Day4 M:N 좋아요 구현

자료	DB
구분	DB
과목	

django_test.zip

1. 기초설정

가상환경을 on 한다. 그리고 배포되는 스켈레톤 파일을 가지고 시작하자.
스켈레톤 파일은 movies 그리고 accounts 앱으로 구성되어 있으며
해당 파일은 지금까지 학습한 게시글 CRUD 그리고 댓글의 생성과 삭제 기능이
미리 구현이 되어 있는 파일이다.

오늘 구현할 내용은 좋아요 :+1 기능 구현이다.

2. 좋아요 구현하기

“좋아요”를 구현하기 앞서 그전에는 댓글 기능을 구현하면서 1:N 관계를 알아보았다.
1:N 관계란 데이터베이스 간이 관계를 의미하며
하나의 게시글에 여러 개의 댓글이 달릴 수 있는 경우를 구현했다.

“좋아요” 기능을 구현 하는데 사용하는 DB의 Cardinality는 M:N (다대다) 관계이다.
M:N 관계를 살펴보면 다음과 같다.
A 그리고 B 라는 Entity(엔티티)가 있다면 하나의 A는 여러 개의 B를 가질 수 있고
B도 여러 개의 A를 가질 수 있는 관계를 말한다.

유저의 입장에서 보면, 한 명의 유저는 여러 개의 게시글에 “좋아요” 누를 수 도 있고

특정 게시글의 입장에서 보면, 하나의 특정 게시글에는 여러 명으로부터 “좋아요”를 받을 수 있다.

예를들어,

유저가 “좋아요” 를 눌렀을 경우 User 의 DB 테이블은 아래와 같이 생겼을 것이다.

id	name	article_Id (좋아요를 누른 게시글 ID)
1	kevin	1
2	james	2
3	kate	3

위 테이블에서 보면 kevin이 1번 아티클에 좋아요를 눌렀다는 것을 확인 할 수 있다.

그런데 만약에 kevin이 2번 아티클에도 좋아요를 누른다면 아래와 같이 데이터베이스가 추가로 저장이 될 것이다.

id	name	article_Id (좋아요를 누른 게시글 ID)
4	kevin	2

User 테이블 전체를 보았을 때 kevin은 ID가 1번 그리고 4번으로 중복으로 데이터 베이스에 저장이 된다는 문제가 발생 한다. (유저 1명이 Database에 중복되는 id를 가질 수 없다.)

id	name	article_Id (좋아요를 누른 게시글 ID)
1	kevin	1, 2

그렇다고 위의 테이블과 같이 kevin 이라는 유저가 “좋아요”를 누른 게시글의 ID를 여러개를 한번에 저장 하는 것은 DB의 데이터타입 문제로 불가능 하다.

따라서 다대다 (M:N) 관계를 구현하기 위해서는 “좋아요”를 위한 테이블 `중개모델`을 따로 만들어서 (M:N) 관계를 구현을 해 준다.

Django에서 `ManyToManyField` 를 제공해 줌으로써 우리는 중개 모델을 쉽게 구현할 수 있다.

또한 차후에 `through` 아규먼트를 통해서 중개 테이블에 원하는 정보도 추가할 수 있다.

실습해보자.

사람들이 게시판에 movie에 대한 정보를 업로드 하면 해당 게시글에 좋아요 버튼을 달아 볼 것이다.

먼저 models.py 의 movie 클래스에 like_users 를 추가함으로

중개모델을 하나 추가로 만들자. (ManyToManyField 추가)

```
# movies/models.py

class Movie(models.Model):
    user = models.ForeignKey(settings.AUTH_USER_MODEL,
                             on_delete=models.CASCADE)

    like_users = models.ManyToManyField(settings.AUTH_USER_MODEL,
                                        related_name='like_movies')
    title = models.CharField(max_length=20)
    description = models.TextField()
```

우리가 like_users 필드를 생성하면 자동으로 Django에서 **.movie_set** 라는 매니저가 생성된다.

우리가 저번 시간에 댓글을 구현하면서 `.comment_set.all()`를 이용해서 역참조를 했던 것과 같은 원리다.

복습해보자.

```
class Comment(models.Model):
    movie = models.ForeignKey(Movie, on_delete=models.CASCADE)
    user = models.ForeignKey(settings.AUTH_USER_MODEL,
                             on_delete=models.CASCADE)
    content = models.CharField(max_length=100)
```

- 특정 댓글의 작성자를 조회 할 때에는 comment.user로 조회를 하면 된다.
- 특정 댓글의 게시글을 조회 할 때에는 comment.movie로 조회하면 된다.
- 그리고 특정User가 작성한 모든 댓글을 확인하고자 한다면 user.comment_set.all() 이 될 것이고

- 특정 영화 게시글에 달린 모든 댓글을 확인하고자 한다면 movie.comment_set.all() 가 될 것이다.

복습은 마치고 다시 돌아오자.

```
# movies/models.py

class Movie(models.Model):
    user = models.ForeignKey(settings.AUTH_USER_MODEL,
                             on_delete=models.CASCADE)

    like_users = models.ManyToManyField(settings.AUTH_USER_MODEL,
                                        related_name='like_movies')
    title = models.CharField(max_length=20)
    description = models.TextField()
```

우리가 like_users 필드를 생성하면 자동으로 Django에서 **.movie_set** 라는 매니저가 생성된다고 했다.

그러나 지금 방금 Movie 모델클래스에 like_users 필드를 추가 한 후에는

- movie.user 는 영화 게시글을 작성한 사람이 될 것이며
- **user.movie_set** 이 의미하는 것이 [user가 작성한 글] 이 되겠다. (역참조)

- movie.like_users 는 영화 게시글에 “좋아요를 누른 사람"이 될 것이지만
- **user.movie_set**가 의미하는 것이 [user가 “좋아요”를 누른 글]이 되어야 하지만 (역참조)

여기서 문제가 발생했다.

user.movie_set 이 의미하는 것이 [user가 작성한 글] 을 의미 하는지, 또는

user.movie_set 가 의미하는 것이 [user가 “좋아요”를 누른 글] 을 의미 하는지 구분할 수가 없게 되었다.

[user가 작성한 글] 과 [user가 "좋아요"를 누른 글] 을 구분하기 위해서 우리는 추가로 User 모델과 관계된 ForeignKey 또는 ManyToManyField 중 한 곳에 related_name을 추가로 작성해서 서로를 각각 구분을 해줘야 한다.

따라서 중계모델인 like_users 속성에 `related_name='like_movies'` 를 추가해 주었다.

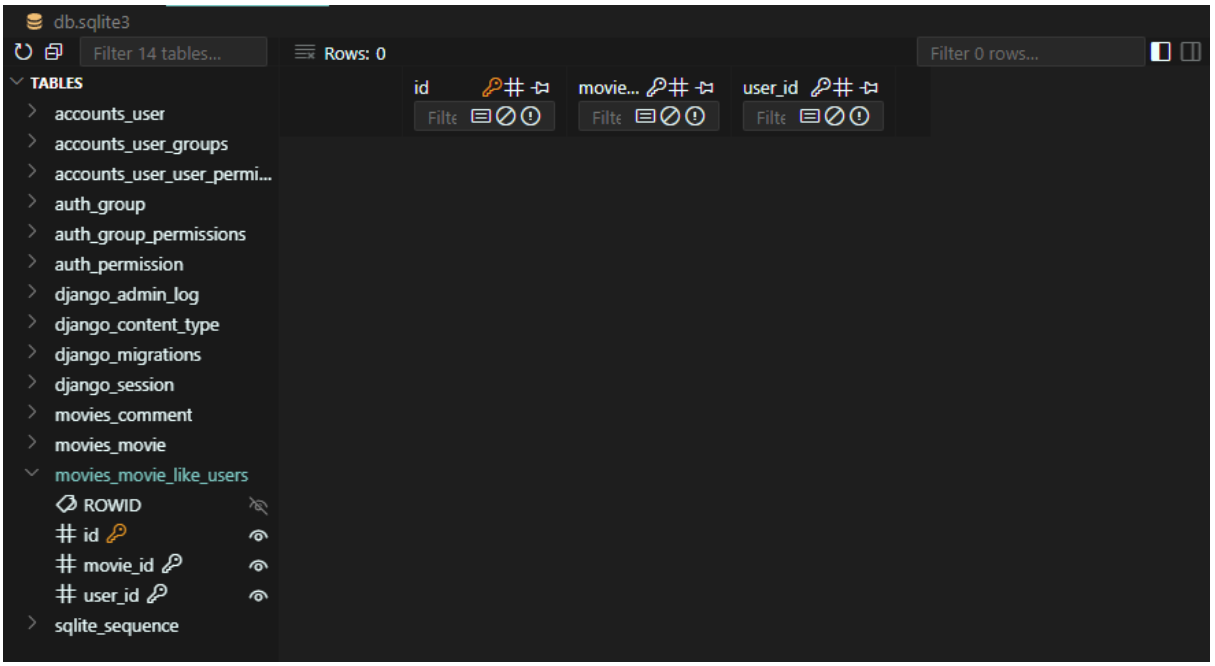
이렇게 되면

- [user가 작성한 글] 은 기존대로 **user.movie_set** 으로 조회를 하면 되고
- [user가 "좋아요"를 누른 글]을 조회하기 위해서는 이제 **user.like_movies** 로 조회를 하면 된다.

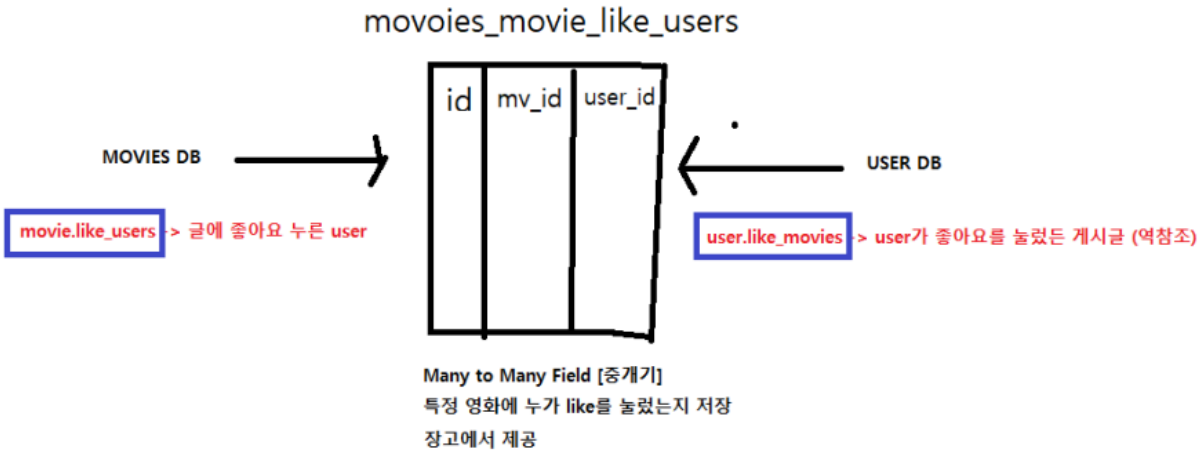
like_users 필드를 모델에 추가 함으로써 model 변경이 있으니 migrate를 해주자.

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

생성된 중개 테이블을 확인 할 수 있다.



아래 그림은 Movies 앱의 movie_like_users 라는 "중개테이블"을 그림으로 나타낸 것이다.



ManyToManyField 중개기는 특정 영화 게시글에 누가 "좋아요"를 눌렀는지 저장하는 중개테이블이 되겠다.

이쯤에서 movie와 user 모델관계를 한번 정리를 해보자.

키워드가 나중에 혼란스럽지 않도록 중간 정리를 하는 것이다.

- 게시글 입장에서 보자.

`movie.user` 라고 표현을 한다면 특정 [영화 게시글을 작성한 유저]가 될 것이고 (N:1)

`movie.like_users` 는 특정 [게시글을 "좋아요"한 유저]가 될 것이다. (M:N)

- 유저 입장에서는 (이 부분 부터는 역참조 이다.)

`user.movie_set.all` 라고 표현을 한다면 특정 [유저가 작성한 모든 게시글]을 의미하며 (N:1)

`user.like_movies` 는 특정 [유저가 "좋아요"한 게시글]이 되는 것이다. (M:N)

자 이제 본격적으로 좋아요를 구현을 해보자.

먼저 경로를 추가하고 likes라고 함수를 정의 하자.

```
# movies/urls.py

urlpatterns = [
```

```
    ...
    path('<int:movie_pk>/likes/', views.likes, name='likes'),
]
```

```
# movies/views.py

@require_POST
def likes(request, movie_pk):
    if request.user.is_authenticated:
        movie = Movie.objects.get(pk=movie_pk)
        if movie.like_users.filter(pk=request.user.pk).exists():
            movie.like_users.remove(request.user)
        else:
            movie.like_users.add(request.user)
        return redirect('movies:detail', movie_pk)
    return redirect('accounts:login')
```

likes 함수를 살펴보자.

- if 만약에 로그인된 user 라면
    - 해당 게시글에 좋아요를 누른 유저중 filter를 통해서 특정 유저가 좋아요를 눌러는지 확인

      해당 유저의 라이크를 지우고
    - else 그게 아니라면 해당 유저의 라이크를 더한 후 해당 게시물의 디테일 페이지로 간다.
- 로그인이 되어 있지 않다면 로그인 페이지로 리다이렉트 한다.

참고로 filter() remove() add() 함수들은 모두 Django ORM 문법이다. (파이썬 문법x)

.exists() 함수는 QuerySet에 결과가 포함되어 있으면 True를 반환하고 그렇지 않으면 False를 반환한다.

이 함수는 큰 QuerySet에 있는 특정 개체의 존재 여부를 확인할 때 유용하다.

덧붙여 filter를 통해 like 를 취소하고 추가하는 기능도 설정했다.

다음에는

게시글의 detail 페이지에 [좋아요 버튼]을 달아보자.

```
# movies/detail.html

<div>
  <h5>{{ movie.title }}</h5>

  <form action="{% url 'movies:likes' movie.pk %}" method="POST">
    {% csrf_token %}
    {% if request.user in movie.like_users.all %}
      <input type="submit" value="좋아요 취소">
    {% else %}
      <input type="submit" value="좋아요">
    {% endif %}
  </form>

  <p>{{ movie.description }}</p>

    ...
```

- if 만약에 `user.movie_set.all` (해당 영화에 라이크를 누른 모든 유저중)
  - `in` 방금 라이크를 누른 유저가 존재 한다면 "좋아요 취소" 가 작동
- else 그게 아니면 "좋아요"가 눌릴 것이다.

이제 서버를 켜서 게시글의 디테일 페이지에 들어가서 좋아요를 눌러보자.

"좋아요"를 클릭할 수 있는 기능을 구현 했는데 "좋아요" 버튼을 여러 번 클릭을 하다보면 버튼을 클릭 할 때 마다 해당 웹페이지에 새로고침이 발생 한다는 것을 알 수 있다. 부드러운 사용자 경험을 위해서 웹페이지에 새로고침이 발생하지 않고 해당 버튼을 통한 값만 바뀌었으면 하는데 이는 차후에 javascript를 학습한 후 "비동기통신"을 통해 새로고침이 없는 "좋아요"가 구현 될 수 있도록 업그레이드 해 볼 것이다.

추가로 "좋아요"의 개수도 표시 하려면 방금 추가로 작성한 form 태그 아래에

```
<span>좋아요 : {{ movie.like_users.count }}개</span>
```

도 추가를 해 주면 되겠다.

지금까지 "좋아요" 기능 구현을 살펴 보았다.

"좋아요" 기능은 M:N 관계로 Movie 게시글 과 user 사이에 중계기를 통해서 ManyToMany 관계를 설정 해 보았다.

다음 번에는 Follow / Unfollow 기능을 구현 해 보자.

이 또한 M:N 관계로 User 와 User 사이에서의 M:N 관계를 살펴 볼 것이다.

[참고] 좋아요 까지 구현한 완성본이다.

django_test.zip