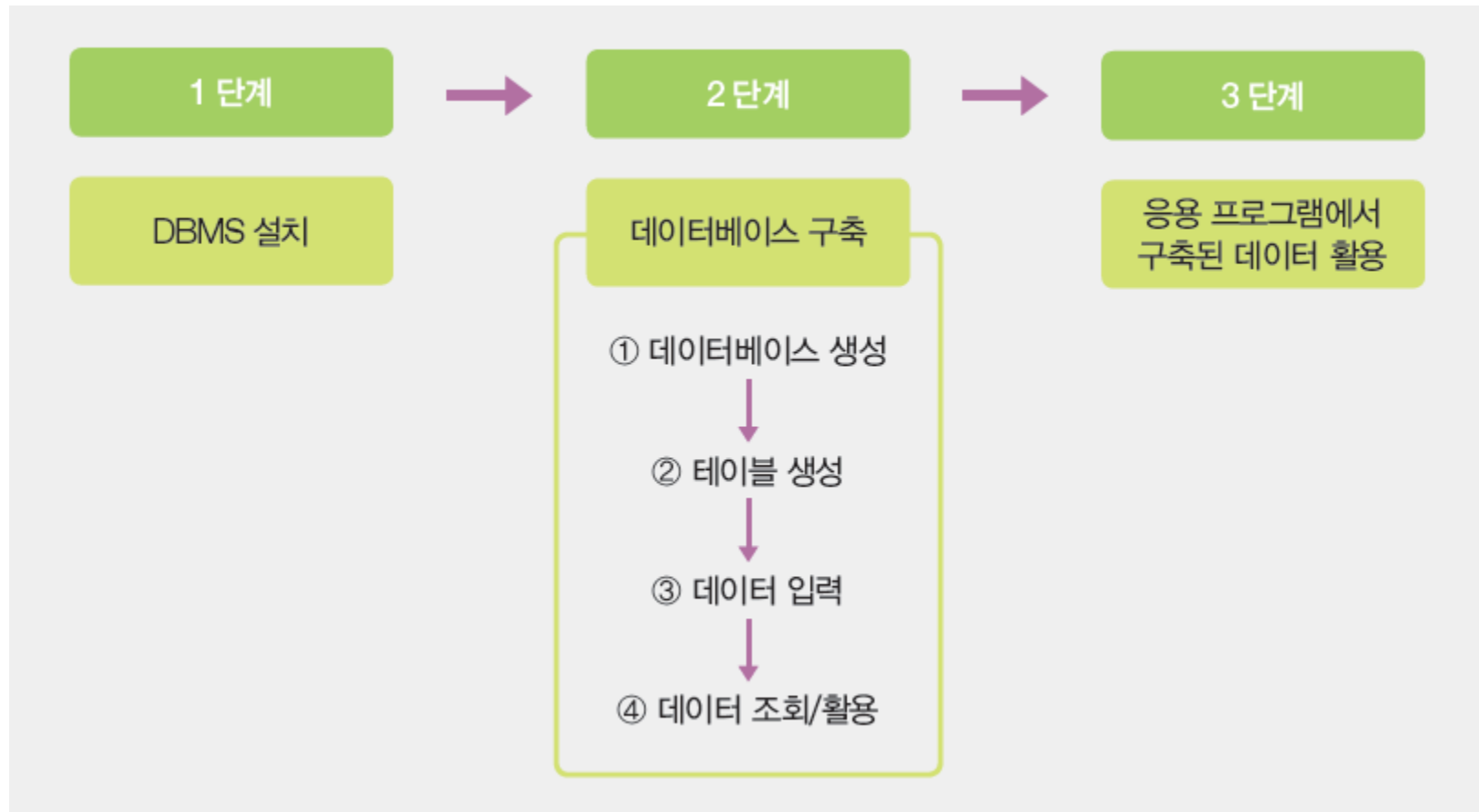




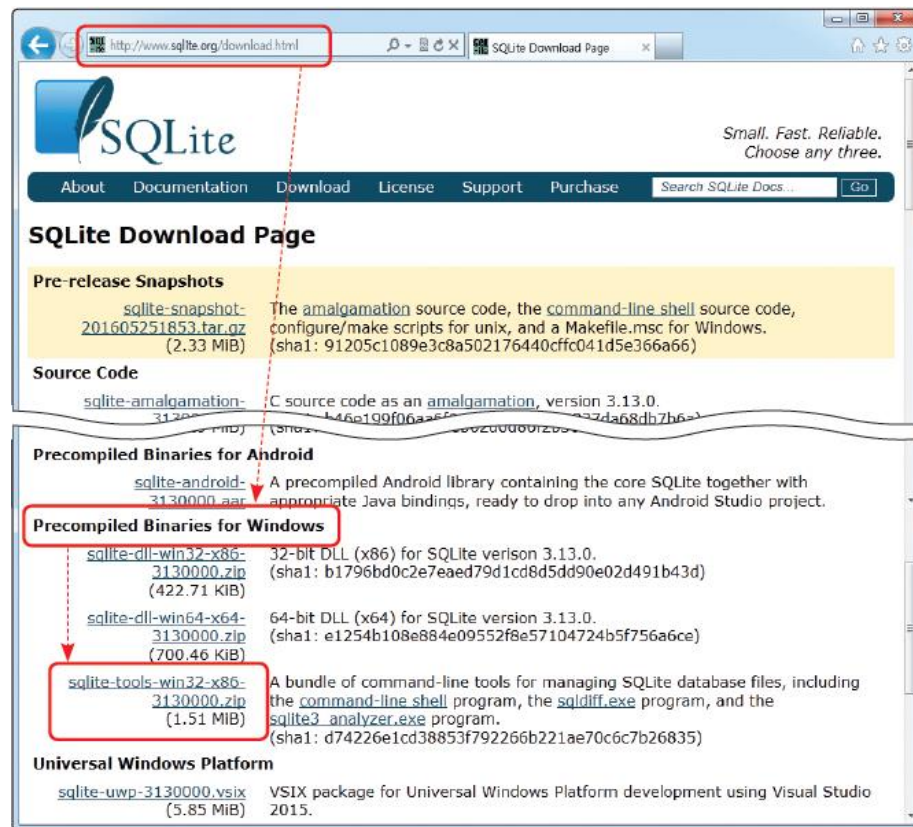
데이터베이스

SQLite를 이용한 데이터베이스

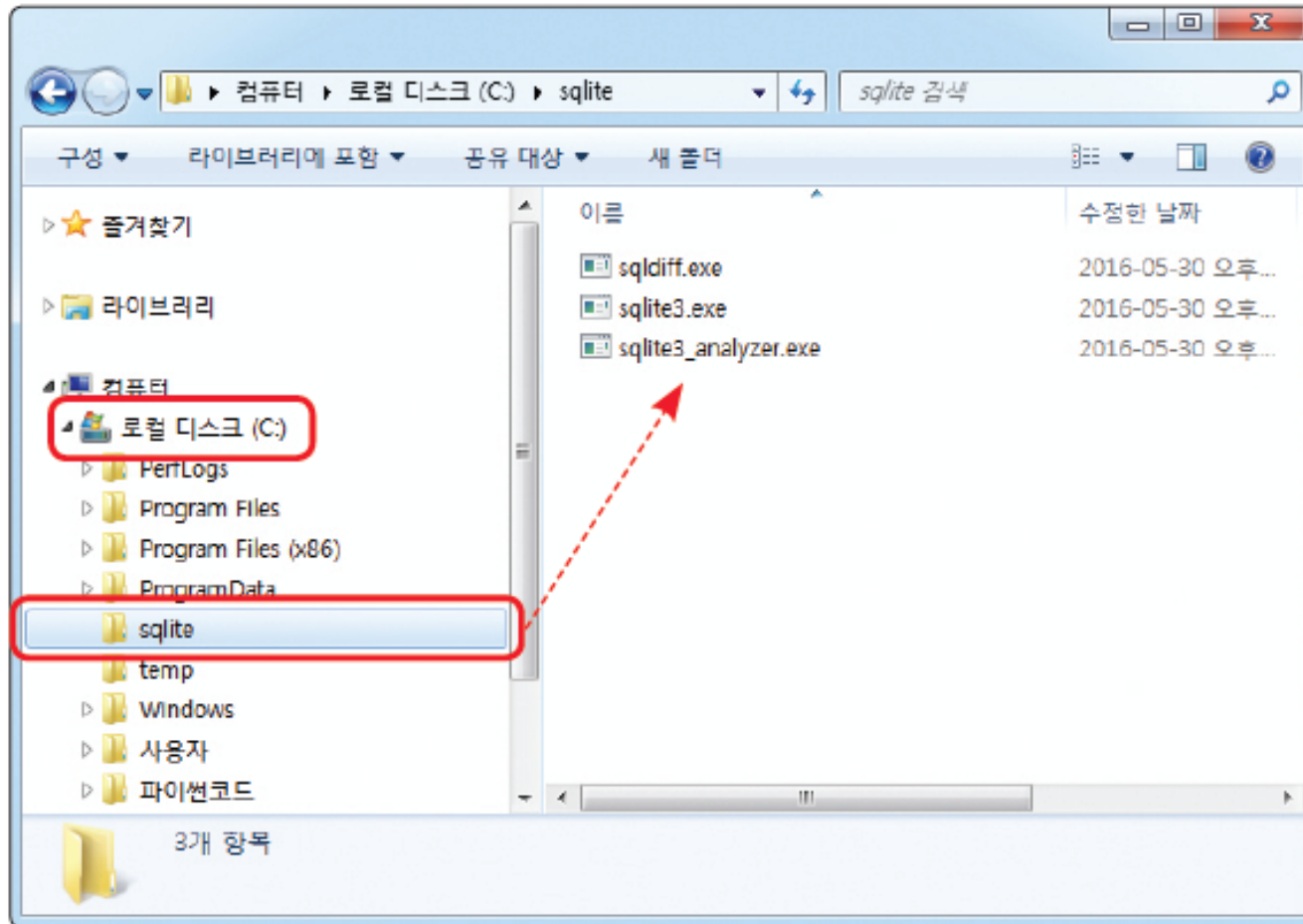


SQLite를 이용한 데이터베이스

- 1 단계: DBMS 설치 - SQLite를 설치
 - <http://www.sqlite.org/download.html>에 접속
 - `sqlite-tools-win32-x86-버전.zip` 파일을 다운로드 한 후 압축을 풀어 폴더 이름을 `sqlite`로 변경

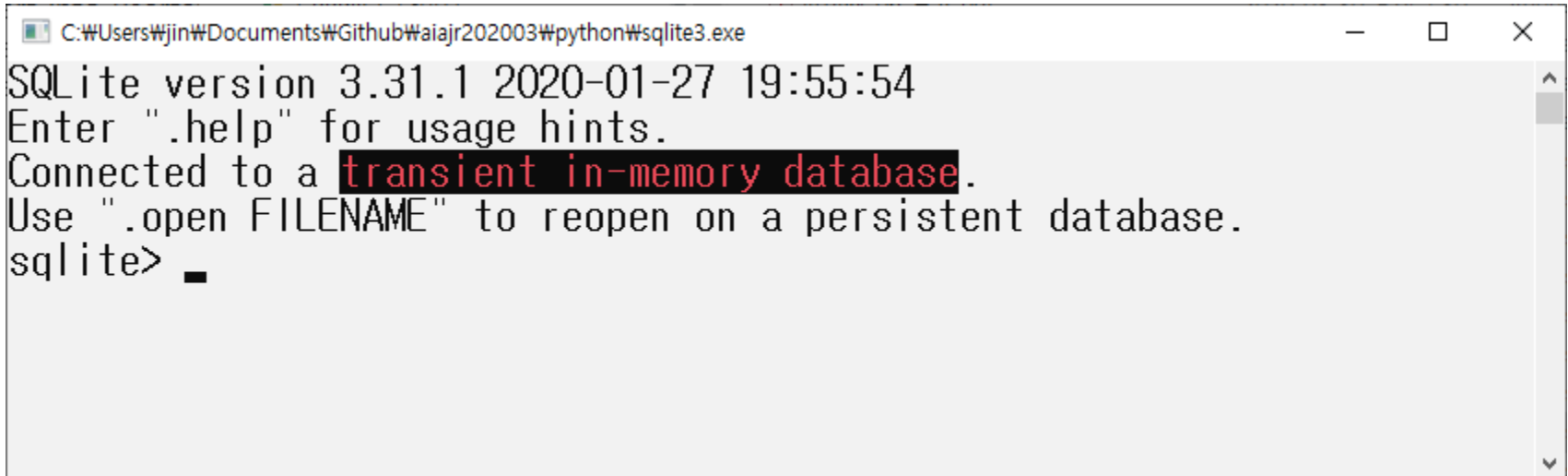


SQLite를 이용한 데이터베이스



SQLite를 이용한 데이터베이스

- 2 단계 : 데이터베이스 구축
 - 파일 탐색기에서 sqlite3.exe를 더블 클릭하면 명령 프롬프트가 실행되고
sqlite> 로 프롬프트가 표시됨



```
C:\Users\Wjin\Documents\Github\aiar202003\python\sqlite3.exe
SQLite version 3.31.1 2020-01-27 19:55:54
Enter ".help" for usage hints.
Connected to a transient in-memory database.
Use ".open FILENAME" to reopen on a persistent database.
sqlite> _
```

SQLite를 이용한 데이터베이스

- SQLite에서 네이버 데이터베이스 완성

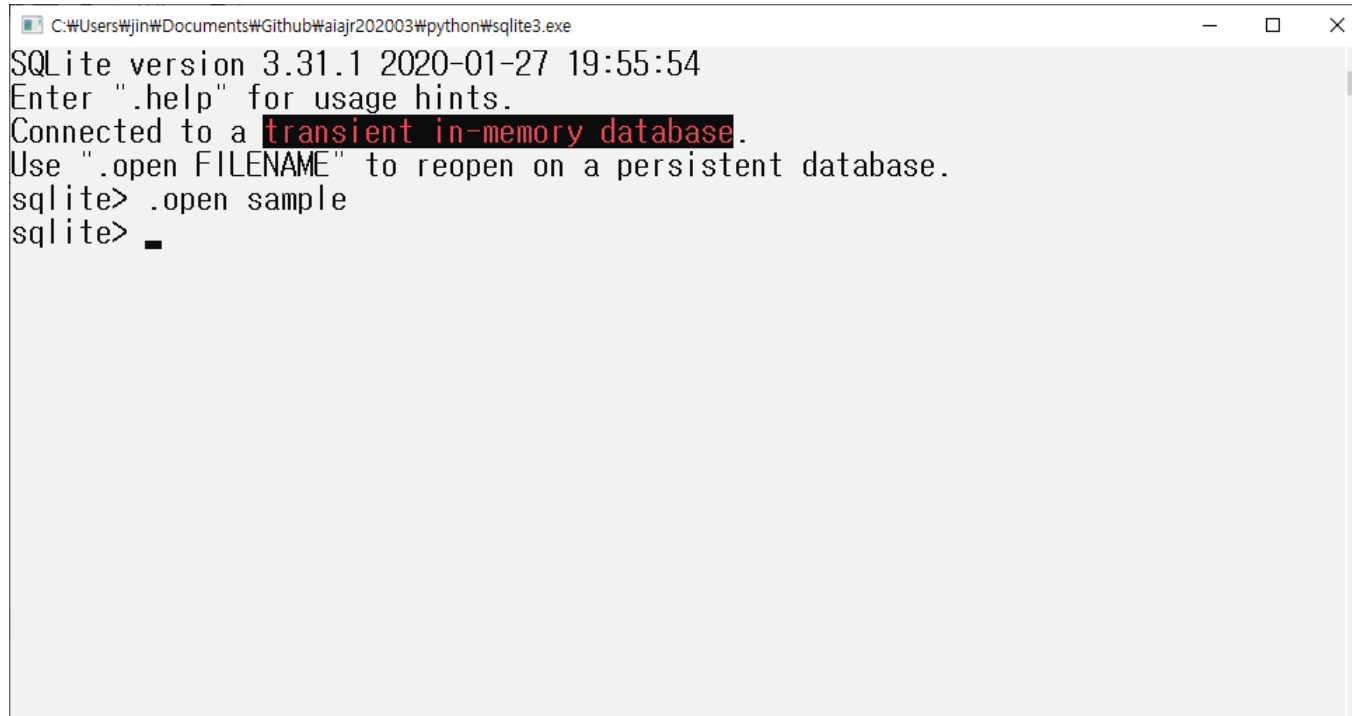
- 2-1 단계 : 데이터베이스 생성

- '.open 데이터베이스이름' 명령 : 데이터베이스를 생성하거나 열어줌

```
.open naverDB
```

- SQL문은 대소문자를 구분하지 않음. 모든 SQL문의 끝에는 세미콜론(;)을 붙임
 - .table : 현재 데이터베이스의 테이블 목록을 보여줌.
 - .schema 테이블이름 : 테이블의 열 및 데이터 형식 등의 정보를 보여줌.
 - .header on : SELECT문으로 출력할 때 헤더를 보여줌.
 - .mode column : SELECT문으로 출력할 때 컬럼 모드로 해줌.
 - .exit : SQLite를 종료함.

SQLite를 이용한 데이터베이스

A screenshot of a SQLite3 command-line interface window. The window title bar shows the file path: C:\Users\Wjin\Documents\Github\aiar202003\python\sqlite3.exe. The main text area displays the following information: 'SQLite version 3.31.1 2020-01-27 19:55:54', 'Enter ".help" for usage hints.', 'Connected to a transient in-memory database.', and 'Use ".open FILENAME" to reopen on a persistent database.' Below this, the prompt 'sqlite>' is shown twice, with the first line followed by the command '.open sample' and the second line followed by a cursor character. A vertical scrollbar is visible on the right side of the text area.

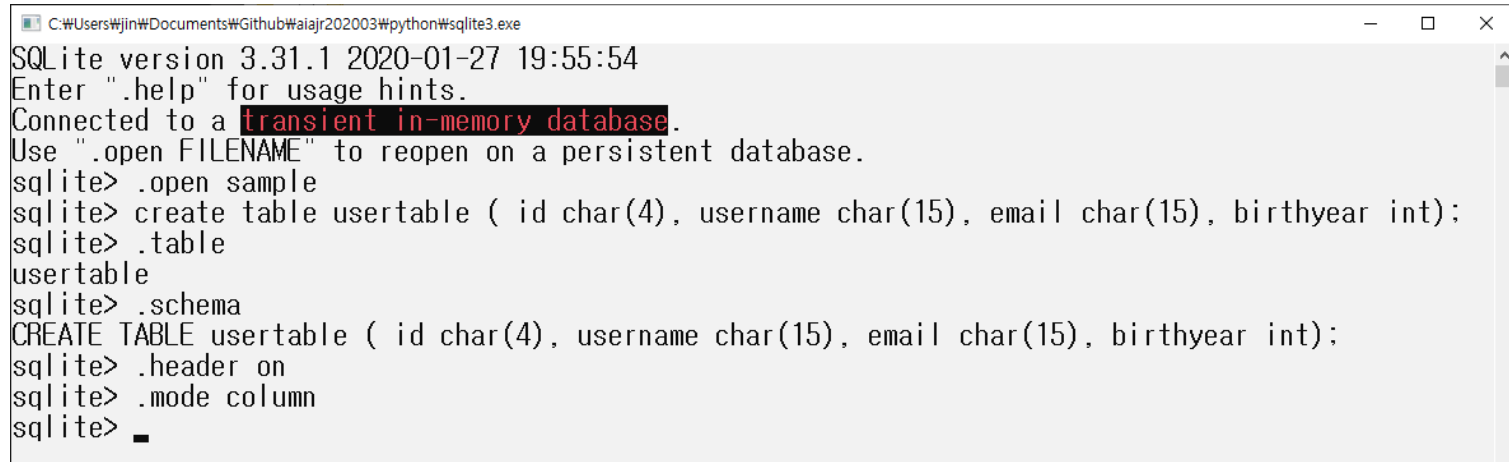
```
C:\Users\Wjin\Documents\Github\aiar202003\python\sqlite3.exe
SQLite version 3.31.1 2020-01-27 19:55:54
Enter ".help" for usage hints.
Connected to a transient in-memory database.
Use ".open FILENAME" to reopen on a persistent database.
sqlite> .open sample
sqlite> _
```

SQLite를 이용한 데이터베이스

- 2-2 단계 : 테이블 생성

```
CREATE TABLE 테이블이름 (열이름1 데이터형식, 열이름2 데이터형식 ...);
```

```
CREATE TABLE userTable (id char(4), userName char(15), email  
char(15), birthYear int);  
.table  
.schema userTable
```



A screenshot of a SQLite3 command-line interface window. The title bar shows the file path: C:\Users\#jin\Documents\Github\#aijr\202003\python\#sqlite3.exe. The window content displays the following text:

```
SQLite version 3.31.1 2020-01-27 19:55:54  
Enter ".help" for usage hints.  
Connected to a transient in-memory database.  
Use ".open FILENAME" to reopen on a persistent database.  
sqlite> .open sample  
sqlite> create table usertable ( id char(4), username char(15), email char(15), birthyear int);  
sqlite> .table  
usertable  
sqlite> .schema  
CREATE TABLE usertable ( id char(4), username char(15), email char(15), birthyear int);  
sqlite> .header on  
sqlite> .mode column  
sqlite> _
```


SQLite를 이용한 데이터베이스

- 생성된 테이블을 삭제하려면 'DROP TABLE 테이블이름;'을 사용

– 2-3 단계 : 데이터 입력

```
INSERT INTO 테이블이름 VALUES(값1, 값2, ...);
```

```
C:\Users\jin\Documents\Github\aijr202003\python\sqlite3.exe
SQLite version 3.31.1 2020-01-27 19:55:54
Enter ".help" for usage hints.
Connected to a transient in-memory database.
Use ".open FILENAME" to reopen on a persistent database.
sqlite> .open sample
sqlite> create table usertable ( id char(4), username char(15), email char(15), birthyear int);
sqlite> .table
usertable
sqlite> .schema
CREATE TABLE usertable ( id char(4), username char(15), email char(15), birthyear int);
sqlite> .header on
sqlite> .mode column
sqlite> insert into usertable values('scott', 'SCOTT', 'scott@gmail.com', 1970);
sqlite> insert into usertable values('king', 'KING', 'king@gmail.com', 1980);
sqlite> insert into usertable values('adam', 'ADAM', 'adam@gmail.com', 1990);
sqlite> _
```

SQLite를 이용한 데이터베이스

– 2-4 단계 : 데이터 조회/활용

```
SELECT * FROM 테이블이름 ;
```

```
C:\Users\Hjin\Documents\Github\Wajr202003\python\sqlite3.exe
Connected to a transient in-memory database.
Use ".open FILENAME" to reopen on a persistent database.
sqlite> .open sample
sqlite> create table usertable ( id char(4), username char(15), email char(15), birthyear int);
sqlite> .table
usertable
sqlite> .schema
CREATE TABLE usertable ( id char(4), username char(15), email char(15), birthyear int);
sqlite> .header on
sqlite> .mode column
sqlite> insert into usertable values('scott', 'SCOTT', 'scott@gmail.com', 1970);
sqlite> insert into usertable values('king', 'KING', 'king@gmail.com', 1980);
sqlite> insert into usertable values('adam', 'ADAM', 'adam@gmail.com', 1990);
sqlite> select * from usertable;
```

id	username	email	birthyear
scott	SCOTT	scott@gmail.com	1970
king	KING	king@gmail.com	1980
adam	ADAM	adam@gmail.com	1990

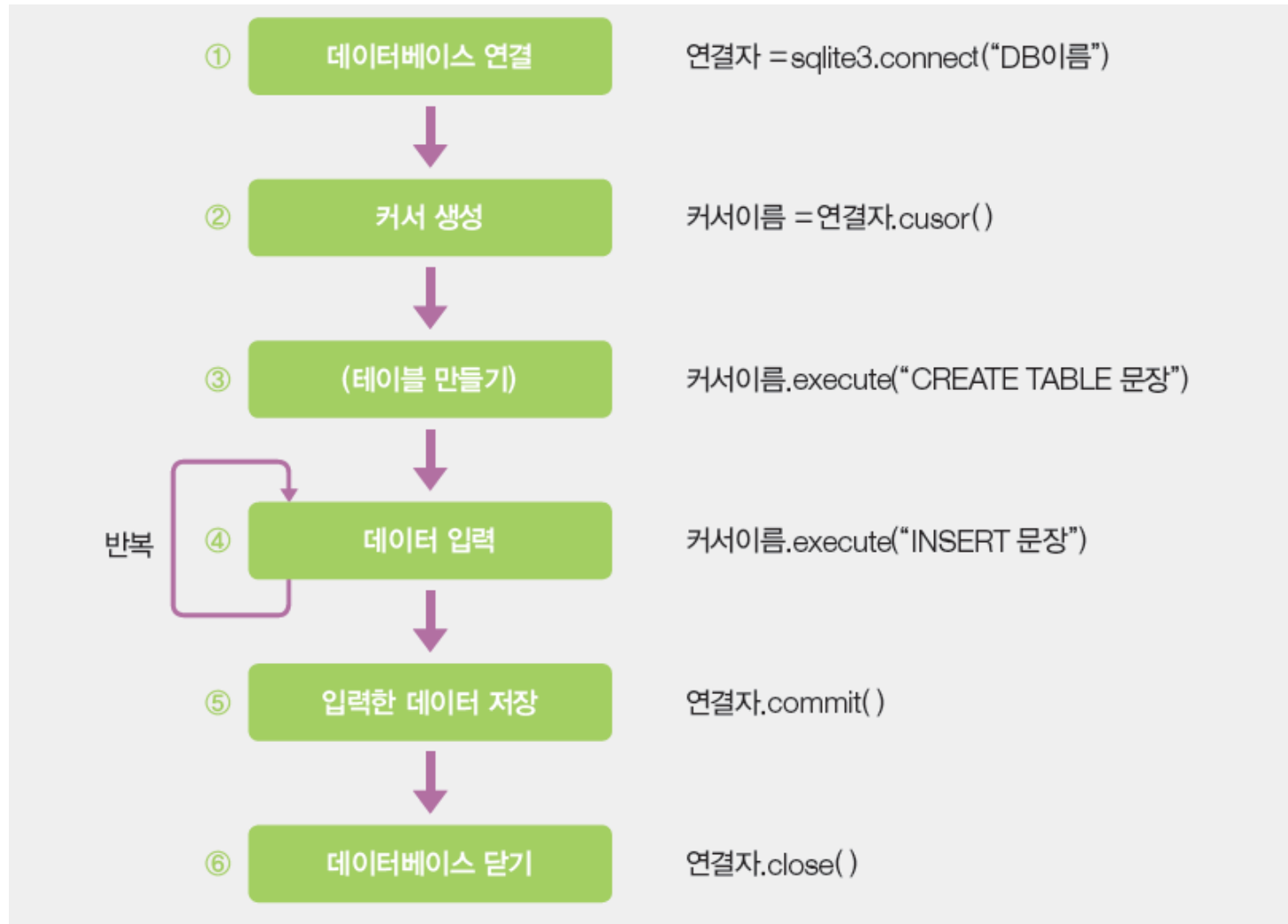
```
sqlite>
```

– SQLite를 종료

```
.exit
```

파이썬에서 SQLite를 활용

- SQLite에 데이터를 입력하기 위한 파이썬 코딩 순서



파이썬에서 SQLite를 활용

– ① 데이터베이스 연결

- SQLite를 사용하기 위해 관련 모듈인 sqlite3를 임포트한 다음 'sqlite3.connect("DB 이름")'으로 데이터베이스와 연결함

```
import sqlite3  
con = sqlite3.connect("naverDB")
```

출력 결과

아무것도 나오지 않음

– ② 커서 생성

- 커서(Cursor)는 데이터베이스에 SQL문을 실행하거나, 실행된 결과를 돌려받는 통로임. ①에서 연결한 연결자에 커서를 만들어야 함

```
cur = con.cursor()
```

출력 결과

아무것도 나오지 않음

파이썬에서 SQLite를 활용

– ③ 테이블 만들기

- 테이블을 만드는 SQL문을 커서이름.execute() 함수의 매개변수로 넘겨주면 SQL문이 데이터베이스에 실행

```
cur.execute("CREATE TABLE userTable (id char(4), userName char(15), email char(15), birthYear int)")
```

출력 결과

```
<sqlite3.Cursor object at 개체번호>
```

– ④ 데이터 입력

- 데이터 입력은 필요한 만큼 반복함. 데이터 입력도 SQL문을 사용해야 하므로 커서이름.execute() 함수를 사용

파이썬에서 SQLite를 활용

```
cur.execute("INSERT INTO userTable VALUES( 'john' , 'John Bann' , 'john@naver.com' , 1990)")
cur.execute("INSERT INTO userTable VALUES( 'kim' , 'Kim Chi' , 'kim@daum.net' , 1992)")
cur.execute("INSERT INTO userTable VALUES( 'lee' , 'Lee Pal' , 'lee@paran.com' , 1988)")
cur.execute("INSERT INTO userTable VALUES( 'park' , 'Park Su' , 'park@gmail.com' , 1980)")
```

출력 결과

<sqlite3.Cursor object at 개체번호>가 각각 4회 출력됨

– ⑤ 입력한 데이터 저장

- ④에서 입력한 4건의 데이터는 임시로 저장된 상태임. 이를 확실하게 저장하는 것을 커밋(Commit)이라고 함

```
con.commit()
```

출력 결과

아무것도 나오지 않음

파이썬에서 SQLite를 활용

– ⑥ 데이터베이스 닫기

```
con.close()
```

출력 결과

아무것도 나오지 않음

파이썬에서 SQLite를 활용

- 데이터 출력 프로그램 작성

```
# 테이블의 질의문 처리
# sqlite3
import sqlite3

# DB 연결
con = sqlite3.connect('sample')

cur = con.cursor()

sql_select = 'select * from usertable'

cur.execute(sql_select)

print('아이디\t이름\t이메일\t\t태어난해')
print('-----')

while True:
    row = cur.fetchone() # 반환 행이 없으면 None 을 반환
    if row == None:
        break
    print('{}\t{}\t{}\t{}'.format(row[0], row[1], row[2], row[3]))

con.close()
```

아이디	이름	이메일	태어난해
scott	SCOTT	scott@gmail.com	1970
king	KING	king@gmail.com	1980
adam	ADAM	adam@gmail.com	1990

파이썬에서 SQLite를 활용

- 데이터 입력 프로그램 작성

```
# DML : insert, update, delete
import sqlite3

# DB 연결
con = sqlite3.connect('sample')
cur = con.cursor()

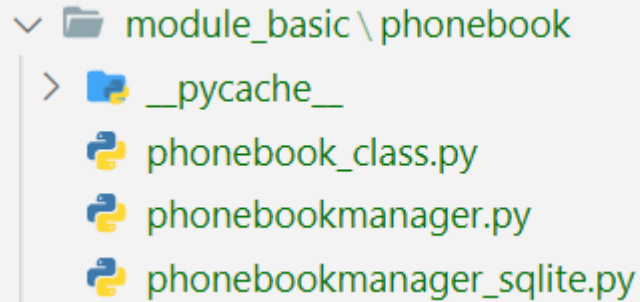
# sql 실행
sql_insert = 'insert into user_table values (\'son\', \'son\', \'son@naver.com\',
1990)'
cur.execute(sql_insert)

con.commit()
con.close()
```

아이디	이름	이메일	태어난해
scott	SCOTT	scott@gmail.com	1970
king	KING	king@gmail.com	1980
adam	ADAM	adam@gmail.com	1990
son	son	son@naver.com	1990

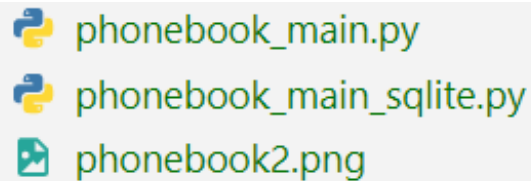
파이썬에서 SQLite를 활용

- 폰북 프로그램 만들기 : github 참고



module_basic\phonebook

- > __pycache__
- phonebook_class.py
- phonebookmanager.py
- phonebookmanager_sqlite.py



- phonebook_main.py
- phonebook_main_sqlite.py
- phonebook2.png



Python + mysql

- MySQL
 - 오픈 소스 관계형 데이터베이스
 - SQLite와 달리 MySQL은 실제 서버로 사용
 - 클라이언트는 네트워크를 통해 MySQL 서버에 접근

Mysql

- mysqlclient
 - url : <https://mysqlclient.readthedocs.io>
 - pip package : mysql-connector-python
 - import : MySQLdb
- MySQL Connector
 - url : <http://bit.ly/mysql-cpdg>
 - pip package : mysql-connector-python
 - import : mysql.connector
- PYMySQL
 - url : <https://github.com/petehunt/PyMySQL>
 - pip package : pymysql
 - import : pymysql
- oursql
 - url : <http://pythonhosted.org/oursql>
 - pip package : oursql
 - import : oursql

- PyMySQL 패키지 설치

```
PS C:\Users\Wjin\Documents\Github\aija202003\python> pip install PyMySQL
Requirement already satisfied: PyMySQL in
c:\Users\Wjin\AppData\Local\Programs\Python\Python38-32\Lib\site-packages (0.9.3)
PS C:\Users\Wjin\Documents\Github\aija202003\python> pip show PyMySQL
Name: PyMySQL
Version: 0.9.3
Summary: Pure Python MySQL Driver
Home-page: https://github.com/PyMySQL/PyMySQL/
Author: yutaka.matsubara
Author-email: yutaka.matsubara@gmail.com
License: "MIT"
Location: c:\Users\Wjin\AppData\Local\Programs\Python\Python38-32\Lib\site-packages
Requires:
Required-by:
```

Mysql

- DB 연결 - connect()

```
import pymysql

project_db = pymysql.connect(
    user='bit',
    passwd='bit',
    host='127.0.0.1',
    db='project',
    charset='utf8'
)
```

- connect() 함수를 이용하면 MySQL host내 DB와 직접 연결할 수 있습니다.
 - user: user name
 - passwd: 설정한 패스워드
 - host: DB가 존재하는 host
 - db: 연결할 데이터베이스 이름
 - charset: 인코딩 설정

Mysql

- DB 연결 - connect()

```
import pymysql

project_db = pymysql.connect(
    user='bit',
    passwd='bit',
    host='127.0.0.1',
    db='project',
    charset='utf8'
)
```

- connect() 함수를 이용하면 MySQL host내 DB와 직접 연결할 수 있습니다.
 - user: user name
 - passwd: 설정한 패스워드
 - host: DB가 존재하는 host
 - db: 연결할 데이터베이스 이름
 - charset: 인코딩 설정

- cursor 설정 - cursor()

```
# 일반 튜플 형태로 값이 반환  
cursor1 = project_db.cursor()  
# 딕셔너리 형태로 값이 반환  
cursor2 = project_db.cursor(pymysql.cursors.DictCursor)
```

- 다양한 커서의 종류가 있지만, 데이터 분석에 적용할 데이터프레임 형태로 결과를 쉽게 변환할 수 있도록 딕셔너리 형태로 결과를 반환해주는 DictCursor를 사용. 딕셔너리 형태로 반환 하면 pandas를 이용한 분석에서 쉽게 사용

- 데이터 조회 - SELECT

```
sql = "SELECT * FROM `project`.`member`;"  
cursor1.execute(sql)  
result1 = cursor1.fetchall()  
  
cursor2.execute(sql)  
result2 = cursor2.fetchall()
```

메서드	설명
fetchall()	모든 데이터를 한 번에 가져올 때 사용
fetchone()	한 번 호출에 하나의 행만 가져올 때 사용
fetchmany(n)	n개만큼의 데이터를 가져올 때 사용

- 데이터 조회 - SELECT

```
print('튜플 형태로 출력')
for item in result1:
    print(item)
print('-----')
print('딕셔너리 형태로 출력')
for item in result2:
    print(item)
```

튜플 형태로 출력

```
(5, 'cool', '1111', '쿨', '/upload/users/360034722685500_erd.png', datetime.datetime(2020, 7, 8, 0, 39, 19))
(6, 'hot', '1234', '핫', None, datetime.datetime(2020, 7, 8, 0, 39, 19))
```

딕셔너리 형태로 출력

```
{'idx': 5, 'uid': 'cool', 'upw': '1111', 'uname': '쿨', 'uphoto': '/upload/users/360034722685500_erd.png', 'regdate': datetime.datetime(2020, 7, 8, 0, 39, 19)}
{'idx': 6, 'uid': 'hot', 'upw': '1234', 'uname': '핫', 'uphoto': None, 'regdate': datetime.datetime(2020, 7, 8, 0, 39, 19)}
```

- 데이터 삽입/변경/삭제 - INSERT/UPDATE/DELETE

```
sql = "insert or update or delete"  
cursor.execute(sql)  
project_db.commit()
```

- execute()를 이용해 INSERT 쿼리를 실행한 후, commit()을 실행.
- commit()을 하지 않으면 execute()를 해도 결과가 DB에 반영되지 않음.
- 처음에 DB 커넥션을 만들 때 autocommit=True 파라미터의 속성값으로 설정 가능.

- `execute()/executemany()`에 Placeholder 사용
 - Placeholder
 - 위와 같이 동적 SQL문을 구성할 때 유용하게 사용할 수 있는 기능으로 동적 값이 들어갈 위치에 %s를 이용해 SQL문을 만들어 놓고, `execute()` 메서드 첫 번째 파라미터에는 SQL을, 두 번째 파라미터에 실제 데이터를 리스트나 튜플 형태로 삽입.
 - `excute(SQL, a-data)` : 하나의 데이터만 적용할 때 사용
 - `executemany(SQL, multiple-data)` : 여러 개의 데이터를 한번에 적용할 때
 - 두 번째 파라미터에 들어간 데이터 순서대로 SQL이 적용
 - 특히 문자의 경우 따옴표 등의 특수문자들이 자동으로 이스케이프(Escape)되어 처리
 - 문자열, 숫자 등에 관계 없이 대치할 값은 모두 %s로 적용해서 사용.

- 데이터 삽입/변경/삭제 - INSERT/UPDATE/DELETE

```
data = ('ramen', 1)
# SELECT
sql = "SELECT * FROM `food` WHERE name = %s AND id = %s;"
cursor.execute(sql, data)

# DELETE
sql = "DELETE FROM `food` WHERE `name` = %s AND `id` = %s;"
cursor.execute(sql, data)
db.commit()
```

- 데이터 삽입/변경/삭제 - INSERT/UPDATE/DELETE

```
data = [['ramen', 1], ['fried rice', 2], ['chicken', 3]]
# SELECT
sql = "SELECT * FROM `food` WHERE name = %s AND id = %s;"
cursor.execute(sql, data)

# DELETE
sql = "DELETE FROM `food` WHERE `name` = %s AND `id` = %s;"
cursor.execute(sql, data)
db.commit()
```

- 반복문 + execute() 메서드를 사용하는 것보다 속도와 메모리 면에서 효율적.