데이터베이스 분석

스프레드시트와 마찬가지로 데이터 베이스는 비즈니스 전반에 널리 사용되고 있다.

회사들은 데이터베이스를 이용하여 고객, 재고 품목, 근무자 등에 대한 데이터를 저장한다.

또한 데이터베이스는 운영, 영업, 재무 등을 추적하는데 필수적이다.

데이터베이스를 단일 스프레드시트나 통합 문서와 다른 점은, 스프레드시트 내 하나의 행이 다른 스프레드시트의 행/열과 연결될 수 있듯 데이터 베이스는 테이블 단위로 서로 연결된다는 점이다.

흔한 예로 고객명, 주소 등의 데이터는 주문상품을 기록하는 '주문' 스프레드시트의 한 행으로 연결할 수 있다. (고객 ID를 이용) . 이어서 이 상품들을 '공급 업체' 스프레드시트 내 데이터와 연결하여 주문 추적 및 배송을 관리할 수 있고 더 심층적인 분석도 시도할 수 있다.

csv와 엑셀 파일은 널리 쓰이는 동시에 중요한 데이터 소스로서 파이썬으로 대규모 처리를 자동화할 수 있다.

파이썬에는 in-memory 데이터베이스를 만드는 것을 지원하는 sqlite3 모듈이 내장되어 있다.

이 모듈을 사용하면 특정 데이터베이스를 다운로드하고 설치할 필요없이 파이썬 코드 내부에서 직접적으로 데이터가 채워진 데이터베이스와 테이블을 생성할 수 있다.

데이터베이스 테이블을 만들고, 여기에 데이터를 삽입하고, 출력될 데이터의 수를 합치고 계산한다.

파일명 : 1db_count_rows.py

```
#!/usr/bin/env python3
import sqlite3

# Create an in-memory SQLite3 database                    메모리에 SQLite3 데이터베이스를 만든다.
# Create a table called sales with four attributes        4개 속성을 지닌 sales 테이블을 만든다.
con = sqlite3.connect(':memory:')
query = """CREATE TABLE sales
                        (customer VARCHAR(20),
                         product VARCHAR(40),
                         amount FLOAT,
                         date DATE);"""
con.execute(query)
con.commit()

# Insert a few rows of data into the table                sales 테이블에 데이터를 삽입한다.
data = [('Richard Lucas', 'Notepad', 2.50, '2019-01-02'),
                ('Jenny Kim', 'Binder', 4.15, '2019-01-15'),
                ('Svetlana Crow', 'Printer', 155.75, '2019-02-03'),
                ('Stephen Randolph', 'Computer', 679.40, '2019-02-20')]
statement = "INSERT INTO sales VALUES(?, ?, ?, ?)"
con.executemany(statement, data)
con.commit()

# Query the sales table                                   sales 테이블에 질의한다.
cursor = con.execute("SELECT * FROM sales")
rows = cursor.fetchall()

# Count the number of rows in the output                  출력된 데이터의 수를 센다.
row_counter = 0
for row in rows:
        print(row)
        row_counter += 1
print('Number of rows: {}'.format(row_counter))
```

☞ **구문 해석**

import sqlite3

2 행에서 sqlite3 모듈을 임포트한다.
이 모듈은 별도의 서버 처리 과정이 필요 없는 디스크 기반의 가벼운 데이터 베이스를 제공하고,
여러 종류의 SQL 쿼리 언어를 사용하여 데이터베이스에 접근할 수 있게 해준다.

con = sqlite3.connect(':memory:')

6 행에서 데이터베이스를 나타내는 con 이라 부르는 접속 객체를 생성하였다.
이번 실습에서는 **:memory:** 를 사용하여 RAM 내부에 데이터베이스를 생성했다.
만약, 데이터베이스를 지속시키려면 다른 문자열을 사용해야 한다.
예를 들어 **:memory:** 대신에 my_database.db 또는 C:₩Users₩TJ₩Desktop₩my_databse.db 와 같은 문자열을 사용한다면 이 데이터베이스 객체는 현재 디렉토리 또는 바탕화면에 저장 할 수 있다.

query = """CREATE TABLE sales

7 ~ 11 행은 삼중 큰따옴표(""")를 이용하여 여러 줄에 걸쳐 단일 문자열을 만들고 변수 query에 할당한다.
이 문자열이 바로 SQL 명령어로서, 이 명령은 데이터베이스에 sales 라는 테이블을 만들며,
sales 테이블은 customer, product, amount, date라는 이름의 4가지 속성을 갖는다.
customer 속성은 최대 길이가 20문자인 문자열 필드이다.
product 속성은 최대 40자 길이의 문자열 필드이다.
amount 속성은 실수 형식의 필드이다.
date 속성은 날짜 형식 필드이다.

con.execute(query)

12 행에서 접속 객체의 execute( ) 함수를 사용하여 query 변수에 들어 있는 SQL 명령어를 실행한다.
이 명령어를 실행하면 인메모리(:memory:) 데이터베이스내에 sales 테이블이 만들어진다.

con.commit()

13 행에서 접속 객체의 commit( ) 함수를 사용하여 데이터베이스의 업데이트를 저장한다.
데이터베이스를 변경하였다면 업데이트된 상태를 저장하기 위해 반드시 commit( ) 함수를 사용해야 한다.
이 명령을 사용하지 않은 경우에는 변경된 상태가 데이터베이스에 반영되지 않는다.

data = [('Richard Lucas', 'Notepad', 2.50, '2019-01-02'),

16행에서 튜플로 이루어진 리스트를 만들어 data라는 변수에 할당한다.
리스트 내 각 원소는 4개의 변수를 포함한 하나의 튜플이다.
4개 중 3개는 문자열이고 하나는 실수값이다.
이들 변수는 위치에 따라 각 테이블 속성(테이블의 4개 열)에 해당한다.
동시에 각 튜플은 테이블 내 하나의 행을 구성하는 데이터이기도 하다.
data 리스트가 4개의 튜플을 포함하므로 테이블은 4행의 데이터를 포함하게 된다.

statement = "INSERT INTO sales VALUES(?, ?, ?, ?)"

20 행은 7 행과 마찬가지로 문자열을 생성하고 statement 변수에 이 문자열을 할당한다.
이 문자열은 한 줄로 작성되었기 때문에 7행처럼 여러 줄 작성용 삼중 큰 따옴표 대신 그냥 큰 따옴표를 사용해 문자열을 할당했다. data 변수에 포함된 여러 행의 데이터를 sales 테이블을 삽입하기 위해 insert 구문을 사용했다.

con.executemany(statement, data)

21 행에서 연결 객체의 executemany( ) 함수를 이용하여 data에 포함된 모든 튜플 데이터에 대해 statement에 있는 SQL 명령어를 실행한다. data에는 네 개의 튜플이 있으므로, executemany( ) 함수는 네 번의 INSERT 명령을 실행하여 sales 테이블에 네 행의 데이터를 삽입한다.

con.commit()

22 행에서 commit 함수를 써서 데이터베이스에 변경값을 저장하였다.

cursor = con.execute("SELECT * FROM sales")

25 행은 한 줄의 SQL 명령어를 실행하려면 execute 함수를 이용하고, 그 결과를 cursor 이라는 커서 객체에 할당한다. 커서 객체에는 execute, executemany, fetchone, fetchmany, fetchall 등 여러 함수가 있다.
execute 함수로 실행한 SQL 명령어의 모든 결과물을 보거나 조작하는 할 수 이쓰며, fetchall( ) 함수를 사용하여 모든 결과 데이터를 반환할 수 있다.

rows = cursor.fetchall()

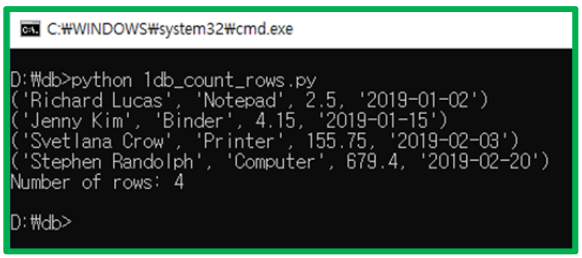
26 행에서 이전에 실행한 SQL 명령어의 모든 결과 데이터를 반환하는 fetchall( ) 함수를 사용해 rows라는 리스트 변수에 이 데이터를 할당했다.

row_counter = 0

29 ~ 32 행에서는 rows에 있는 데이터의 수를 세기 위해 row_counter 변수를 생성하고,
for 문으로 rows의 각 데이터를 순회하면 rows 내 데이터 수만큼 row_counter의 갓을 증가시킨다.
for 문이 종료된 후에는 명령 프롬프트(터미널) 창에 Number of rows: 라는 문자열과 row_counter의 값을 출력한다. rows에는 네 개의 데이터 행이 출력될 것이다.

스크립트 실행

cmd> python 1db_count_rows.py



테이블에 새 레코드 삽입하기

테이블에 각각 20 ~30가지 속성을 지닌 10,000개 레코드들을 직접 입력 해야한다고 하면 절대 수동으로 데이터를 입력하지는 못할것이다.

대부분의 경우, 데이터베이스 테이블에 입력해야 할 데이터는 데이터베이스 쿼리 결과물이거나 이미 만들어진 하나 이상의 엑셀 또는 CSV 파일로 존재한다.
주요한 모든 데이터베이스들에서 쿼리 결과물을 CSV 파일로 내보내는 것은 상대적으로 쉽다.

이번에는 CSV 파일을 사용하여 대규모 데이터를 데이터베이스 테이블에 입력하는 방법에 대하여 실습한다.

파일명 : 2db_insert_rows.py

```
#!/usr/bin/env python3
import csv
import sqlite3
import sys

# Path to and name of a CSV input file        메모리에 SQLite3 데이터베이스를 만든다.
input_file = sys.argv[1]

# Create an in-memory SQLite3 database
# Create a table called Suppliers with five attributes
con = sqlite3.connect('Suppliers.db')
c = con.cursor()
create_table = """CREATE TABLE IF NOT EXISTS Suppliers
                                    (Supplier_Name VARCHAR(20),
                                    Invoice_Number VARCHAR(20),
                                    Part_Number VARCHAR(20),
                                    Cost FLOAT,
                                    Purchase_Date DATE);"""
c.execute(create_table)
con.commit()

# Read the CSV file                                CSV 파일을 읽는다.
# Insert the data into the Suppliers table         읽은 데이터를 Suppliers 테이블에 삽입한다.
file_reader = csv.reader(open(input_file, 'r'), delimiter=',')
```

```
header = next(file_reader, None)
for row in file_reader:
        data = []
        for column_index in range(len(header)):
                data.append(row[column_index])
        print(data)
        c.execute("INSERT INTO Suppliers VALUES (?, ?, ?, ?, ?);", data)
con.commit()

# Query the Suppliers table                    Suppliers 테이블에 질의한다.
output = c.execute("SELECT * FROM Suppliers")
rows = output.fetchall()
for row in rows:
        output = []
        for column_index in range(len(row)):
                output.append(str(row[column_index]))
        print(output)
```

구문 해석

csv와 sys 모듈을 사용하여 실습하였다.

2 행에서 CSV 모듈을 임포트해서 CSV 입력 파일을 읽고 파싱하는 함수들을 사용할 수 있게 한다.

3행은 sqlite3 모듈을 임포트해서 간단한 로컬 데이터베이스와 테이블을 만들고 SQL 쿼리를
실행할 수 있는 함수들을 사용할 수 있게 한다.

4행은 sys 모듈을 임포트하여 파일 경로와 파일명을 명령줄에서 받아올 수 있게 한다.

7 행에서는 sys 모듈을 사용하여 명령줄에서 파일 경로와 파일명을 읽어온 후,
input_file 변수에 그값을 할당한다.

11행에서 Suppliers.db 라는 간단한 로컬 데이터베이스에 연결을 한다.
여기에서는 "memory: 키워드를 사용하는 대신 데이터베이스 명칭을 사용했다.
이렇게 하면 컴퓨터를 재부팅하더라도 데이터베이스를 유지할 수 있게 된다.

12 ~ 18 행에서는 커서 객체와 여러 줄로 된 SQL 문을 만든다.
이 SQL문은 다섯 가지 속성을 지닌 Suppliers 라는 테이블을 생성한다.
이 명령을 실행하고, 변경된 정보를 데이터베이스에 저장한다.

24 ~ 32 행에서는 두 가지 작업을 수행한다.
첫째, 데이터베이스 테이블에 입력할 데이터를 CSV 입력파일로부터 읽어오는 것이고,
둘째, 읽어온 데이터의 각 행별로 SQL 문을 실행하여 데이터베이스 테이블에 삽입하는 것이다.

24 행은 CSV 모듈을 이용하여 file_reader 객체를 만든다.

25 행은 next( ) 함수를 사용하여 입력 파일의 첫번째 행을 읽고 header라는 변수에 그 값을 할당한다.

26 행에서는 for 문을 사용하여 입력파일의 모든 행을 반복처리 한다.

27 행에서는 data라는 빈 리스트 변수를 채운다.
입력 데이터의 각 행별로, 이 data 변수에 31 행의 INERT 구문에 필요한 값을 채울 것이다.

28 행에서는 각 행의 모든 속성을 반복 처리할 for 문을 사용했다.

29 행에서는 리스트의 append( ) 함수를 이용하여 입력 파일 각 행의 모든 값을 data 변수에 채운다.

30 행에서는 data 변수에 추가된 해당 행 데이터를 명령프롬프트나 터미널 창에 출력한다.
이 줄은 들여쓰기에 주의해야 한다.
입력 데이터의 모든 행과 모든 속성 대신 모든 행에 대해 나타나야 하기 때문에,
이 줄의 들여쓰기는 내부 for 문이 아닌 외부 for 문에 맞춰져야 한다.
이 줄은 디버깅을 하는데 도움이 된다.
하지만 코드가 정확하게 작동할 것이라고 확신한다면, 화면에 많은 출력 결과물을 표시하지 않기 위해 이줄을 지우거나 주석으로 처리해도 된다.

31 행에서는 실제로 데이터베이스 테이블에 각 행별 데이터를 입력한다.
이 줄은 커서 객체의 execute( ) 함수를 이용하여 Suppliers 테이블에 한 행의 변수들을 입력하는 INSERT 구문을 실행한다.

이 구문의 물음표들은 입력할 각 변수들의 위치를 표시하는 플레이스홀더이다.
물음표의 수는 입력파일에 있는 열의 수와 일치해야 하고, 테이블의 열의 수와 일치해야 한다.
또한, 입력 파일 내 열의 순서는 테이블 내 열의 순서와 일치해야 한다.
물음표의 위치에 대입되는 값들은 execute( ) 함수 내 쉼표 뒤에 있는 data 변수에 할당된 값 리스트에 들어 있다. 입력팡리 데이터의 각 행별로 data 변수에 값들이 채워지고 이에 대해 INSERT 구문이 실행되므로 이 코드는 효과적으로 입력파일로 부터 데이터 행들을 읽은 후,
데이터베이스 테이블에 채운다.

32 행에서는 이 데이터베이스에 변경된 정보를 저장하기 위해 commit( ) 명령은 실행한다.

36 ~ 42 행은 Suppliers 테이블의 모든 데이터를 선택하고 그 결과를 명령프롬프트나 터미널 창에 출력하는
방법을 보여준다.
Suppliers 테이블의 모든 데이터를 선택하는 SQL 문을 실행하고, 그 결과(output)의 행들을 변수 rows에 가져온다.

38 행은 rows 변수의 각 행들을 반복 처리하는 for 문이다.

40 행은 각 행의 열에 대해 반복 처리하는 for 문이다.

41 행은 output이라는 리스트에 각 행의 값들을 추가한다.

42 행에서는 print 문을 사요하여 SQL 문을 실행한 결과의 각 행을 새로운 줄에 출력한다.

이제 데이터베이스 테이블에 사용할 CSV 파일만 있으면 된다.
이 실습을 위해 supplier_data.cav 파일을 사용한다.

스크립트 실행

cmd> python 2db_insert_rows.py supplier_data.csv

```
D:₩db>python 2db_insert_rows.py supplier_data.csv
['Supplier X', '001-1001', '2341', '$500.00 ', '1/20/14']
['Supplier X', '001-1001', '2341', '$500.00 ', '1/20/14']
['Supplier X', '001-1001', '5467', '$750.00 ', '1/20/14']
['Supplier X', '001-1001', '5467', '$750.00 ', '1/20/14']
['Supplier Y', '50-9501', '7009', '$250.00 ', '1/30/14']
['Supplier Y', '50-9501', '7009', '$250.00 ', '1/30/14']
['Supplier Y', '50-9505', '6650', '$125.00 ', '2/17/14']
['Supplier Y', '50-9505', '6650', '$125.00 ', '2/17/14']
```

결과물의 첫번째 구역은 CSV 파일을 파싱한 데이터 행들이고, 두번째 구역은 sqlite3 테이블에서 불러온 동일한 행들이다.

테이블 내 레코드 갱신하기

이번에는 CSV 파일을 사용하여 데이터베이스 테이블에 있는 기존 레코드들을 갱신하는 방법에 대해 실습한다.

파일명 : 3db_update_rows.py

```
#!/usr/bin/env python3
import csv
import sqlite3
import sys

# Path to and name of a CSV input file                    CSV 입력 파일 경로와 파일명
input_file = sys.argv[1]

# Create an in-memory SQLite3 database                    메모리에 SQLite3 데이터베이스를 만든다.
# Create a table called sales with four attributes        네 가지 속성을 지닌 sales 테이블을 만든다.
con = sqlite3.connect(':memory:')
query = """CREATE TABLE IF NOT EXISTS sales
                        (customer VARCHAR(20),
                                product VARCHAR(40),
                                amount FLOAT,
                                date DATE);"""
con.execute(query)
con.commit()

# Insert a few rows of data into the table                  데이터 입력
data = [('Richard Lucas', 'Notepad', 2.50, '2019-01-02'),
                ('Jenny Kim', 'Binder', 4.15, '2019-01-15'),
                ('Svetlana Crow', 'Printer', 155.75, '2019-02-03'),
                ('Stephen Randolph', 'Computer', 679.40, '2019-02-20')]
for tuple in data:
        print(tuple)
statement = "INSERT INTO sales VALUES(?, ?, ?, ?)"
con.executemany(statement, data)
con.commit()
```

```
# Read the CSV file and update the specific rows
file_reader = csv.reader(open(input_file, 'r'), delimiter=',')
header = next(file_reader, None)
for row in file_reader:
        data = []
        for column_index in range(len(header)):
                data.append(row[column_index])
        print(data)
        con.execute("UPDATE sales SET amount=?, date=? WHERE customer=?;", data)
con.commit()

# Query the sales table
cursor = con.execute("SELECT * FROM sales")
rows = cursor.fetchall()
for row in rows:
        output = []
        for column_index in range(len(row)):
                output.append(str(row[column_index]))
        print(output)
```

CSV 파일을 읽고, 특정 행의 데이터를 갱신한다.

slaes 테이블에 질의한다.

구문 해석

2 ~ 4 행에서 세 개의 내장 모듈을 임포트해서 명령줄의 입력을 읽고, CSV 파일을 읽고, 인메모리 데이터베이스 및 테이블과 상호작용할 수 있게 했다.

7 행은 CSV 입력파일을 input_file 변수에 할당한다.

11 ~ 18 행에서 네 가지 속성을 가진 sales 테이블과 인메모리 데이터베이스를 만든다.

21 ~ 29 행에서 sales 테이블에 네 개의 레코드 집합을 만들고, 이 레코드들을 테이블에 입력한다.

35 행은 UPDATE 문을 사용했다. update 문은 갱신하고 싶은 레코드와 열 속성을 지정해야 한다.

여기서는 특정 고객(coustomer)의 amount와 date 값을 갱신할 것이다.
갱신을 위해서는 쿼리 내 값들의 위치를 가리키는 물음표 플레이스홀더들이 있어야 하며,
CSV 입력파일의 데이터 순서는 쿼리 내 속성들의 순서와 동일해야 한다.
여기에서는 왼족에서 오른쪽 순서대로 쿼리 내 속성은 amount, date, customer 이다.
따라서, CSV 입력 파일에 있는 열들의 순서도 왼쪽부터 오른쪽으로 amount, date, customer여야 한다.

43 ~ 49 행은 sales 테이블의 모든 행들을 불러오고,
각 열 값들 사이에 공백문자를 추가하여 명령프롬프트나 터미널 창에 각 행을 출력한다.

실습을 위해 데이터베이스 테이블의 특정 레코드를 갱신할 데이터가 포함된 CSV 입력 파일이 필요하다.
다음 순서에 따라 파일을 생성한다.

1. 스프레드시트(엑셀)을 연다.

2. 데이터를 추가한다.

	A	B	C
1	amount	date	customer
2	4.25	5/11/2014	Richard Lucas
3	6.75	5/12/2014	Jenny Kim

3. data_for_updating.csv 파일로 저장한다.

스크립트 실행

cmd> python 3db_update_rows.py data_for_updating.csv

```
D:₩db>python 3db_update_rows.py data_for_updating.csv
('Richard Lucas', 'Notepad', 2.5, '2019-01-02')
('Jenny Kim', 'Binder', 4.15, '2019-01-15')
('Svetlana Crow', 'Printer', 155.75, '2019-02-03')
('Stephen Randolph', 'Computer', 679.4, '2019-02-20')
['4.25', '5/11/2014', 'Richard Lucas']
['6.75', '5/12/2014', 'Jenny Kim']
['Richard Lucas', 'Notepad', '4.25', '5/11/2014']
['Jenny Kim', 'Binder', '6.75', '5/12/2014']
['Svetlana Crow', 'Printer', '155.75', '2019-02-03']
['Stephen Randolph', 'Computer', '679.4', '2019-02-20']
```

출력 결과의 첫 4 행은 초기 데이터이고, 다음 2행은 CSV 파일로부터 읽어온 데이터이고,
마지막 4행은 특정 행은 갱신한 후 데이터베이스 테이블로부터 가져온 데이터이다.

MySQL 데이터베이스

이번 실습을 위해서는 명령프롬프트에서 mysql 을 입력하여 클라이언트를 연다
이제 명령줄 인터페이스를 통해 MaridDB 데이터베이스 프로그램과 상호작용할 수 있다.

cmd> pip install mysqlclient

MariaDB [(none)> SHOW DATABASES;

```
Enter password: ****
Welcome to the MariaDB monitor.  Commands end with ; or ₩g.
Your MariaDB connection id is 10
Server version: 10.3.15-MariaDB mariadb.org binary distribution

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '₩h' for help. Type '₩c' to clear the current input statement.

MariaDB [(none)]> SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| homework           |
| information_schema |
| mysql              |
| performance_schema |
| test               |
+--------------------+
5 rows in set (0.002 sec)
```

데이터베이스를 생성한다.

MariaDB [(none)]> CREATE DATABASE my_suppliers;

```
MariaDB [(none)]> CREATE DATABASE my_suppliers;
Query OK, 1 row affected (0.001 sec)
```

생성된 데이터베이스를 확인한다.

MariaDB [(none)> SHOW DATABASES;

```
MariaDB [(none)]> SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| homework           |
| information_schema |
| my_suppliers       |
| mysql              |
| performance_schema |
| test               |
+--------------------+
6 rows in set (0.001 sec)
```

사용할 데이터베이스를 선택한다.

MariaDB [(none)]> USE my_suppliers;

```
MariaDB [(none)]> use my_suppliers;
Database changed
```

공급자(Suppliers) 테이블을 생성한다.

MariaDB [my_suppliers]> CREATE TABLE Suppliers
(Supplier_Name VARCHAR(20),
Invoice_Number VARCHAR(20),
Part_Number VARCHAR(20),
Cost FLOAT,
Purchase_Date DATE);

```
MariaDB [my_suppliers]> CREATE TABLE Suppliers
    -> (Supplier_Name VARCHAR(20),
    -> Invoice_Number VARCHAR(20),
    -> Part_Number VARCHAR(20),
    -> Cost FLOAT,
    -> Purchase_Date DATE);
Query OK, 0 rows affected (0.019 sec)
```

데이터베이스 테이블이 정확하게 만들어졌는지 확인한다.

MariaDB [my_suppliers]> DESCRIBE Suppliers;

```
MariaDB [my_suppliers]> DESCRIBE Suppliers;
+---------------+-------------+------+-----+---------+-------+
| Field         | Type        | Null | Key | Default | Extra |
+---------------+-------------+------+-----+---------+-------+
| Supplier_Name | varchar(20) | YES  |     | NULL    |       |
| Invoice_Number| varchar(20) | YES  |     | NULL    |       |
| Part_Number   | varchar(20) | YES  |     | NULL    |       |
| Cost          | float       | YES  |     | NULL    |       |
| Purchase_Date | date        | YES  |     | NULL    |       |
+---------------+-------------+------+-----+---------+-------+
5 rows in set (0.014 sec)
```

새로운 사용자를 만들고 이 사용자에게 데이터베이스와 테이블 간에 상호작용할 수 있는 권한을 설정한다.

MariaDB [my_suppliers]> CREATE USER '사용자명'@'localhost' IDENTIFIED BY '암호';

```
MariaDB [my_suppliers]> CREATE USER 'tj'@'localhost' IDENTIFIED BY 'apple';
Query OK, 0 rows affected (0.004 sec)
```

실습에서는 TJ 계정에 apple 암호를 사용했다.

새로운 사용자에게 모든 권한을 부여하기 한다.

MariaDB [my_suppliers]> GRANT ALL PRIVILEGES ON my_suppliers.* TO '사용자명'@'localhost';

```
MariaDB [my_suppliers]> GRANT ALL PRIVILEGES ON my_suppliers.* TO 'tj'@'localhost';
Query OK, 0 rows affected (0.001 sec)
```

MariaDB [my_suppliers]> FLUSH PRIVILEGES;

```
MariaDB [my_suppliers]> FLUSH PRIVILEGES;
Query OK, 0 rows affected (0.000 sec)
```

이제 준비가 되었으므로 파이썬에서 테이블에 데이터를 입력하는 방법에 대해 실습한다.

테이블에 새 레코드 입력하기

스크립트를 작성한다.
작성할 스크립트는 CSV 파일의 데이터를 위에서 작업한 데이터베이스 테이블에 입력하고,
테이블에 있는 데이터를 출력할 것이다.
명령프롬프트/터미널 창에 데이터를 출력하는 것은 많은 자료가 출력될 수 있으므로 **필수단계는 아님에 주의한다.**

파일명 : 4db_mysql_load_from_cav.py

```
#!/usr/bin/env python3
import csv
import MySQLdb
import sys
from datetime import datetime, date

# Path to and name of a CSV input file
input_file = sys.argv[1]

# Connect to a MySQL database
con = MySQLdb.connect(host='localhost', port=3306, db='my_suppliers', user='계정명', passwd='암호')
c = con.cursor()

# Read the CSV file
# Insert the data into the Suppliers table
file_reader = csv.reader(open(input_file, 'r'), delimiter=',')
header = next(file_reader)
for row in file_reader:
        data = []
        for column_index in range(len(header)):
                if column_index < 4:
                        data.append(str(row[column_index]).lstrip('$')₩
                        .replace(',', '').strip())
                else:
                        a_date = datetime.date(datetime.strptime(₩
                        str(row[column_index]), '%m/%d/%y'))
```

```
                        # %Y: year is 2016; %y: year is 15
                        a_date = a_date.strftime('%Y-%m-%d')
                        data.append(a_date)
        print(data)
        c.execute("""INSERT INTO Suppliers VALUES (%s, %s, %s, %s, %s);""", data)
con.commit()

# Query the Suppliers table
c.execute("SELECT * FROM Suppliers")
rows = c.fetchall()
for row in rows:
        row_list_output = []
        for column_index in range(len(row)):
                row_list_output.append(str(row[column_index]))
        print(row_list_output)
```

구문 해석

이 스크립트에서는 csv, datetime, sys 모듈을 사용한다.

2 행은 CSV 파일을 읽고 파싱하는 함수를 사용하기 위해 csv 모듈ㅇ르 임포트한다.

4 행은 이 스크립트에서 사용할 파일명과 경로를 명령줄로부터 읽어오기 위해 sys 모듈을 임포트한다.

5 행은 날짜를 다루고 포맷팅하기 위해 datetime 모듈에서 date와 datetime 함수를 임포트한다.
데이터베이스 테이블의 열에서는 실수만 허용되므로 달러 기호와 쉼표를 제거할 필요가 있다.

3 행에서는 설치한 MariaDB 모듈을 임포트해서 MariaDB 데이터베이스와 테이블에 연결하는
함수를 사용할 수 있게 한다.

8 행에서 sys 모듈을 이용하여 명령줄에서 파일의 경로와 이름을 읽어오고 input_file 변수에 할당한다.

11 헹에서 MariaDB 모듈의 connect 함수를 이용하여 앞에서 만든 my_suppliers 데이터베이스에 접속한다.
인플레이스로 읽고 수정하고 삭제할 수 있는 CSV 또는 엑셀파일 작업과는 다르게,
MariaDB는 별도의 컴퓨터 처럼 우리가 접속하거나 데이터를 주고 받을 수 있도록 지원한다.
데이터베이스에 접속하기 위해서는 host, port, db, user, passwd 등 일반적인 인수들을 설정한다.

host는 데이터베이스가 위치한 컴퓨터의 호스트 주소이다.
MariaDB 서버가 현재 컴퓨터에 설치되었기에 host는 localhost 값을 갖는다.
다른 컴퓨터에 접속할 때에는 그 서버의 주소로 변경해야 한다.
port는 MariaDB 서버의 TCP/IP 접속을 위한 포트 번호이다. 기본값 3306을 사용한다.
db는 접속할 데이터베이스의 이름이다. (my_suppliers 데이터베이스에 접속할 것이다.)
user는 데이터베이스에 접속할 사용자 이름이다.
passwd는 데이터베이스에서 설정한 계정의 암호를 지정한다.

12 행은 커서를 만들어서 my_suppliers 데이터베이스의 Suppliers 테이블에서 SQL문을 실행하고
그 변화를 데이터베이스에 저장하는데 이용할 수 있게 한다.

16 ~ 23 행에서 데이터베이스 테이블에 저장하기 위한 데이터를 CSV 파일로부터 읽어옥,
이 데이터의 각 행을 데이터베이스 테이블에 입력하는 SQL 문을 실행한다.

16 행에서는 csv 모듈을 이용하여 file_reader 객체를 만든다.

17 행에서는 next( ) 함수를 이용하여 입력파일의 첫 행인 헤더 정보를 읽어온 후 header 변수에 할당한다.

18행에서는 입력팡리의 모든 행을 반복 처리하기 위한 for 문을 사용한다.

19 행은 data라는 빈 리스트 변수를 만들어 각 입력 행에 대해 31행의 INSERT 문에서 필요한 값을 채운다

20 행은 각 행에 있는 모든 열들을 반복 처리하기 위한 for 문이다.

21 행은 if – else 문은 해당 열 인덱스가 4보다 작은지 확인한다.
입력파일은 다섯개의 열로, 이 중 날짜 정보는 다섯 번째 열에 있으므로 날짜열의 인덱스는 4이다.
따라서 이 줄애서는 마지막 날짜 열 앞에 있는 열들을 처리할 지 여부를 판단한다.

22 행에서는 0, 1, 2, 3의 인덱스 값을 지닌 열에 대해 실행된다.
해당 열의 값을 문자열로 반환하고, 달러($) 기호가 있으면 이를 제거하고,
리스트 변수인 data에 저장한다.

25 행은 인덱스 값이 4인 마지막 날짜 열을 문자열로 바꾸고, 입력 형식에 따라 해당 문자열을
datetime 객체로 만든 후, datetime 객체를 date 객체 (이 객체는 오직 년, 월, 일 요소만 유지한다)로 변환하여 a_date 변수에 할당한다.

28 행은 date 객체를 MariaDB 데이터베이스에 입력하기 위해 새로운 입력 형식(YYYY-MM-DD)의
문자열로 변환하여 다시 a_date 변수에 할당한다.

29 행에서는 이렇게 변환한 문자열을 data에 저장한다.

30 행은 명령프롬프트/터미널 창에 data에 저장된 데이터 행을 출력한다. (들여쓰기에 주의한다)

31 행은 실제로 데이터베이스 테이블에 각 행별 데이터를 입력한다.
이 줄은 커서 객체의 execute( ) 함수를 이용하여 Suppliers 테이블에 한 행의 변수들을 입력하기 위한 INSERT 구문을 실행한다. 각각의 %s는 입력할 각 변수의 위치를 표시하는 플레이스홀더이다.
그 개수는 입력파일에 있는 열의 수와 일치해야 하고, 테이블의 열의 수와 일치해야 한다.
또한, 입력파일 내 열의 순서는 테이블 내 열의 순서와 일치해야 한다. %s의 위치에 대입되는 값들은 execute( ) 함수 내 쉼표 뒤에 있는 data 변수에 할당된 값들 리스트에 있다.
입력파일 데이터의 각 행별로 data 변수에 값들이 채워지고 INSERT 구문이 실행되기 때문에,
이 코드들은 효과적으로 입력파일로부터 데이터의 행들을 읽은 후 데이터베이스 테이블에 채운다.

32 행에서 이 데이터베이스에 변경된 정보를 저장하기 위해 commit( ) 문을 수행한다.

36 ~ 42 행은 Suppliers 테이블의 모든 데이터를 선택하고 그 결과를 명령프롬프트나 터미널 창에 출력하는
방법을 보여준다.

36 ~ 37 행은 Suppliers 테이블의 모든 데이터를 선택하는 SQL 문을 실행하고,
그 결과 행들을 변수 rows에 가져온다.

38 행은 rows 변수의 각 행들을 반복 처리하는 for 문을 사용한다.

39 행은 orw_output 이라는 빈 변수를 만들어서 SQL 쿼리로부터 가져온 각 행들의 값들을 포함시킨다.

40 행은 각 행의 열(속성)을 반복 처리하는 for 문이다.

41 행은 각 열에 있는 값들을 문자열로 변환한 후, row_list_output 변수에 저장한다.

42 행에서는 row_list_outoput 변수에 저장된 이 행의 데이터를 화면에 출력한다

스크립트 실행

cmd> python 4db_mysql_load_from_csv.py supplier_data.csv

```
D:₩db>python 4db_mysql_load_from_csv.py supplier_data.csv
['Supplier X', '001-1001', '2341', '500.00', '2014-01-20']
['Supplier X', '001-1001', '2341', '500.00', '2014-01-20']
['Supplier X', '001-1001', '5467', '750.00', '2014-01-20']
['Supplier X', '001-1001', '5467', '750.00', '2014-01-20']
['Supplier Y', '50-9501', '7009', '250.00', '2014-01-30']
['Supplier Y', '50-9501', '7009', '250.00', '2014-01-30']
['Supplier Y', '50-9505', '6650', '125.00', '2014-02-17']
['Supplier Y', '50-9505', '6650', '125.00', '2014-02-17']
['Supplier Z', '920-4803', '3321', '615.00', '2014-02-17']
['Supplier Z', '920-4804', '3321', '615.00', '2014-02-17']
['Supplier Z', '920-4805', '3321', '615.00', '2014-02-17']
['Supplier Z', '920-4806', '3321', '615.00', '2014-02-24']
['Supplier X', '001-1001', '2341', '500.0', '2014-01-20']
['Supplier X', '001-1001', '2341', '500.0', '2014-01-20']
['Supplier X', '001-1001', '5467', '750.0', '2014-01-20']
['Supplier X', '001-1001', '5467', '750.0', '2014-01-20']
['Supplier Y', '50-9501', '7009', '250.0', '2014-01-30']
['Supplier Y', '50-9501', '7009', '250.0', '2014-01-30']
['Supplier Y', '50-9505', '6650', '125.0', '2014-02-17']
['Supplier Y', '50-9505', '6650', '125.0', '2014-02-17']
['Supplier Z', '920-4803', '3321', '615.0', '2014-02-17']
['Supplier Z', '920-4804', '3321', '615.0', '2014-02-17']
['Supplier Z', '920-4805', '3321', '615.0', '2014-02-17']
['Supplier Z', '920-4806', '3321', '615.0', '2014-02-24']
```

결과물의 첫번째 부분은 CSV 파일을 파싱한 데이터이고,
두번째 부분은 데이터베이스 테이블에서 불러온 동일한 데이터이다.

이 결과는 CSV 파일의 헤더 행을 제외한 12 개의 데이터로부터 생성된 12 개의 리스트를 보여준다.
리스트는 화면에 출력될 때 각 값들이 쉼표로 구분되고 대괄호로 묶여 표현된다.

데이터베이스에서 이 결과를 확인한다.

MariaDB [my_suppliers]> SELECT * FROM Suppliers;

```
MariaDB [my_suppliers]> SELECT * FROM Suppliers;
+---------------+----------------+-------------+------+---------------+
| Supplier_Name | Invoice_Number | Part_Number | Cost | Purchase_Date |
+---------------+----------------+-------------+------+---------------+
| Supplier X    | 001-1001       | 2341        |  500 | 2014-01-20    |
| Supplier X    | 001-1001       | 2341        |  500 | 2014-01-20    |
| Supplier X    | 001-1001       | 5467        |  750 | 2014-01-20    |
| Supplier X    | 001-1001       | 5467        |  750 | 2014-01-20    |
| Supplier Y    | 50-9501        | 7009        |  250 | 2014-01-30    |
| Supplier Y    | 50-9501        | 7009        |  250 | 2014-01-30    |
| Supplier Y    | 50-9505        | 6650        |  125 | 2014-02-17    |
| Supplier Y    | 50-9505        | 6650        |  125 | 2014-02-17    |
| Supplier Z    | 920-4803       | 3321        |  615 | 2014-02-17    |
| Supplier Z    | 920-4804       | 3321        |  615 | 2014-02-17    |
| Supplier Z    | 920-4805       | 3321        |  615 | 2014-02-17    |
| Supplier Z    | 920-4806       | 3321        |  615 | 2014-02-24    |
+---------------+----------------+-------------+------+---------------+
12 rows in set (0.000 sec)
```

테이블 검색 및 결과물을 CSV 파일로 출력하기

이번 실습에서는 Suppliers 데이터베이스 테이블에서 특정 레코들 검색하고, 이 결과를 CSV 파일로 저장한다.

조건 : Cost 열이 700 보다 큰 레코드를 출력한다.

파일명 : 5db_mysql_write_to_file.py

```
#!/usr/bin/env python3
import csv
import MySQLdb
import sys

# Path to and name of a CSV output file
output_file = sys.argv[1]

# Connect to a MySQL database
con = MySQLdb.connect(host='localhost', port=3306, db='my_suppliers', user='root', passwd='암호')
c = con.cursor()

# Create a file writer object and write the header row
filewriter = csv.writer(open(output_file, 'w', newline=''), delimiter=',')
header = ['Supplier Name','Invoice Number','Part Number','Cost','Purchase Date']
filewriter.writerow(header)

# Query the Suppliers table and write the output to a CSV file
c.execute("""SELECT *
                    FROM Suppliers
                    WHERE Cost > 700.0;""")
rows = c.fetchall()
for row in rows:
        filewriter.writerow(row)
```

구문 해석

2 ~ 4 행에서 csv, MySQLdb, sys 모듈을 임포트해서 MySQL 데이터베이스와 상호작용하고 CSV 파일에 검색결과를 출력할 함수를 사용할 수 있게 한다.

7 행에서 sys 모듈을 이용하여 명령줄로부터 입력파일의 경로와 파일명을 읽어오고, 이 값들을 output_file 변수에 할당한다.

10 행에서 MariaDB의 connect( ) 함수를 이용하여 앞에서 만든 MariaDB 데이터베이스인 my_suppliers에 접속한다.

10 ~ 12 행에서 데이터베이스 테이블에서 SQL 문을 실행하고 그 결과를 저장하기 위한 커서를 만든다.

15 행에서 이 테이블의 열을 나타내는 문자열 다섯 개가 들어 있는 리스트 변수인 header를 만든다.

16 행에서 filewriter의 writerow( ) 함수를 사용하여 쉼표로 구분된 이 문자열의 리스트를 CSV 형식의 출력파일에 작성한다.
데이터베이스 쿼리는 열의 헤더 정보를 제외한 데이터만 출력하기 때문에, 이 코드로 출력 파일에 헤더 정보를 작성한다.

21 ~ 25 행은 쿼리의 결과물을 명령프롬프트/터미널 창에 출력하는 것을 제외한다.

25 행은 화면에 결과를 화면에 출력하지 않고 CSV 파일에 저장한다.

스크립트 실행

cmd> python 5db_mysql_write_to_file.py 5output.csv

```
D:₩db>
D:₩db>python 5db_mysql_write_to_file.py 5outout.csv

D:₩db>dir 5outout.csv
 D 드라이브의 볼륨: DATA-1TB
 볼륨 일련 번호: D477-8498

 D:₩db 디렉터리

2019-07-20  오전 02:47               147 5outout.csv
               1개 파일                 147 바이트
               0개 디렉터리  65,036,242,944 바이트 남음

D:₩db>type 5outout.csv
Supplier Name,Invoice Number,Part Number,Cost,Purchase Date
Supplier X,001-1001,5467,750.0,2014-01-20
Supplier X,001-1001,5467,750.0,2014-01-20
```

테이블 내 레코드 갱신하기

CSV 파일을 사용하여 데이터베이스 테이블에 있는 기존 레코드들을 갱신하는 방법을 실습한다.

파일명 : 6db_mysql_update_from_csv.py

```
#!/usr/bin/env python3
import csv
import MySQLdb
import sys

# Path to and name of a CSV input file
input_file = sys.argv[1]

# Connect to a MySQL database
con = MySQLdb.connect(host='localhost', port=3306, db='my_suppliers', user='root', passwd='암호')
c = con.cursor()

# Read the CSV file and update the specific rows
file_reader = csv.reader(open(input_file, 'r', newline=''), delimiter=',')
header = next(file_reader, None)
for row in file_reader:
        data = []
        for column_index in range(len(header)):
                data.append(str(row[column_index]).strip())
        print(data)
        c.execute("""UPDATE Suppliers SET Cost=%s, Purchase_Date=%s WHERE Supplier_Name=%s;""",
data)
con.commit()

# Query the Suppliers table
c.execute("SELECT * FROM Suppliers")
rows = c.fetchall()
for row in rows:
        output = []
        for column_index in range(len(row)):
                output.append(str(row[column_index]))
        print(output)
```

구문 해석

2 ~ 4 행에서 csv, MySQLdb, sys 모듈을 임포트해서 MySQL 데이터베이스와 상호작용하고 CSV 파일에 검색결과를 출력할 함수를 사용할 수 있게 한다.

7 행에서 CSV 입력 파일을 input_file 변수에 할당한다.

10 행에서는 my_suppliers 데이터베이스에 접속한다.

11 행에서는 SQL 문을 실행하고 결과를 데이터베이스에 저장하기 위한 커서를 만든다.

15 ~ 23 행은 다른 코드들과 거의 동일하다.
22 행의 UPDATE 문이 INSERT 문으로 대체되었다.
UPDATE 문은 갱신할 레코드와 속성을 지정해야 한다.
이번 실습에서는 특정 Supplier Name에 대해 Cost와 Purchase Date 속성을 UPDATE 한다.

26 ~ 32 행에서는 Suppliers 테이블의 모든 행을 불러오고, 각 열 상이에 공백문자를 넣어 명령프롬프트나 터미널 창에 각 행을 출력한다.

데이터베이스 테이블에서 UPDATA할 레코드가 들어 있는 CSV 입력 파일을 생성한다.

1. 엑셀을 연다.
2. 다음과 같이 데이터를 추가한다.

	A	B	C
1	Cost	Purchase Date	Supplier Name
2	600	2014-01-22	Supplier X
3	200	2014-02-01	Supplier Y

3. data_for_updating_mysql.csv 파일로 저장한다.

스크립트 실행

cmd> python 6db_mysql_update_from_csv.py data_for_updating_mysql.csv

```
D:₩db>python 6db_mysql_update_from_csv.py data_for_updating_mysql.csv
['600.00', '2014-01-22', 'Supplier X']
['200.00', '2014-02-01', 'Supplier Y']
['Supplier X', '001-1001', '2341', '600.0', '2014-01-22']
['Supplier X', '001-1001', '2341', '600.0', '2014-01-22']
['Supplier X', '001-1001', '5467', '600.0', '2014-01-22']
['Supplier X', '001-1001', '5467', '600.0', '2014-01-22']
['Supplier Y', '50-9501', '7009', '200.0', '2014-02-01']
['Supplier Y', '50-9501', '7009', '200.0', '2014-02-01']
['Supplier Y', '50-9505', '6650', '200.0', '2014-02-01']
['Supplier Y', '50-9505', '6650', '200.0', '2014-02-01']
['Supplier Z', '920-4803', '3321', '615.0', '2014-02-17']
['Supplier Z', '920-4804', '3321', '615.0', '2014-02-17']
['Supplier Z', '920-4805', '3321', '615.0', '2014-02-17']
['Supplier Z', '920-4806', '3321', '615.0', '2014-02-24']
```

출력결과의 처음 두 행은 CSV 파일에 있는 데이터이고, 나머지는 행들을 갱신한 후 데이터베이스 테이블로부터 가져온 데이터이다.

CSV 파일의 헤더 행을 제외한 두 행으로부터 생성된 2개의 리스트를 보여준다.
리스트는 화면에 출력될 때 각 값들이 쉼표로 구분되고 대괄호로 묶여 표현된다.
Supplier X의 Cost 값은 600 이고, Purchase Date는 2014-01-22 이다.
Supplier Y의 Cost 값은 200 이고, Purchase Date는 2014-02-01 이다.

두 리스트 아래에는 갱신 후 데이터베이스 테이블로부터 가져온 12행의 데이터가 있다.
각 행은 새로운 행에 출력되고, 각 행의 값은 쉼표로 구분된다.

MariaDB 데이터베이스에서 UPDATE 된 정보를 확인한다.

MariaDB [my_suppliers]> SELECT * FROM Suppliers;

```
MariaDB [my_suppliers]> SELECT * FROM Suppliers;
+---------------+----------------+-------------+------+---------------+
| Supplier_Name | Invoice_Number | Part_Number | Cost | Purchase_Date |
+---------------+----------------+-------------+------+---------------+
| Supplier X    | 001-1001       | 2341        |  600 | 2014-01-22    |
| Supplier X    | 001-1001       | 2341        |  600 | 2014-01-22    |
| Supplier X    | 001-1001       | 5467        |  600 | 2014-01-22    |
| Supplier X    | 001-1001       | 5467        |  600 | 2014-01-22    |
| Supplier Y    | 50-9501        | 7009        |  200 | 2014-02-01    |
| Supplier Y    | 50-9501        | 7009        |  200 | 2014-02-01    |
| Supplier Y    | 50-9505        | 6650        |  200 | 2014-02-01    |
| Supplier Y    | 50-9505        | 6650        |  200 | 2014-02-01    |
| Supplier Z    | 920-4803       | 3321        |  615 | 2014-02-17    |
| Supplier Z    | 920-4804       | 3321        |  615 | 2014-02-17    |
| Supplier Z    | 920-4805       | 3321        |  615 | 2014-02-17    |
| Supplier Z    | 920-4806       | 3321        |  615 | 2014-02-24    |
+---------------+----------------+-------------+------+---------------+
12 rows in set (0.000 sec)
```