

Database

◆ Python과 데이터베이스 연동

정수아

Contents

01 데이터베이스 작업하기

02 Python과 연동하기

03 데이터 추가 및 수정하기

04 동적 SQL 만들기



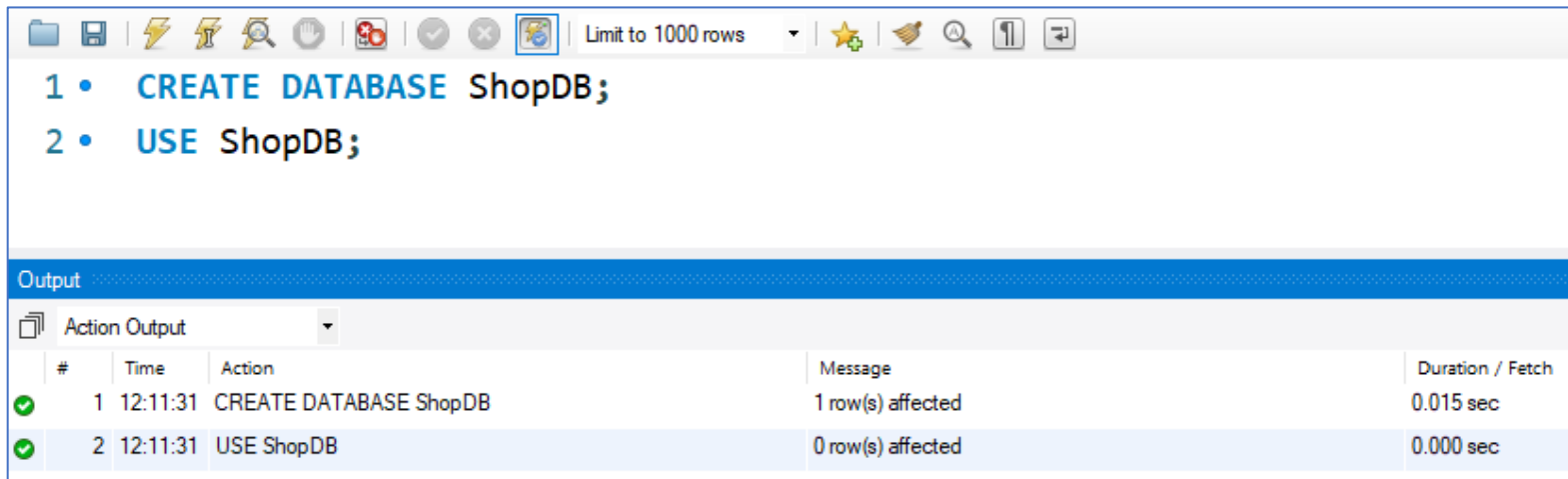
01

데이터베이스 작업하기

데이터베이스 생성하기

❖ 'ShopDB' 데이터베이스 생성

```
$ CREATE DATABASE ShopDB;
```



The screenshot shows a database management tool interface. The top toolbar includes icons for file operations, execution, and search. The main area displays two SQL commands:

- 1 • **CREATE DATABASE** ShopDB;
- 2 • **USE** ShopDB;

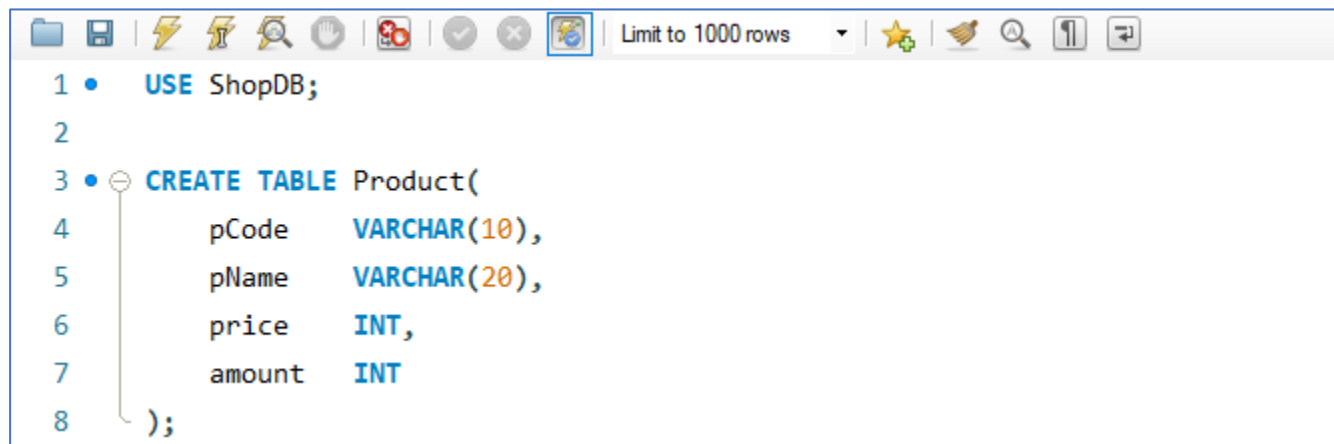
Below the commands is an "Output" section with a table showing the execution results:

#	Time	Action	Message	Duration / Fetch
✓ 1	12:11:31	CREATE DATABASE ShopDB	1 row(s) affected	0.015 sec
✓ 2	12:11:31	USE ShopDB	0 row(s) affected	0.000 sec

테이블 생성하기

❖ 'Product' 테이블 생성

```
CREATE TABLE Product(  
    pCode    VARCHAR(10),  
    pName    VARCHAR(20),  
    price    INT,  
    amount   INT  
);
```



데이터 추가하기

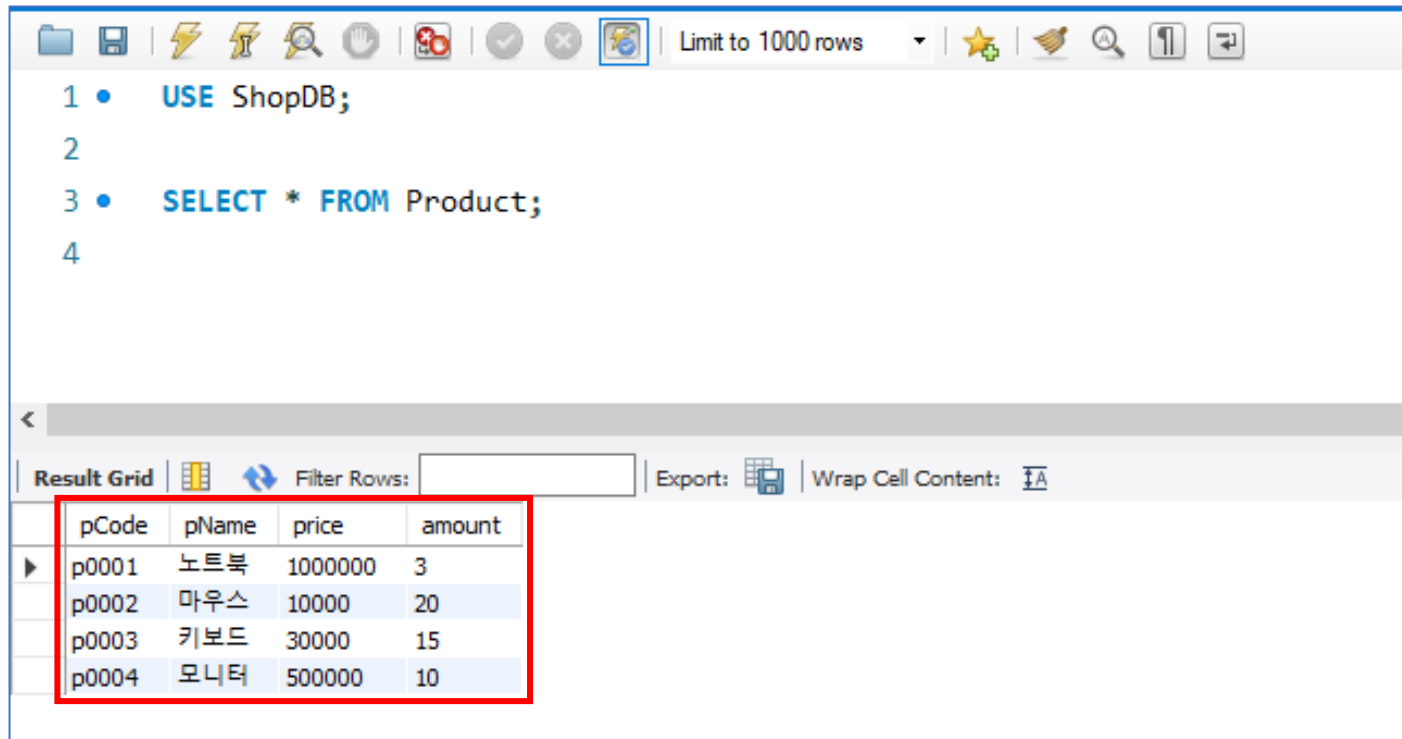
❖ 'Product' 테이블에 데이터 추가

pCode	pName	price	amount
p0001	노트북	1000000	3
p0002	마우스	10000	20
p0003	키보드	30000	15
p0004	모니터	500000	10

데이터 조회하기

❖ 'Product' 테이블에 저장된 전체 데이터 조회

```
SELECT * FROM Product;
```



The screenshot shows a database query tool interface. The top toolbar includes icons for file operations, execution, and settings. The main area displays the following SQL script:

```
1 • USE ShopDB;  
2  
3 • SELECT * FROM Product;  
4
```

Below the script, the 'Result Grid' tab is active, showing the query results. The grid has columns for pCode, pName, price, and amount. The data is as follows:

pCode	pName	price	amount
p0001	노트북	1000000	3
p0002	마우스	10000	20
p0003	키보드	30000	15
p0004	모니터	500000	10



02

Python과 연동하기

라이브러리 설치

❖ PyMySQL 라이브러리

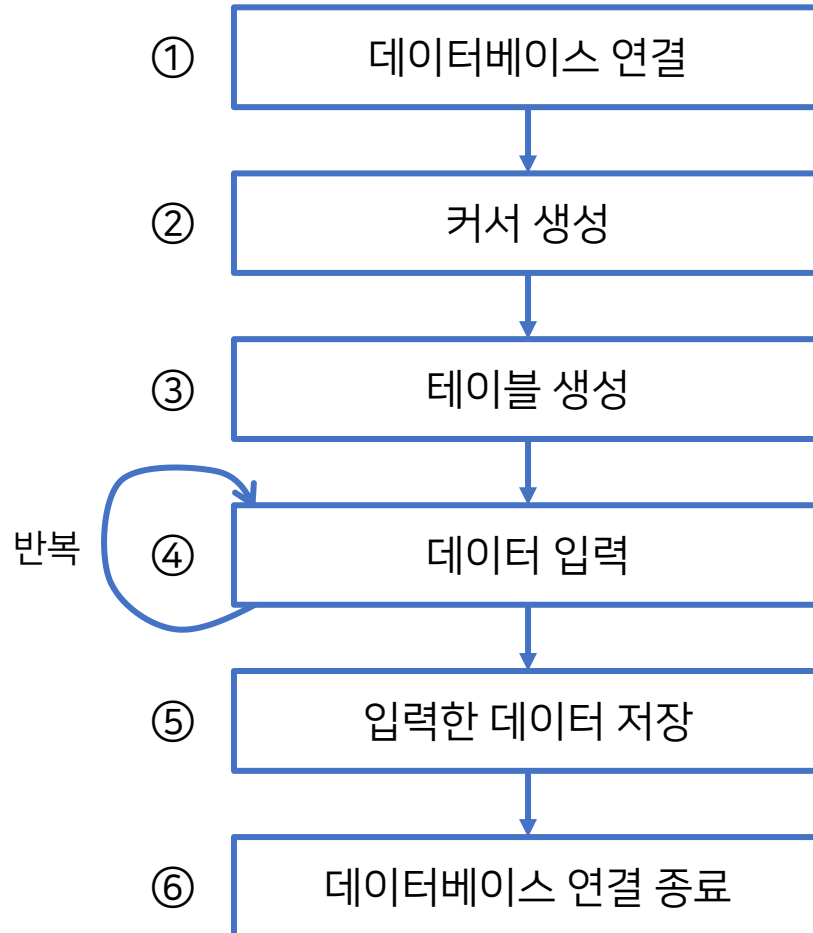
- Python이 MySQL 데이터베이스와 원활하게 상호 작용할 수 있도록 도와주는 라이브러리

❖ 설치

```
$ pip install PyMySQL
```

데이터 입력과 조회

❖ 데이터 입력 순서



데이터베이스 연결

❖ 데이터베이스 연결

- PyMySQL 라이브러리 import

```
import pymysql
```

데이터베이스 연결

❖ 데이터베이스 연결

- connect() 함수로 데이터베이스와 연동하여 연결자 생성

```
연결자 = pymysql.connect(옵션)
```

- 옵션

항목	설명
host	서버 IP 주소
user	사용자
password	비밀번호
db	데이터베이스 이름
charset	인코딩

[실습] 데이터베이스 연결

❖ shopDB와 연결하기

```
import pymysql

conn = pymysql.connect(host='localhost',
                       user='root',
                       password='root',
                       db='ShopDB',
                       charset='utf8')
```

커서 생성

❖ 커서(Cursor)란?

- 데이터베이스에 SQL문을 실행하거나 실행된 결과를 돌려받는 통로
- 앞에서 만든 연결자에 cursor() 함수를 연결하여 생성

```
커서 = 연결자.cursor()
```

- 커서 생성하기

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

SQL문 실행

❖ SQL문 실행 방법

- 실행할 SQL문을 커서.execute() 함수에 매개변수로 전달하면 SQL문이 데이터베이스에 실행됨

```
커서.execute("SQL문")
```

❖ 예시

- 테이블 생성 SQL문 실행

```
cur.execute("테이블 생성 SQL문")
```

- 테이블 조회 SQL문 실행

```
cur.execute("테이블 조회 SQL문")
```

[실습] shopDB 데이터 조회하기

```
# 라이브러리 추가
import pymysql

# 연결자 생성
conn = pymysql.connect(host='localhost', user='root',
                       password='root', db='ShopDB', charset='utf8')

# 커서 생성
curs = conn.cursor()

# SQL문 실행
sql = "SELECT * FROM Product"
curs.execute(sql)
```


[실습] shopDB 데이터 조회하기

❖ 데이터 조회

함수	설명
fetchone()	cursor에 저장된 데이터를 한 행 씩 추출
fetchmany(size)	cursor에 저장된 데이터를 size개의 행 추출
fetchall()	cursor에 저장된 데이터를 모두 추출

[실습] shopDB 데이터 조회하기 - fetchall()

```
# 생략
```

```
# SQL문 실행
```

```
sql = "SELECT * FROM Product"
```

```
curs.execute(sql)
```

```
# 모든 데이터 가져오기
```

```
result = curs.fetchall()
```

```
# result 타입 확인
```

```
print(type(result))
```

❖ 실행 결과

```
<class 'tuple'>
```

[실습] shopDB 데이터 조회하기 - fetchall()

```
# 생략
```

```
# 모든 데이터 가져오기  
result = curs.fetchall()
```

```
# result 타입 확인  
print(type(result))
```

```
# 출력  
for data in result:  
    print(data)
```

❖ 실행 결과

```
<class 'tuple'>  
( 'p0001', '노트북', 1000000, 3)  
( 'p0002', '마우스', 10000, 20)  
( 'p0003', '키보드', 30000, 15)  
( 'p0004', '모니터', 500000, 10)
```

[실습] shopDB 데이터 조회하기 - fetchmany(n)

```
# 생략
```

```
# 2줄의 데이터만 가져오기  
result = curs.fetchmany(2)
```

```
# 출력  
for data in result:  
    print(data)
```

❖ 실행 결과

```
('p0001', '노트북', 1000000, 3)  
('p0002', '마우스', 10000, 20)
```

[실습] shopDB 데이터 조회하기 - fetchone()

```
# 생략
```

```
# 1줄의 데이터만 가져오기  
result = curs.fetchone()
```

```
# 출력  
print(result)
```

❖ 실행 결과

```
('p0001', '노트북', 1000000, 3)
```

fetch 함수 - 추가1

```
# 라이브러리 추가
# 연결자 생성
# 커서 생성

# SQL문 실행
sql = "SELECT * FROM Product"
curs.execute(sql)

# 모든 데이터 가져오기
result = curs.fetchall()
print("데이터 출력 : ", result)

# 데이터 1개 가져오기
result = curs.fetchone()
print("데이터 출력 : ", result)
```

fetch 함수 - 추가1

❖ 실행 결과

```
데이터 출력 : (('p0001', '노트북', 1000000, 3),  
               ('p0002', '마우스', 10000, 20),  
               ('p0003', '키보드', 50000, 15),  
               ('p0004', '모니터', 500000, 10))  
데이터 출력 : None
```

❖ fetchone()의 결과가 None인 이유

- fetchall() 함수가 cursor에 저장된 모든 데이터를 출력했기 때문

fetch 함수 - 추가2

```
# 라이브러리 추가
# 연결자 생성
# 커서 생성

# SQL문 실행
sql = "SELECT * FROM Product"
curs.execute(sql)

# 데이터 1개 가져오기
result = curs.fetchone()
print("데이터 출력 : ", result)

# 모든 데이터 가져오기
result = curs.fetchall()
print("데이터 출력 : ", result)
```


fetch 함수 - 추가2

❖ 실행 결과

```
데이터 출력 : ('p0001', '노트북', 1000000, 3)
데이터 출력 : (('p0002', '마우스', 10000, 20),
               ('p0003', '키보드', 50000, 15),
               ('p0004', '모니터', 500000, 10))
```

데이터베이스 연결 종료

❖ 데이터베이스 연결 종료

- 데이터베이스의 사용이 끝난 경우, 연결한 데이터베이스를 닫아 주어야 함

- 방법

```
연결자.close()
```

데이터베이스 연결 종료

라이브러리 추가

연결자 생성

커서 생성

SQL문 실행

데이터베이스 연결 종료

`conn.close()`



03

데이터 추가 및 수정하기

데이터 추가하기

❖ 데이터 추가

```
curs.execute("데이터 추가 SQL문")
```

- 실행한 SQL문은 데이터베이스에 임시로 저장되어 있는 상태
- 데이터베이스에 최종 저장하려면 commit을 실행해야 함

❖ commit 실행

```
연결자.commit()
```

[실습] 데이터 추가하기

pCode	pName	price	amount
p0005	핸드폰	800000	5

```
# 라이브러리 추가
# 연결자 생성
# 커서 생성

# SQL문 실행
sql = "INSERT INTO Product(pCode, pName, price, amount) VALUES('p0005', '핸드폰',
    800000, 5)"
curs.execute(sql)

# commit 실행
conn.commit()

# 데이터베이스 연결 종료
conn.close()
```

데이터 수정하기

❖ 데이터 수정

```
curs.execute("데이터 수정 SQL문")
```

- 실행한 SQL문은 데이터베이스에 임시로 저장되어 있는 상태
- 데이터베이스에 최종 저장하려면 commit을 실행해야 함

[실습] 데이터 수정하기

❖ 'Product' 테이블 기존 데이터

pCode	pName	price	amount
p0001	노트북	1000000	3
p0002	마우스	10000	20
p0003	키보드	30000	15
p0004	모니터	500000	10
p0005	핸드폰	800000	5

- 제품번호 p0003 키보드의 가격을 50000원으로 수정

[실습] 데이터 수정하기

pCode	pName	price	amount
p0003	키보드	50000	15

```
# 라이브러리 추가
# 연결자 생성
# 커서 생성

# SQL문 실행
sql = "UPDATE Product SET price=50000 WHERE pCode='p0003'"
curs.execute(sql)

# commit 실행
conn.commit()

# 데이터베이스 연결 종료
conn.close()
```

[실습] 데이터 수정하기

❖ 실행 결과

```
('p0001', '노트북', 1000000, 3)
('p0002', '마우스', 10000, 20)
('p0003', '키보드', 50000, 15)
('p0004', '모니터', 500000, 10)
('p0005', '핸드폰', 800000, 5)
```

데이터 삭제하기

❖ 데이터 삭제

```
curs.execute("데이터 삭제 SQL문")
```

- 실행한 SQL문은 데이터베이스에 임시로 저장되어 있는 상태
- 데이터베이스에 최종 저장하려면 commit을 실행해야 함

[실습] 데이터 삭제하기

pCode	pName	price	amount
p0005	핸드폰	800000	5

```
# 라이브러리 추가
# 연결자 생성
# 커서 생성

# SQL문 실행
sql = "DELETE FROM Product WHERE pCode='p0005'"
curs.execute(sql)

# commit 실행
conn.commit()

# 데이터베이스 연결 종료
conn.close()
```

[실습] 데이터 삭제하기

❖ 실행 결과

```
('p0001', '노트북', 1000000, 3)  
( 'p0002', '마우스', 10000, 20)  
( 'p0003', '키보드', 50000, 15)  
( 'p0004', '모니터', 500000, 10)
```



04

동적 SQL 만들기

동적 SQL 만들기

❖ SQL문에 변수 사용

- execute() 함수에 매개변수 전달
 - 매개변수는 튜플(tuple) 형식으로 전달됨
- 예) pCode가 p0002인 상품 정보 조회

```
sql = "SELECT * FROM Product WHERE pCode = %s"  
curs.execute(sql, ('p0002'))
```

[실습] 동적 SQL 만들기 - 1

❖ 가격이 50만원~100만원 사이의 제품을 검색

pCode	pName	price	amount
p0001	노트북	1000000	3
p0004	모니터	500000	10

[실습] 동적 SQL 만들기 - 2

❖ 가격이 10만원 미만인 제품들의 평균가와 수량의 합을 계산

pCode	pName	price	amount
p0002	마우스	10000	20
p0003	키보드	50000	15

- 실행 결과

```
(Decimal('30000.0000'), Decimal('35'))
```

THANK 😊 YOU