DB

Many to one relationships 01

모델 관계

Many to one relationships

N:1 or 1:N

- 한 테이블의 0개 이상의 레코드가 다른 테이블의 레코드 한 개와 관련된 관계

Comment(N) - Article(1)

- 0개 이상의 댓글은 1개의 게시글에 작성될 수 있다.

테이블 관계

Comment

id
content
created_at
updated_at
Article에 대한 외래 키

Article

id
title
content
created_at
updated_at

댓글 모델 정의

- `ForeignKey()` ; 한 모델이 다른 모델을 참조하는 관계를 설정하는 필드
  - N:1 관계 표현
  - 데이터베이스에서 외래 키로 구현
- ForeignKey 클래스의 인스턴스 이름은 참조하는 모델 클래스 이름의 단수형으로 작성하는 것을 권장
- 외래 키는 ForeignKey 클래스를 작성하는 위치와 관계없이 테이블의 마지막 필드로 생성됨

```python
# articles/models.py

class Comment(models.Model):
    article = models.ForeignKey(Article, on_delete=models.CASCADE)
    # ForeignKey 생성 시 _id가 자동을 붙기 때문에 요구사항에
    # article_id로 되어있더라도 변수명은 article로 해야함
    content = models.CharField(max_length=200)
    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)
```

ForeignKey(to, on_delete)

- to ; 참조하는 모델 class 이름
- on_delete ; 외래 키가 참조하는 객체(1)가 사라졌을 때, 외래 키를 가진 객체(N)를 어떻게 처리할 지를 정의하는 설정 (데이터 무결성)

on_delete의 'CASCADE'

- 참조 된 객체(부모 객체)가 삭제 될 때 이를 참조하는 모든 객체도 삭제되도록 지정
- 기타 on_delete 설정 값 참고

Migration 이후 댓글 테이블 확인

- 댓글 테이블의 article_id 외래 키 필드 확인
- 만들어지는 필드 이름
  - '참조 대상 클래스 이름' + '_' + 'id'
- 참조하는 클래스 이름을 소문자(단수형)로 작성하는 것이 권장된 이유

댓글 생성 연습

1. shell_plus 실행 및 게시글 작성

```
# shell_plus 실행
$ python manage.py shell_plus

# 게시글 생성
Article.objects.create(title='title', content='content')
```

2. 댓글 생성

```
# Comment 클래스의 인스턴스 comment 생성
comment = Comment()

# 인스턴스 변수 저장
comment.content = 'first comment'

# DB에 댓글 저장
comment.save()

# 에러 발생
django.db.utils.IntegrityError: NOT NULL constraint failed:
articles_comment.article_id
#=> articles_comment 테이블의 ForeignKeyField, article_id 값이 저장 시 누락되었기 때문
```

3. shell_plus 실행 및 게시글 작성

```
# 게시글 조회
article = Article.objects.get(pk=1)

# 외래 키 데이터 입력
comment.article = article
# 또는 comment.article_id = article.pk 표현으로
# pk 값을 직접 외래 키 컬럼에 넣어 줄 수도 있지만 권장하지 않음

# 댓글 저장 및 확인
comment.save()
```

4. comment 인스턴스를 통한 article 값 참조하기

```
In [9]: comment.pk
Out[9]: 1

In [10]: comment.content
Out[10]: 'first comment'

# 클래스 변수명인 article로 조회 시 해당 참조하는 게시물 객체를 조회할 수 있음
In [11]: comment.article
Out[11]: <Article: Article object (1)>

# article_pk는 존재하지 않는 필드이기 때문에 사용 불가
In [12]: comment.article_id
Out[12]: 1
```

5. comment 인스턴스를 통한 article 값 참조하기

```
# 1번 댓글이 작성된 게시물의 pk 조회
In [13]: comment.article.pk
Out[13]: 1

# 1번 댓글이 작성된 게시물의 content 조회
In [14]: comment.article.content
Out[14]: 'content'
```

6. 두번째 댓글 생성

```
In [19]: comment = Comment(content='second comment', article=article)
In [20]: comment.save()

In [21]: comment.pk
Out[21]: 2

In [22]: comment
Out[22]: <Comment: Comment object (2)>

In [23]: comment.article.pk
Out[23]: 1
```

7. 작성된 댓글 데이터 확인

관계 모델 참조

역참조

- N:1 관계에서 1에서 N을 참조하거나 조회하는 것(1->N)
- 모델 간의 관계에서 관계를 정의한 모델이 아닌, 관계의 대상이 되는 모델에서 연결된 객체들에 접근하는 방식
- N은 외래 키를 가지고 있어 물리적으로 참조가 가능하지만, 1은 N에 대한 참조 방법이 존재하지 않아 별도의 역참조 키워드가 필요

역참조 사용 예시

- article.comment_set.all()
  - 모델 인스턴스 . related manager(역참조 이름) . QuerySet API
- 특정 게시글에 작성된 댓글 전체를 조회하는 요청

related manager

- N:1 혹은 M:N 관계에서 역참조 시에 사용하는 매니저
- 'objects' 매니저를 통해 QuerySet API를 사용했던 것처럼 related manager를 통해 QuerySet API를 사용할 수 있게 됨

related manager 이름 규칙

- N:1 관계에서 생성되는 Related manager의 이름은 "모델명_set" 형태롤 자동 생성됨
  - 관계를 직접 정의하지 않은 모델에서 연결된 객체들을 조회할 수 있게 함
- 특정 댓글의 게시글 참조 (Comment -> Article)
  - comment.article
- 특정 게시글의 댓글 목록 참조 (Article -> Comment)
  - article.comment_set.all()

related manager 연습

- shell_plus 실행 및 1번 게시글 조회

```
$ python manage.py shell_plus

In [24]: article = Article.objects.get(pk=1)
```

- 1번 게시글에 작성된 모든 댓글 조회하기 (역참조)

```
In [25]: article.comment_set.all()
Out[25]: <QuerySet [<Comment: Comment object (1)>, <Comment: Comment object (2)>]>
```

- 1번 게시글에 작성된 모든 댓글 내용 출력

```
comments = artcile.comment_set.all()

for comment in comments:
    print(comment.content)
```

댓글 구현

댓글 CREATE

1. 사용자로부터 댓글 데이터를 입력받기 위한 CommentForm 정의

```python
# articles/forms.py

from .models import Article, Comment

class CommentForm(forms.ModelForm):
    class Meta:
        model = Comment
        fields = '__all__'
```

2. detail view 함수에서 CommentForm을 사용하여 detail 페이지에 렌더링

```python
# articles.views.py

from .forms import ArticleForm, CommentForm

def detail(request, pk):
    article = Article.objects.get(pk=pk)
    comment_form = CommentForm()
    context = {
        'article': article,
        'comment_form': comment_form,
    }
    return render(request, 'articles/detail.html', context)
```

```html
<!-- articles/detail.html -->

<form action="#" method="POST">
  {% csrf_token %}
  <input />
</form>
```

- Comment 클래스의 외래 키 필드 article 또한 데이터 입력이 필요한 필드이기 때문에 출력 되는 것
- 하지만, 외래 키 필드 데이터는 사용자로부터 입력 받는 값이 아닌 view 함수 내에서 다른 방법으로 전달 받아 저장되어야 함

3. Comment Form의 출력 필드 조정하여 외래 키 필드가 출력되지 않도록 함

```python
# articles/forms.py

from .models import Article, Comment

class CommentForm(forms.ModelForm):
    class Meta:
        model = Comment
        fields = ('content',)
```

- 출력에서 제외된 외래키는 해당 게시글 URL에서 pk 값을 가져와서 사용

4. url 작성 및 action 값 작성

```
# articles/urls.py

urlpatterns = [
    ...,
    path('<int:pk>/comments/', views.comments_create, name='comments_create'),
]
```

```
<!-- articles/detail.html -->

<form action="{% url "article.comments_create" article.pk %}" method="POST">
{% csrf_token %}
{{ comment_form }}
<input type='submit'/>
</form>
```

5. comments_create view 함수 정의

- url로 받은 pk 인자를 게시글을 조회하는 데 사용

```
# articles/views.py

def comments_create(request, pk):
    article = Article.objects.get(pk=pk)
    comment_form = CommentForm(request.POST)
    if comment_form.is_valid():
        comment_form.save()
        return redirect('articles:detail', article.pk)
    context = {
        'article': article,
        'comment_form': comment_form,
    }
    return render(request, 'articles/detail.html', context)
```

6. article 객체는 어떻게/언제 저장?

save(commit=False)

- DB에 저장 요청을 보내지 않고 인스턴스만 반환

7. save의 commit 인자를 활용해 외래 키 데이터 추가 입력

```
# articles/views.py

def comments_create(request, pk):
    article = Article.objects.get(pk=pk)
    comment_form = CommentForm(request.POST)
    if comment_form.is_valid():
        comment=comment_form.save(commit=False)
        comment.article = article
        comment.save()
        return redirect('articles:detail', article.pk)
    context = {
        'article': article,
```

```
        'comment_form': comment_form,
    }
    return render(request, 'articles/detail.html', context)
```

8. 댓글 작성 후 테이블 확인

댓글 READ

1. detail view 함수에서 전체 댓글 데이터를 조회

```
# articles/views.py

def detail(request, pk):
    article = Article.objects.get(pk=pk)
    comment_form = CommentForm()
    comments = article.comment_set.all()
    context = {
        'article': article,
        'comment_form': comment_form,
        'comments': comments,
    }
    return render(request, 'articles/detail.html', context)
```

2. 전체 댓글 출력 및 확인

```
<!-- articles/detail.html -->

<h4>댓글 목록</h4>
<ul>
  {% for comment in comments %}
  <li>{{ comment.content }}</li>
  {% endfor %}
</ul>
```

댓글 DELETE

1. 댓글 삭제 url 작성

```python
# articles/urls.py

app_name = 'articles'
urlpatterns = [
    ....,
    path(
        '<int:article_pk>/comments/<int:comment_pk>/delete',
        views.comments_delete,
        name='comments_delete'
    ),
]
```

2. 댓글 삭제 view 함수 정의

```python
# articles/views.py

from .models import Article, Comment

def comments_delete(reqest, article_pk, comment_pk):
    comment = Comment.objects.get(pk=comment_pk)
    comment.delete()
    return redirect('articles:detail', article_pk)
```

3. 댓글 삭제 버튼 작성

```
<!-- articles/detail.html -->
<h4>댓글 목록</h4>
<ul>
{% for comment in comments %}
    <li>{{ comment.content }}</li>
    <form action="{% url "articles:comment_delete" article.pk comment.pk %}"
method="POST">
    {% csrf_token %}
    <input type="submit" value="DELETE">
    </form>
{% endfor %}
</ul>
```

참고

데이터 무결성

- 데이터베이스에 저장된 데이터의 정확성, 일관성, 유효성을 유지하는 것
- 데이터베이스에 저장된 데이터 값의 정확성을 보장하는 것
- 중요성
    1. 데이터의 신뢰성 확보

2. 시스템 안전성
3. 보안 강화

admin site 댓글 등록

- Comment 모델을 admin site에 등록해 CRUD 동작 확인하기

```
# articles/admin.py

from .models import Article, Comment

admin.site.register(Article)
admin.site.register(Comment)
```

댓글 추가 구현

1. 댓글이 없는 경우 대체 콘텐츠 출력
   - DTL의 'for empty' 태그 활용

```
{% for comment in comments %}
<li>
    {{ comment.content }}
    <form action="{% url "articles:comment_delete" article.pk comment.pk %}"
method="POST">
    {% csrf_token %}
    <input type="submit" value="DELETE">
    </form>
</li>
{% empty %}
<p>댓글이 없어요...</p>
{% endfor %}
```

2. 댓글 개수 출력하기

- DTL filter - 'length' 사용

```
{{ comments|length }}

{{ article.comment_set.all|length}}
```

- QuerySet API 'count()' 사용

```
{{ article.comment_set.count }}
```