Day4 (Django ORM)

자료	Django
구분	Django
과목	

오늘은 “QuerySet API” 라는 것을 사용해서 DB(데이터베이스)를 조작 하는 것을 연습한다.

조작한다는 것의 의미는 DB에 데이터를 생성하고 조회하고 수정하고 지우는 행위를 말한다.

원래 DB에서는 SQL (Structured Query Language) 이라는 DB 언어를 통해서 데이터를 읽고 쓰고 수정하고 지운다.

하지만, 오늘 우리가 하려고 하는 것은 SQL문을 통한 DB의 데이터를 조작하는 것이 아니다.

Django 프레임워크에서 사용하는 파이썬을 사용해서 DB에있는 데이터를 조작하고자 한다.

Database를 조작하는데 SQL문이 아닌, 파이썬과 같은 객체지향 언어로 데이터를 조작이 가능한 이유는 바로 Django에 있는 ORM (Object-Relational-Mapping) 이라는 컴퓨팅 기술 덕분이다.

Django　←—　ORM　—→ Database

(python)　　(data)

우리는 Django에서 파이썬으로 DB의 데이터를 읽고 쓰고 수정하고 삭제하는 요청을 보낼 것이다.

이를 “쿼리문을 작성한다.” (쿼리를 날린다.) 라고 이야기 한다.

- Django 에서 python 문법으로 작성한 코드가 어떻게 DB로 전달이 되는지 살펴보자.

1. 우리는 Django에서 파이썬을 이용해서 Database 조작을 요청한다.
2. ORM이 “QuerySet API” 라는 도구를 사용해서 우리가 작성한 파이썬 코드를 가지고 “QuerySet” 이라는 객체를 생성한다.
3. ORM은 QuerySet 객체를 SQL 쿼리로 변환해서 Database에 요청을 보낸다.

- 우리가 DB에 정상적인 요청을 보냈다면 DB는 우리에게 응답 결과를 돌려줄 것이다.

  그 과정을 살펴 보자.

1. Database가 요청에 대한 응답 결과를 뱉으면 ORM이 응답 결과를 QuerySet 객체의 형태로 가지고 있다가, 우리가 파이썬으로 응답받은 데이터를 조작을 하려고 할 때,
2. ORM은 QuerySet 객체를 우리가 응답받은 결과를 쉽게 사용할 수 있도록

QuerySet 객체를 파이썬 객체로 바로 변환을 해준다.

3. 그래서 우리는 응답받은 파이썬 객체를 리스트로 만들거나 for와 같은 반복문 사용을 해서 데이터를 조작하게 된다.

==========

[참고]

그렇다면, 우리가 파이썬으로 쿼리를 보내면 ORM이 파이썬을 바로 sql문으로 바꾸지 않고

중간에 QuerySet 객체로 변환하는 과정을 가져가는 이유는 무엇일까?

그 이유는QuerySet 객체를 사용하면 데이터가 실제로 필요할 때만 쿼리를 실행할 수 있다.

```
# QuerySet 객체만 생성하고 이 시점에서는 쿼리가 실행되지 않는다.
books = Book.objects.filter(author="MINHO")

다른 소스코드 생략 ...

# 데이터가 실제로 필요한 시점에 쿼리가 실행된다.
for book in books:
    print(book.title)  # 이 시점에서 SQL 쿼리가 실행된다.
```

파이썬 코드가 실행되는 즉시 DB에 쿼리를 보내면 당장 사용하지 않을 데이터 까지 모두 쿼리를

보내게 된다. 이렇게 되면 불필요하게 반복적으로 DB에 요청을 보내게 된다.

위의 소스코드를 보면 SQL 쿼리가 한 번만 실행되고, 이후에는 메모리에 로드된 데이터를 사용하여

반복문이나 다른 작업을 처리를 한다.

이처럼 즉시실행을 하지 않아도 됨으로써 불필요한 데이터 접근을 피할 수 있는데 이를

지연로딩 (lazy loading) 이라고 한다.

자, 이제는 본격적인 실습에 앞서, 실습 환경부터 조성하자.

[어제 PDF 파일 실습내용]

test.zip

==========

먼저 migrations migrate 이후에 db.sqlite3 를 통해서 DB 테이블은 한번 확인해 보고 시작하자.

ipython 그리고 django-extensions를 설치하자.

```
# 파이썬에서 사용하는 대화형 인터프리터 패키지

$ pip install ipython

# Django 편의기능들이 들어있는 패키지로 shell_plus 사용하기 위해 설치하자

$ pip install django-extensions
```

```
# settings.py

INSTALLED_APPS = [

'articles',
'django_extensions',              # 언더바 주의
...

]
```

```

$ pip freeze > requirements.txt

$ python manage.py shell_plus
```

Django shell 을 실행 했다.

자 이제 django shell 에서 python 코드를 써서 database를 조작할 것이다.
(이 과정을 도와주는 것이 ORM 이라고 앞서 설명했다.)

1. Query문으로 database에 data 객체를 생성(create) 볼 것이다. 그 다음
2. 데이터 들을 조회 (Read) 할 것이다. “all” “filter” “get” 을 사용해서 조회를 해 볼 것이다.
3. 데이터를 수정(Update)를 할 것이며,
4. 데이터를 삭제(Delete)도 할 것이다.

- Create (작성 방법이 3가지가 있다.)

```
# 방법 1
```

```
# models.py에서 정의한 Article 클래스를 통한 인스턴스 생성
article = Article()

# 값 채워 넣고
article.title = 'first'
article.content = 'django!!'

# 저장하면 database에 해당 내용이 적용이 된다.
article.save()

# objects 라는 이름을 가진 장고매니저 야~ (해리포터에서의 도비)
# test 에 Articles 클래스의 모든 정보를 가져와!
test = Article.objects.all()

# 원하는 값 출력
test[0].title
```

```
# 방법2

article = Article(title='second', content='django!!')

article.save()
article.title
```

```
# 방법3

Article.objects.create(title='third',content='django22')

# save 안해도 자동으로 database에 저장됨
# 하지만 나중에 유효성 검사 때문에 방법1 또는 방법2를 더 자주 사용할 것임
```

- Read

```
# 다 읽어 오기
Article.objects.all()

# content가 'django!!' 인것 다 읽어 오기
Article.objects.filter(content='django!!')
```

```
# pk가 1인 단일데이터 읽어 오기 (여러개 복수의 데이터는 못 불러옴)
Article.objects.get(pk=1)

[참고]
Django ORM에서 QuerySet을 사용할 때의 접근 방식이 다르다.
all, filter 로 읽어 온 것은 [0] 인덱스로 접근하되
.get()으로 읽어 온 것은 [0] 사용하면 안된다.
```

- Update

```
# 수정할 인스턴스 조회
>>> article = Article.objects.get(pk=1)

# 인스턴스 변수를 변경
>>> article.title = 'byebye'

# 저장
>>> article.save()

# 정상적으로 변경된 것을 확인
>>> article.title
'byebye'
```

- delete

```
# 삭제할 인스턴스 조회 후 삭제

>>> article = Article.objects.get(pk=1)

>>> article.delete()
```

[끝]