학습 내용

2부. 프로그래밍 언어 활용

- 10장. N차원 배열 다루기
- 11장. 데이터프레임과 시리즈
- 12장. 데이터 시각화
- 13장. 총괄 예제
- 14장. 웹 데이터 수집
- **15장. 데이터베이스 연동**
  - **1. SQLite 데이터베이스 연결**
  - **2. 오라클 데이터베이스 연결**

1.1. SQLite와 파이썬

1절. SQLite 데이터베이스 연결

- SQL 쿼리 언어의 비표준 변형을 사용하여 데이터베이스에 액세스 할 수 있는 디스크 기반 데이터베이스를 제공하는 C 라이브러리
- SQLite를 사용하여 응용 프로그램의 프로토타입을 만든 후 Oracle 또는 PostgreSQL등 더 큰 데이터베이스로 코드를 이식 할 수 있음
- sqlite3 모듈
  - 파이썬에서 데이터베이스에 연결하기 위한 모듈
  - PEP 249에 설명된 SQLite 데이터베이스를 위한 DB-API 2.0 사양을 준수하는 SQL 인터페이스를 제공함

```
1  import sqlite3
2  sqlite3.sqlite_version
```

```
'3.21.0'
```

1.2. 데이터베이스 연결

1절. SQLite 데이터베이스 연결

- 데이터베이스 연결 객체 Connection을 생성

```
sqlite3.connect( database [, timeout,
                 detect_types, isolation_level,
                 check_same_thread, factory,
                 cached_statements, uri ] )
```

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

```
1  print(conn)
```

```
<sqlite3.Connection object at 0x0000000051813B0>
```

1.3. SQLite API

- 모듈은 PEP 249에 설명된 SQLite 데이터베이스를 위한 DB-API 2.0 사양을 준수하는 SQL 인터페이스를 제공
- sqlite3.Connection
- sqlite3.Cursor
- sqlite3 예외 클래스

1) sqlite3.Connection

- 데이터베이스 연결 객체이며 데이터베이스마다 객체를 만드는 방법은 다름

속성/메서드	설명
cursor( )	커서 객체를 생성
commit( )	현재 트랜잭션을 커밋(변경사항 적용). 이 메서드를 호출하지 않으면 마지막 호출 이후 수행 한 작업이 다른 데이터베이스 연결에서 볼 수 없음.
rollback( )	마지막 호출 이후 데이터베이스에 대한 모든 변경 사항을 롤백(취소)
close( )	데이터베이스 연결이 닫힙니다. 이 작업은 자동으로 호출되지 않음. commit()을 먼저 호출하지 않고 데이터베이스 연결을 닫으면 변경 사항이 손실됨.
execute(*sql*[, *parameters*])	cursor() 메서드 를 호출하여 커서 객체를 생성하고 주어진 매개 변수로 커서의 execute() 메서드를 호출 한 다음 커서를 반환함. Connection의 execute(), executemany(), executescript() 메서드는 비표준임.

2) sqlite3.Cursor

- 데이터베이스 질의문을 실행하고 결과를 가져오기 위한 객체

속성/메서드	설명
execute(*sql*[, *parameters*])	SQL 문을 실행. SQL 문을 매개변수화 할 수 있음 - qmark 스타일 : `cur.execute("insert into people values (?, ?)", (who,age))` - named 스타일 : `cur.execute("select * from people where name_last=:who and age=:age", {"who":who,"age":age})`
executemany(*sql*, *seq_of_parameters*)	시퀀스 seq_of_parameters 에 있는 모든 매개 변수 시퀀스 또는 매핑에 대해 SQL 명령을 실행.
executescript(*sql_script*)	한 번에 여러 SQL 문을 실행하기 위한 비표준 메서드. 각 SQL 스크립트는 라인의 끝이 세미콜론(;)으로 끝남.
fetchone( )	쿼리 결과 집합의 다음 행을 가져 오거나, 단일 시퀀스를 반환하거나, 더 이상 데이터를 사용할 수 없는 경우 None을 가져옴
fetchmany(*size=cursor.arraysize*)	질의 결과의 다음 행 집합을 가져 와서 목록을 반환함. 행이 더 이상 사용 가능하지 않으면 빈 목록이 리턴. 호출 당 패치(fetch)할 행의 수는 size 매개 변수로 지정. 지정되어 있지 않은 경우, 커서의 배열 크기가 가져올 행의 수를 결정. 이 메서드는 size 매개 변수가 나타내는 수만큼의 행을 가져옴. 지정된 행의 수를 사용할 수 없어서 이것이 가능하지 않으면, 더 적은 행이 리턴될 수 있음. 최적의 성능을 위해서는 arraysize 속성을 사용하는 것이 가장 좋음
fetchall( )	질의 결과의 모든(나머지) 행을 가져와 목록을 반환. 커서의 arraysize 속성은 이 작업의 성능에 영향을 줄 수 있음. 사용할 수 있는 행이 없으면 빈 목록이 반환됨.
close( )	커서를 닫음(__del__() 메서드가 호출될 때는 커서가 닫히지 않음). 이 시점부터는 커서를 사용할 수 없음. 이후 커서에 어떤 작업이 시도되면 ProgrammingError 예외가 발생.
rowcount	sqlite3 모듈의 Cursor 클래스가 이 속성을 구현 하더라도 데이터베이스 엔진 자체의 지원은 "영향을 받는 행"/"선택된 행"으로 결정. 예를 들어 executemany() 문장에서 수정된 행의 수는 rowcount에 합해 짐. 파이썬 DB API 스펙에서 요구하는 대로, rowcount 속성은 executeXX() 커서에서 아무 것도 수행 되지 않았거나 마지막 작업의 행 개수가 인터페이스에 의해 결정될 수 없는 경우 -1임

4) sqlite3 예외 클래스

- 예외 발생 시 발생하는 예외 별로 처리를 할 수 있음

예외 클래스	설명
sqlite3.Warning	StandardError의 하위 클래스.
sqlite3.Error	이 모듈의 다른 예외의 기본 클래스. StandardError의 하위 클래스.
sqlite3.DatabaseError	데이터베이스와 관련된 오류에 대해 발생하는 예외. Error의 하위 클래스.
sqlite3.IntegrityError	데이터베이스의 관계형 무결성이 영향을 받을 때 예외가 발생(예: 외래키 검사 실패). DatabaseError의 하위 클래스.
sqlite3.ProgrammingError	프로그래밍 오류에 대한 예외(예 : 테이블을 찾을 수 없거나 이미 존재 함). SQL 문에 구문 오류가 있으며, 지정된 매개 변수 수가 잘못 되었을 경우 발생. DatabaseError의 하위 클래스.
sqlite3.OperationalError	예기치 않은 단절이 발생하거나, 데이터 소스 이름을 찾을 수 없거나, 트랜잭션을 처리 할 수 없는 등의 이유로 데이터베이스 작동과 관련된 프로그래머가 제어하지 않는 오류에 대해 예외가 발생. DatabaseError의 하위 클래스.
sqlite3.NotSupportedError	rollback() 트랜잭션을 지원하지 않거나 트랜잭션이 꺼져있는 연결에서 메서드를 호출하는 경우와 같이 데이터베이스에서 지원하지 않는 메서드 또는 데이터베이스 API가 사용 된 경우 예외가 발생. DatabaseError의 하위 클래스.

1.4. SQLite 데이터베이스에 데이터 입력하기

- CRUD
  - Create, Read, Update, Delete
  - 데이터베이스에 데이터를 입력, 조회, 수정, 삭제하는 것

- 데이터를 데이터베이스에 넣기 위해서는 다음 과정을 따라야 함
  1. **sqlite3.connect() 함수를 이용해서 데이터베이스 연결(Connection) 객체를 생성.**
  2. **Cursor 객체 생성**
  3. **Cursor 객체의 execute() 메서드를 이용하여 SQL 구문 실행**
  4. **Connection 객체의 commit() 메서드를 이용하여 변경된 내용을 데이터베이스에 반영(커밋; Commit)하거나 변경된 내용을 취소(롤백; Rollback)**
  5. **데이터베이스 연결 닫기**

1.4. SQLite 데이터베이스에 데이터 입력하기

1절. SQLite 데이터베이스 연결

```
conn = sqlite3.connect("contact.db")
```

```
cursor = conn.cursor()
```

```
try :
    cursor.execute("DROP TABLE contact")
except:
    print("삭제할 테이블이 존재하지 않습니다.")
```

```
cursor.execute("""CREATE TABLE contact(
    name text,
    age int,
    email text)
""")
```

데이터베이스에 데이터를 저장할 테이블 생성

```
<sqlite3.Cursor at 0x5c32260>
```

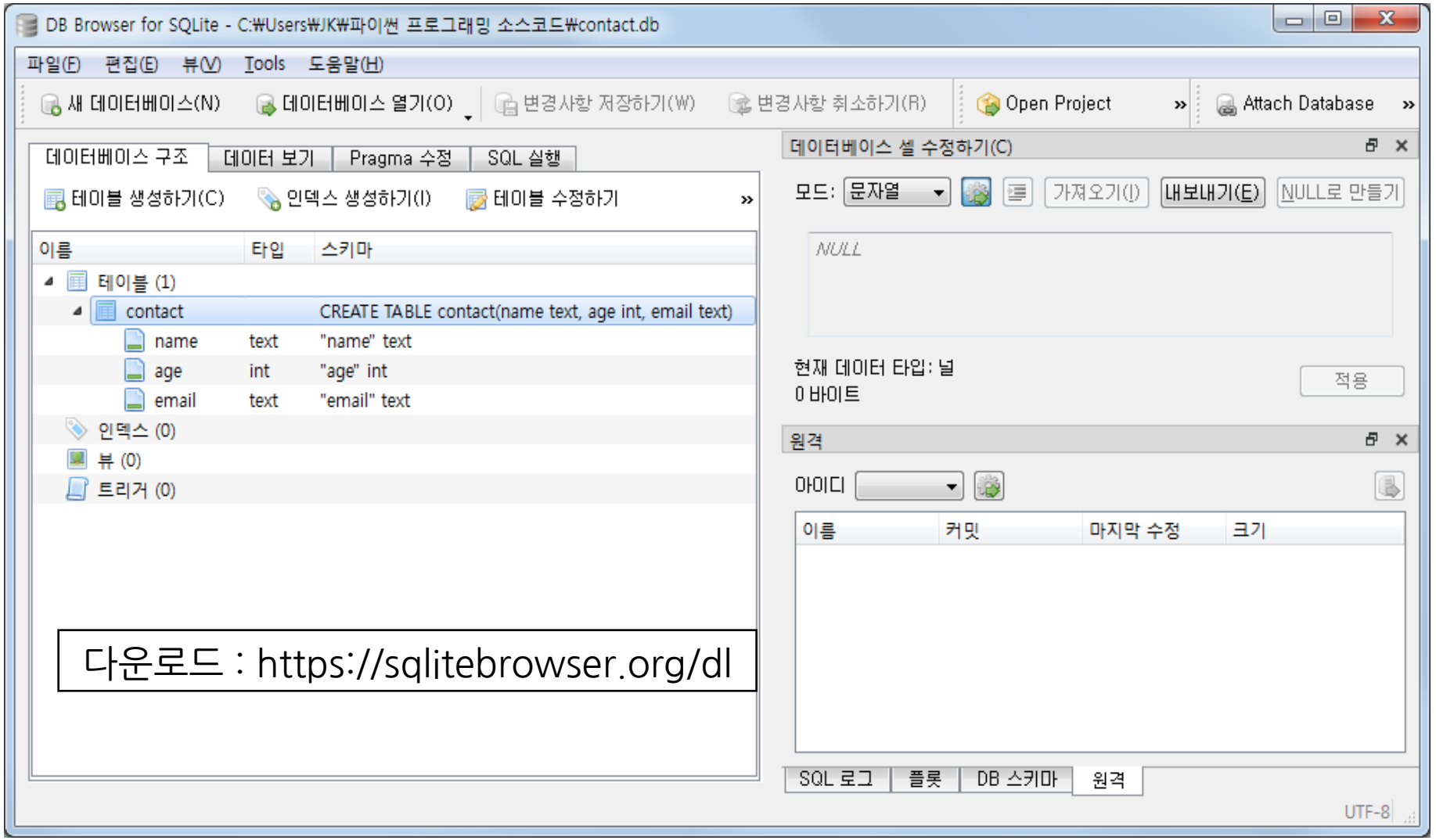
테이블에 데이터 입력

```
cursor.execute("INSERT INTO contact VALUES('kim', 30, 'kim@naver.com')")
cursor.execute("INSERT INTO contact VALUES('lee', 35, 'lee@daum.net')")
cursor.execute("INSERT INTO contact VALUES('park', 40, 'heo@coderby.com')")
```

```
<sqlite3.Cursor at 0x5c32260>
```

DB Browser for SQLite

1절. SQLite 데이터베이스 연결



1.5. SQLite 데이터베이스에서 데이터 조회하기

- 데이터베이스에서 데이터 읽기 절차
  - sqlite3.connect 함수를 이용해서 데이터베이스 연결(Connection) 객체 생성
  - Connection객체를 이용해 커서 객체 생성
  - 커서 객체의 execute() 메서드를 이용해 SELECT 구문을 실행
  - 커서 객체의 fetchone(), fetchall() 등의 메서드와 반복문을 이용해 데이터 처리
    - 커서로부터 조회(fetch)된 데이터는 다시 조회하려면 SQL 구문을 다시 실행시켜야 함
  - 데이터베이스 연결 닫기

```
1  conn = sqlite3.connect("contact.db")
```

```
1  cursor = conn.cursor()
```

```
1  cursor.execute("SELECT * FROM contact")
```

```
<sqlite3.Cursor at 0x5c322d0>
```

```
1  for row in cursor:
2      print(row)
```

```
('kim', 30, 'kim@naver.com')
('lee', 35, 'lee@daum.net')
('park', 40, 'heo@coderby.com')
```

1.6. SQL 구문에 파라미터 사용하기

- qmark 스타일
  - SQL 구문의 매개변수를 포함해야 할 값에 물음표(?)로 표시한 후 튜플을 통해 물음표에 전달할 값을 지정
  - cur.execute("insert into people values (?, ?)", (who, age))

```
#qmark 스타일 파라미터 인수
cursor.execute("SELECT * FROM contact WHERE email=?",
               ("heo@coderby.com",))
cursor.fetchall()
```

```
[('park', 40, 'heo@coderby.com')]
```

- named 스타일
  - SQL 구문의 매개변수를 포함해야 할 값에 콜론(:)과 값을 받을 이름을 표시한 후 딕셔너리를 이용해 이름에 값을 전달
  - cur.execute("select * from people where name_last=:who and age=:age", {"who":who,"age":age})

```
# named 스타일 파라미터 인수
user_email = "heo@coderby.com"
cursor.execute("SELECT * FROM contact WHERE email=:email",
               {"email":user_email})
cursor.fetchall()
```

```
[('park', 40, 'heo@coderby.com')]
```

1.7. SQLite 데이터베이스에서 데이터 수정/삭제하기


```
1  conn = sqlite3.connect("contact.db")
```

```
1  conn.execute("UPDATE contact SET name='heo' WHERE email='heo@coderby.com'")
```

```
<sqlite3.Cursor at 0x4ef6d50>
```

```
1  cursor = conn.execute("SELECT * FROM contact")
2  cursor.fetchall()
```

```
[('kim', 30, 'kim@naver.com'),
 ('lee', 35, 'lee@daum.net'),
 ('heo', 40, 'heo@coderby.com')]
```

- SQL 구문을 실행할 때에는 커서 객체가 반드시 필요한 것은 아님
- Connection 객체의 execute() 함수를 이용해서 SQL 구문을 실행시킬 수 있지만 표준은 아니므로 커서를 통해 SQL 구문을 실행시킬 것을 권장

```
1  conn.rollback()
```

- INSERT, UPDATE, DELETE 구문의 경우 commit() 또는 rollback() 함수를 이용해 변경사항을 저장(commit) 또는 취소(rollback)시킬 수 있음

```
1  cursor = conn.execute("SELECT * FROM contact")
2  cursor.fetchall()
```

```
[('kim', 30, 'kim@naver.com'),
 ('lee', 35, 'lee@daum.net'),
 ('park', 40, 'heo@coderby.com')]
```

```
1  conn.execute("UPDATE contact SET name='heo' WHERE email='heo@coderby.com'")
```

```
<sqlite3.Cursor at 0x4ef6f80>
```

```
1  conn.commit()
```

2.1. cx_Oracle 패키지

- 오라클 데이터베이스에 연결하는 패키지
- 설치
  - pip install cx_Oracle
  - conda install cx_Oracle

오라클 데이터베이스 등 상용 데이터베이스에 연결하기 위해서 import 하는 패키지 또는 모듈의 이름과 connect() 함수를 이용해 Connection 객체를 생성하는 것만 다르고 나머지는 SQLite 데이터베이스에서 설명한 내용과 같음

2.2. 오라클 데이터베이스 연결

2절. 오라클 데이터베이스 연결

- cx_Oracle 모듈은 makedsn 함수와 connect 함수를 이용해 데이터베이스 서버주소, 포트번호, SID, 사용자이름, 비밀번호 등을 설정해야 함

```
cx_Oracle.makedsn(host, port, sid=None)
```

```
cx_Oracle.connect(user=None, password=None, dsn=None)
```

- 구문에서...
  - host : 데이터베이스가 설치되어 있는 컴퓨터의 주소, 자신의 컴퓨터에 오라클 데이터베이스를 설치했다면 “localhost” 사용
  - port : 데이터베이스의 포트번호, 오라클은 주로 1521번 사용
  - sid : 데이터베이스 인스턴스의 고유 이름(System ID) 입니다. 오라클 Express Edition을 설치했다면 sid는 xe
  - user : 데이터베이스 사용자 아이디
  - password : 데이터베이스에 접속하기 위한 사용자의 비밀번호

2.3. EMPLOYEES 테이블 데이터 조회하기


```
1 oracle_dsn = cx_Oracle.makedsn(host='localhost', port=1521, sid='xe')
2 conn = cx_Oracle.connect(dsn=oracle_dsn, user="scott", password="tiger")
3 cursor = conn.cursor()
```

```
1 sql = "SELECT EMPNO, ENAME, MGR FROM EMP WHERE DEPTNO=20"
2 emps = cursor.execute(sql)
```

```
1 for emp in emps:
2     print(emp)
```

```
(7369, 'SMITH', 7902)
(7566, 'JONES', 7839)
(7788, 'SCOTT', 7566)
(7876, 'ADAMS', 7788)
(7902, 'FORD', 7566)
```

[실습]EMP 테이블 데이터 조회하기

```
1 cursor.close()
2 conn.close()
```

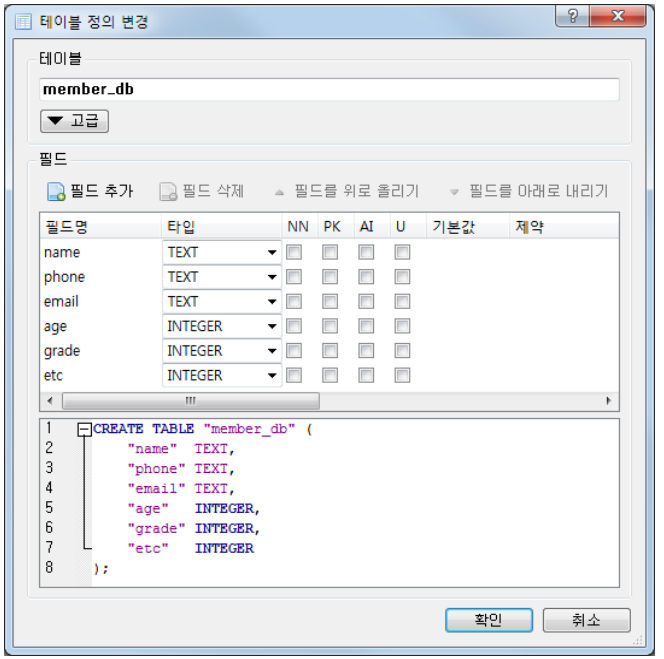
연습문제(실습형)

- 제출 파일 : 실행화면SQLite.html, data.db, ~.csv, 실행화면oracle.html

- 회원관리 애플리케이션 요구사항
  - 파이썬으로 회원관리 애플리케이션을 작성하세요.
  - 회원 정보를 저장하기 위한 Member 클래스를 정의하세요.
  - 회원 정보는 이름, 전화번호, 나이, 고객 등급(1~5), 이메일, 특징입니다.
  - 회원 정보들은 데이터베이스를 이용해 저장되어야 합니다.
  - 고객의 정보는 입력, 고객정보 전체 조회, 검색(이름으로 검색), 삭제(이메일로 삭제) 할 수 있어야 합니다.
  - 데이터는 CSV 파일로 내보내기 기능이 있어야 합니다.
- SQLite 데이터베이스 테이블 구조

```
CREATE TABLE "member" (
"name" TEXT,
"phone" TEXT,
"email" TEXT,
"age" INTEGER,
"grade" INTEGER,
"etc" TEXT
);
```

DB Browser for SQLite를 실행하고 [새 데이터베이스]를 선택한 후 파일 이름을 입력하면 쉽게 테이블을 만들 수 있습니다. 오른쪽 그림은 테이블을 생성할 때 추가한 필드 정보입니다.



연습문제(실습형)

10장. 데이터베이스 연동

```python
if __name__=='__main__':
    import sqlite3
    global conn
    conn = sqlite3.connect('data.db')
    main()
```

```
1:입력 | 2.전체조회 | 3.이름찾기 | 4.메일삭제 | 5.CSV내보내기 | 0.종료메뉴선택 : 2
입력된 회원이 없습니다
1:입력 | 2.전체조회 | 3.이름찾기 | 4.메일삭제 | 5.CSV내보내기 | 0.종료메뉴선택 : 1
이름 :홍길동
전화 :010-9999-9999
이메일 :abc@hong.com
나이 :20
고객등급(1~5) :5
기타정보 :abc
1:입력 | 2.전체조회 | 3.이름찾기 | 4.메일삭제 | 5.CSV내보내기 | 0.종료메뉴선택 : 2
('홍길동', '010-9999-9999', 'abc@hong.com', 20, 5, 'abc')
1:입력 | 2.전체조회 | 3.이름찾기 | 4.메일삭제 | 5.CSV내보내기 | 0.종료메뉴선택 : 5
파일명을 입력하세요abc
1:입력 | 2.전체조회 | 3.이름찾기 | 4.메일삭제 | 5.CSV내보내기 | 0.종료메뉴선택 : 0
```

연습문제(실습형)

10장. 데이터베이스 연동

```
class Member:
    def __init__(self, name, phone, email, age, grade, etc):
    def __str__(self):
    def as_dict(self):
def to_member(*row): # 튜플 데이터를 매개변수로 받아 Member형 객체로 반환

# 1.입력
def insert_member():

# 2. 전체 출력
def select_all():

# 3. 이름 검색
def select_name():

# 4. 메일 삭제
def delete_email():

# 5. CSV 내보내기
def save_csv():

# 0. 종료
def close_sql():
```

```
def main():
    while True:
        print("1.입력", "2.전체 조회", "3.이름 찾기", "4.메일 삭제", "5.CSV 내보내기", "0.종료",
              sep=" | ")
        try:
            menu = int(input("메뉴 선택: "))
        except:
            print("유효하지 않은 값을 입력하였습니다. 다시 선택해주세요.")
        if menu==1:
            insert_member()
        elif menu==2:
            select_all()
        elif menu==3:
            select_name()
        elif menu==4:
            delete_email()
        elif menu==5:
            save_csv()
        elif menu==0:
            close_sql()
            break
```

연습문제(실습형)

```
if __name__=='__main__':
    import sqlite3
    global conn
    conn = sqlite3.connect(‘data/ch10_data.db')
    main()
```

연습문제(문제풀이형)

10장. 데이터베이스 연동

1.파이썬의 데이터베이스 연동에 대해 잘 못 설명한 것은?
① 데이터베이스에 연결하기 위한 모듈들은 SQLite 데이터베이스를 위한 DB-API 2.0 사양을 준수한다.
② COnnection 객체를 이용해서 변경사항을 저장(cOmmit)하거나 취소(rOllback)할 수 있 다.
③ COnnection 객체를 생성하는 방법은 데이터베이스마다 다를 수 있다.
④ cOmmit()을 호출하지 않고 데이터베이스 연결 객체를 clOse() 하면 변경사항은 자동으로 저장된다.

2.파이썬의 CursOr 객체에 대해 잘 못 설명한 것은?
① 커서는 SQL 구문을 실행시키고 데이터를 조회하는데 사용한다.
② 커서 객체의 execute() 메소드는 결과 set으로 반환한다.
③ 실행할 SQL 구문을 매개변수화 할 수 있다.
④ 커서 객체는 COnnection의 cursOr() 메소드를 이용해 사용할 수 있다.