데이터베이스

SQLite3

- 디스크 기반의 가벼운 데이터베이스 라이브러리
  - 데이터베이스 연결(connect) 관련 함수
  - Connection 클래스
    - 연결된 데이터베이스를 동작시키는 역할
  - Cursor 클래스
    - 실질적으로 데이터베이스에 SQL 문장을 수행하고, 조회된 결과를 가져오는 역할
  - Row 클래스
    - 조회된 결과집합에서 Row 객체는 관계형 데이터베이스 모델에서 튜플을 나타냄
      - Join 연산을 이용해 2개 이상의 테이블을 조회한 결과인 경우, Row 객체는 결과 뷰의 한 행을 나타냄

데이터베이스 연결

- 데이터베이스를 사용하려면 실제 저장된 데이터베이스 파일을 반영하는 connection 객체를 생성해야 함

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

  - 해당 경로에 파일을 생성하며, 파일이 이미 존재할 경우 그 DB 파일을 그대로 사용함.

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

  - ":memory:" 키워드는 메모리 상에 DB를 만들 수 있음
    - 연결이 종료될 경우 현재까지 작업한 모든 내용이 사라짐
    - 물리적인 DB 파일에 기록하는 것보다는 연산속도가 빠름

SQL문 수행

- SQL문은 cursor 클래스의 execute 구문을 사용함
- **테이블 생성**

```
cur = con.cursor()
sql='''create table
        phonebook(name text, phoneNum text);'''
cur.execute(sql)
```

```
cur = con.cursor()
sql='''create table if not exists
        phonebook(name text, phoneNum text);'''
cur.execute(sql)
```

```
dropsql='''drop table if exists phonebook;'''
cur.execute(dropsql)
```

SQL문 수행

- 레코드 삽입

```
insertsql='''insert into phonebook
                values('greenjoa1','010-1111-2222');'''
cur.execute(insertsql)
```

  - **?** : 인자 전달 순서에 맞추어 시퀀스 객체 전달

```
name='greenjoa2'
phoneNumber='010-2222-2222'
insertsql='''insert into phonebook values(?,?);'''
cur.execute(insertsql, (name, phoneNumber))
```

SQL문 수행

- 레코드 삽입
  - 각 인자에 이름을 부여해서 수행하는 방법

```
name='greenjoa3'
phoneNumber='010-3333-3333'
insertsql='''insert into phonebook
        values(:inputName, :inputNum);'''
cur.execute(insertsql,
        {"inputNum":phoneNumber, "inputName":name})
```

SQL문 수행

- 레코드 삽입
  - Iterator 객체를 통한 삽입

```
insertsql='''insert into phonebook values(?,?);'''
datalist=(('greenjoa4','010-4444-4444'),
                    ('greenjoa5','010-5555-5555'))
cur.executemany(insertsql, datalist)
```

  - 제너레이터를 통한 삽입

```
def dataGenerator():
    datalist=(('greenjoa6','010-6666-6666'),('greenjoa7','010-7777-7777'))
    for item in datalist:
        yield item
cur.executemany(insertsql, dataGenerator())
```

*yield : return과 같이 값을 반환하지만, 종료되지 않음

SQL문 수행

- 레코드 조회
  - 입력된 데이터를 데이터베이스로 부터 가져오는 메소드

```
cur.execute("select * from phoneBook;")
for row in cur:
    print(row)
    print(row[0])
```

  - 1개의 레코드 조회, n개의 레코드 조회, 모든 레코드 조회

```
cur.fetchone()

cur.fetchmany(2)

cur.fetchall()
```

트랜잭션

- 트랜잭션?
  - 데이터베이스에서 논리적 작업의 단위

1) A통장에서 100만원 출금 ← 수행 완료 후 정전
2) B통장으로 출금한 100만원 입금 ← 100만원은????

  - 두 개별 작업이 하나의 연산처럼 트랜잭션으로 묶어서 처리하도록 만들며, 이러한 논리적 작업 단위를 트랜잭션이라 함

1) 트랜잭션 시작
2) A통장에서 100만원 출금
3) B통장으로 출금한 100만원 입금
4) 트랜잭션 커밋(commit)/롤백(rollback)

  - 데이터베이스에 반영(commit)되어 영구히 저장되거나, 롤백해 트랜잭션 수행 이전 상태로 복귀

트랜잭션

- 트랜잭션 commit()

```
con = sqlite3.connect("test.db")
cur = con.cursor()
sql='''create table
        phonebook(name text, phoneNum text);'''
cur.execute(sql)
insertsql='''insert into phonebook
                values('greenjoa1','010-1111-2222');'''
cur.execute(insertsql)
con.commit()
```

  - 자동으로 커밋 모드 설정
    - con.isolation_level = None

레코드 정렬

- order by 구문

```
cur.execute("select * from phoneBook order by name;")

cur.execute("select * from phoneBook order by name desc;")
```

- 사용자 지정 정렬 방식

```
def OrderFunc(str1, str2):
    s1 = str1.upper()
    s2 = str2.upper()
    return (s1 > s2) - (s1 < s2)   # 앞 (음수), 같음(0), 뒤(양수)
con.create_collation('myordering', OrderFunc)
cur.execute("select * from phoneBook order by name collate
myordering;")
```

내장 집계 함수

함수	설명
abs(x)	인자의 절대값을 반환
length(x)	문자열의 길이 반환
lower(x)	인자로 받은 문자열을 소문자로 반환. 원본 문자열은 변화 없음
upper(x)	인자로 받은 문자열을 대문자로 반환. 원본 문자열은 변화 없음
min(x,y,…)	인자 중 가장 작은 값을 반환
max(x,y,…)	인자 중 가장 큰 값을 반환
random(*)	임의의 정수를 반환
count(x)	조회 결과 중 필드 인자가 NULL이 아닌 튜플의 개수를 반환
count(*)	조회 결과의 튜플의 개수를 반환
sum(x)	조회 결과 중 필드 인자의 합을 반환

내장 집계 함수

```
cur.execute("insert into phonebook(phoneNum) values('010-9999-9999');")


cur.execute("select count(*) from phoneBook;")
print(cur.fetchone()[0])


cur.execute("select count(name) from phoneBook;")
print(cur.fetchone()[0])
```

사용자 집계 함수

```
class Average:
    def __init__(self):
        self.sum = 0
        self.cnt = 0

    def step(self, value):
        self.sum += value
        self.cnt +=1

    def finalize(self):
        return self.sum/self.cnt

con1.create_aggregate("avg", 1, Average) # DB에 등록
cur1.execute("select avg(Age) from user;")
```

자료형

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

*SQLite3 와 파이썬 자료형은 특별히 변환하지 않고도 사용가능

사용자정의 자료형

```
class Point:
    def __init__(self, x, y):
        self.x = x
        self.y = y
    # Point 객체의 내용 출력
    def __repr__(self):
        return "Point(%f, %f)" % (self.x, self.y)

# 클래스 객체에서 SQLite3 입력 가능한 자료형으로 변환
def PointAdapter(point):
    return "%f:%f"%(point.x, point.y)

# SQLite3에서 조회한 결과를 클래스 객체로 변환
def PointConverter(s):
    x, y = list(map(float, s.decode().split(":")))
    return Point(x,y)
```

사용자정의 자료형

```
# 클래스 이름과 변환 함수 등록
sqlite3.register_adapter(Point, PointAdapter)

# SQL 구문에서 사용할 자료형 이름과 변환 함수 등록
sqlite3.register_converter("point", PointConverter)

p1 = Point(4,3)
p2 = Point(3,4)
con1 = sqlite3.connect(":memory:")
cur1 = con1.cursor();

cur1.execute("create table test(p point);")
cur1.execute("insert into test values(?);", (p1,))
cur1.execute("insert into test(p) values(?);", (p2,))

cur1.execute("select p from test")
print(cur1.fetchone())
```