역참조 && 댓글달기

1. 두 모델의 관계

```
from django.db import models

# Create your models here.


class Article(models.Model):
    title = models.CharField(max_length=10)
    content = models.TextField()
    updated_at = models.DateTimeField(auto_now=True)
    created_at = models.DateTimeField(auto_now_add=True)
```

이번 시간에는 댓글 다는 것을 볼 것이다.
배포하는 파일을 먼저 열어보자. Articls > models.py를 보면

다음과 같은 모델이 이미 존재한다. `Article` 은 게시판의 게시글을 의미한다.

이 `models.py` 파일은 단 하나의 모델, `Article` 만 존재하는데, 만약 게시글에 댓글을 달고자 하면 `Comment` 모델을 추가로 만들어야한다.

일단, 다음 ERD 를 보자.

Article		
PK	id	Int
	title	Char(10)
	content	Text
	updated_at	DateTime
	created_at	DateTime

Comment		
PK	id	Int
	content	Char(200)
	created_at	DateTime
	updated_at	DateTime

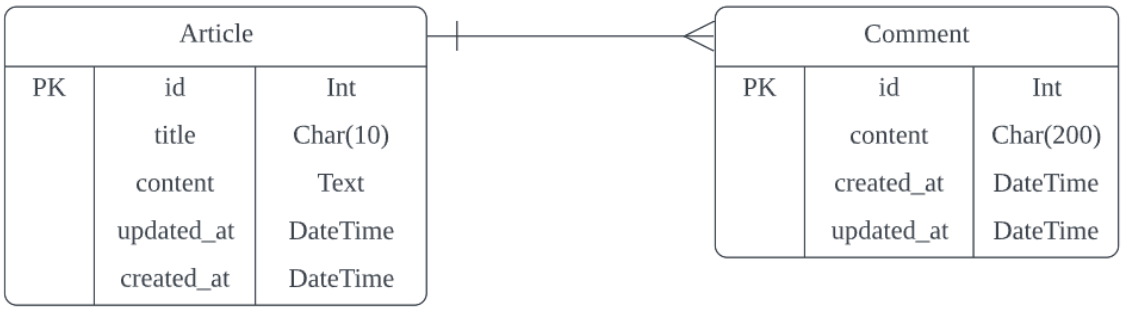
기존에 `Article` 모델과, 곧 추가될 `Comment` 모델이다.

두 테이블은 분명 관계가 있다. 댓글은 게시글에 달리기 때문이다.

그러나, 관계를 잇기 전에, ERD 로 따지면 선을 그리기 전에 판단해야 할 것이 있다. 누가 Many 인지, 누가 One 인지를 알아야한다.

- 하나의 게시글은 여러개 댓글을 가지나? ⇒ O
- 하나의 댓글은 여러개 게시글을 가지나? ⇒ X. 하나의 댓글은 특정 게시글에만 쓰인다.

따라서, 관계는 다음과 같이, Crow's foot (까마귀발) 방식으로 도식화할 수 있다.



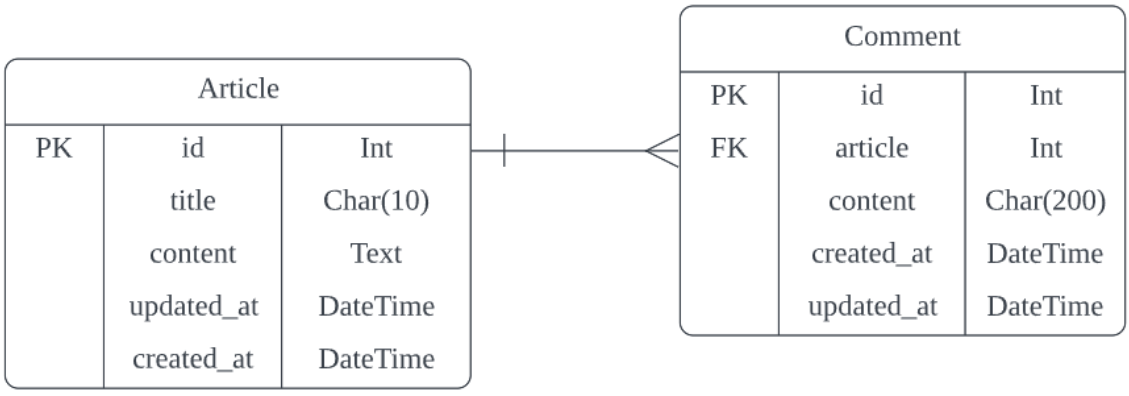
이것을 Many To One 관계, 다른 말로 1:N 관계라고 한다. 한쪽은 Many, 한쪽은 One 인 관계이다.

관계의 Many, One 을 왜 따졌는가? 관계가 형성된다는 것은, FK 가 생긴다는 것인데 위 ERD 에는 FK 가 존재하지 않는다.

FK 는 관계의 Many, One 을 따진 다음에야 존재할 수 있으며, 반드시 두 모델 중 한 쪽에만 두어야 한다.

이 관계에서는 공식이 있는데, FK 는 반드시 Many 에 위치해야 한다.

위 예시로 보면, `Comment` 에 위치하며, 도식은 다음과 같이 완성된다.



`Comment` 모델에서 FK, 즉 `Article` 의 PK 를 담고 있는 컬럼을 만들어준다.

수정된 모델은 다음과 같다.

```
from django.db import models

# Create your models here.


class Article(models.Model):
    title = models.CharField(max_length=10)
    content = models.TextField()
    updated_at = models.DateTimeField(auto_now=True)
    created_at = models.DateTimeField(auto_now_add=True)


class Comment(models.Model):
    article = models.ForeignKey(Article, on_delete=models.CASCADE)
    content = models.CharField(max_length=200)
    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)
```

```
on_delete=models.CASCADE
on_delete
참조하는 객체가 사라졌을때
CASCADE
부모를 참조하는 객체도 모두 삭제
```

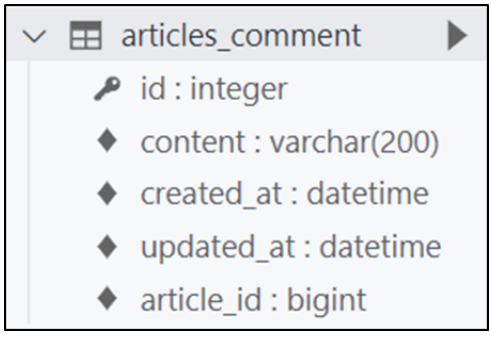
`on_delete` 은 참조하는 객체가 사라졌을때 어떻게 할 것인지를 적어줄 것이다. 즉 게시글을 삭제하면 관련 댓글들은 어떻게 할 것인지 정해 줄 것인데

`CASCADE`를 통해서 부모를 참조하는 객체도 모두 삭제 하겠다고 적어준 것이다.

모델이 바뀌었으므로, 가상환경을 on 하고 다음 두 명령어를 실행하자.

```
$ python manage.py makemigrations
$ python manage.py migrate
```

그리고 SQLITE EXPLORER 를 통해 확인해보면,



`Comment` 모델에서 FK 로 `article` 필드를 설정했지만, 실제 DB 엔 `article_id` 로 들어간 것을 확인할 수 있다.

admin page 에서 조작하기 위해, `admin.py` 를 다음과 같이 수정하자.

```
from django.contrib import admin
from .models import Article, Comment

# Register your models here.
admin.site.register(Article)
admin.site.register(Comment)
```

python manage.py createsuperuser 를 통해서 admin 을 새로 등록 한 다음 간단한 댓글 샘플을 admin page 을 켜서 몇 개 만들고, SQLITE EXPLORER 를 작동시켜 실제 테이블을 확인해보면 다음과 같다.

id	title	content	updated_at	created_at
1	안녕 싸피 여러분	반가워~~	2023-03-22 01:48:07.306929	2023-03-22 01:48:07.306929
2	오늘 수업 너무힘듦	엉엉 싸탈언제해	2023-03-22 01:48:54.472785	2023-03-22 01:48:54.472785

id	content	created_at	updated_at	article_id
1	나도 안녕	2023-03-22 07:14:39.448084	2023-03-22 07:15:30.347554	1
2	싸탈 빨리해~~	2023-03-22 07:15:19.100768	2023-03-22 07:15:19.100768	2
3	하이	2023-03-22 07:15:40.461675	2023-03-22 07:15:40.461675	1

일단 게시글은 총 두 개이며, 1번 게시글은 총 두 개의 댓글, “나도 안녕” 과 “하이” 를 가지고 있고, 2번 게시글은 총 한 개의 댓글, “싸탈 빨리해~~” 를 가지고 있는 것을 알 수 있다.

무엇을 보고 판단 가능한 것인가? `articles_comment` 테이블의 `article_id` 컬럼을 보고 판단 가능하다. article 에서의 pk이자 comments의 fk 이다.

2. Comment Create

먼저, `articles/forms.py` 를 다음과 같이 수정하자.

```
from django import forms
from .models import Article, Comment


class ArticleForm(forms.ModelForm):
    class Meta:
        model = Article
        fields = '__all__'


class CommentForm(forms.ModelForm):
    class Meta:
        model = Comment
        exclude = ('article',)
```

`article` 필드는 왜 제외했는가? 댓글 쓰는 사람이 어떤 게시글에 댓글을 쓸지 결정 수는 없다. 해당 댓글 쓰는 양식은 123번 게시글의 디테일 페이지에 들어간다면, 반드시 123 번 게시글에 대한 댓글이 되어야만 한다. (즉, 댓글을 작성한다면 해당 게시글 페이지에 들어간 다음에 해당 게시글에 대한 댓글을 단다면 어떤 게시글에 대한 댓을을 작성할 것인지 결정할 필요가 없다는 말이다.)

다음, 해당 글의 디테일 페이지에서도 댓글을 작성하기 위하여 댓글 작성 폼을 추가하자

`articles/views.py` 로 가서 `detail` 함수를 다음과 같이 수정하자.

```
from .forms import ArticleForm, CommentForm

@require_safe
def detail(request, pk):
    article = get_object_or_404(Article, pk=pk)
    comment_form = CommentForm()
    context = {
        'article': article,
        'comment_form': comment_form,
    }
    return render(request, 'articles/detail.html', context)
```

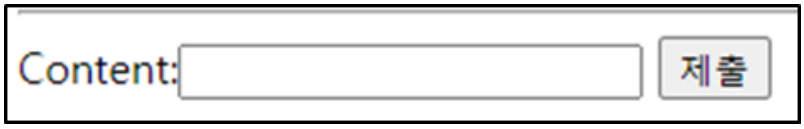
크게 바뀐 건 없다. 우리가 방금 전에 만들었던 `CommentForm` 을 `context` 에 추가해 `articles/detail.html` 로 보낸다는 게 전부다.

그리고 게시글의 디테일 페이지에서도 작성한 댓글을 post를 할 수 있도록
`articles/detail.html` 을 다음과 같이 수정한다.

```
{% extends 'base.html' %}

{% block content %}
    <h2>DETAIL</h2>
    <h3>{{ article.pk }} 번째 글</h3>
    <hr>
    <p>제목: {{ article.title }}</p>
    <p>내용: {{ article.content }}</p>
    <p>작성 시각: {{ article.created_at }}</p>
    <p>수정 시각: {{ article.updated_at }}</p>
    <hr>
    <a href="{% url 'articles:update' article.pk %}">UPDATE</a>
    <form action="{% url 'articles:delete' article.pk %}" method="POST">
        {% csrf_token %}
        <input type="submit" value="DELETE">
    </form>
    <a href="{% url 'articles:index' %}">[back]</a>
    <hr>
    <form action="#" method="POST">
        {% csrf_token %}
        {{ comment_form }}
        <input type="submit">
    </form>
{% endblock content %}
```

[back] 링크 아래에 댓글 작성 폼을 추가했다.



다음과 같이, 맨 아래 댓글 작성란이 추가되었음을 알 수 있다. 물론, 현재 `action` 의 경로는 `#` 이기 때문에, 작동은 되지 않는다.

`actions` 의 `#` 대신, 다음과 같이 수정한다.

```
<form action="{% url 'articles:comments_create' article.pk %}" method="POST">
```

'articles:comments_create' 는 차후에 `articles/views.py` 에서 만들어 줄 것이다.

중요한 건, 어떤 게시글에 대한 댓글인지 알아야 한다는 것이다. 따라서, 맨 뒤에 `article.pk` 를 추가해주었다.

댓글 생성 요청이다. URL 이 명확해졌으니, URL 을 만들기 위해 article의
`urls.py` 로 향하자.

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

해당 URL 을 자세히 보면, `<int:pk>` 는 “어떤 게시글” 에 대한 댓글을 생성할지를 나타낸다.

우리는 `article.pk` 를 `<form>` 에서 주었고, 그걸 URL 로 받은 것이다.

이제 구현 부인 `comments_create` 를 만들어보자. `articles/views.py` 로 가자.

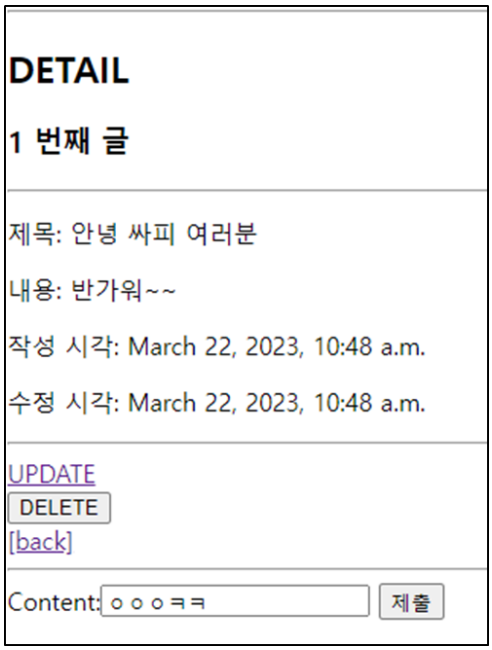
```
from django.views.decorators.http import require_safe, require_http_methods, require_POST

@require_POST
def comments_create(request, pk):
    article = get_object_or_404(Article, 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)
```

comment_form.is_valid()를 통해서 댓글에 아무것도 작성을 하지 않고 제출버튼을 누르면 경고창이 뜰 것이다.

comment_form.save(commit=False) 를 통해서 `commit=False` 를 하게 되면 데이터베이스에 댓글을 `당장` 저장하지 않는다. 왜 이것을 하느냐? `commit=False` 은 DB에 데이터를 저장하기 전에 특정 행위를 하고 싶을 때 사용한다. 즉 CommentForm에 바로 저장하지 않고 어떤 아티클의 comment인지 속성을 지정한 후 (comment.article = article) 그리고 나서 최종적으로 `comment.save()` 저장을 하였다.

이제 댓글을 작성하고, DB 에서 직접 확인해보자.



id	content	created_at	updated_at	article_id
1	나도 안녕	2023-03-22 07:14:39.448084	2023-03-22 07:15:30.347554	1
2	싸탈 빨리해~~	2023-03-22 07:15:19.100768	2023-03-22 07:15:19.100768	2
3	하이	2023-03-22 07:15:40.461675	2023-03-22 07:15:40.461675	1
4	ㅇㅇㅇㅋㅋ	2023-03-22 08:04:25.611819	2023-03-22 08:04:25.611819	1

3. Comment Read (역참조)

오늘 하고자 하는 가장 중요한 목표는, 하나의 게시글에 해당하는 댓글을 전부 가져오기다.

디테일 페이지에서 해당 게시글에 대한 댓글들을 모두 보여줘야 한다.

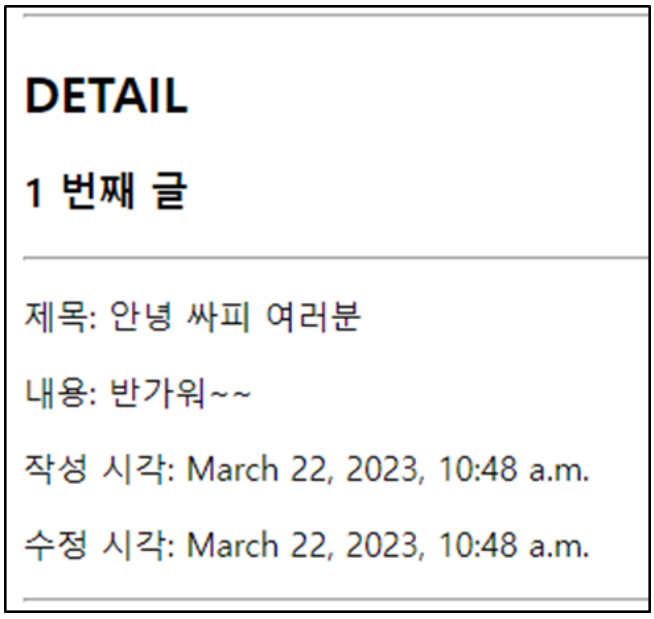
id	title	content	updated_at	created_at
1	안녕 싸피 여러분	반가워~~	2023-03-22 01:48:07.306929	2023-03-22 01:48:07.306929
2	오늘 수업 너무힘듦	엉엉 싸탈언제해	2023-03-22 01:48:54.472785	2023-03-22 01:48:54.472785

id	content	created_at	updated_at	article_id
1	나도 안녕	2023-03-22 07:14:39.448084	2023-03-22 07:15:30.347554	1
2	싸탈 빨리해~~	2023-03-22 07:15:19.100768	2023-03-22 07:15:19.100768	2
3	하이	2023-03-22 07:15:40.461675	2023-03-22 07:15:40.461675	1

아까 추가한 댓글, 예를들어 “asdfasdf” 가 추가되기 전이라고 가정하자.

위 그림에서 보면 1번 게시글의 경우, 총 두 개의 댓글을 가지고 있다. 그러나 잘 생각해보면, 게시글 테이블엔 어떤 댓글이 달려있는지 그 정보가 없다! 모르면 대체 어떻게 가져 온다는 말인가?!?!.

`/articles/1` 에 접속했다고 가정하겠다.



이것은 분명 게시글 페이지다. 이 페이지에 들어갔을 때, 1번 게시글에 해당하는 댓글도 가져와서 전부 다 보여주는 게 우리의 목표다. 그러나 분명 게시글 테이블 그 어디에도, 어떤 댓글을 가지고 있는지 알려주지 않는다.

이런 문제를 해결하기 위해, Django 에선 “역참조” 를 제공한다.

역참조: 본인을 FK 로 참조 중인 다른 테이블에 접근

1:N 관계에서 1(하나의 게시글)에는 N(여러개의 댓글)이 달려있다. 그래서 댓글(N)을 작성할 때에는 어떠한 게시글(1)에 대한 댓글 인지를 알려주기 위해 FK를 사용 했다면

하나의 게시글(1)에 달린 모든 댓글들(N)을 가져 올 때는 “역참조”라는 것을 사용해서 가져 올 수 있다.

역참조는 개념적으로 설명하기 조금 어려울 수도 있는 개념이라, 실제 코드로 보면서 파악 하는 것이 좋을듯 싶다. (C언어로 따지면 약간 포인터 너낌이다ㅎ)

우선, `articles/views.py` 의 `detail` 함수를 수정하자.

```
from .models import Article, Comment

@require_safe
def detail(request, pk):
    article = get_object_or_404(Article, 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)
```

눈 뜨고 다른 걸 찾아보면, `article.comment_set.all()` 이라는 처음보는 함수를 사용해 `comments` 에 담고, 그걸 `context` 에 딸려 보내는 걸 알 수 있다.

잠시 Model 을 살펴보자. `articles/models.py` 를 연다.

```
from django.db import models
# Create your models here.


class Article(models.Model):
    title = models.CharField(max_length=10)
    content = models.TextField()
    updated_at = models.DateTimeField(auto_now=True)
    created_at = models.DateTimeField(auto_now_add=True)


class Comment(models.Model):
    article = models.ForeignKey(Article, on_delete=models.CASCADE)
    content = models.CharField(max_length=200)
    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)
```

`Article` 모델에선 눈 씻고 살펴봐도 `comment_set` 이란 건 보이지 않는다.

`comment_set` 은 `Comment` 모델의 FK 인 `article` 필드가 생길 때, `Article` 모델에서 만들어지는 하나의 객체다. 이 객체에서는 다양한 메서드를 제공하는데, 그 중 `all()` 메서드를 호출하면 다음과 같은 일이 발생한다.

```
1번 게시글에 해당하는 모든 댓글을 가져와!
```

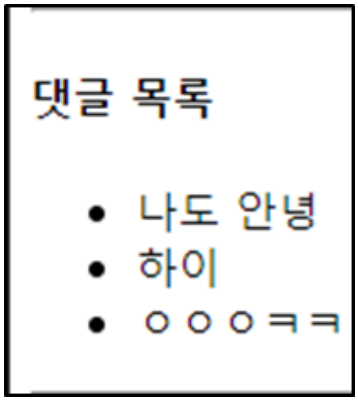
게시글 모델인 `Article` 에는 어떤 댓글이 있는지 알 수 없지만, Django 에서 제공하는 이 객체를 사용하면 복잡한 문제를 생각할 필요 없이 바로 알 수 있다. 이 과정에서, 반대편 테이블을 참조하기 때문에 “역참조” 라는 단어를 사용한다.

- 만약 모델 이름이 `Comment` 가 아니라 `Review` 라면, `review_set` 으로 이름지어진다. 즉, 뒤에 `_set` 이 붙는다.

이제 `articles/detail.html` 에서 해당 게시글의 댓글을 전부 보여주자.

```
{% extends 'base.html' %}

{% block content %}
    ...
    <a href="{% url 'articles:index' %}">[back]</a>
    <hr>
    <h4>댓글 목록</h4>
    <ul>
        {% for comment in comments %}
            <li>{{ comment.content }}</li>
        {% endfor %}
    </ul>
    <hr>
    ...
{% endblock content %}
```



4. 역참조 추가내용

상황에 따라서, `_set` 이라고 자동으로 지어주는 네이밍이 마음에 안 들수도 있다. 그럴 경우, `related_name` 을 사용한다.

지금은 모델 변경이 복잡 하다고 느낄 수 도 있겠지만, 알아 두자.

models.py > Comment 에 related_name을 달아보자.

```
class Comment(models.Model):
    article = models.ForeignKey(Article, on_delete=models.CASCADE, related_name='comments')
    content = models.CharField(max_length=200)
    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)
```

`article` 필드 맨 끝에 `related_name='comments'` 라고 추가했다. 이는 역참조 시 `comment_set` 이라고 자동으로 만들어주는 객체라는 이름이 마음에 안 들기에, 앞으로는

`comments` 라는 이름으로 사용하겠다는 뜻이다.

이 이름을 설정한 이후엔 `comment_set` 은 존재하지 않고 오로지 `comments` 라는 객체만 존재한다. 따라서 related_name을 작성한 후에는 `comment_set`

을 섞어서 사용하지 않는다.

그렇다면, `articles/views.py` 의 `detail` 함수는 어떻게 바뀌겠는가?

```
from .models import Article, Comment

@require_safe
def detail(request, pk):
    article = get_object_or_404(Article, pk=pk)
    comment_form = CommentForm()
    # comments = article.comment_set.all()
    comments = article.comments.all()
    context = {
        'article': article,
        'comment_form': comment_form,
        'comments': comments,
    }
    return render(request, 'articles/detail.html', context)
```

주석과 비교해보자.

comments = article.comment_set.all() 라는 Django에서 제공해주는 객체의 속성을 이용하지 않고 아까 이름을 재정의 한 `comments`을 사용하였다.

댓글을 삭제해 보자

5. Comments Delete

```
# articles/urls.py

urlpatterns = [

...,
```

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

```
# articles/views.py

def comments_delete(request, article_pk, comment_pk):

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

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

{% block content %}

...
<h4>댓글 목록</h4>
    <ul>
        {% for comment in comments %}
            <li>
                {{ comment.content }}
                <form action="{% url 'articles:comments_delete' article.pk comment.pk %}" method="POST">
                    {% csrf_token %}
                    <input type="submit" value="DELETE">
                </form>
            </li>
        {% endfor %}
    </ul>
    <hr>

{% endblock content %}
```

이상으로 댓글 삭제 까지 보았다. 댓글 수정은 댓글 수정 후 수정된 웹 페이지를 새로 불러보는 것이 아니라 페이지 일부 내용만 업데이트 하는 것이라서 차후에 javascript할 때 사용할 것이다.

여기까지 했다면 수업 교재를 보고 몇개의 댓글이 달렸는지도 총 댓글 개수를 디테일스 페이지에 표시 해 보고 라이브 교재를 보면서 댓글과 게시글을 지우고 수정하는 권한을 해당 댓글 또는 게시글을 작성한 사람만 가능하도록 반드시 연습해보자.