8기 파이썬반 PJT - 8 (1)

Created	@2022년 10월 11일 오후 9:44
Last Edited Time	@2022년 10월 11일 오후 11:54
Type	
Status	
Created By	
Last Edited By	
Stakeholders	

1. 기초 설정

가상환경 생성, 패키지 설치

```
$ python -m venv ~/venv
$ source ~/venv/Scripts/activate
$ pip install -r ./requirements.txt
```

프로젝트와 필요한 앱을 생성한다.

```
$ django-admin startproject pjt08 .
$ python manage.py startapp accounts
$ python manage.py startapp community
$ python manage.py startapp movies
```

일단, `settings.py` 부터 건드려준다.

```
# settings.py

INSTALLED_APPS = [
    'accounts',
    'community',
    'movies',
]

TEMPLATES = [
    {
        'DIRS': [BASE_DIR / 'templates', ],
```

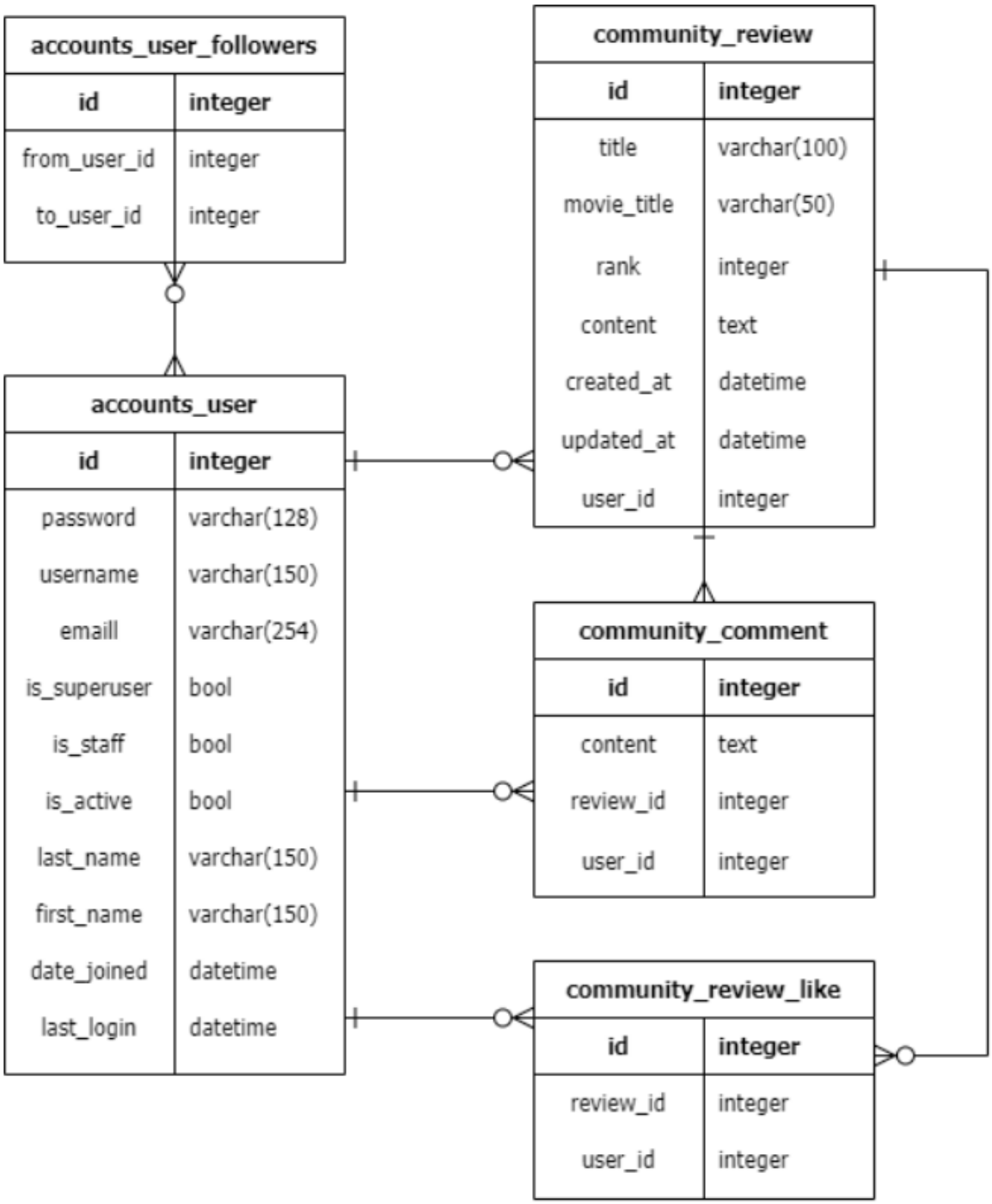
```
    }
]

LANGUAGE_CODE = 'ko-kr'

TIME_ZONE = 'Asia/Seoul'

AUTH_USER_MODEL = 'accounts.User'
```

다음, 전역 `urls.py` 를 설정하자.

```
from django.contrib import admin
from django.urls import path, include

urlpatterns = [
    path('admin/', admin.site.urls),
    path('community/', include('community.urls')),
    path('accounts/', include('accounts.urls')),
    path('movies/', include('movies.urls')),
]
```

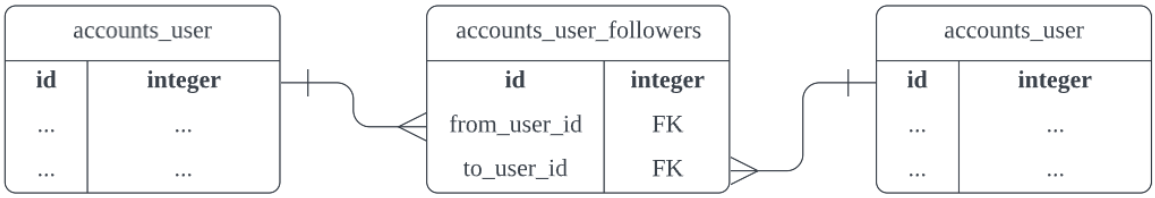
이제 `accounts` 앱을 설정하자.

ERD 를 살펴보면 다음과 같다.



자세히 살펴보면, `community_review`, `community_comment` 는 `accounts_user` 에서 FK 를 가져와서 쓸 것이기 때문에 `accounts/models.py` 에선 구현할 게 없고, `community_review_like` 는 `accounts_user` 와 `community_review` 의 브릿지 테이블일 뿐이라서 `ManyToManyField` 로 구성하면 된다.

우리가 `accounts/models.py` 에서 신경써야 할 건 `accounts_user_followers` 인데, 이것은 `ManyToManyField` 로 구성되어야 한다. 테이블을 쪼개보면 다음과 같아지는데,



즉, 양쪽에 `accounts_user` 가 있고, 하나의 브릿지테이블이 차지하고 있는 형태가 된다.

다시 말하자면, 자기 자신에게 ManyToMany 관계를 준 것이라고 할 수 있다.

그래서, `ManyToManyField` 함수를 사용하기 위해 고려해야 할 사항은 다음과 같다.

첫번째로, 유저가 유저에게 ManyToMany 를 한 경우이므로, 첫번째 인자에 자기 자신을 의미하는 `'self'` 를 넣어줘야한다.

두번째, 대칭인지 비대칭인지 고려해서 `symmetrical` 을 지정해야 한다.

자세히보면, `from_user_id` 에서 `to_user_id` 로, 오로지 한쪽에서만 다른 유저에게로 팔로우를 할 수 있는것으로 확인되는데, 한쪽이 팔로우를 했다고 해서 다른 유저가 똑같이 팔로우함을 의미하진 않는다. 즉, “대칭이 되지 않는다.” 이를 비대칭 재귀 참조라고 하며, 이런 테이블을 만들 땐 `symmetrical=False` 옵션을 지정해주면 된다.

- 그렇다면, `symmetrical=True` 가 되는 관계는 대칭되는 관계란 뜻인데, 어떤 경우가 있을까? 한쪽이 친구 추가를 하면 다른 쪽도 자동으로 친구가 되어서 양쪽에서 친구로 등록되는 경우 대칭이라고 할 수 있다.

이에, `accounts_user_followers` 를 포함한 모델을 만들어보면 다음과 같다.

```
# accounts/models.py

from django.db import models
from django.contrib.auth.models import AbstractUser


class User(AbstractUser):
    followings = models.ManyToManyField('self', symmetrical=False, related_name='followers')
```

테이블 이름은 `followings` 이고, 이 안에 양쪽의 FK 가 담길 것이며, 접근은 `followers` 로 한다.

`related_name='followers'` 를 지정해주었으므로 해당 테이블에 접근할 땐 `followers` 라는 이름으로 접근한다.

`accounts/admin.py` 는 다음과 같다.

```
# accounts/admin.py

from django.contrib import admin
from django.contrib.auth.admin import UserAdmin
from .models import User


admin.site.register(User, UserAdmin)
```

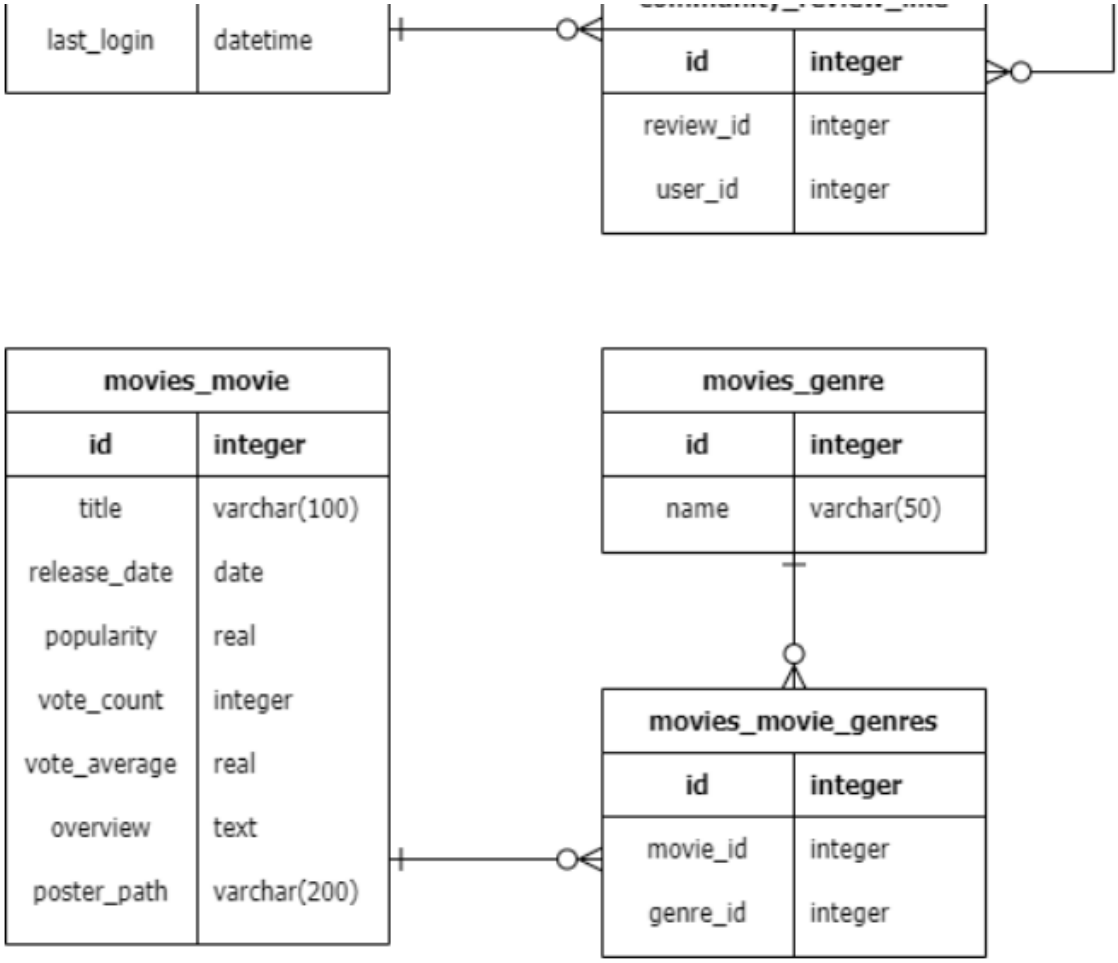
어드민 사이트에 커스텀 유저 모델을 등록한다.

마지막으로, `accounts/urls.py` 는 다음과 같이 작성한다.

```
# accounts/urls.py

from django.urls import path
from . import views

app_name = 'accounts'

urlpatterns = []
```

다음, `movies` 앱을 설정하자.

우리의 ERD 를 확인해보면 독특한 점을 볼 수 있다.



ERD 상으로, `movies` 는 완전히 따로 논다. `community` 와 `accounts` 와는 아무 관련 없는것이 확인된다.

단, `movies_movie_genres` 는 단순히 브릿지테이블이기때문에 `ManyToManyField` 로 설정할 수 있다는 것과, `real` 은 실수 타입을 의미하므로 `IntegerField` 클래스를 이용해야 한다는 것만 알아두자.

따라서, 이를 기반으로 `movies/models.py` 를 설계해보면 다음과 같은 결론이 나온다.

```
# movies/models.py

from django.db import models

class Genre(models.Model):
    name = models.CharField(max_length=50)

class Movie(models.Model):
    title = models.CharField(max_length=100)
    release_date = models.DateField()
    popularity = models.FloatField()
    vote_count = models.IntegerField()
    vote_average = models.FloatField()
    overview = models.TextField()
```

```
    poster_path = models.CharField(max_length=200)
    genres = models.ManyToManyField(Genre)
```

`movies/admin.py` 는 다음과 같다.

```
# movies/admin.py

from django.contrib import admin
from .models import Genre, Movie

# Register your models here.
admin.site.register(Genre)
admin.site.register(Movie)
```

자, 이제 사용자가 `localhost:8000/movies/` 로 접속했을 때 보일 간단한 인덱스 창을 만들자.

```
# movies/urls.py

from django.urls import path
from . import views

app_name = 'movies'

urlpatterns = [
    path('', views.index, name='index'),
]
```

`movies/views.py` 는 다음과 같다.

```
# movies/views.py

from django.shortcuts import render

# Create your views here.
def index(request):
    return render(request, 'movies/index.html')
```

프로젝트 루트 경로 바로 아래에, `base.html` 을 다음과 같이 작성한다.

```
<!DOCTYPE html>
<html lang="en">
  <head>
```

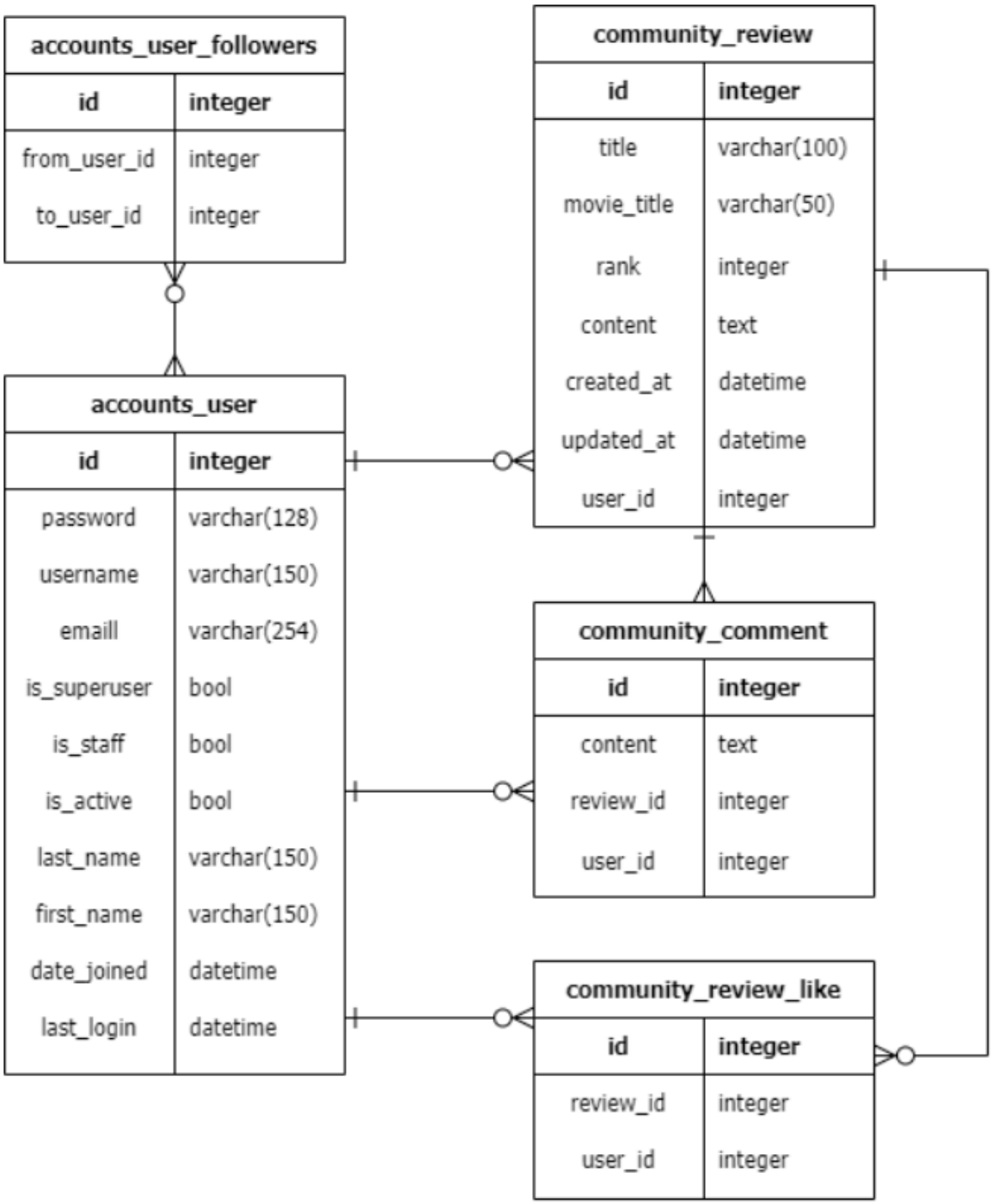
```
    <meta charset="UTF-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Document</title>
  </head>
  <body>
    <div>
      {% block content %}
      {% endblock content %}
    </div>
  </body>
</html>
```

`movies/templates/movies/index.html` 을 생성하고, 다음과 같이 작성한다.

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

{% block content %}
  <h1>MOVIE INDEX</h1>
{% endblock %}
```

마지막으로, `community` 앱을 설정할 차례다.

우리가 구현할 ERD 의 형태를 보면 다음과 같다.



딱 봐도 알겠지만, `accounts_user` 와 매우 밀접한 관련이 있다. `community_review_like` 라는 하나의 브릿지테이블이 끼여있고, 나머지 두 개 테이블은 `accounts_user` 와 1대N 관계이며, 자기들도 1대N 관계이다.

이를 기반으로 구현한 `community/models.py` 는 다음과 같다.

```
# community/models.py

from django.db import models
```

```
from django.conf import settings


class Review(models.Model):
    title = models.CharField(max_length=100)
    movie_title = models.CharField(max_length=50)
    rank = models.IntegerField()
    content = models.TextField()
    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)
    user = models.ForeignKey(settings.AUTH_USER_MODEL, on_delete=models.CASCADE)
    like_users = models.ManyToManyField(settings.AUTH_USER_MODEL, related_name='like_reviews')


class Comment(models.Model):
    content = models.TextField()
    review = models.ForeignKey(Review, on_delete=models.CASCADE)
    user = models.ForeignKey(settings.AUTH_USER_MODEL, on_delete=models.CASCADE)
```

- 질문. `Comment` 는 브릿지테이블이 아닌가? 1대N, 1대N 사이에 끼여있지않은가.

  브릿지테이블의 조건은 PK 하나, 양쪽 FK 하나씩만 보유하고 있어야 한다. 총 세 개의 필드만 있어야한다. 그러나 이 경우는 실제 댓글이 존재하기에 브릿지로 처리하면 안되고 별도의 테이블로 존재해야한다.

`community/admin.py` 는 다음과 같다.

```
# community/admin.py

from django.contrib import admin
from .models import Review, Comment

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

`movies` 와 마찬가지로, 사용자가 `localhost:8000/community` 로 접속했을 때 보일 간단한 창을 만들자.

`community/urls.py` 생성 후, 다음과 같이 작성한다.

```
# community/urls.py

from django.urls import path
from . import views

app_name = 'community'

urlpatterns = [
    path('', views.index, name='index'),
]
```

`community/views.py` 는 다음과 같다.

```
# community/views.py

from django.shortcuts import render

# Create your views here.
def index(request):
    return render(request, 'community/index.html')
```

`community/templates/community/index.html` 을 생성하고, 다음과 같이 작성한다.

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

{% block content %}
  <h1>COMMUNITY INDEX</h1>
{% endblock %}
```

이제 마이그레이션 생성 및 마이그레이트, 관리자 생성하자.

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

이후, 주어진 fixture 파일을 로드하자.

```
# json 파일은 movies/fixtures 에 위치한다.
$ python manage.py loaddata movies.json
```

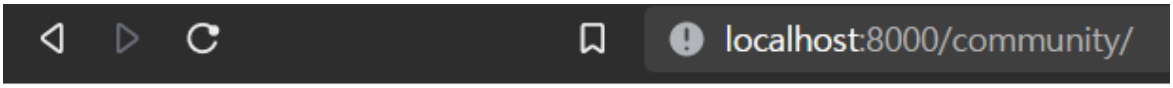
그리고 SQLite 를 열어, `movie` 쪽의 테이블들을 확인해보자.

id	title	release_date	popularity	vote_count	vote_average	
1	가브리엘의 지옥 파트 2	2020-07-31	12.087	883	8.9	
2	가브리엘의 지옥	2020-05-29	12.701	1476	8.9	
3	Dedicada a mi ex	2019-11-01	21.833	246	8.8	
4	쇼생크 탈출	1994-09-23	40.468	17292	8.7	촉망받는 은행
5	용감한 자가 신부를 데려가리	1995-10-20	15.604	2405	8.7	영국에서 유학
6	대부	1972-03-14	41.163	13088	8.7	시실리에서 이

서버 동작시켜보자.

```
$ python manage.py runserver
```

`localhost:8000/community` 접속 시 다음과 같이 보여야하며,

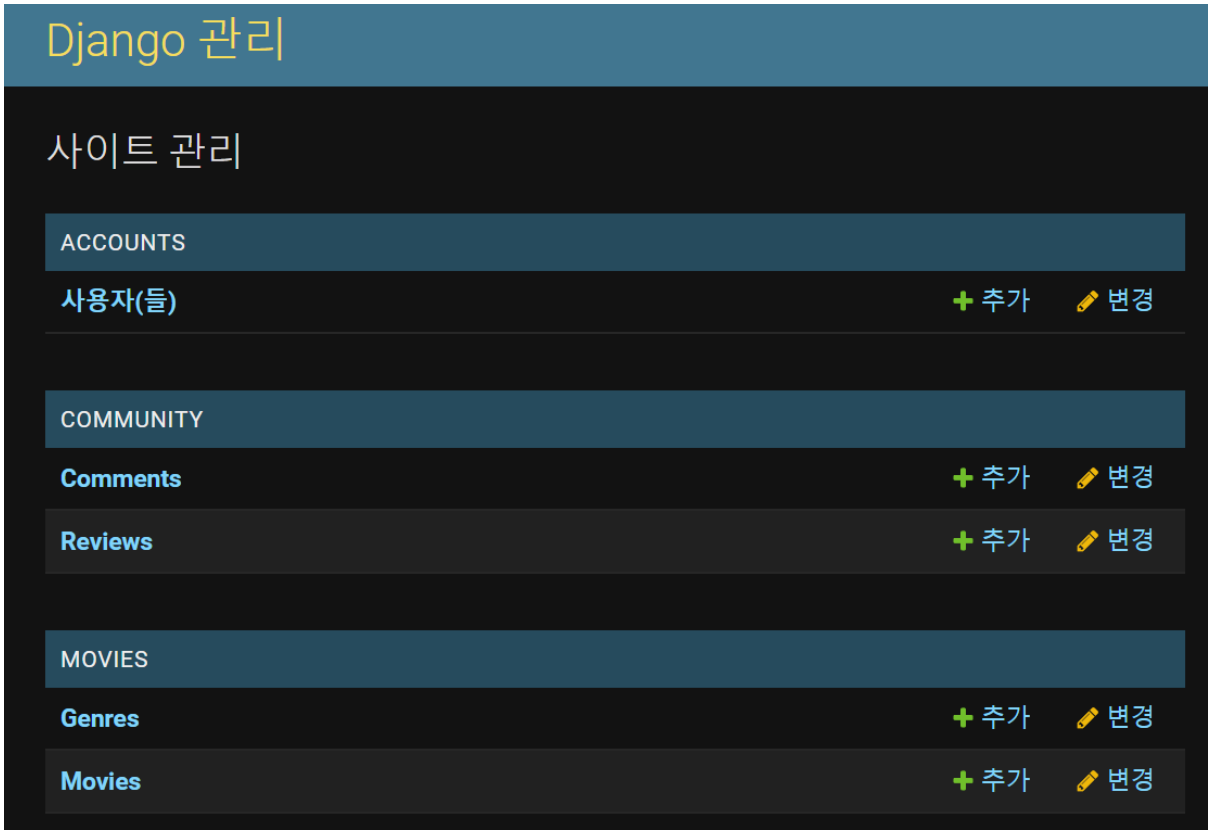


COMMUNITY INDEX

`localhost:8000/movies` 접속 시 다음과 같이 보여야 한다.

MOVIE INDEX

그리고 관리자 페이지에서, 다음 테이블들이 확인되어야한다.


Django 관리
사이트 관리
ACCOUNTS
사용자(들)
추가
변경
COMMUNITY
Comments
추가
변경
Reviews
추가
변경
MOVIES
Genres
추가
변경
Movies
추가
변경