python

SQLite3 / Oracle

- SQLite3
  - SQLite DB 생성 및 Table 생성
  - 기본 DML 실습
  - 트랜잭션
- Oracle
  - 모듈 설치
  - 기본 DML 실습

❑ SQLite 특징 및 용도

- SQLite는 세계에서 많이 배포된 데이터베이스 엔진으로 스마트 폰, 임베디드 디바이스, 사물 인터넷, 데이터 분석, 작은 규모의 웹사이트사용하기에 적합하다.
  - 디스크 또는 메모리 기반의 경량 데이터베이스 엔진 라이브러리
  - 임베디드 SQL DB엔진 기반으로 특별한 서버가 필요하지 않음
  - 설치 과정이 없고, 설정 파일도 존재하지 않는다.
  - 테이블, 인덱스, 트리거, 뷰 등 모든 객체가 디스크 상에 단 하나의 파일로 존재한다.
  - 개인적 또는 상업적 목적으로 사용할 수 있다.
  - 모바일 디바이스에 기본적으로 설치되어 있음
  - 파이썬에 기본적으로 포함되어 있음

- 상용 DBMS 활용이전에 SQLite를 사용하여 응용 프로그램의 prototype 작성
- 사이트 : https://www.sqlite.org/

❑ 콘솔에서 실습

- Python 이 설치되어 있다면 바로 실습이 가능하다.
- SQLite DataType 관련 : https://www.sqlite.org/datatype3.html
  - NULL, INTEGER, REAL, TEXT, BLOB 만 존재
  - 다양한 별칭 존재 (TEXT: VARCHAR, CHARACTER, NCHAR, CLOB, ...)

```
-- sqlite3에서 데이터베이스 작성
-- 저장되는 확장자는 임으로 해도 상관없으나 보통  .db .sqlite .sqlite3 .db3 로 한다.
$ sqlite3 mydb.db3

-- .help 를 통하여 사용하는 명령어 이해 ( 앞에 dot(.)이 있다.)
-- .database, .table,.exit 등
sqlite> .help

-- 테이블 생성
sqlite> CREATE TABLE member (
   ...>   mem_id VARCHAR(30),
   ...>   mem_age NUMBER);


--  데이터 입력
sqlite> INSERT INTO member ( mem_id, mem_age ) VALUES ( 'malja, 28 );

-- 데이터 조회
sqlite> SELECT * FROM member WHERE mem_id = 'malja';
```

❑ Window 에서 SQLite 설정

- 다운로드에서 : dll, tools 가 존재한다.
- dll 은 Windos의 system32 폴더에 복사하면 된다.
- tools 는 SQLiite에 연결하의 작업을 하는 콘솔 클라이언트이다
- GUI 클라이언트 프로그램을 사용할 것이므로 tools는 필요없다.

❑ DB Browser for SQLite 설치

- 사이트 : http://sqlitebrowser.org/
- 사이트에서 사용자 OS에 따라서 설치한다.
- Ubuntu 에서 설치

```
sudo apt-get update
sudo apt-get install sqlitebrowser
```

❑ 파이썬에서 SQLite3 사용하기

- Python SQLite3 라이브러리 Document(한글) : https://docs.python.org/ko/3/library/sqlite3.html

❑ 일반적인 DB 처리 순서

- 첫번째 Connection 구하기
- 두번째 Cursor 객체를 생성
- 세번째 Cursor 객체의 execute() 메서드를 호출하여 SQL 명령을 수행
- 마지막으로 Connection 닫기

❑ sqlite3 모듈함수

함수	설명
sqlite3.**version()**	이 모듈의 버전 번호(문자열). SQLite 라이브러리의 버전이 아닙니다.
sqlite3.**sqlite_version()**	SQLite 라이브러리의 버전 번호(문자열)
sqlite3.**connect** (database [, timeout, isolation_level, ...])	database에 대한 연결 및 Connection 객체를 반환
sqlite3.**complete_statement(sql)**	완전한 SQL문이면 True를 반환 SQL문이 문법적으로 올바른지 확인하지는 않음

❑ sqlite3.Connection 클래스

속성 및 메서드	설 명
Connection.isolation_level	트랜잭션의 격리 수준을 가져오거나 설정 자동 커밋 모드를 뜻하는 None 이나 "DEFERRED", "IMMEDIATE" 또는 "EXCLUSIVE" 중 하나
Connection.in_transaction	트랜잭션이 활성화 상태 리턴 (커밋되지 않은 변경 사항이 있으면) True, 그렇지 않으면 False
Connection.**cursor()**	Cursor 객체 생성
Connection.**commit()**	현재 트랜잭션의 변경내역을 DB에 반영
Connection.**rollback()**	가장 최근의 commit() 이후 지금까지 작업한 내용에 대해서 되돌림
Connection.**close()**	DB 연결을 종료
Connection.execute(sql[, parameters])	비표준, Cursor 객체의 메서드 참조
Connection.executemany(sql[, parameters])	비표준, Cursor 객체의 메서드 참조
Connection.executescript(sql_script)	비표준, Cursor 객체의 메서드 참조

❑ sqlite3.Cursor 클래스 속성 및 메서드

메서드	설 명
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 데이터(튜플)를 리스트로 반환
Cursor.**fetchall()**	조회된 결과 모두를 리스트 형태로 반환
Cursor.close()	커서를 닫음
Cursor.**description**	조회된 결과의 열 이름, 데이터 형식 등의 메타 정보를 리스트로 반환

❑ sqlite3 모듈 및 DB 생성

- 파이썬은 기본적으로 SQLite를 사용하기 위한 sqlite3모듈이 내부모듈로 설치 되어 있음
- 메모리 DB 접속 (일회성, **:memory:**)

```
1. import sqlite3
2. conn = sqlite3.connect(':memory:')
```

- 파일 DB 접속 (현재 경로에 example.db 생성)

```
1. import sqlite3
2. conn = sqlite3.connect('example.db')
```

❑ SQLite 기본정보 확인

- 버전 및 SQLite 버전 확인

```
1. import sqlite3
2. conn = sqlite3.connect('example.db')
3. print(sqlite3.version)
4. print(sqlite3.sqlite_version)
5. conn.close()
```

- 연습문제
  - 본인의 home 디렉토리에 dev 폴더가 있다고 가정
  - dev 하위에 example.db 생성
    - os 라이브러리 environ
    - for key in os.environ 으로 확인 및 get 메서드 활용

❑ Table 생성하기

- Cursor객체를 받아서 execute 메서드로 SQL 구문 구문 실행

```
1. import sqlite3
2. conn = sqlite3.connect('example.db')
3. cur = conn.cursor()
4. cur.execute('''CREATE TABLE stocks
5.                (date text, trans text, symbol text, qty real, price real)''')
6. conn.close()
```

❑ 데이터 삽입

- 문자열을 사용하여 Query 호출

```
1. cur = con.cursor()
2. cur.execute("INSERT INTO stocks VALUES ('2006-01-05','BUY','RHAT',100,35.14)")
```

- Parameter: Tuple 사용

```
1. date = '2006-01-05'
2. trans = 'BUY'
3. symbol = 'RHAT'
4. qty = 100
5. price = 35.14
6. cur = conn.cursor()
7. cur.execute('INSERT INTO stocks VALUES( ?, ?, ?, ?, ? )', (date, trans, symbol, qty, price))
```

- Named Parameter 사용(: 사용)

```
1. date = '2006-01-05'
2. trans = 'BUY'
3. symbol = 'RHAT'
4. qty = 100
5. price = 35.14
6. in_data = {"date" : date, "trans" : trans, "symbol" : symbol , "qty" : qty , "price" : price}
7. cur = conn.cursor()
8. cur.execute('INSERT INTO stocks VALUES( :date, :trans, :symbol, :qty, :price)', in_data )
```

- List 사용 (여러건의 튜플 등록, executemany 메서드 실행)

```
1. purchases = [('2006-03-28', 'BUY', 'IBM', 1000, 45.00),
2.                ('2006-04-05', 'BUY', 'MSFT', 1000, 72.00),
3.                ('2006-04-06', 'SELL', 'IBM', 500, 53.00),
4.              ]
5. cur = conn.cursor()
6. cur.executemany("INSERT INTO stocks VALUES( ?, ?, ?, ?, ? );", purchases )
```

❑ 데이터 조회

- 커서 객체는 파일 포인터처럼 한 번 읽은 데이터는 다시 읽을 수 없으므로 다시 SELECT 쿼리를 전송해야 한다.

- 커서에서 바로 레코드 조회

```
1. cur.execute('SELECT * FROM stocks')
2. for row in cur:
3.       print(row)
```

❑ Fetch를 사용하여 조회

- 단건 조회

```
1. cur.execute('SELECT * FROM stocks')
2. row = cur.fetchone()
3. print(row)
```

- 다건 조회

```
1. rows = cur.fetchmany(2)
```

- 모두 조회

```
1. rows = cur.fetchall()
```

❑ try와 with 문의 사용

- Connection, Cursor 와 같은 리소스들이 Leak 되는 것을 방지하기 위하여 try...finally 혹은 with 문을 사용할 수 있다. try...finally와 with 문은 블럭 내 에러가 발생하더라도 마지막에 리소스를 해제하는 역활을 하기 때문에, 데이타베이스 코딩에서 자주 사용된다.

- 아래 예제는 Connection 리소스를 닫기 위해 with 문을 사용하는 예이다.

```
1. import sqlite3
2.
3. conn = sqlite3.connect("test.db")
4.
5. with conn:
6.     cur = conn.cursor()
7.     cur.execute("select * from customer")
8.     rows = cur.fetchall()
9.
10.    for row in rows:
11.        print(row)
```

❑ 쿼리의 Meta 정보 조회

- 쿼리(SELECT)를 실행 한 후 결과 객체에 대한 커서의 description 사용하여 열 이름 및 데이터 형식과 같은 메타 정보 튜플을 리스트로 구한다.

```
import sqlite3

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

cur = conn.cursor()
cur.execute("SELECT * FROM stocks ORDER BY price DESC LIMIT 0, 5")

# 실행 이후 메타정보를 구한다.
for col in cur.description:
    print(col)

print("-" * 50)

rows = cur.fetchall()
for row in rows:
    print(row)

cur.close()
conn.close()
```

❑ Oracle 연동 라이브러리 설치

- Python에서 Oracle DBMS에 접속하기 위한 라이브러리 설치
  - cx_Oracle 사이트 : http://cx-oracle.sourceforge.net/
  - cx_Oracle 도큐먼트 : https://cx-oracle.readthedocs.io/en/latest/index.html
  - Oracle Instant Client 다운로드 :
    https://www.oracle.com/database/technologies/instant-client/linux-x86-64-downloads.html
  - 설치 (터미널 또는 가상환경 터미널에서 실행)
    1. pip install cx_Oracle

- DB 연결 및 확인

```
1. import cx_Oracle
2. conn = cx_Oracle.connect('java', 'oracle', 'localhost:1521/XE')

3. # 다른 방법 1
4. # conn = cx_Oracle.connect('java/oracle@localhost:1521/XE')
5. # 다른 방법 2
6. # dsn_tns = cx_Oracle.makedsn('localhost', 1521, 'XE')
7. # conn = cx_Oracle.connect('java', 'oracle', dsn_tns)

8. print(conn.version)
9. conn.close()
```

❑ cx_Oracle 연동 관련

- sqlite3 모듈을 사용해 보았으므로 기본적인 작업은 거의 유사하므로 관련 메서드에 대한 가이드는 하지 않음
  - execute, executemany, fetchone, fetch

❑ 주의할 사항

- 한글이 깨질 때 아래 방법 중 선택하여 처리
  1. os.environ["NLS_LANG"] = ".AL32UTF8"
  2. connect('java', 'oracle', 'localhost:1521/XE', encoding="utf-8")
- 입력파라미터(또는 placeholder 중 sqlite3에서는 ? , :key 를 사용했지만
- cx_Oracle 작업시 :key , :seq 만 사용 (:name 또는 :1, :2)
- 연결테스트 오류시
  - Oracle Instant Client Zip Basic Light 파일 64bit 용 다운
  - sudo unzip instantclient-basiclite-linux.x64-19.6.0.0.0dbru.zip
  - sudo mkdir -p /opt/oracle
  - sudo mv instantclient_19_6/ /opt/oracle/

  - .bashrc 환경변수 추가
  - export LD_LIBRARY_PATH=/opt/oracle/instantclient_19_6:$LD_LIBRARY_PATH

  - 공유 라이브러리 등록 및 캐시 재설정
  - sudo sh -c "echo /opt/oracle/instantclient_19_6 > /etc/ld.so.conf.d/oracle-instantclient.conf"
  - sudo ldconfig

- cursor 의 주요 메서드
- **Execute**
  - cx_Oracle.Cursor.execute(statement, [parameters], **keyword_parameters)
  - 이 메소드는 데이터베이스에 대해 직접 실행될 단일 인수 인 SQL 문을 허용 할 수 있습니다. 매개 변수 또는 keyword_parameters 인수를 통해 지정된 바인드 변수는 사전, 순서 또는 키워드 인수 세트로 지정할 수 있습니다. 사전 또는 키워드 인수가 제공되면 값은 이름에 바인딩됩니다. 순서가 주어지면 값은 위치에 의해 결정됩니다. 이 메서드는 쿼리 인 경우 변수 개체의 목록을 반환하고 그렇지 않은 경우 없음을 반환합니다
  - cx_Oracle.Cursor.executemany(statement, parameters)
  - 특히 대량 삽입 작업에 필요한 Oracle 실행 작업의 수를 단일 단일 작업으로 제한 할 수 있기 때문에 유용합니다. 이를 사용하는 방법에 대한 자세한 내용은 아래의 "한꺼번에"섹션을 참조하십시오

- Fetch (optional) 쿼리 (select 문)문에서 사용가능
- cx_Oracle.Cursor.fetchall() 결과 집합의 나머지 행을 모두 튜플 목록으로 가져옵니다

- cx_Oracle.Cursor.fetchmany([rows_no]) 데이터베이스에서 다음 rows_no 행을 가져옵니다.
- cx_Oracle.Cursor.fetchone() 데이터베이스에서 단일 튜플을 가져 오거나 더 이상 사용할 수있는 행이 없으면 아무 것도 가져 오지 않습니다.

- 바인드 변수를 이름으로 전달하려면 execute 메소드의 parameters 인수가 사전 또는 키워드 인수 세트 여야합니다. 아래의 query1과 query2는 동일합니다.
- named_params = {'dept_id':50, 'sal':1000}
- query1 = cursor.execute('SELECT * FROM employees WHERE department_id=:dept_id AND salary>:sal', named_params)
- query2 = cursor.execute('SELECT * FROM employees WHERE department_id=:dept_id AND salary>:sal', dept_id=50, sal=1000)

❑ 객체를 사용하여 관리

- 데이터에 대한 처리를 클래스로 구성하여 처리
  - member.py

```
manager = DBManager()
conn = manager.get_connection()

memberDao = MemberDao()

mem =  memberDao.get_member(conn, "b001")
print(mem.mem_id, mem.mem_name, mem.mem_pass)

memList = memberDao.get_member_list(conn, "김")
for mem in memList:
    print(mem.mem_id, mem.mem_name, mem.mem_pass)

manager.db_free()
```