1. 파이썬 데이터 베이스 연동(SQLite)

1-1 테이블 생성

In [11]:
```
import sqlite3
import datetime
```

In [110…
```
print('sqlite3 version :', sqlite3.version)
```

```
sqlite3 version : 2.6.0
```

In [111…
```
# 데이터 추가 날짜 값 추출하기
now = datetime.datetime.now()
print('now : ', now)
nowDatetime = now.strftime('%Y-%m-%d %H:%M:%S')
print('nowDatetime : ', nowDatetime)
```

```
now :  2021-10-24 18:49:41.822415
nowDatetime :  2021-10-24 18:49:41
```

1-2 DB 연결

- sqlite3는 db연결시 filedb가 만들어짐(ex: database.db)

In [112…
```
#conn = sqlite3.connect("sqlite_db.db")
#conn = sqlite3.connect("sqlite_db.db")

# auto commit : 실행하면 db에 바로 반영
#conn = sqlite3.connect("sqlite_db.db", isolation_level=None)
conn = sqlite3.connect("sqlite_db.db", isolation_level=None)
```

1-3 db cursor 식별자 생성

In [113…
```
cur = conn.cursor()
print('cursor type : ', type(cur))
```

```
cursor type :  <class 'sqlite3.Cursor'>
```

1-4 DB 테이블 생성

In [114…
```
cur.execute("create table if not exists usersdb(id integer primary key, \
           username text, email text, phone text, website text, regdate text)")
```

Out[114…
```
<sqlite3.Cursor at 0x7ff9d84d3810>
```

1-5 데이터 CRUD

- create(insert), Read(select), Update(update), Delete(delete)

1-5-1. Create(insert)

In [115…
```
# 테이블 생성(Data Type : TEXT, NUMERIC, INTEGER, REAL, BLOB)
cur.execute("INSERT INTO usersdb VALUES(1, 'Kim', 'kim@cozlab.com', '010-1234-5678', 'cozlab.com', ?)", (nowDatetime,))
```

Out[115…
```
<sqlite3.Cursor at 0x7ff9d84d3810>
```

In [116…
```
# 데이터 insert 튜플형식
cur.execute("INSERT INTO usersdb (id, username, email, phone, website, regdate) VALUES (?,?,?,?,?,?)", \
            (2, 'Park', 'park@naver.com', '010-3456-4567', 'park.com', nowDatetime))
```

Out[116…
```
<sqlite3.Cursor at 0x7ff9d84d3810>
```

In [117…
```
# 데이터 insert many 형식(튜플, 리스트)
# 많은 양의 데이터를 한꺼번에 넣는 방법

userList = (
    (3, 'Lee', 'lee@naver.com', '010-3333-3333', 'lee.com', nowDatetime),
    (4, 'Cho', 'cho@naver.com', '010-4444-4444', 'cho.com', nowDatetime),
    (5, 'Yue', 'yue@naver.com', '010-5555-5555', 'yue.com', nowDatetime),
```

```
    (6, 'Sea', 'sea@naver.com', '010-6666-6666', 'sea.com', nowDatetime),
)
cur.executemany("INSERT INTO usersdb(id, username, email, phone, website, regdate) VALUES (?,?,?,?,?,?)", userList)
```

Out[117…
```
<sqlite3.Cursor at 0x7ff9d84d3810>
```

In [118…
```
# 테이블데이터 보두 삭제
cur.execute("DELETE FROM usersdb")
```

Out[118…
```
<sqlite3.Cursor at 0x7ff9d84d3810>
```

In [119…
```
# 데이터 insert many 형식(튜플, 리스트)
userList = (
    (1, 'kim', 'kim@naver.com', '010-3456-4567', 'kim.com', nowDatetime),
    (2, 'Park', 'park@naver.com', '010-3456-4567', 'park.com', nowDatetime),
    (3, 'Lee', 'lee@naver.com', '010-3333-3333', 'lee.com', nowDatetime),
    (4, 'Cho', 'cho@naver.com', '010-4444-4444', 'cho.com', nowDatetime),
    (5, 'Yue', 'yue@naver.com', '010-5555-5555', 'yue.com', nowDatetime),
    (6, 'Sea', 'sea@naver.com', '010-6666-6666', 'sea.com', nowDatetime),
)
cur.executemany("INSERT INTO usersdb(id, username, email, phone, website, regdate) VALUES (?,?,?,?,?,?)", userList)
```

Out[119…
```
<sqlite3.Cursor at 0x7ff9d84d3810>
```

In [120…
```
# DB 접속을 더이상 하지 않는다면, 접속 해제, 자원 반환하기
cur.close()
```

커밋, 롤백

conn = sqlite3.connect("database.db")

- conn.commit() 명령을 실행해야함.
- 취소하고 싶을 경우, conn.rollback() 바로 이전에 실행 취소함.

conn = sqlite3.connect("database.db", isolation_level=None)

- isolation_level = None 일 경우 자동(오토 커밋)

In [ ]:

1-5-2. Read(select)

다양한 데이터 조회

- select
- where
- where 조건의 tuple, dict Mapping

In [121…

```
import sqlite3
import datetime

# 삽입 날자 샛업
now = datetime.datetime.now()
print('now : ', now)
nowDatetime = now.strftime('%Y-%m-%d %H:%M:%S')
print('nowDatetime : ', nowDatetime)

# auto commit(그때그때 DB에 반영), # rollback (되돌림))
conn = sqlite3.connect("sqlite_db.db")   # 로컬 db
cur = conn.cursor()  # cursor binding
```

```
now :  2021-10-24 18:49:52.847791
nowDatetime :  2021-10-24 18:49:52
```

In [122…

```
cur.execute("SELECT * FROM usersdb")
```

Out[122…

```
<sqlite3.Cursor at 0x7ff9d84cd3b0>
```

In [123…

```
#커서위 위치 변경
#1개  로우 선택
print('one -> \n', cur.fetchone())
```

```
one ->
 (1, 'kim', 'kim@naver.com', '010-3456-4567', 'kim.com', '2021-10-24 18:49:41')
```

In [124…

```
print('one -> \n', cur.fetchone())
```

```
one ->
 (2, 'Park', 'park@naver.com', '010-3456-4567', 'park.com', '2021-10-24 18:49:41')
```

In [125…

```
# 현재에서 3개 가져오기
print('Three -> \n', cur.fetchmany(size=3))
```

```
Three ->
 [(3, 'Lee', 'lee@naver.com', '010-3333-3333', 'lee.com', '2021-10-24 18:49:41'), (4, 'Cho', 'cho@naver.com', '010-4444-4444', 'cho.com', '20
21-10-24 18:49:41'), (5, 'Yue', 'yue@naver.com', '010-5555-5555', 'yue.com', '2021-10-24 18:49:41')]
```

In [126…

```
# 현재 cur가 있는 위치 이후,남은 것 모두
print('All -> \n', cur.fetchall())
```

```
All ->
 [(6, 'Sea', 'sea@naver.com', '010-6666-6666', 'sea.com', '2021-10-24 18:49:41')]
```

select ~ 순회(retriveve) 방법

DB 테이블을 돌며서 데이터를 읽어오는 것

In [127…

```
cur.execute("SELECT * FROM usersdb")
```

Out[127…

```
<sqlite3.Cursor at 0x7ff9d84cd3b0>
```

In [128…

```
# 순회 1:
rows = cur.fetchall()
for row in rows:
    print('retrivel -> ', row)
```

```
retrivel ->  (1, 'kim', 'kim@naver.com', '010-3456-4567', 'kim.com', '2021-10-24 18:49:41')
retrivel ->  (2, 'Park', 'park@naver.com', '010-3456-4567', 'park.com', '2021-10-24 18:49:41')
retrivel ->  (3, 'Lee', 'lee@naver.com', '010-3333-3333', 'lee.com', '2021-10-24 18:49:41')
retrivel ->  (4, 'Cho', 'cho@naver.com', '010-4444-4444', 'cho.com', '2021-10-24 18:49:41')
retrivel ->  (5, 'Yue', 'yue@naver.com', '010-5555-5555', 'yue.com', '2021-10-24 18:49:41')
retrivel ->  (6, 'Sea', 'sea@naver.com', '010-6666-6666', 'sea.com', '2021-10-24 18:49:41')
```

In [129…

```
# 순회2 : 커서가 제일 끝에 있기 때문에 실행을 보려면 select를 다시 해야함.
for row in cur.fetchall():
```

```
    print('retrivel -> ', row)
```

In [130…

```
# 순회3
for row in cur.execute('SELECT * FROM usersdb ORDER BY id DESC'):
    print('retrivel -> ', row)
```

```
retrivel ->  (6, 'Sea', 'sea@naver.com', '010-6666-6666', 'sea.com', '2021-10-24 18:49:41')
retrivel ->  (5, 'Yue', 'yue@naver.com', '010-5555-5555', 'yue.com', '2021-10-24 18:49:41')
retrivel ->  (4, 'Cho', 'cho@naver.com', '010-4444-4444', 'cho.com', '2021-10-24 18:49:41')
retrivel ->  (3, 'Lee', 'lee@naver.com', '010-3333-3333', 'lee.com', '2021-10-24 18:49:41')
retrivel ->  (2, 'Park', 'park@naver.com', '010-3456-4567', 'park.com', '2021-10-24 18:49:41')
retrivel ->  (1, 'kim', 'kim@naver.com', '010-3456-4567', 'kim.com', '2021-10-24 18:49:41')
```

select ~ where retriever(튜플형, format문, dict형)

In [131…

```
# WHERE 조건1 튜플형
param1 = (3,)
cur.execute("SELECT * FROM usersdb WHERE id=?", param1)
print('param1', cur.fetchone()) # 1개
print('param1', cur.fetchall()) # 데이터 없음
```

```
param1 (3, 'Lee', 'lee@naver.com', '010-3333-3333', 'lee.com', '2021-10-24 18:49:41')
param1 []
```

In [132…

```
#WHERE 조건 2 데이터 타입의 format 문으로
param2 = 4
cur.execute("SELECT * FROM usersdb WHERE id='%s'" % param2)  # %s %f %d
print('param2', cur.fetchone()) #1개
#print('param2', cur.fetchall()) #1개
```

```
param2 (4, 'Cho', 'cho@naver.com', '010-4444-4444', 'cho.com', '2021-10-24 18:49:41')
```

In [133…

```
# WHERE 조건3 dict 형
cur.execute("SELECT * FROM usersdb WHERE id=:id", {"id":5})  #WHERE id=:id  <= dict형 조건
print('param3', cur.fetchone()) # 1개
print('param3', cur.fetchall()) # 1개
```

```
param3 (5, 'Yue', 'yue@naver.com', '010-5555-5555', 'yue.com', '2021-10-24 18:49:41')
param3 []
```

In [134…

```
# WHERE 조건4
param4 = (3,5)
cur.execute("SELECT * FROM usersdb WHERE id IN(?,?) ", param4)
print('param4 : ', cur.fetchall())
```

```
param4 :  [(3, 'Lee', 'lee@naver.com', '010-3333-3333', 'lee.com', '2021-10-24 18:49:41'), (5, 'Yue', 'yue@naver.com', '010-5555-5555', 'yue.com', '2021-10-24 18:49:41')]
```

In [135…

```
# WHERE 조건5
cur.execute("SELECT * FROM usersdb WHERE id IN('%d','%d') " % (3,4))
print('param5 : ', cur.fetchall())
```

```
param5 :  [(3, 'Lee', 'lee@naver.com', '010-3333-3333', 'lee.com', '2021-10-24 18:49:41'), (4, 'Cho', 'cho@naver.com', '010-4444-4444', 'cho.com', '2021-10-24 18:49:41')]
```

In [136…

```
# WHERE 조건6 WHERE OR
 #WHERE id=:id  <= dict형 조건
cur.execute("SELECT * FROM usersdb WHERE id=:id1 OR id=:id2", {"id1":2, "id2":5})
print('param6', cur.fetchall()) # 1개
```

```
param6 [(2, 'Park', 'park@naver.com', '010-3456-4567', 'park.com', '2021-10-24 18:49:41'), (5, 'Yue', 'yue@naver.com', '010-5555-5555', 'yue.com', '2021-10-24 18:49:41')]
```

Dump 출력

- dump : db를 백업받는 것, 다른 시스템으로 db를 재구성할때 사용.

In [137…

```
with conn: # db 연결, file로 저장
    with open('./dump.sql', 'w') as f:
        for line in conn.iterdump():
            f.write('%s\n' % line)
        print('Dump print Complete')
```

```
Dump print Complete
```

In [140…

```
# db 연결 종료
conn.close()
```

1-5-3. DB Update

DB 테이블 내용을 수정하는 것

In [2]:
```
conn = sqlite3.connect("sqlite_db.db")
cur = conn.cursor()
```

In [3]:
```
# kim -> joy롤 바꾸기, 튜플
cur.execute("UPDATE usersdb SET username = ? WHERE id =?", ('joy', 1))
```

Out[3]:
```
<sqlite3.Cursor at 0x7f9b9024c1f0>
```

In [4]:
```
#2 park -> 2 joy 로 바꾸기 , dict
cur.execute("UPDATE usersdb SET username = :name WHERE id = :id", {'name' : 'good', 'id' : 2})
```

Out[4]:
```
<sqlite3.Cursor at 0x7f9b9024c1f0>
```

In [5]:
```
conn.commit()
```

1-5-4 DB Delete

DB 테이블 데이터 삭제하기

In [1]:
```
import sqlite3
import datetime
now = datetime.datetime.now()
print('now : ', now)
nowDatetime = now.strftime('%Y-%m-%d %H:%M:%S')
print('nowDatetime : ', nowDatetime)

conn = sqlite3.connect("sqlite_db.db")
cur = conn.cursor()
```

```
now :  2021-10-24 19:14:33.502625
nowDatetime :  2021-10-24 19:14:33
```

In [6]:
```
# 테이블 데이터 모두 삭제
#cur.execute("DELETE FROM users")
```

```
# 삭제 한것 카운트 해서 반환함 : cur.execute("DELETE FROM usersdb").rowcount
print("user db delete : ", cur.execute("DELETE FROM usersdb").rowcount)
```

```
user db delete :  5
```

In [8]:
```
# 데이터 insert many 형식(튜플, 리스트)
userList = (
    (1, 'kim', 'kim@naver.com', '010-3456-4567', 'kim.com', nowDatetime),
    (2, 'Park', 'park@naver.com', '010-3456-4567', 'park.com', nowDatetime),
    (3, 'Lee', 'lee@naver.com', '010-3333-3333', 'lee.com', nowDatetime),
    (4, 'Cho', 'cho@naver.com', '010-4444-4444', 'cho.com', nowDatetime),
    (5, 'Yue', 'yue@naver.com', '010-5555-5555', 'yue.com', nowDatetime),
    (6, 'Sea', 'sea@naver.com', '010-6666-6666', 'sea.com', nowDatetime),
)
cur.executemany("INSERT INTO usersdb(id, username, email, phone, website, regdate) VALUES (?,?,?,?,?,?)", userList)
```

Out[8]:
```
<sqlite3.Cursor at 0x7f993032af80>
```

In [4]:
```
# 삭제 : 튜플형으로
cur.execute("DELETE FROM usersdb WHERE id = ? ", (2,))
```

Out[4]:
```
<sqlite3.Cursor at 0x7f993032af80>
```

In [11]:
```
# 확인하기
for user in cur.execute("SELECT * FROM usersdb"):
    print(user)
```

```
(1, 'kim', 'kim@naver.com', '010-3456-4567', 'kim.com', '2021-10-24 19:05:12')
(2, 'Park', 'park@naver.com', '010-3456-4567', 'park.com', '2021-10-24 19:05:12')
(3, 'Lee', 'lee@naver.com', '010-3333-3333', 'lee.com', '2021-10-24 19:05:12')
(5, 'Yue', 'yue@naver.com', '010-5555-5555', 'yue.com', '2021-10-24 19:05:12')
(6, 'Sea', 'sea@naver.com', '010-6666-6666', 'sea.com', '2021-10-24 19:05:12')
```

In [10]:
```
# 삭제 : dict 형으로
cur.execute("DELETE FROM usersdb WHERE id =:id", {"id" : 4})
```

Out[10]:
```
<sqlite3.Cursor at 0x7f993032af80>
```

In [12]:

```
# 확인하기
for user in cur.execute("SELECT * FROM usersdb"):
    print(user)
```

```
(1, 'kim', 'kim@naver.com', '010-3456-4567', 'kim.com', '2021-10-24 19:05:12')
(2, 'Park', 'park@naver.com', '010-3456-4567', 'park.com', '2021-10-24 19:05:12')
(3, 'Lee', 'lee@naver.com', '010-3333-3333', 'lee.com', '2021-10-24 19:05:12')
(5, 'Yue', 'yue@naver.com', '010-5555-5555', 'yue.com', '2021-10-24 19:05:12')
(6, 'Sea', 'sea@naver.com', '010-6666-6666', 'sea.com', '2021-10-24 19:05:12')
```

In [143…

```
# 삭제 : 스트링 format형으로
cur.execute("DELETE FROM users WHERE id = %d" % 1)
```

Out[143…

```
<sqlite3.Cursor at 0x7f0fb08a6ab0>
```

In [144…

```
# 확인하기
for user in cur.execute("SELECT * FROM users"):
    print(user)
```

```
(3, 'Lee', 'lee@naver.com', '010-3333-3333', 'lee.com', '2021-09-21 17:30:00')
(5, 'Yue', 'yue@naver.com', '010-5555-5555', 'yue.com', '2021-09-21 17:30:00')
(6, 'Sea', 'sea@naver.com', '010-6666-6666', 'sea.com', '2021-09-21 17:30:00')
```

In [150…

```
# 삭제 : WHERE 조건으로
cur.execute("DELETE FROM users WHERE id=:id AND username=:name", {"id":3, "name":"Lee"})  #WHERE id=:Id  <= dict형 조건
```

Out[150…

```
<sqlite3.Cursor at 0x7f0fb08a6ab0>
```

In [151…

```
# 확인하기
for user in cur.execute("SELECT * FROM users"):
    print(user)
```

```
(5, 'Yue', 'yue@naver.com', '010-5555-5555', 'yue.com', '2021-09-21 17:30:00')
(6, 'Sea', 'sea@naver.com', '010-6666-6666', 'sea.com', '2021-09-21 17:30:00')
```

In [14]:

```
# 데이터 전체 삭제 : 삭제된 행 카운트 출력
print("user db deleted : ", conn.execute("DELETE FROM usersdb").rowcount, " rows")
```

```
user db deleted :  5  rows
```

In [15]:
```
conn.commit()
conn.close()
```

DB 사용 권장 이유

최신 데이터를 통합관리를 할 수 있기 때문
요즘은 데이터가 자산인 시대
새로운 서비스 창출 할 수 있음.
데이터는 4차 산업혁명 시대에 원유와 같음.

In [ ]: