Python 연동

Maria DB 연동

- Python DB API
  - ✓ Python에서 데이터베이스를 엑서스하기 위해서는 Python DB API를 사용
  - ✓ Python DB API는 데이터베이스를 엑세스하는 표준 API 로서 여러 DB 엑세스 모듈에서 이 최소한의 API 인터페이스 표준을 따름
  - ✓ 표준 API는 크게 데이터베이스를 연결하고, SQL 문을 실행하고, 연결을 닫는 등의 기본적인 DB 작업과 관련된 기능들을 정의
  - ✓ 데이터베이스를 엑세스하는 또 다른 방식으로 ORM(Object Relational Mapping)이 있는데 Django ORM, PonyORM, peewee 등이 ORM 방식을 사용한 데이터 엑세스를 제공
  - ✓ Python에서 지원하는 데이터베이스는 매우 다양하기 때문에 Python에서 각 데이터베이스를 사용하기 위해서는 각각의 DB에 상응하는 별도의 DB 모듈을 다운받아야 함(sqlite3 는 Python 2.5 이상에서 기본 내장).
  - ✓ 수많은 DB 모듈들이 있지만 이들이 거의 모두 Python DB API 표준을 따르고 있으므로 동일한 API를 사용해 데이터베이스를 사용할 수 있음
  - ✓ python에서는 거의 모든 데이터베이스를 사용할 수 있는데 MySQL, PostgreSQL, MSSQL, Sqlite, Oracle, Sybase, Informix 등과 같은 대표적인 DB 들을 모두 지원
  - ✓ MariaDB 연동 모듈 설치

    pip install pyMySQL

Maria DB 연동

❖ 데이터베이스 접속

```
import pymysql
#데이터베이스 연결
con = pymysql.connect(host='데이터베이스 위치', port=포트번호, user='계정', passwd='비밀번호', db='데이터베이스 이름', charset ='인코딩방식')
print(con)
con.close()
```

✓ 데이터베이스 접속이 되지 않는 경우 아래와 같은 예외 발생

```
Traceback (most recent call last):

  File "C:₩Users₩Administrator₩python₩test₩__main__.py", line 12, in <module>

exception: (<class 'pymysql.err.OperationalError'>, OperationalError(2003, "Can't connect to MySQL server on '211.183.2.253' ([WinError 10060] 연결된 구성원으로부터 응답이 없어 연결하지 못했거나, 호스트로부터 응답이 없어 연결이 끊어졌습니다)"), <traceback object at 0x00C839E0>)
    if con != None:
NameError: name 'con' is not defined
```

Maria DB 연동

❖ Maria DB에 접속해서 테이블 생성

```
create database sample;

use sample;

CREATE TABLE usertbl(
userid char(15) not null primary key,
name varchar(20) not null,
birthyear int not null,
addr char(100),
mobile char(11),
mdate date);

insert into usertbl values('ailee', '에일리',1989,'미국','01000000000', '1989-5-30');

commit;

select * from usertbl;
```

Maria DB 연동

- 파이썬에서 DML 수행
  - ✓ 연결 객체의 cursor() 메서드를 호출해서 sql 실행 객체를 생성
  - ✓ execute(실행 할 sql문장)
    - 데이터는 직접 설정 가능
    - SQL 문장에 %로 작성한 후 다음 매개변수를 튜플로 해서 튜플에 값을 대입하는 것도 가능 - 권장
  - ✓ 연결 객체의 commit() 을 호출하면 작업 내용이 반영되고 rollback()을 호출하면 작업 취소

Maria DB 연동

❖ 데이터 삽입

```
import sys, pymysql

#데이터베이스 연결
try:
    con = pymysql.connect(host='localhost', port=3306, user='root', password='wnddkd', database='sample', charset ='utf8mb4')
    cursor = con.cursor()
    cursor.execute("insert into usertbl values('ljy', '이진연', 1970, '서울', '01012345678', '1970-10-31')")
    #cursor.execute("insert into usertbl values(%s, %s, %s, %s, %s, %s)",  ('sohye','김소혜', 1999, '서울', '01012121212','1999-7-19'))
    con.commit()
    print("삽입 성공")
except:
    print('exception:', sys.exc_info())
finally:
    con.close()
```

Maria DB 연동

❖ 데이터 수정 과 삭제

```
import sys, pymysql

#데이터베이스 연결
try:
    con = pymysql.connect(host='localhost', port=3306, user='root', password='wnddkd', database='sample', charset ='utf8mb4')
    cursor = con.cursor()
    cursor.execute("update usertbl set name='이지연' where name = '이진연'")
    cursor.execute("delete from usertbl where name = '구하라'")
    con.commit()
    print("수정 성공")

except:
    print('exception:', sys.exc_info())
finally:
    con.close()
```

Maria DB 연동

- 데이터 검색
  - ✓ 연결 객체의 cursor() 메서드를 호출해서 sql 실행 객체 생성
  - ✓ execute(실행 할 sql문장)
  - ✓ cursor 객체를 가지고 fetchall 메서드를 호출하면 튜플들의 튜플로 결과가 리턴되며 fetchone 메서드를 호출하면 첫번째 데이터 1개만 튜플로 리턴

Maria DB 연동

❖ 데이터 검색

```
import sys, pymysql

#데이터베이스 연결
try:
    con = pymysql.connect(host='localhost', port=3306, user='root', password='wnddkd',
database='sample', charset ='utf8mb4')
    cursor = con.cursor()
    cursor.execute("select * from usertbl")
    data = cursor.fetchone()
    print(data)
    for imsi in data:
        print(imsi)

except:
    print('exception:', sys.exc_info())
finally:
    con.close()
```

Maria DB 연동

❖ 데이터 검색

```
import sys, pymysql

#데이터베이스 연결
try:
    con = pymysql.connect(host='localhost', port=3306, user='root', password='wnddkd',
database='sample', charset ='utf8mb4')
    cursor = con.cursor()
    cursor.execute("select * from usertbl")
    data = cursor.fetchone()
    print(data)
    for imsi in data:
        print(imsi)

except:
    print('exception:', sys.exc_info())
finally:
    con.close()
```

Maria DB 연동

❖ 데이터 검색

```
import sys, pymysql

#데이터베이스 연결
try:
    con = pymysql.connect(host='localhost', port=3306, user='root', password='wnddkd',
database='sample', charset ='utf8mb4')
    cursor = con.cursor()
    cursor.execute("select * from usertbl")
    data = cursor.fetchall()
    for imsi in data:
        print(imsi)
except:
    print('exception:', sys.exc_info())
finally:
    con.close()
```

Maria DB 연동

❖ 프로시저 연동

✓ 프로시저 생성

```
DELIMITER //
 CREATE PROCEDURE myproc(IN _userid CHAR(15),
  IN _name VARCHAR(20), _birthyear int, _addr CHAR(10), _mobile char(11), _mdate date)
   BEGIN
   insert into usertbl values(_userid, _name, _birthyear, _addr, _mobile, _mdate);
   END //
DELIMITER ;
```

Maria DB 연동

- 프로시저 연동
  - ✓ 프로시저 실행

```
import sys, pymysql

#데이터베이스 연결
try:
    con = pymysql.connect(host='localhost', port=3306, user='root', password='wnddkd', database='sample', charset ='utf8mb4')
    cursor = con.cursor()
    cursor.callproc('myproc', args=('momo', '모모', 1996, '교토', '01098765432', '1996-11-9'))
    con.commit()
except:
    print('exception:', sys.exc_info())
finally:
    con.close()
```

Maria DB 연동

- BLOB 연동
  - ✓ 테이블 생성

```
create table member(userid char(15) not null primary key,filename
varchar(1000),filecontent longblob);


select * from member;
```

Maria DB 연동

- BLOB 연동
  - ✓ BLOB 저장

```
import sys, pymysql
#자신의 이미지 파일 경로를 사용하세요
# read file
filename = './image1.png'
f =  open(filename, 'rb')
photo = f.read()
f.close()
sp = filename.split('/')
query = 'insert into member values(%s, %s, %s)'
args = ('태연', sp[len(sp)-1], photo)
```

Maria DB 연동

- BLOB 연동
  - ✓ BLOB 저장

```
try:
    con = pymysql.connect(host='localhost',
    port=3306, user='root', password='wnddkd', database='sample', charset ='utf8mb4')
    cursor = con.cursor()
    cursor.execute(query, args)
    con.commit()
    print("삽입성공")
except:
    print('exception:', sys.exc_info())
finally:
    cursor.close()
    con.close()
```

Maria DB 연동

- BLOB 연동
  - ✓ BLOB 읽기

```
import sys, pymysql
try:
    con = pymysql.connect(host='localhost',
    port=3306, user='root', password='wnddkd', database='sample', charset ='utf8mb4')
    cursor = con.cursor()
    cursor.execute("select * from member")
    data = cursor.fetchall()
    for imsi in data:
        print(imsi[0])
        print(imsi[1])
        f = open(imsi[1], 'wb')
        f.write(imsi[2])
        f.close()
```

Maria DB 연동

- BLOB 연동
  - ✓ BLOB 읽기

```
except:
    print('exception:', sys.exc_info())
finally:
    cursor.close()
    con.close()
```

ORM

- 개요(Object Relational Mapping)
  - ✓ 객체 지향 패러다임을 관계형 데이터베이스에 보존하는 기술
  - ✓ 객체와 관계형 데이터베이스의 테이블을 매핑해서 사용하는 방법
  - ✓ 관계형 데이터베이스에서는 Table(Table)을 설계하는데 새로운 Table에는 칼럼을 정의하고 칼럼에 맞는 데이터 타입을 지정해서 데이터를 보관하는 틀을 만든다는 의미에서 Class와 상당히 유사
  - ✓ 클래스는 데이터베이스의 테이블과 매핑하기 위해 만들어진 것이 아니기 때문에 RDB 테이블과 어쩔 수 없는 불일치가 존재하는데 ORM은 이 둘의 불일치와 제약사항을 해결하는 역할을 담당

ORM

❖ 개요

- ✓ Class 와 Table이 유사하듯이 Instance 와 Row(Record, Tuple)도 유사하며 객체 지향에서는 Class를 이용해서 객체(Instance)를 생성해서 데이터를 보관하는데 관계형 데이터베이스에서는 Table에 Row를 이용해서 데이터를 저장하며 차이는 객체 라는 단어가 데이터 + 행위(메서드)라는 의미라면 Row는 데이터 만을 의미
- ✓ 관계(relation) 와 참조(reference) 라는 의미도 유사한데 관계형 데이터베이스는 Table 사이의 관계를 통해서 구조적인 데이터를 표현한다면 객체 지향에서는 참조를 통해서 어떤 객체가 다른 객체들과 어떤 관계를 맺고 있는지를 표현
- ✓ 객체 지향과 관계형 데이터베이스는 유사한 특징을 가지고 있어서 객체 지향을 자동으로 관계형 데이터베이스에 맞게 처리해 주는 기법에 대해서 아이디어를 내기 시작했고 그것이 ORM의 시작
- ✓ ORM은 완전히 새로운 패러다임을 주장하는 것이 아니라 객체 지향 과 관계형 사이의 변환 기법을 의미하는 것으로 특정 언어에 국한되는 개념이 아니고 관계형 패러다임을 가지고 있다면 데이터베이스의 종류를 구분하지 않음
- ✓ 대다수의 객체 지향 언어에는 ORM을 위한 여러 프레임워크들이 존재

ORM

- ❖ 개요
  - ✓ 장점
    - 특정 데이터베이스에 종속되지 않음
    - 객체 지향적 프로그래밍
    - 생산성 향상
  - ✓ 단점
    - 복잡한 쿼리 처리
    - 성능 저하 위험
    - 학습 시간

ORM

- SQLAlchemy
  - ✓ SQLAlchemy는 파이썬에서 가장 많이 사용되는 ORM(Object-Relational Mapping) 라이브러리
  - ✓ 다른 ORM 라이브러리와 다른 점 중 하나로는 자체적으로 스키마를 생성하지 않는다는 것인데 그렇기 때문에 애플리케이션 코드나 데이터베이스 시스템에 간섭하지 않는다는 특징이 있음
  - ✓ 패키지: sqlalchemy

ORM

- SQLAlchemy
  - ✓ SQLAlchemy는 파이썬에서 가장 많이 사용되는 ORM(Object-Relational Mapping) 라이브러리
  - ✓ 다른 ORM 라이브러리와 다른 점 중 하나로는 자체적으로 스키마를 생성하지 않는다는 것인데 그렇기 때문에 애플리케이션 코드나 데이터베이스 시스템에 간섭하지 않는다는 특징이 있음
  - ✓ 패키지: sqlalchemy

ORM

- SQLAlchemy
  - ✓ 데이터 삽입 과 조회

```
from sqlalchemy import create_engine, Column, Integer, String
from sqlalchemy.orm import declarative_base, sessionmaker


# 1. 연결 URL 설정
# 형식: mysql+pymysql://사용자명:비밀번호@호스트/DB이름?charset=utf8mb4
DB_URL = "mysql+pymysql://root:wnddkd@localhost:3306/sample?charset=utf8mb4"


# 2. Engine 생성 (DB와의 실제 연결 통로)
engine = create_engine(DB_URL, echo=True)  # echo=True는 실행되는 SQL문을 터미널에 출력함


# 3. 세션 설정 (DB 작업을 위한 단위)
SessionLocal = sessionmaker(autocommit=False, autoflush=False, bind=engine)
```

ORM

- SQLAlchemy
  - ✓ 데이터 삽입 과 조회

```
# 4. 모델 생성을 위한 기본 클래스 (Base)
Base = declarative_base()

# 5. 테이블 모델 정의
class User(Base):
    __tablename__ = "users"

    id = Column(Integer, primary_key=True, autoincrement=True)
    name = Column(String(50), nullable=False)
    email = Column(String(100), unique=True)

# 6. DB에 테이블 실제 생성 (이미 있으면 생성 안 함)
Base.metadata.create_all(bind=engine)
```

ORM

- SQLAlchemy
  - ✓ 데이터 삽입 과 조회

```
# 세션 열기
db = SessionLocal()
try:
    # [C] 데이터 추가
    new_user = User(name="홍길동", email="hong@example.com")
    db.add(new_user)
    db.commit()  # 커밋해야 실제 DB에 반영됨
    print(f"사용자 생성 완료: {new_user.name}")

    # [R] 데이터 조회
    user = db.query(User).filter(User.name == "홍길동").first()
    print(f"조회된 사용자: {user.name} ({user.email})")

finally:
    # 세션 닫기
    db.close()
```

ORM

- SQLAlchemy
  - ✓ 최근 권장하는 모델 생성

```
from sqlalchemy.orm import Mapped, mapped_column


class User(Base):
    __tablename__ = "users"

    # 2.0 스타일: 타입 힌트로 더 안전하게 정의
    id: Mapped[int] = mapped_column(primary_key=True)
    name: Mapped[str] = mapped_column(String(50))
    email: Mapped[str] = mapped_column(String(100), unique=True)
```