Ask Company

리액트와 함께 장고 시작하기 / 장고 Models

관계를 표현하는 모델 필드

여러분의 파이썬/장고 페이스메이커가 되겠습니다.

당부의 말

ORM은 어디까지나, SQL 생성을 도와주는 라이브러리

ORM이 DB에 대한 모든 것을 알아서 처리해주진 않습니다.

보다 성능높은 애플리케이션을 만들고자 하신다면, 사용하실 DB엔진과 SQL에 대한 높은 이해가 필요합니다.

RDBMS에서의 관계 예시

설계하기 나름입니다.

1 : N 관계 ➔ models.ForeignKey로 표현

1명의 유저(User)가 쓰는 다수의 포스팅(Post)

1명의 유저(User)가 쓰는 다수의 댓글(Comment)

1개의 포스팅(Post)에 다수의 댓글(Comment)

1 : 1 관계 ➔ models.OneToOneField로 표현

1명의 유저(User)는 1개의 프로필(Profile)

M : N 관계 ➔ models.ManyToManyField로 표현

1개의 포스팅(Post)에는 다수의 태그(Tag)

1개의 태그(Tag)에는 다수의 포스팅(Post)

ForeignKey

ForeignKey

1 : N 관계에서 N측에 명시

ex) Post:**Comment**, User:**Post**, User:**Comment**,

ForeignKey(to, on_delete)

to : 대상모델

클래스를 직접 지정하거나,

클래스명을 문자열로 지정. 자기 참조는 "self" 지정

on_delete : Record 삭제 시 Rule → https://docs.djangoproject.com/en/2.1/ref/models/fields/#django.db.models.ForeignKey.on_delete

CASCADE : FK로 참조하는 다른 모델의 Record도 삭제 (장고 1.X에서의 디폴트값)

PROTECT : ProtectedError (IntegrityError 상속) 를 발생시키며, 삭제 방지

SET_NULL : null로 대체. 필드에 null=True 옵션 필수.

SET_DEFAULT : 디폴트 값으로 대체. 필드에 디폴트값 지정 필수.

SET : 대체할 값이나 함수 지정. 함수의 경우 호출하여 리턴값을 사용.

DO_NOTHING : 어떠한 액션 X. DB에 따라 오류가 발생할 수도 있습니다.

https://docs.djangoproject.com/en/2.1/ref/models/fields/#django.db.models.ForeignKey

올바른 User 모델 지정

```
# django/contrib/auth/models.py
class User(AbstractBaseUser):
    ...

# blog/models.py
class Post(models.Model):
    author = models.ForeignKey(settings.AUTH_USER_MODEL,
                               on_delete=models.CASCADE)
    title = models.CharField(max_length=100)


>>> user.post_set.all()
```

FK에서의 reverse_name

reverse 접근 시의 속성명 : 디폴트 → "모델명소문자_set"

```
from django.db import models

class Post(models.Model):
    title = models.CharField(max_length=100)
    content = models.TextField()

class Comment(models.Model):
    post = models.ForeignKey(Post, on_delete=models.CASCADE)
    message = models.TextField()


>>> comment.post

>>> post.comment_set.all() ⇔ Comment.objects.filter(post=post)
```

reverse_name 이름 충돌이 발생한다면?

reverse_name 디폴트 명은 앱이름 고려 X, 모델명만 고려

다음의 경우, user.**post_set** 이름에 대한 충돌

blog앱 Post모델, author = FK(User)

shop앱 Post모델, author = FK(User)

이름이 충돌이 날 때, makemigrations 명령이 실패

이름 충돌 피하기

1. 어느 한 쪽의 FK에 대해, reverse_name을 포기 → related_name='+'
2. 어느 한 쪽의 (혹은 모두) FK의 reverse_name을 변경
    1. ex) FK(User, …, related_name="blog_post_set")
    2. ex) FK(User, …, related_name="shop_post_set")

ForeignKey.limit_choices_to 옵션

Form을 통한 Choice 위젯에서 선택항목 제한 가능.

dict/Q 객체를 통한 지정 : 일괄 지정

dict/Q 객체를 리턴하는 함수 지정 : 매번 다른 조건 지정 가능

ManyToManyFieId에서도 지원

```
staff_member = models.ForeignKey(
    User,
    on_delete=models.CASCADE,
    limit_choices_to={'is_staff': True},
)
```

OneToOneField

OneToOneField

1 : 1 관계에서 어느 쪽이라도 가능

User:**Profile**

ForeignKey(unique=True)와 유사하지만, reverse 차이

User:Profile를 FK로 지정한다면 → profile.user_set.first() → user

User:Profile를 O2O로 지정한다면 → profile.user → user

OneToOneField(to, on_delete)

```
# django/contrib/auth/models.py
class User(AbstractBaseUser):
    ...

# accounts/models.py
class Profile(models.Model):
    author = models.OneToOneField(settings.AUTH_USER_MODEL, on_delete=models.CASCADE)
```

https://docs.djangoproject.com/en/3.0/ref/models/fields/#onetoonefield

O2O에서의 related_name

reverse 접근 시의 속성명 : 디폴트 → 모델명소문자

```
# accounts/models.py
class Profile(models.Model):
    author = models.OneToOneField(settings.AUTH_USER_MODEL,
                                  on_delete=models.CASCADE)
    phone = models.CharField(max_length=11, blank=True)
    birth = models.DateField(null=True)


>>> profile.author

>>> user.profile
```

ManyToManyField

ManyToManyField

M : N 관계에서 어느 쪽이라도 필드 지정 가능

ManyToManyField(to, blank=False)

```python
# 방법 1)

class Post(models.Model):
    tag_set = models.ManyToManyField('Tag', blank=True)

class Article(models.Model):
    tag_set = models.ManyToManyField('Tag', blank=True)

class Tag(models.Model):
    name = models.CharField(max_length=100, unique=True)
```

```python
# 방법 2)

class Post(models.Model):
    ...

class Article(models.Model):
    ...

class Tag(models.Model):
    name = models.CharField(max_length=100, unique=True)
    post_set = models.ManyToManyField('Post', blank=True)
    article_set = models.ManyToManyField('Article',
                                         blank=True)
```

https://docs.djangoproject.com/en/2.1/ref/models/fields/#manytomanyfield

RDBMS지만, DB따라 NoSQL기능도 지원

ex) 하나의 Post 안에 다수의 댓글 저장 가능

djkoch/jsonfield

- 대개의 DB엔진에서 사용 가능
- TextField/CharField를 래핑
- dict 등의 타입에 대한 저장을 직렬화하여 문자열로 저장
  - 내부 필드에 대해 쿼리 불가

django.contrib.postgres.fields.JSONField

- 내부적으로 PostgreSQL의 jsonb 타입
- 내부 필드에 대해 쿼리 지원

adamchainz/django-mysql

- MySQL 5.7 이상에서 json 필드 지원

Life is short.
You need Python and Django.

I will be your pacemaker.