

Python과 데이터베이스

Python DB API

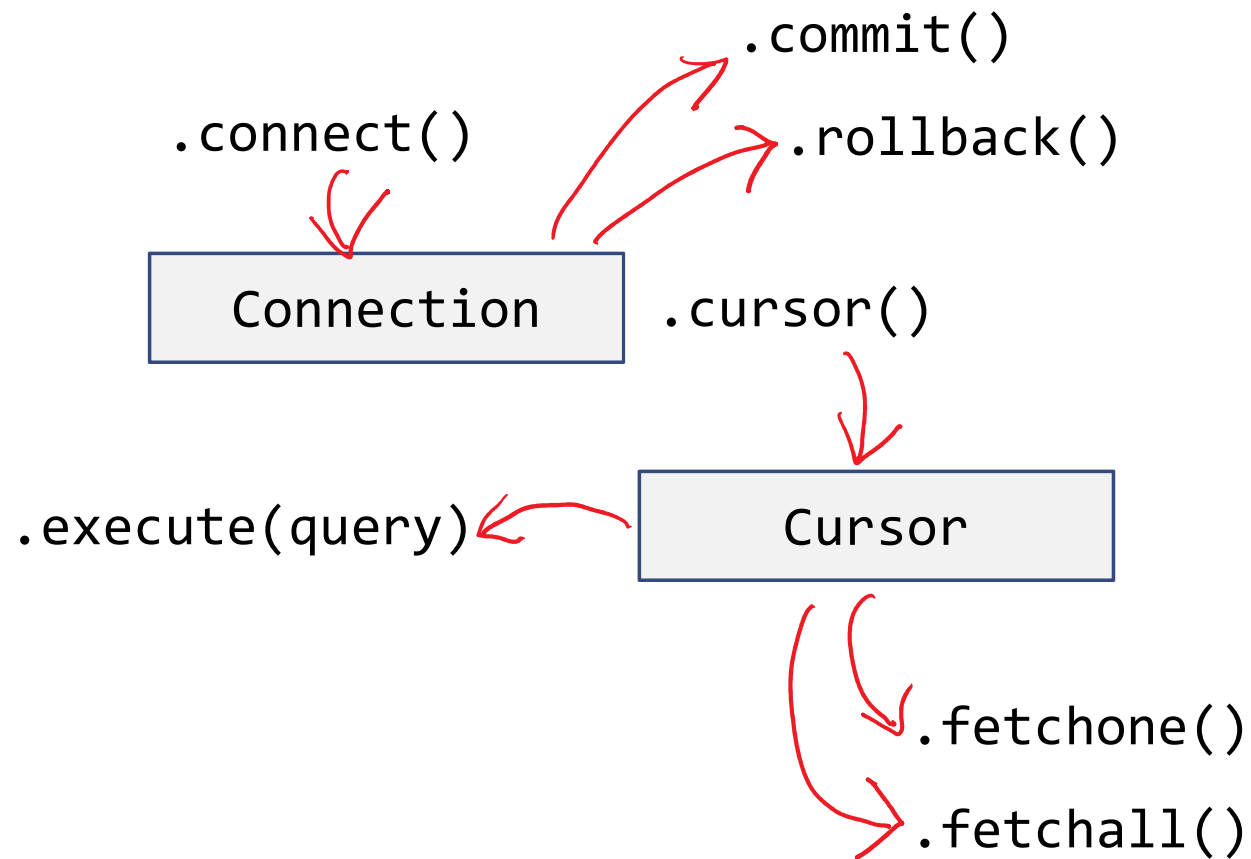
▶ Python DB API

- ▶ 여러 데이터베이스를 액세스하는 표준 **API**
 - ▶ 여러 데이터베이스 액세스 모듈에서 따르는 최소한의 **API** 인터페이스 표준
 - ▶ 현재 버전 **2.0**을 사용하며, 스펙은 **PEP 249**에 소개

▶ DBMS 시스템을 이용하기 위해서는 각각 **DB**에 상응하는 모듈을 다운 받아야 함

- ▶ 하지만 대부분 **Python DB API** 표준을 따르고 있어 동일한 방식으로 **DBMS**를 사용할 수 있음
- ▶ **MySQL**, **PostgreSQL**, **MSSQL**, **Oracle**, **Sybase**, **Informix**, **mSQL** 등 대표적 **DB** 모두 지원
- ▶ **SQLite**는 **Embedded SQL DB** 엔진으로 **Python 2.5** 이상에 기본 내장

Python DB API



SQLite

▶ SQLite

- ▶ Embedded SQL DB 엔진으로 별도의 DB 서버가 필요 없는 파일 기반 엔진
- ▶ 별도의 설치가 필요 없고, 쉽고 편리하게 사용할 수 있다는 점에서 널리 사용
- ▶ Mac OS X, Linux, Android 등에 기본으로 내장
- ▶ Windows 등, 기본 내장되어 있지 않은 OS의 경우 다음 경로에서 다운로드 받아 사용
 - ▶ <https://sqlite.org/download.html>
 - ▶ sqlite-tools-win32-x86-xxxxxxx.zip
 - ▶ SQLite Viewer for Windows : <https://sqlitebrowser.org/>

SQLite

: 데이터베이스 접속

▶ SQLite 접속과 해제

- ▶ `sqlite3 import`
- ▶ 접속 : `.connect(데이터베이스명)`
- ▶ 해제 : `.close()`

▶ SQLite3는 `connect`시 데이터베이스가 없으면 자동으로 생성

```
import sqlite3
from sqlite3 import Error

# SQLite DB 연결
conn = sqlite3.connect("test.db")

# Sqlite DB 연결 해제
conn.close()
```

SQLite

: 쿼리의 수행

▶ 쿼리의 수행

- ▶ Connection 객체의 `execute()` 메서드로 실행
- ▶ 쿼리 수행 결과로 `Cursor`를 반환

▶ 반환 받은 결과는 다음 메서드로 확인

- ▶ `fetchone` : 결과 하나를 반환
- ▶ `fetchmany` : 결과 여러 개를 반환
- ▶ `fetchall` : 전체 결과를 반환

```
conn = sqlite3.connect("./database/mysqlite.db")  
  
cursor = conn.execute("SELECT * FROM customer")  
  
for customer in cursor.fetchall():  
    print(customer)
```

▶ 수행 결과 영향을 받은 레코드의 개수는 커서의 `.rowcount` 필드로 확인

SQLite

: commit과 rollback

- ▶ 데이터가 추가(**INSERT**)되거나, 변경(**UPDATE**)되거나, 삭제(**DELETE**)되어도 실제 데이터베이스에 반영되지는 않음
 - ▶ 해당 변화 반영을 위해서는 `.commit()`
 - ▶ 해당 변화 취소를 위해서는 `.rollback()`

SQLite

: Parameterized Query

- ▶ 실제 개발에서는 **SQL** 문장을 **String** 연결로 하기보다 포맷을 만들어 두고 동적으로 컬럼 데이터 값을 집어 넣는 경우가 많음
 - ▶ Parameterized Query를 이용하면 편리
 - ▶ Anonymous Placeholder : tuple로 바인딩

```
conn.execute("SELECT * FROM customer WHERE category=?", (1,))
```

- ▶ Named Placeholder : dict로 바인딩

```
conn.execute("SELECT * FROM customer WHERE category=:cat", {"cat": 2})
```


MySQL

:pymysql

- ▶ pymysql 모듈은 MySQL 데이터베이스를 파이썬에서 사용하기 위한 모듈 중 하나
 - ▶ Python DB API 표준을 따르므로 다른 데이터베이스 모듈과 사용 방식이 거의 같다
- ▶ pymysql 모듈의 설치

```
pip install pymysql
```

- ▶ pymysql 연결

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

MySQL

:pymysql

▶ 기본적인 SELECT 문의 수행

```
# Connection으로부터 Cursor 생성
cursor = conn.cursor()

# SQL문 실행
sql = "SELECT * FROM actor"
cursor.execute(sql)

# Data Fetch
rows = cursor.fetchall()

# Print
for actor in rows:
    print(actor)

# Connection Close
conn.close()
```

MySQL

:pymysql

▶ Parameter Placeholder

- ▶ 동적으로 컬럼 데이터 값을 집어넣고자 할 때 **Parameter Placeholder %s**를 사용
 - ▶ 일반 문자열 포매팅에 사용되는 %s 와는 다른 것이며, 매칭될 값은 튜플로 넣어준다
 - ▶ Placeholder 문자는 컬럼 값을 대치할 때만 사용할 수 있으며 SQL 문장 다른 부분에 Placeholder를 사용할 수는 없음

```
...  
cursor = conn.cursor()  
sql = "SELECT * FROM actor WHERE last_name = %s or last_name = %s"  
cursor.execute(sql, ("PENN", "GRANT"))  
...
```

MySQL

:pymysql

▶ Dictionary Cursor

- ▶ Cursor를 획득할 때, 파라미터로 DictCursor를 지정하면 결과 셋을 사전 형태로 반환 받을 수 있다

```
...  
cursor = conn.cursor(pymysql.cursors.DictCursor)  
sql = "SELECT * FROM actor WHERE last_name = %s or last_name = %s"  
cursor.execute(sql, ("PENN", "GRANT"))  
...
```

PEEWEE

▶ TODO: 내용 작성