Python

담당 : 이도연

교육기간 : 2019.5.18~19.6.8

>> 데이터베이스 – SQLite3

- SQLite는 오픈소스이자 모든 기능을 갖춘 독립형(외부 라이브러리를 거의 필요로 하지 않음), 서버리스(서버가 데이터베이스 엔진을 실행하지 않아도 되고 로컬에 저장된 데이터베이스), 제로 설정(설치하거나 설정할 것이 아무것도 없음), SQL 기반 경량 데이터베이스 관리 시스템(SQL 쿼리가 SQLite 테이블을 대상으로 실행될 수 있음)으로서 *하나의* 데이터 파일을 사용해 데이터를 저장합니다.
- SQLite의 좋은 점은 SQLite가 구글, 애플, 마이크로소프트 등의 대기업에서 사용되므로 매우 안정적이라는 것입니다. 이 튜토리얼에서는 SQLite를 사용해 데이터베이스와 상호작용하고 더 구체적으로 파이썬에서 sqlite3 모듈을 사용할 것입니다.

Sqlite3 란?

- 디스크 기반의 가벼운 데이터베이스 라이브러리
- 서버가 필요치 않음
- 모바일 디바이스에 기본적으로 사용되고 있음
- 파이썬에도 기본적으로 포함되어 있음

>> 수행 순서(단계)

1. 선택한 데이터베이스에 대한 *연결(connection)*을 설정
2. 데이터 전달을 위한 *커서(cursor)*를 생성
3. SQL을 이용해 데이터를 조작*(상호작용)*
4. SQL 조작을 데이터에 적용한 후 이를 영구적으로 반영하거나(*커밋*) 그러한 조작을 중단시켜(*롤백*) 상호작용이 발생하기 전의 상태로 데이터를 되돌리도록 연결에 지시
5. 데이터베이스에 대한 연결을 *닫음(close)*

- import sqlite3
- conn = sqlite3.connect('company.db')
- curs = conn.cursor()
- curs.execute('create table employee (name, age)')
- curs.execute("insert into employee values ('Ali', 28)")
- values = [('Brad',54), ('Ross', 34), ('Muhammad', 28), ('Bilal', 44)]
- curs.executemany('insert into employee values(?,?)', values)
- conn.commit()
- conn.close()

>> SQLite3 기본

모듈 함수

함수	설명
sqlite3. connect(database[timeout, isolation_level, detect_types, factory])	SQLite3 DB에 연결
sqlite3.complete_statement(sql)	SQL 문장에 대해서 True를 반환
sqlite3.register_adapter(type, callable)	사용자 정의 파이썬 자료형을 SQLite3에서 사용하도록 등록
sqlite3.register_converter(typename, callable)	SQLite3에 저장된 자료를 사용자 정의 자료형으로 변환하는 함수를 등록

Connection 클래스

메서드	설명
Connection.cursor()	Cursor 객체 생성
Connection.commit()	현재 트랜잭션의 변경내역을 DB에 반영(commit)함
Connection.rollback()	가장 최근의 commit() 이후 지금까지 작업한 내용에 대해서 되돌림
Connection.close()	DB 연결을 종료
Connection.isolation_level()	트랜잭션의 격리 수준(isolation level)을 확인/설정

Connection 클래스

메서드	설명
Connection.execute(sql[, parameters]) Connection.executemany(sql[, parameters]) Connection.executescript(sql_script)	임시 Cursor 객체를 생성하여 해당 execute 계열 메서드를 수행
Connection.create_aggregate(name, num_params, aggregate_class)	사용자 정의 집계(aggregate) 함수를 생성
Connection.create_collation(name, callable)	문자열 정렬시 SQL 구문에서 사용될 이름(name) 과 정렬 함수를 지정
Connection.iterdump()	연결된 DB의 내용을 SQL 질의로 형태로 출력

Cursor 클래스

메서드	설명
Cursor.execute(sql[, parameters])	SQL 문장을 실행
Cursor.executemany(sql, seq_of_ parameters)	동일한 SQL 문장을 파라미터만 변경하며 수행
Cursor.executescript(sql_script)	세미콜론으로 구분된 연속된 SQL 문장을 수행
Cursor.fetchone()	조회된 결과(Record Set)로부터 데이터 1개를 반환
Cursor.fetchmany([size=cursor. arraysize])	조회된 결과로부터 입력받은 size 만큼의 데이터를 리스트 형태로 반환
Cursor.fetchall()	조회된 결과 모두를 리스트 형태로 반환

>> SQLite3 사용법

데이터베이스 연결

실제 파일을 이용한 Connection 객체 생성

```
>>> import sqlite3
>>> con = sqlite3.connect("test.db")
```

메모리를 이용한 Connection 객체 생성

```
>>> con = sqlite3.connect(":memory:")
```

SQL문 수행

Cursor.execute 메서드를 이용하여 SQL문을 수행

```
>>> import sqlite3
>>> con = sqlite3.connect(":memory:")
>>> cur = con.cursor()
>>> cur.execute("CREATE TABLE PhoneBook(Name text, PhoneNum text);")
>>> cur.execute("INSERT INTO PhoneBook VALUES('Someone', '010-1234-5678');")
```

인자를 전달하여 SQL문을 수행

```
>>> name = "Someone"
>>> phoneNumber = '010-5678-1234'
>>> cur.execute("INSERT INTO PhoneBook
VALUES(?, ?);", (name, phoneNumber))
```

SQL문 수행

사전을 이용한 인자 전달

```
>>> cur.execute("INSERT INTO PhoneBook VALUES(:inputName, :inputNum);",
{"inputNum":phoneNumber, "inputName":name})
```

동일한 문장을 매개변수만 바꾸며 연속적으로 수행하는 경우

```
>>> datalist = (('Tom', '010-543-5432'), ('DSP', '010-123-1234'))
>>> cur.executemany("INSERT INTO PhoneBook VALUES(?, ?);", datalist)
```

레코드 조회

fetch를 이용한 레코드 조회 방법

```
>>> cur.execute("SELECT * FROM PhoneBook;")
>>> for row in cur:
        print(row)
('Derick', '010-1234-5678')
('Someone', '010-5670-2343')
('Tom', '010-543-5432')
('DSP', '010-123-1234')
```

레코드 조회

fetchone, fetchmany를 이용한 레코드 조회 방법

```
>>> cur.execute("SELECT * FROM PhoneBook;")
>>> cur.fetchone()
('Derick', '010-1234-5678')
>>> cur.fetchmany(2)
[('Someone', '010-5670-2343'), ('Tom', '010-543-5432')]
```

fetchall의 사용법

```
>>> cur.execute("SELECT * FROM PhoneBook;")
>>> cur.fetchone()
('Derick', '010-1234-5678')
>>> cur.fetchall()
[('Someone', '010-5670-2343'), ('Tom', '010-543-5432'), ('DSP', '010-123-1234')]
```

트랜잭션 처리

작업한 내용이 커밋되지 않는 예제

```
01 import sqlite3
02 con = sqlite3.connect("./test.db")
03 cur = con.cursor()
04 cur.execute("CREATE TABLE PhoneBook(Name text, PhoneNum text);")
05 cur.execute("INSERT INTO PhoneBook VALUES('Derick', '010-1234-5678');")
06 cur.execute("SELECT * FROM PhoneBook;")
07 print(cur.fetchall())
```

트랜잭션 처리

작업한 내용이 커밋되지 않는 결과 확인

```
01 import sqlite3
02 con = sqlite3.connect("./test.db")
03 cur = con.cursor()
04 cur.execute("SELECT * FROM PhoneBook;")
05 print(cur.fetchall())
```

트랜잭션 처리

작업한 내용을 커밋하는 예제

```
01 import sqlite3
02 con = sqlite3.connect("./commit.db")
03 cur = con.cursor()
04 cur.execute("CREATE TABLE PhoneBook(Name text, PhoneNum text);")
05 cur.execute("INSERT INTO PhoneBook VALUES('Derick', '010-1234-5678');")
06 con.commit()
```

트랜잭션 처리

작업한 내용을 커밋한 내용을 확인하는 예제

```
01 import sqlite3
02 con = sqlite3.connect("./commit.db")
03 cur = con.cursor()
04 cur.execute("SELECT * FROM PhoneBook;")
05 print(cur.fetchall())
```

레코드 정렬

ORDER BY 사용 예제

```
>>> cur.execute("SELECT * FROM PhoneBook ORDER BY Name")
>>> [r for r in cur]
[('DSP', '010-123-1234'), ('Derick', '010-1234-5678'), ('Someone', '010-5670-2343'),
('Tom', '010-543-5432')]

>>> cur.execute("SELECT * FROM PhoneBook ORDER BY Name DESC")
>>> [r for r in cur]
[('Tom', '010-543-5432'), ('Someone', '010-5670-2343'), ('Derick', '010-1234-5678'),
('DSP', '010-123-1234')]
```

레코드 정렬

사용자 임의로 정렬 방식을 변경하는 경우

```
>>> def OrderFunc(str1, str2): #대소문자 구별 없이 정렬하는 함수
        s1 = str1.upper()
        s2 = str2.upper()
        return (s1 > s2) - (s1 < s2)
>>> con.create_collation('myordering', OrderFunc) #SQL 구문에서 호출할 이름과 함수를 등록
>>> cur.execute("SELECT Name FROM PhoneBook ORDER BY Name COLLATE myordering")
>>> [r[0] for r in cur]
['apple', 'Derick', 'DSP', 'Someone', 'Tom']
```

내장/집계 함수

함수	설명
abs(x)	절대값을 반환
length(x)	문자열의 길이를 반환
lower(x)	소문자로 변환해서 반환
upper(x)	대문자로 변환해서 반환
min(…)	최소값을 반환
max(…)	최대값을 반환
random(*)	임의의 정수를 반환
count(x)	NULL이 아닌 튜플의 개수를 반환
count(*)	튜플의 개수를 반환
sum(x)	합을 반환

자료형

◉ SQLite3 자료형과 그에 대응되는 파이썬의 자료형

SQLite3 자료형	파이썬 자료형
NULL	None
INTEGER	int
REAL	float
TEXT	str, float
BLOB	buffer

```
>>> cur.execute("CREATE TABLE tbl_1(Name TEXT, Age INTEGER, Money REAL);")
>>> cur.execute("CREATE TABLE tbl_2(Name str, Age int, Money float);")
```

데이터베이스 덤프 만들기

데이터베이스 덤프 예제 코드

```
01 import sqlite3
02 con = sqlite3.connect(":memory:")
03 cur = con.cursor()
04
05 cur.execute("CREATE TABLE PhoneBook(Name text, PhoneNum text);")
06 cur.execute("INSERT INTO PhoneBook VALUES('Derick', '010-1234-5678');")
07 list = (('Tom', '010-543-5432'), ('DSP', '010-123-1234'))
08 cur.executemany("INSERT INTO PhoneBook VALUES(?, ?);", list)
09
10 for l in con.iterdump():
11             print(l)
```

데이터베이스 덤프 만들기

실행 결과

```
BEGIN TRANSACTION;
CREATE TABLE PhoneBook(Name text, PhoneNum text);
INSERT INTO "PhoneBook" VALUES('Derick','010-1234-5678');
INSERT INTO "PhoneBook" VALUES('Tom','010-543-5432');
INSERT INTO "PhoneBook" VALUES('DSP','010-123-1234');
COMMIT;
```

명령어 프롬프트에서 SQLite3 관리하기

요구사항

- 입력받은 SQL 구문을 실행
- Select 문을 입력받은 경우에는 fetch를 사용해서 결과 출력

실행 결과

```
선택 C:₩Windows₩py.exe
>>CREATE TABLE phonebook(name text, age integer)
Invalid sql statement
>>CREATE TABLE phonebook(name text, age integer);
success
>>INSERT INTO phonebook('someone', 10 )
Invalid sql statement
>>INSERT INTO phonebook('someone', 10 );
Error: near "10": syntax error
>>INSERT INTO phonebook VALUES('someone', 10 );
success
>>INSERT INTO phonebook VALUES('john', 12 );
success
>>select * from phonebook;
[('someone', 10), ('john', 12)]
success
>>
```

소스코드

sqlite.py

```python
import sqlite3
import sys

if len(sys.argv) == 2:
    path = sys.argv[1]
else:
    path = ":memory:"

con = sqlite3.connect( path )
con.isolation_level = None
cur = con.cursor()

while True:
    sql = input( ">>" ).strip()
    if sql=="":
        break

    if sqlite3.complete_statement( sql ):
        try:
            cur.execute( sql )
            if sql.upper().startswith( "SELECT" ):
                print( cur.fetchall() )
        except sqlite3.Error as e:
            print( "Error:", e.args[0] )
        else:
            print( "success" )
    else:
        print( "Invalid sql statement" )

con.close()
print( "quit" )
```

>> file , excel 로 된 파일을 DB에 처리

- Text 파일 불러와서 sqlite3 db에 저장하기
- 엑셀 파일 처리 하기

```
arr = []
with open("C:₩₩Labs₩₩work_python₩₩day9_sqlite₩₩score.txt", "r", encoding='utf-8') as f:
    for i in range(6):
        arr.append(f.readline().split())

for i in range(len(arr)):
    arr[i][1] = int(arr[i][1])
    arr[i][2] = int(arr[i][2])
    arr[i][3] = int(arr[i][3])
    arr[i].append(arr[i][1] + arr[i][2] + arr[i][3])
    arr[i].append(arr[i][4] / 3)

kor=0
eng =0
math = 0

for i in range(len(arr)):
    kor = kor + arr[i][1]
    eng = eng + arr[i][2]
    math = math + arr[i][3]
```

```
22  print("** 성적표 **")
23  print("-----------------------------------")
24  print("이름   국어  영어  수학  총점  평균")
25  print("-----------------------------------")
26
27  for i in range(len(arr)):
28      print("{}   {}  {}  {}  {}  {:.1f}".format(arr[i][0],arr[i][1],arr[i][2],arr[i][3],arr[i][4],arr[i][5]))
29  print("-----------------------------------")
30  print("평균 : {:.1f} {:.1f} {:.1f}   {:.1f}".format(kor/6, eng/6, math/6, (kor+eng+math)/3/6 ))
```

```
32  import sqlite3 as s
33  conn = s.connect("kosmoDB.db")
34  cursor = conn.cursor()
35
36  cursor.execute("drop table if exists student")
37  cursor.execute("create table student(name char(20), kor int, eng int, math int)")
38
39  query = "insert into student(name, kor, eng, math) values(?, ?, ?, ?)"
40
41  for i in range(6):
42      cursor.execute(query, (arr[i][0],arr[i][1],arr[i][2],arr[i][3]))
43
44  conn.commit()
45  cursor.execute("select * from student")
46
47  rows = cursor.fetchall()
```

```
49  print("=======================================")
50  print("이 름   국어  영어  수학")
51  print("=======================================")
52
53  for row in rows :
54      print("{:4s} {:4d} {:4d} {:4d}".format(row[0], row[1],row[2], row[3]))
55
56  print("=======================================")
57  # 김연아 검색하기
58  cursor.execute('select * from student where name="김연아" ')
59  rows = cursor.fetchall()
60  print("₩n search result ==> {} 학생의 성적".format('김연아'))
61
62  for i in rows:
63      print(i)
```

>> ex2) text -> excel => python

```
# pip install openpyxl
# pip freeze
import openpyxl

excelFile = openpyxl.load_workbook('D:₩₩190413_0504_python_basic₩₩workspace₩₩score.xlsx')
aws =excelFile.active
print("++++++++++++++++++++++++++++++++++++++++++")
print("이름  국어  영어  수학  총점  평균")
print("++++++++++++++++++++++++++++++++++++++++++")
for i in aws.rows:
    index = i[0].row
    name = i[0].value
    kor = i[1].value
    eng = i[2].value
    math = i[3].value
    total = kor + eng + math
    avg = total / 3

    aws.cell(row = index, column = 5).value = total
    aws.cell(row=index, column=6).value = avg
    print('{}₩t  {}₩t  {}₩t{}₩t{}₩t{:.1f}'.format(name, kor, eng, math, total, avg))
print("++++++++++++++++++++++++++++++++++++++++++")

excelFile.save('D:₩₩190413_0504_python_basic₩₩workspace₩₩result.xlsx')
excelFile.close()
```