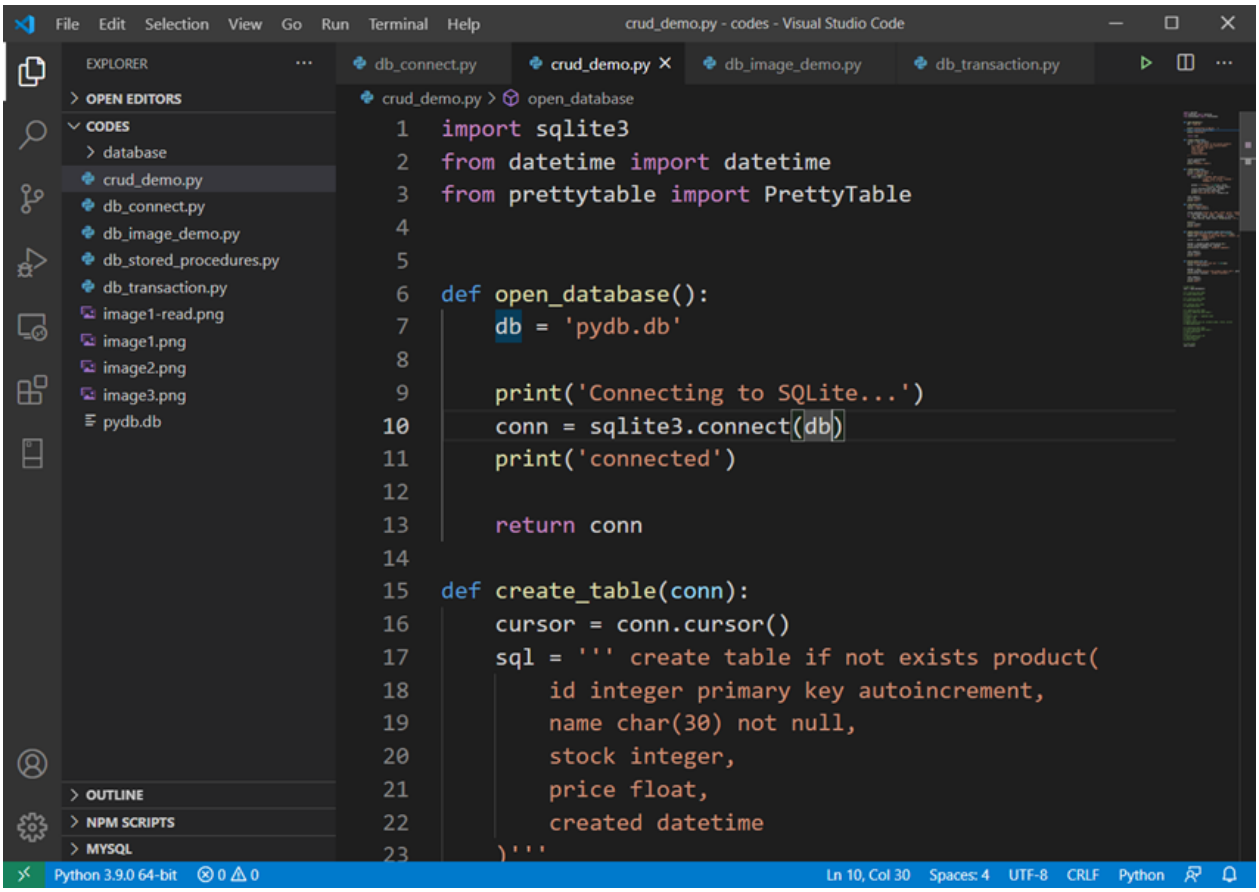
To access SQLite from Python, we need SQLite driver for Python. Fortunately, SQLite driver for Python is already inclued in Python standard library so we don't need to install this.

Next, we start to develop Python program for SQLite.

1.4 Development Tools

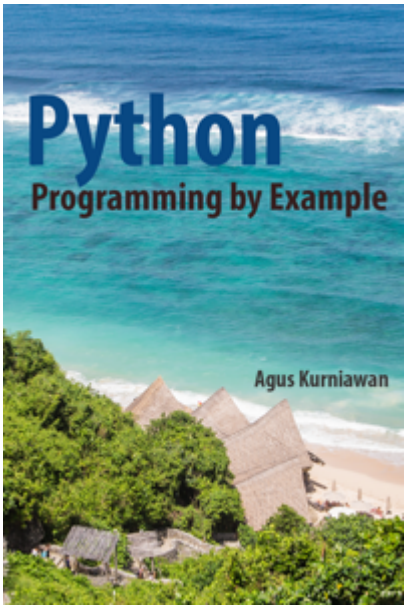
Technically, you can use any editor to develop Python. For demo, I use Visual Studio Code fo writing Python scripts. This tool is free, You can download it for Windows, macOS and Linux. Download it

The following is a sample of Visual Studio Code.



1.5 Python Programming

This book does not cover about basic Python programming. You can learn Python programming from online stuffs. In addition, I also have written a book about Python Programming by Example.



You can get this book on the following online store:

Amazon
Google Play

2. Getting Started - Python and SQLite

This chapter explains how to connect SQLite database via SQLite driver for Python.

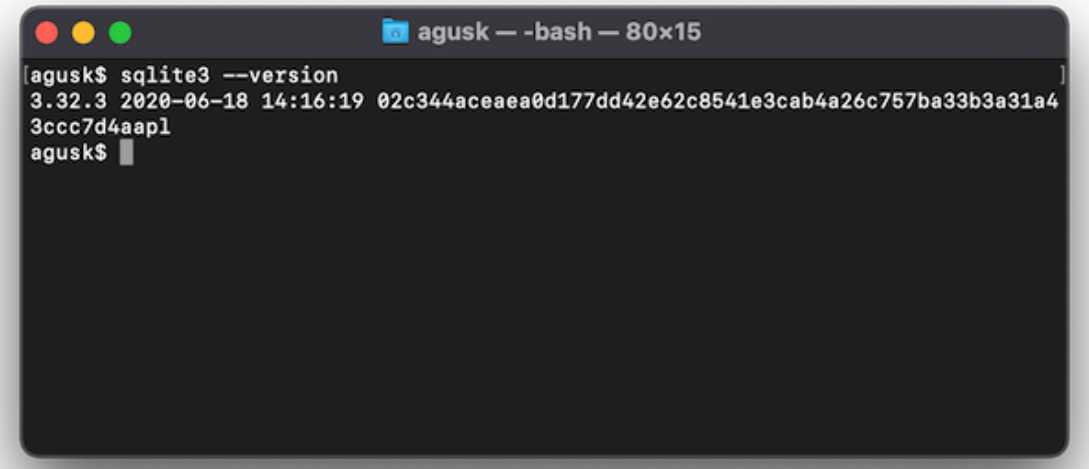
2.1 SQLite Shell

SQLite has a shell that runs on Terminal. We can create database and tables using SQLite shell. You can download SQLite shell for your platform.

After you set up SQLite shell, you can run this shell using sqlite3 command. You can type this.

```
$ sqlite3 --version
```

You should see SQLIte shell version on Terminal. You can see my SQLite shell version in Figure below (macOS version).



If you are working on Windows, you can open Commmand Prompt. Then, navigate to a folder where SQLite shell file was extracted. Then, type sqlite3 --version so you can see SQLite version on Command Prompt.

```
$ sqlite3 --version
```

Next, we are playing with SQLite shell. We can use a memory databse on SQLite. This means we can create some tables and then insert data but our data will loss after we quit from SQLite shell.

You can type sqlite3 on Terminal and then you can see SQLite memory database.

```
$ sqlite3
```

Now we can create a table, called tb1. You can type this command on SQLite shell.

```
> create table tb1(id int, name char(10));
```

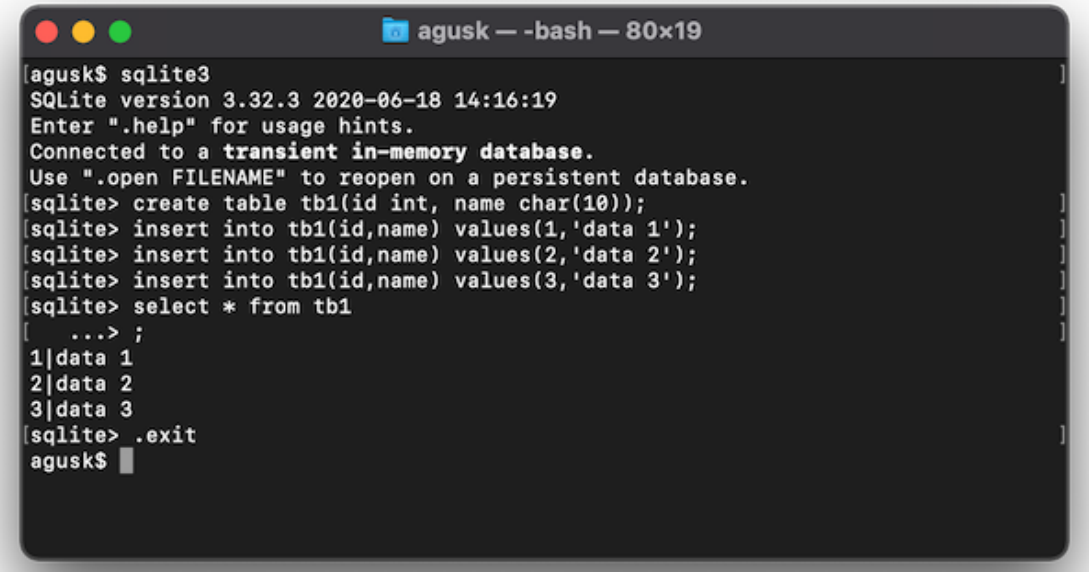
After that, we can insert data into tb1 table. Type these commands on SQLite shell.

```
> insert into tb1(id,name) values(1,'data 1');
> insert into tb1(id,name) values(2,'data 2');
> insert into tb1(id,name) values(3,'data 3');
```

After we inserted data, we show data using select statement. You can type this shell command.

```
> select * from tb1;
```

You can see my program output from SQLite shell in Figure below.



To quit from SQLite shell, we can type .exit command.

```
> .exit
```

Next, we can connect to SQLite database and perform data manipulation.

2.2 Connecting to SQLite

We can connect to SQLite database using SQLite driver for Python with passing our database configuration such as SQLite database file. As we know, SQLite doesn't has database engine like SQL Server, Oracle and MySQL so we can SQLite database through database file directly.

Next, we create Python program to access SQLite database.

2.3 Creating Python Program

In this section, we create Python file, called The objective is to connect SQLite database via SQLite driver for Python.

Firstly, we load SQLite driver for Python library and then define database file, called pydb.db. We use **pydb** database for demo.

```
import sqlite3

db = 'pydb.db'

conn = sqlite3.connect(db)
print('connected')
print(conn)

conn.close()
```

Then we connect to SQLite using **sqlite3.connect()** by passing database file. We should pass database file with full path. If the database file is not exist, SQLite driver will create a database file.

At the end of code, we close connection using **close()** function.

Save all code. Next, we run this program.

2.3 Running

We can run our Python program in Visual Studio Code. Open Terminal on Visual Studio Code (menu View -> Terminal). You also can run this program on Terminal directly.

```
$ python db_connect.py
```

Here is a sample of running Python application.

3. CRUD Operations

This chapter explains how to manipulate data SQLite using Python.

3.1 CRUD Operations

CRUD (Create, Read, Update, Delete) operations are basic data manipulation for database. In this chapter we will develop Python application to execute CRUD operations. Firstly, we create database, called and then create a table, called On previous chapter, we already created pydb database. The following is **product** table schema:

Data Type: integer; Primary Key (PK) and Not null (NN)
Data Type: character(30); Not null (NN)
Data Type: integer
Data Type: float
Data Type: datetime

You also can generate Product table using this query in Python. For CRUD demo, we start by creating a Python file, crud_demo.py. We create **open_database()** function to connect SQLite database file. The following is codes implementation for open_database() function. You can change database file name. I use pydb.db file. After we connected to SQLite database file, we return conn object.

```
import sqlite3
from datetime import datetime
from prettytable import PrettyTable


def open_database():
    db = 'pydb.db'

    print('Connecting to SQLite...')
    conn = sqlite3.connect(db)
```

```
    print('connected')

    return conn
```

open_database() function will return conn object that will be used for CRUD operations.

We create create_table() function to create Product table. create_table() function needs SQLite connection. We define creating table query in sql variable. Firstly, we obtain cursor by calling conn.cursor(). Then, we execute a query using cursor.execute(). After executed, you should call conn.commit() to commit all data changes.

```
def create_table(conn):
    cursor = conn.cursor()
    sql = ''' create table if not exists product(
        id integer primary key autoincrement,
        name char(30) not null,
        stock integer,
        price float,
        created datetime
    )'''

    cursor.execute(sql)
    conn.commit()
    print('created a table')
```

We use PrettyTable library to display a table in Terminal. Further information about PrettyTable library, you can read it on You can install this library using pip.

```
$ pip install PrettyTable
```

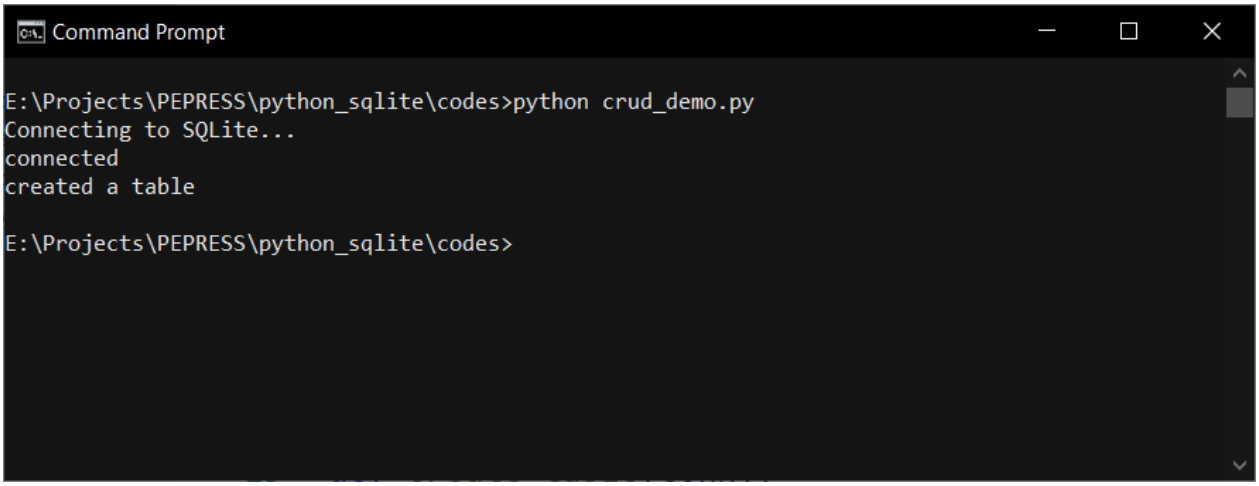
Now we can call open_database() function to connect SQLite database file. Then, we create a table by calling create_table().

```
# open data
conn = open_database()

# creating table demo
create_table(conn)

# close data
conn.close()
```

Save the code and try to run crud_demo.py file.

```
$ python crud_demo.py
```

The following is sample running test on Terminal.



The next step is to create functions for CRUD implementation. We will add some functions to execute CRUD operations on next section.

3.2 Creating Data

In this scenario, we create a new product and try to retrieve inserted ID. We can retrieve inserted id using cursor.lastrowid. We will use SQL statement to insert data but we use parameters to prevent SQL injection. At the end of codes, don't forget to release our usage resources and close connection using **close()**

Now we add function **create_data()** to create new product. Write these codes.

```
def create_data(conn):
    cursor = conn.cursor()
    print('inserting data...')
    for i in range(1,6):
        insert_sql = ("INSERT INTO product "
                      "(name, stock, price, created) "
                      "VALUES(?, ?, ?, ?)")

        params = ("product " + str(i), 3+i*4,
                            0.4+i*8, datetime.now())
        cursor.execute(insert_sql, params)
        product_id = cursor.lastrowid
        print('inserted with id=', product_id)

    conn.commit()
    cursor.close()
    print('done')
```

Save this code.

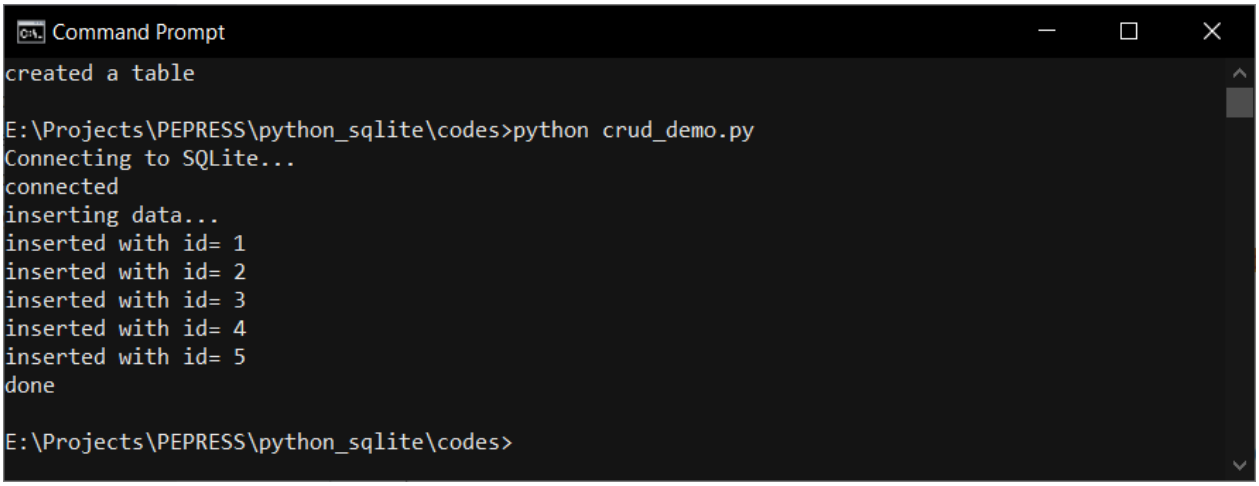
To test this class, we call **create_data()** function. After that, we try to insert 5 data.

```
# open data
conn = open_database()

# creating data demo
create_data(conn)

# close data
conn.close()
```

Save the code and try to run crud_demo.py file.

```
$ python crud_demo.py
```

The following is sample running test on Terminal.



3.3 Reading Data

After inserted data, we read data. The following is a list of reading data steps:

Open connection by calling **open_database()** function
Get cursor object to perform query
Create query with SELECT statement
Execute query using **execute()**
Get all data by looping on cursor object
Print data
Close connection using **close()**

Now we implement our demo. Create a function called write this code:

```
def read_data(conn):
    print('reading data')
    cursor = conn.cursor()

    cursor.execute("select id, name, stock, price, created
from product")
    t = PrettyTable(['ID','Name', 'Stock', 'Price',
'Created'])
    for (id, name, stock, price, created) in cursor:
        t.add_row([id, name, stock, format(price,'.2f'),
created])

    print(t)
    cursor.close()
    print('done')
```

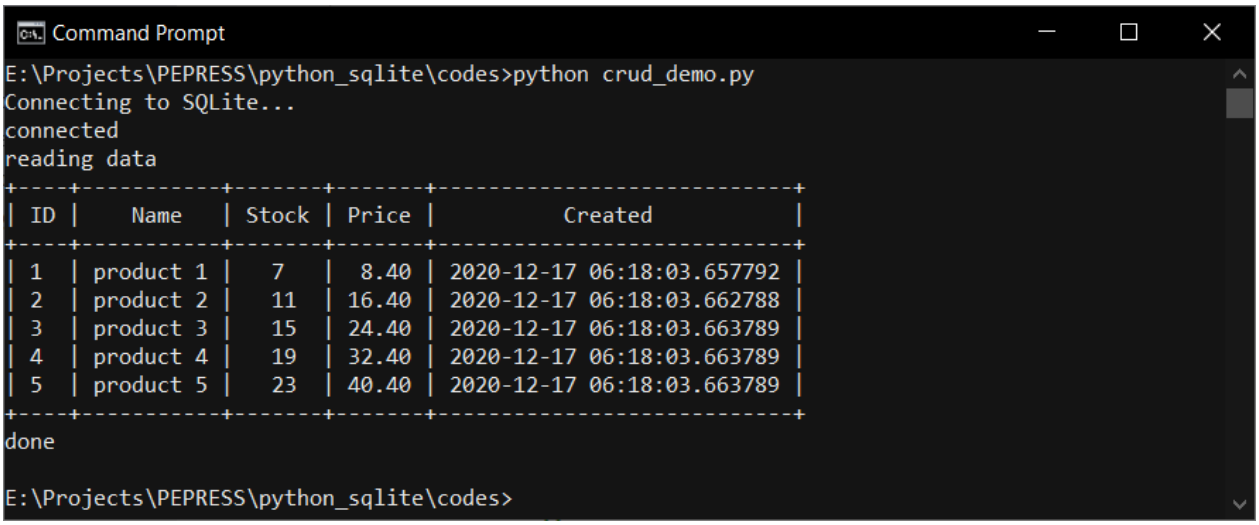
Save this code.

Now we test function Write these codes

```
# open data
conn = open_database()

# reading data demo
read_data(conn)

# close data
conn.close()
```

Save this code.

You can execute **crud_demo.py** file. The following is a sample of program output.



3.4 Updating Data

On this section, we will update data by particular ID. We use UPDATE statement and pass an updated data.

Now we implement our scenario. Create a function called **update_data()** and write this function.

```
def update_data(conn,id,product_name,stock,price):
    print('updating data for product id=' + str(id))
    update_sql = ("UPDATE product SET name=?, stock=?,
price=? "
                  "WHERE id=?")
    cursor = conn.cursor()

    params = (product_name,stock,price,id,)
    cursor.execute(update_sql, params)
    print(cursor.rowcount, ' products updated')

    conn.commit()
    cursor.close()
    print('done')
```

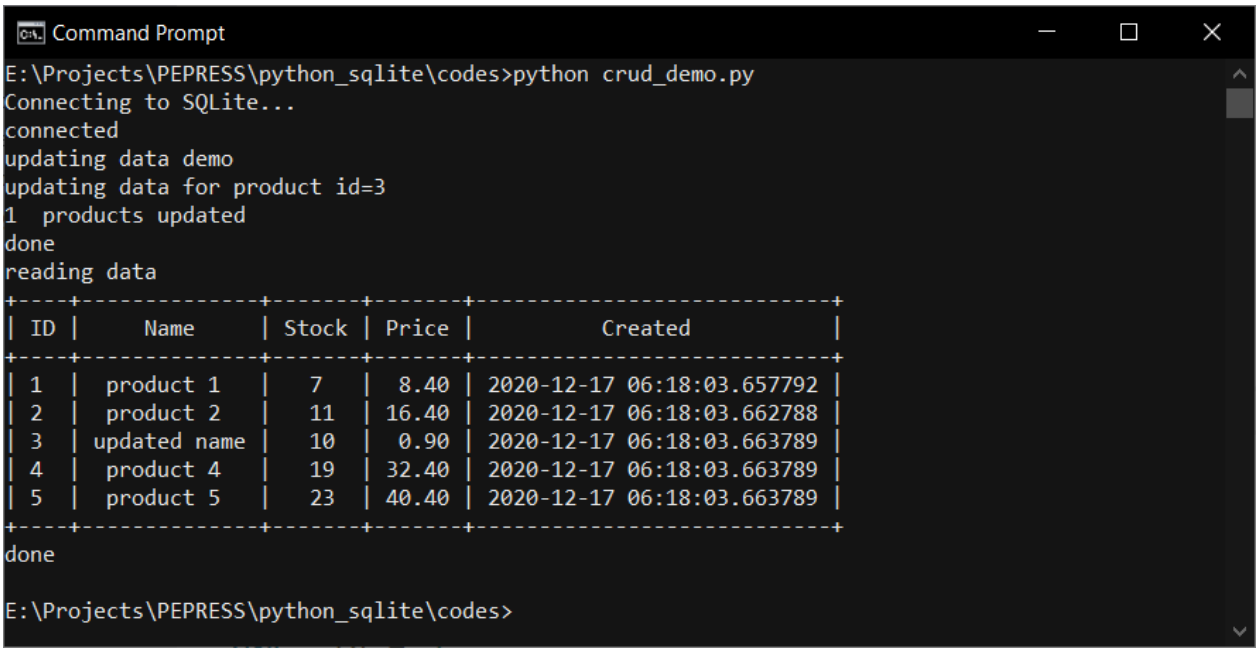
Save this code.

Now we work on file call In this testing, we update product with ID 3. You may change product ID based on your database. After updated data, we show all data on Terminal.

```
# open data
conn = open_database()

# updating data demo
```

```python
print('updating data demo')
id = 3
product_name = 'updated name'
stock = 10
price = 0.9
update_data(conn,id, product_name, stock, price)
read_data(conn)

# close data
conn.close()
```

Save this code. Run file



3.5 Deleting Data

To delete data, we use SQL statement using DELETE and pass a product ID.

We start by creating a function called **delete_data()** and write these codes.

```
def delete_data(conn,id):
    print('deleting data with id=' + str(id))
    cursor = conn.cursor()

    params = (id,)
    cursor.execute("delete from product where id=?", params)
    print(cursor.rowcount, ' product deleted')

    conn.commit()
    cursor.close()
    print('done')
```

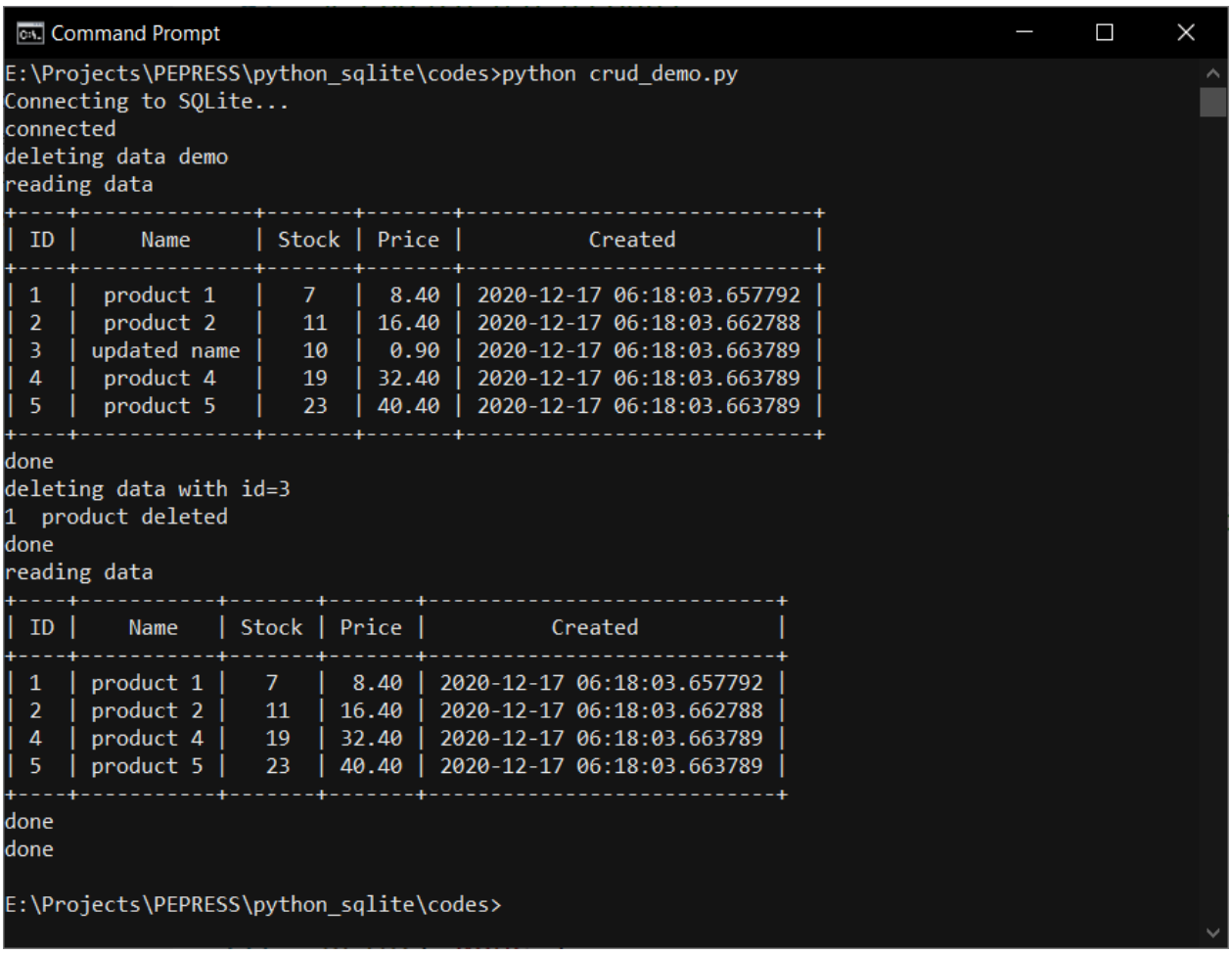
Save this code.

Now we test this function on file Firstly, we open and read existing data. Then, we delete data with ID = 3. You can change this ID based on your database. After that, we show a list of changed data.

```
# open data
conn = open_database()

# deleting data demo
print('deleting data demo')
read_data(conn)
id = 3
```

```
delete_data(conn,id)
read_data(conn)
print('done')

# close data
conn.close()
```

Save this code and



4. Working with Image and Blob Data

This chapter explains how to work with image and blob data on SQLite database and manipulate them using Python.

4.1 Image and Blob Data

SQLite database provides data type for Text and blob for blob data. Image data type can store image file and binary data type can be used to store any file.

In this chapter, we will work with image file on SQLite database. The following is table schema, called

The following Table schema:

Data Type: integer; Primary Key and Auto Increment; Not null
Data Type: char(30); Not null
Data Type: char(15); Not null
Data Type: Text
Data Type: datetime

imagetype is image type of uploading file. This is important while we want to display the image in web, for instance. Firstly, we create Python file, called We set database file configuration and create **open_database()** function to open SQLite database. In this demo, we create pydb.db SQLite database file. You probably change this database file.

```
import sqlite3
from datetime import datetime
from prettytable import PrettyTable
```

```
def open_database():
    db = 'pydb.db'

    print('Connecting to SQLite...')
    conn = sqlite3.connect(db)
    print('connected')

    return conn
```

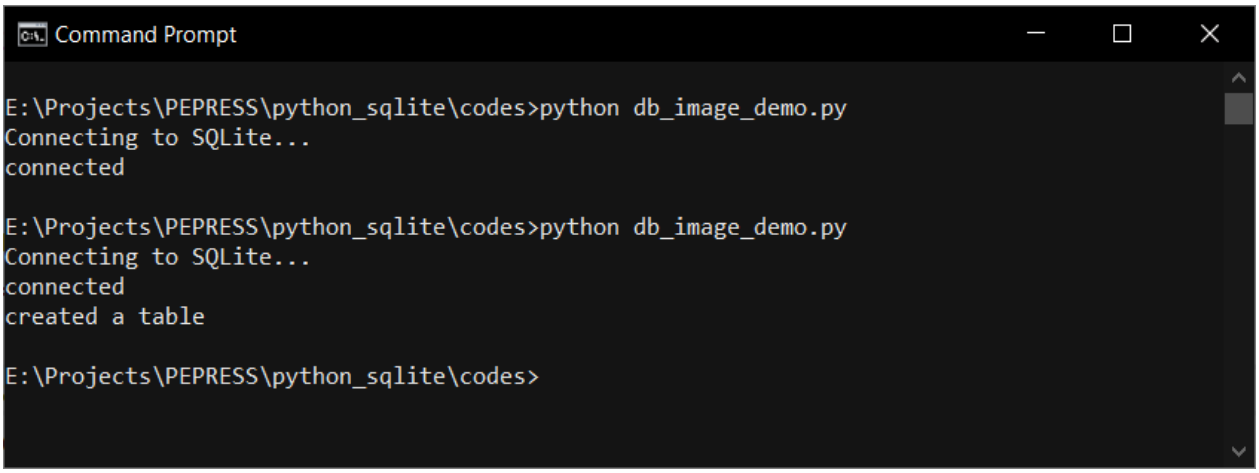
Next, we create **create_table()** function to create ImageFiles table using SQLite query. You can type these scripts.

```
def create_table(conn):
    cursor = conn.cursor()
    sql = ''' create table if not exists imagefiles(
        id integer primary key autoincrement,
        filename char(30) not null,
        imagetype char(30) not null,
        imgfile blob,
        created datetime
    )'''

    cursor.execute(sql)
    conn.commit()
    print('created a table')
```

Now we can call **open_database()** and then call **create_table()** function to create database and table. Type these scripts.

```
# open database
conn = open_database()

# creating data demo
create_table(conn)

# close database
conn.close()
```

You can see my program ouput in Figure below.



The next action we create two functions for uploading image and listing/saving image data.

4.2 Inserting Image File

In this section, we insert an image file into SQLite database. Firstly, we create **insert_image_data()** function. For binary data, we open and read image file using **open()** and **read()** functions. Then, we pass all parameters to SQLite query. Write these codes for implementation.

```
def
insert_image_data(conn,full_file_path,file_name,file_type):
    print('inserting image data')
    cursor = conn.cursor()

    with open(full_file_path, 'rb') as f:
        imagedata = f.read()

    params = (file_name,file_type,imagedata,datetime.now())
    query = ("insert into
imagefiles(filename,imagetype,imgfile,created) "
             "values(?,?,?,?)")

    cursor.execute(query, params)
    img_id = cursor.lastrowid
    print('inserted with id=',img_id)

    conn.commit()
    cursor.close()
```

Save this program.

Now we use the following image file, image1.png, to be inserted into SQLite Database.

We test our function **insert_image_data()** function. We use **image1.png** on the same directory with **db_image_demo.py** program.

```
# open database
conn = open_database()

# inserting image data demo
print('inserting image data demo')
full_file_path = './image1.png'
file_name = 'image1.png'
file_type = 'image/png'
insert_image_data(conn,full_file_path,file_name,file_type)
print('done')
```

Save this program. You can run this file.

```
$ python db_image_demo.py
```

You can see the program output as shown in Figure below.

You can try to upload another image file using this program.

4.3 Reading and Saving Image File

Now we build a function, to show an image data. We save image data into a file using **open()** and **write()** functions. We also print database data with PrettyTable library.

Write these codes for **read_image_data()** implementation.

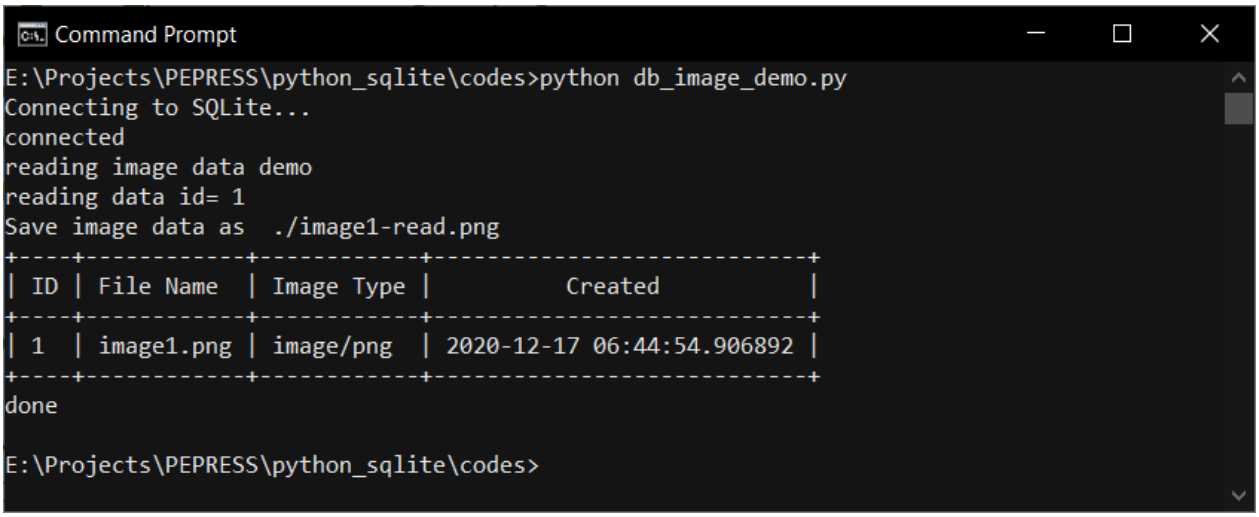
```
def read_image_data(conn, id,save_as_file):
    print('reading data id=',id)
    cursor = conn.cursor()
    try:

        params = (id,)
        query = ("select filename,imagetype,imgfile,created
"
                 "from imagefiles where id=?")

        cursor.execute(query,params)
        t = PrettyTable(['ID','File Name', 'Image
Type','Created'])
        for (filename, imagetype, imgfile, created) in

cursor:
            t.add_row([id, filename, imagetype, created])

            with open(save_as_file, 'wb') as f:
                f.write(imgfile)
            f.close()
            print('Save image data as ',save_as_file)

        print(t)
    except Exception as e:
        print(e)

    finally:
        cursor.close()
        pass
```

Save the code.

We call **read_image_data()** function with passing file name **image1-read.png** and image ID so our image data will be saved as image1-read.png. For instance, I have my image with ID 1. I read this image data and save to local folder.

```
# open database
conn = open_database()

# reading image data demo
print('reading image data demo')
save_as_file = './image1-read.png'
id = 1
read_image_data(conn,id,save_as_file)
print('done')
```

Now open image1-read.png file. You should see the image file. If you cannot open the file, you probably have problems on saving and reading image file. The following is a sample of program output.



5. Transaction

This chapter explains how to work with transaction on SQLite using Python.

5.1 Python and SQLite Transaction

SQLite driver provides transaction operations. To implement a transaction in SQLite, we can perform teh following steps:

open connection to SQLite
set connection with **conn.isolation_level = None**
if you use a cursor, you can call **cursor.execute("BEGIN")**
commit() is used to commit all query operations
rollback() and **cursor.execute("ROLLBACK")** are used to cancel all query operations

Next, we build a Python program to show how to work with transaction in SQLite.

5.2 Demo

To illustrate transaction demo, we create a simple Python application. We insert 5 data into database. If one of them is failed we will call rollback transaction. If succeed, we call **commit()** to store all data. Otherwise, we call **rollback()** to perform rollback operation.

For demo, we use previous database, pydb.db SQLite file. Let's build. Create Python file, called and write the following code:

```
import sqlite3
from datetime import datetime
from prettytable import PrettyTable


def open_database():
    db = 'pydb.db'

    print('Connecting to SQLite...')
    conn = sqlite3.connect(db)
    print('connected')

    return conn

def read_data(conn):
    print('reading data')
    cursor = conn.cursor()

    cursor.execute("select id, name, stock, price, created
from product")
    t = PrettyTable(['ID','Name', 'Stock', 'Price',
'Created'])
    for (id, name, stock, price, created) in cursor:
        t.add_row([id, name, stock, format(price,'.2f'),
created])

    print(t)
    cursor.close()
```

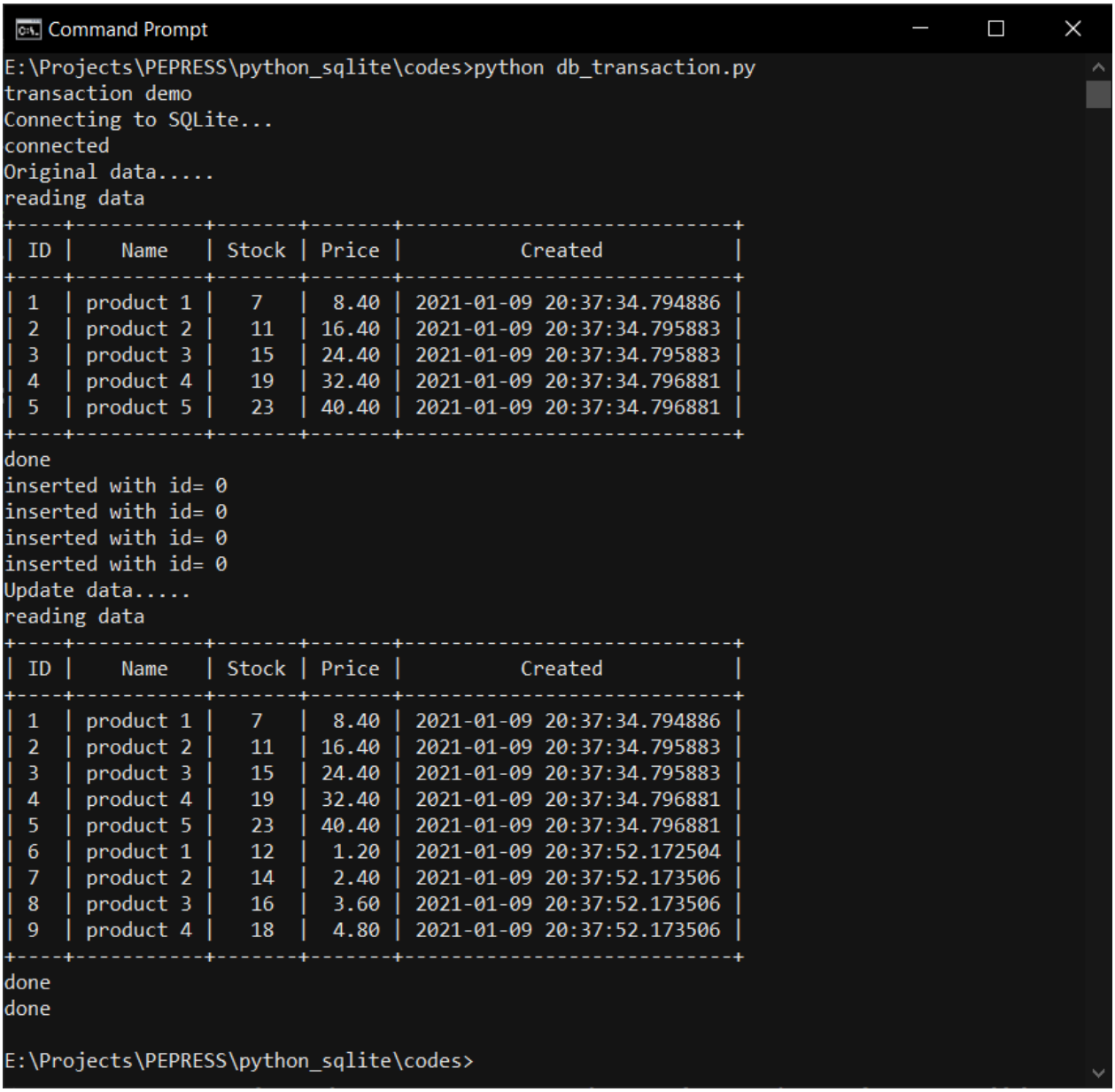
```
    print('done')


# creating data demo
print('transaction demo')

conn = open_database()
print('Original data.....')
read_data(conn)

# set manual transaction
conn.isolation_level = None


try:
    cursor = conn.cursor()
    cursor.execute("BEGIN")
    for index in range(1,5):
        product_name = 'product ' + str(index)
        price = 1.2 * index
        stock = 10 + 2*index

        insert_sql = ("INSERT INTO product "
                "(name, stock, price, created) "
                "VALUES(?, ?, ?, ?)")

        # demo error
        # if index == 3:
        #     insert_sql =
insert_sql.replace('INSERT','INSERT1') # wrong statement

        params = (product_name, stock, price,
datetime.now())
        conn.execute(insert_sql, params)
        product_id = cursor.lastrowid
        print('inserted with id=', product_id)

    conn.commit()
    cursor.close()

except Exception as e:
    cursor.execute("ROLLBACK")
    conn.rollback()
    print('error in inserting data')
    print(e)


print('Update data.....')
read_data(conn)
```

```
conn.close()
print('done')
```

Save this program. Now you can run this file.

```
$ python db_transaction.py
```

If succeed, you can see the response as follows.



For rollback demo, we make error when inserting data on index=3. We only make invalid query. We modify our codes (uncomment codes).

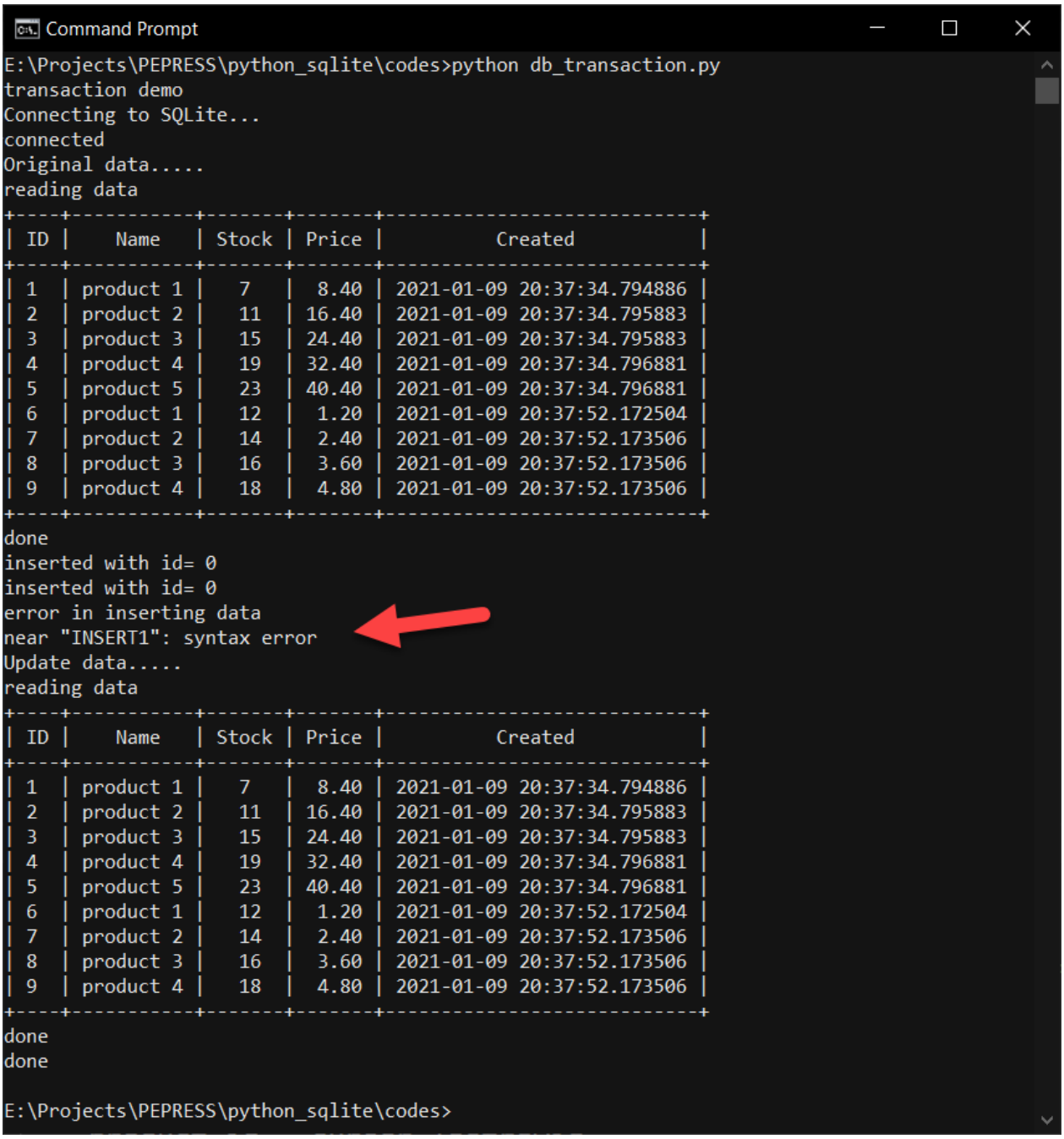
```
import sqlite3
from datetime import datetime
from prettytable import PrettyTable


def open_database():
    db = 'pydb.db'

    print('Connecting to SQLite...')
    conn = sqlite3.connect(db)
    print('connected')

    return conn

def read_data(conn):
    print('reading data')
    cursor = conn.cursor()

    cursor.execute("select id, name, stock, price, created
from product")
    t = PrettyTable(['ID','Name', 'Stock', 'Price',
'Created'])
    for (id, name, stock, price, created) in cursor:
        t.add_row([id, name, stock, format(price,'.2f'),
created])

    print(t)
    cursor.close()
    print('done')


# creating data demo
print('transaction demo')

conn = open_database()
print('Original data.....')
read_data(conn)

# set manual transaction
conn.isolation_level = None


try:
```

```
    cursor = conn.cursor()
    cursor.execute("BEGIN")
    for index in range(1,5):
        product_name = 'product ' + str(index)
        price = 1.2 * index
        stock = 10 + 2*index

        insert_sql = ("INSERT INTO product "
                "(name, stock, price, created) "
                "VALUES(?, ?, ?, ?)")

        # demo error
        if index == 3:
            insert_sql =
insert_sql.replace('INSERT','INSERT1') # wrong statement

        params = (product_name, stock, price,
datetime.now())
        conn.execute(insert_sql, params)
        product_id = cursor.lastrowid
        print('inserted with id=', product_id)

    conn.commit()
    cursor.close()

except Exception as e:
    cursor.execute("ROLLBACK")
    conn.rollback()
    print('error in inserting data')
    print(e)


print('Update data.....')
read_data(conn)

conn.close()
print('done')
```

Save and run this program.

Since our program performs rollback operation, we don't see our inserted data on database.

The following is a program output.


Command Prompt
E:\Projects\PEPRESS\python_sqlite\codes>python db_transaction.py
transaction demo
Connecting to SQLite...
connected
Original data.....
reading data
+----+-----------+-------+-------+----------------------------+
| ID | Name | Stock | Price | Created |
+----+-----------+-------+-------+----------------------------+
1	product 1	7	8.40	2021-01-09 20:37:34.794886
2	product 2	11	16.40	2021-01-09 20:37:34.795883
3	product 3	15	24.40	2021-01-09 20:37:34.795883
4	product 4	19	32.40	2021-01-09 20:37:34.796881
5	product 5	23	40.40	2021-01-09 20:37:34.796881
6	product 1	12	1.20	2021-01-09 20:37:52.172504
7	product 2	14	2.40	2021-01-09 20:37:52.173506
8	product 3	16	3.60	2021-01-09 20:37:52.173506
9	product 4	18	4.80	2021-01-09 20:37:52.173506
+----+-----------+-------+-------+----------------------------+
done
inserted with id= 0
inserted with id= 0
error in inserting data
near "INSERT1": syntax error
Update data.....
reading data
+----+-----------+-------+-------+----------------------------+
| ID | Name | Stock | Price | Created |
+----+-----------+-------+-------+----------------------------+
1	product 1	7	8.40	2021-01-09 20:37:34.794886
2	product 2	11	16.40	2021-01-09 20:37:34.795883
3	product 3	15	24.40	2021-01-09 20:37:34.795883
4	product 4	19	32.40	2021-01-09 20:37:34.796881
5	product 5	23	40.40	2021-01-09 20:37:34.796881
6	product 1	12	1.20	2021-01-09 20:37:52.172504
7	product 2	14	2.40	2021-01-09 20:37:52.173506
8	product 3	16	3.60	2021-01-09 20:37:52.173506
9	product 4	18	4.80	2021-01-09 20:37:52.173506
+----+-----------+-------+-------+----------------------------+
done
done
E:\Projects\PEPRESS\python_sqlite\codes>

6. Python, SQLite and Pandas

This chapter explains how to work with Pandas and SQLite.

6.1 Demo

In this section, we learn how to access SQLite using Pandas. As we know, Pandas is a Python library that we can perform data processing. Before we use Pandas library, we should install Pandas and Numpy libraries on our local computer using pip command. You can type this command

```
$ pip install numpy pandas
```

For testing, we previous a database file, called We open a database file and then pass the database object into Pandas object. After that, we perform queries by calling read_sql_query() function. You can write the following complete codes for our demo.

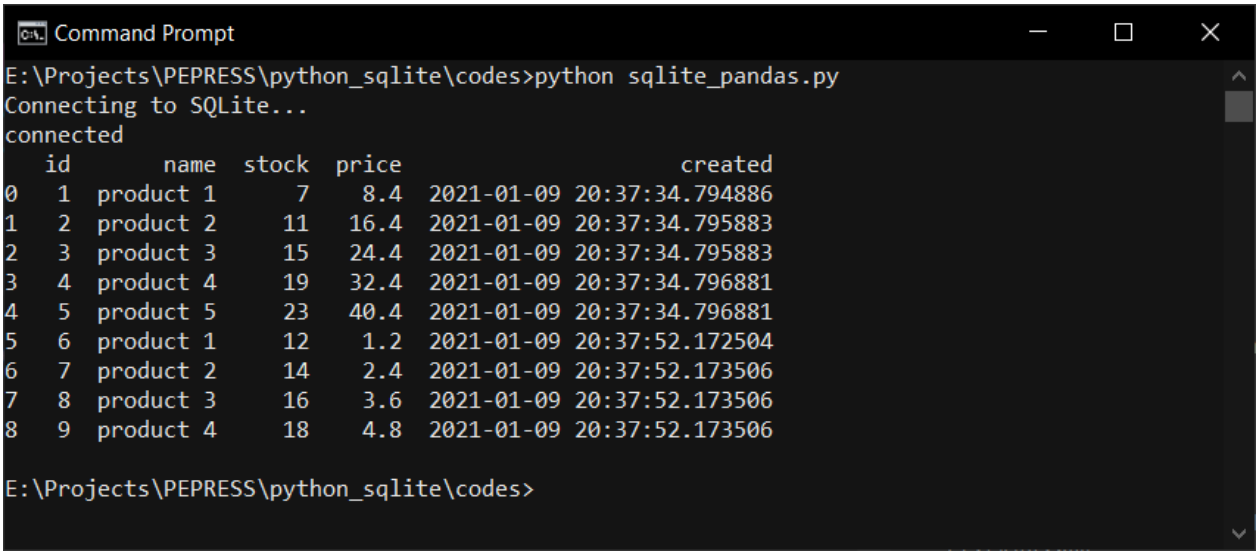
```
import sqlite3
from datetime import datetime
from prettytable import PrettyTable
import pandas as pd


def open_database():
    db = 'pydb.db'

    print('Connecting to SQLite...')
    conn = sqlite3.connect(db)
    print('connected')

    return conn


conn = open_database()
df = pd.read_sql_query("select * from product", conn)
print(df)
```

```
conn.close()
```

Save this file. You can execute file

```
$ python sqlite_pandas.py
```

Here is a sample of program output.



Source Code

You can download source code for this book on http://www.makers.id/ak/pypostgres12195.zip

Contact

If you have question related to this book, please contact me at aguskur@hotmail.com . My blog: http://blog.aguskurniawan.net .