장고 차근차근 시작하기 2/E

당신의 파이썬/장고 페이스메이커가 되겠습니다. ☺

EP-14. 모델을 통한 데이터 생성/수정/삭제

INSERT SQL

다양한 INSERT 예시

```
# 방법1) 각 Post.objects의 create 함수 호출 ➔ 반환값 : 모델 객체
post = Post.objects.create(field1=value1, field2=value2, …)
post.pk                 # DB로부터 할당받은 pk

# 방법2) 각 모델 인스턴스의 save 함수 호출 ➔ 반환값 : None
post = Post(field1=value1, field2=value2)
post.pk                 # .pk => None
post.save()
post.pk                 # DB로부터 할당받은 pk

# 방법3) 관련 ModelForm을 통한 save 함수 호출 ➔ 반환값 : 모델객체
form = PostModelForm(request.POST, request.FILES)
if form.is_valid():     # 유효성 검사 수행
    post = form.save()  # 내부적으로 모델객체.save() 호출하고, 그 객체를 리턴
    post.pk             # DB로부터 할당받은 pk
```

유효성 검사는 언제 이뤄지는 가?

- 모델의 각 필드 코드에 validators를 지정하기는 하지만, 모델 단에서는 유효성 검사를 수행하지 않습니다.
  - 모델객체.save() 시에 필드내역대로 INSERT SQL을 만들어서, DB로 던집니다.
  - DB 처리 중에 에러가 발생할 경우, 이를 예외로 처리 (DB 의존적)
    - 유효성 검사를 최대한 타이트하게 하는 것이 좋습니다.
- 그럼 어디에서 언제?
  - 관련 Form 객체의 is_valid()를 호출할 때 수행
    - ModelForm의 validators는 관련 Model의 validators를 참조합니다.
  - Admin 페이지에서는 모델 별로 기본 ModelForm을 생성합니다.

UPDATE SQL

다양한 UPDATE 예시

```
# 방법1) 개별 모델 인스턴스의 save 함수 호출 ➔ 반환값: None
post = Post.objects.all().first()
post.field1 = new_value1
post.field2 = new_value2
post.save()  # 변경된 필드에 한해서 수행되는 것이 아니라, 모든 필드에 대해서 수행

# 방법2) QuerySet의 update 함수 호출 ➔ 반환값 : 업데이트한 Row 개수 (정수)
qs = Post.objects.all().filtter(...).exclude(...)
qs.update(field1=new_value1, field2=new_value2)

# 방법3) 관련 ModelForm의 save 함수 호출 ➔ 반환값 : 모델객체
form = PostForm(request.POST, request.FILES, instance=post)
if form.is_valid():      # 유효성 검사 수행
    post = form.save()  # 내부적으로 모델객체.save() 호출하고, 그 객체를 리턴
```

비슷한 동작, 다른 성능

같은 값들로 갱신하려 할 때

```
# 1안) 각 인스턴스 별로 별도의 SQL
qs = Post.objects.all()
for post in qs:
    post.title = 'changed title'
    post.save()


# 2안) 하나의 SQL
qs = Post.objects.all()
qs.update(title='changed title')
```

DELETE SQL

다양한 DELETE 예시

```
# 방법1) 개별 모델 인스턴스의 delete 함수 호출 ➔ 반환값 : 삭제된 Record 갯수
post = Post.objects.all().first()                    ex) (1, {'blog.Post': 1})
post.field1 = new_value1
post.field2 = new_value2
post.delete()

# 방법2) QuerySet의 delete 함수 호출 ➔ 반환값 : 삭제된 Record 갯수
qs = Post.objects.all().filtter(...).exclude(...)    ex) (3, {'blog.Post': 3})
qs.delete()
```

웹서비스에서의 주요 병목

대개의 경우, 데이터베이스가 주요 병목

- 같은 작업을 하더라도
  - DB로 전달/실행하는 SQL 개수를 줄이고
  - 각 SQL의 성능/처리속도 최적화가 필요
- RDBMS 외에도 캐싱 솔루션이나 NoSQL 솔루션을 고려해볼 수도 있겠습니다.

➔ 제일 먼저, DB엔진과 서비스에 맞는 적절한 DB 설계가 중요


인생은 짧습니다.
파이썬/장고를 쓰세요.