数据库应用系统设计：影评系统

姓名	学号	担任工作
李明俊	21307140	所有相关工作，包括前期资料收集，代码整体框架的构建，open gauss数据库的配置与连接，后端和前端代码的实现，影评系统的实现，报告的撰写等

一. 影评系统介绍

在当今这个电影作品层出不穷的时代，观众面临着选择过多的困境。从好莱坞大片到独立电影，从经典老片到最新上映，电影的多样性和数量使得找到那些符合个人口味和兴趣的影片变得越来越具有挑战性。正是在这样的背景下，我们迫切需要一个系统，它不仅能帮助用户高效地浏览和筛选出广泛的电影库存，还能与其它影迷共同交流对电影的看法。这样的系统可以极大地丰富观众的观影体验，使他们在电影的海洋中轻松找到那些能触动心灵、引发共鸣的佳作。

本次数据库应用系统设计的影评系统就是旨在为电影爱好者提供一个互动和信息丰富的平台，使他们能够分享自己对电影的看法，并发现适合自己的电影。因此，在我设计的影评系统中主要实现了以下功能：

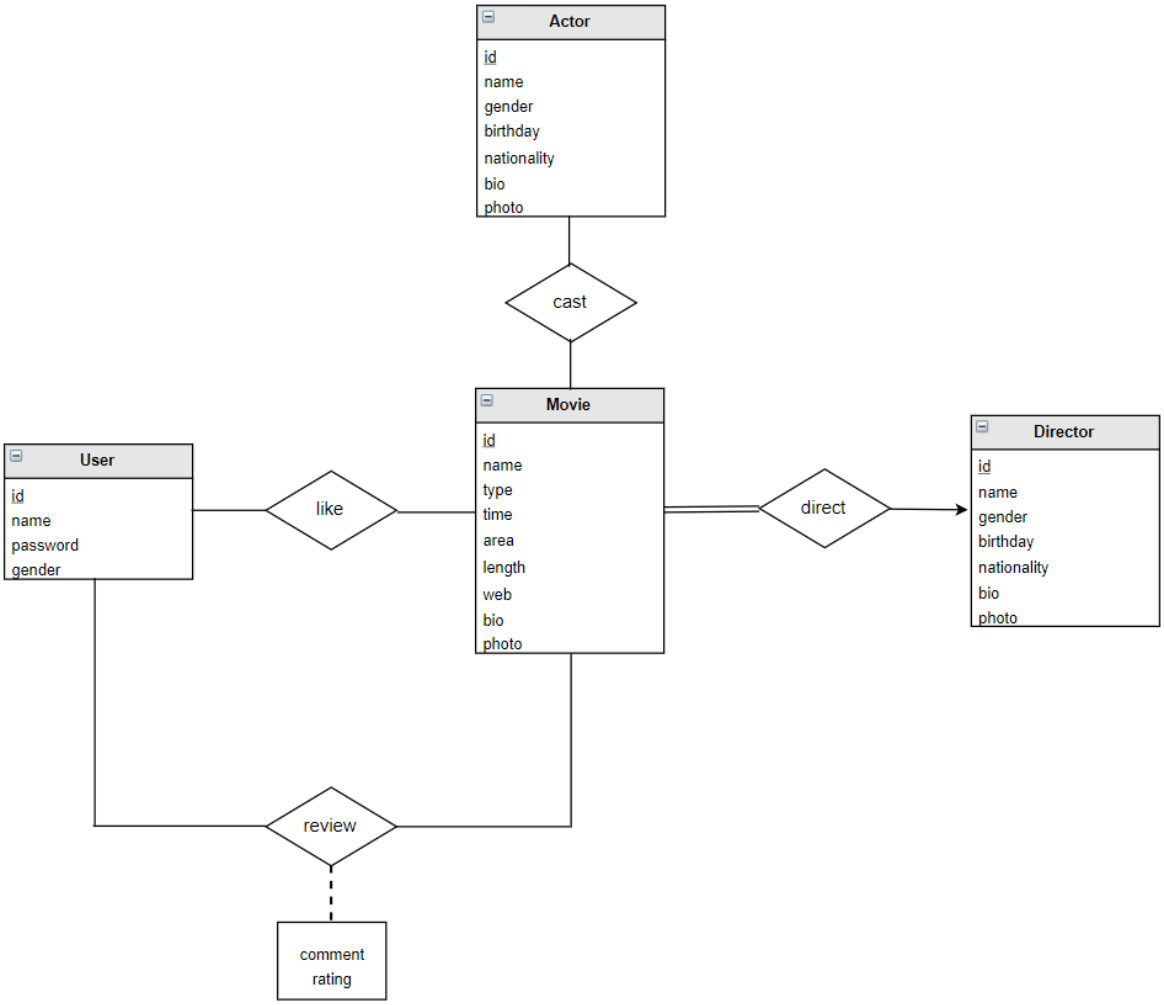
1. **注册和登录功能**：在这个系统中，用户可以轻松创建属于自己的账户，仅需提供一些基本信息，如用户名、密码和性别等。这一简洁而直观的注册流程让每位用户都能享有一个独一无二的个人空间。注册后，系统便能够根据不同用户的登录状态，展现针对性的个性化内容，满足用户的独特喜好和需求。此外，为了保障用户的安全和隐私，登录时必须输入密码。这一安全机制不仅确保了用户信息的保密性，还增强了整个平台的信任度和可靠性。
2. **管理员功能**：设置了一个超级用户作为管理员，管理员登陆后与普通用户登陆后界面不同，可以进入影评系统的管理员界面。在该界面支持管理员对系统的数据库内容（如导演信息，电影信息，用户信息，评论信息，收藏信息，演员信息，出演信息等）进行增、删、查、改的操作，从而有效地监控和管理用户活动，审查内容，更新平台信息，维护社区健康等。这包括对用户账户的管理、对评论内容的审查与过滤、对电影和导演信息的增添与修改等等。通过这些综合性的管理功能，管理员不仅能够丰富平台的内容，还能营造一个和谐的氛围，极大地提升了用户体验的整体质量。
3. **电影检索功能**：在普通用户登陆后就可以进入用户首页，在首页中用户可以根据自己的需求按照电影名称，导演姓名，电影类型，电影上映时间，制片地区等进行电影的个性化检索。这种个性化检索机制大大提高了用户找到合适电影的效率，减少了在海量电影资料中盲目搜索的时间和精力。此外，这种灵活的搜索方式也为用户提供了探索新电影和发现未知佳作的机会，满足不同用户的独特需求，让每个人都能在这个电影的宝库中找到自己的宝藏。
4. **电影信息查看功能**：当用户点击筛选出的电影后，会有单独电影页面显示电影的名称，导演，演员，类型，制片地区，时长，观看网址，电影简介，电影评论等详细的电影信息。此外，用户可以点击导演名字查看导演的详细信息，也可以点击演员名字查看演员的详细信息。这样，用户就可以进一步地浏览电影信息和其他影迷的评论来判断自己是否要观看这部电影，从而满足了用户对于深入了解电影背景和内容的需求。此外，集成的评论功能允许用户获取其他观众的观点和感受，为他们提供社群层面的参考。该功能不仅增强了用户的决策能力，还丰富了他们的观影体验。

5. **电影收藏功能**：用户可以在电影界面点击收藏按钮对该电影进行收藏，收藏后的电影将会列在用户的个人主页中。这个功能满足了用户对于个性化和便捷管理电影喜好的需求。通过收藏功能，用户能够创建一个个人化的电影列表，这不仅帮助他们追踪想要观看或再次观看的电影，而且方便在不同设备和时间里访问这些收藏。此外，这个功能还为用户提供了一个简单的方式来组织和回顾他们的电影观看历史，增强了个人电影体验的连续性和个人投入感。
6. **电影评论功能**：用户可以在电影界面给出自己对这个电影的评分并发表对电影的评论，评论成功后该评论可以被其它所有影迷看到以供参考。种开放的评论机制允许用户之间分享观点、体验和感受，帮助影迷更好地根据他人的观点选择适合自己的电影。

该影评系统集合了上面的所用功能，从用户的需求出发来设计数据库应用系统。

二. 影评系统设计

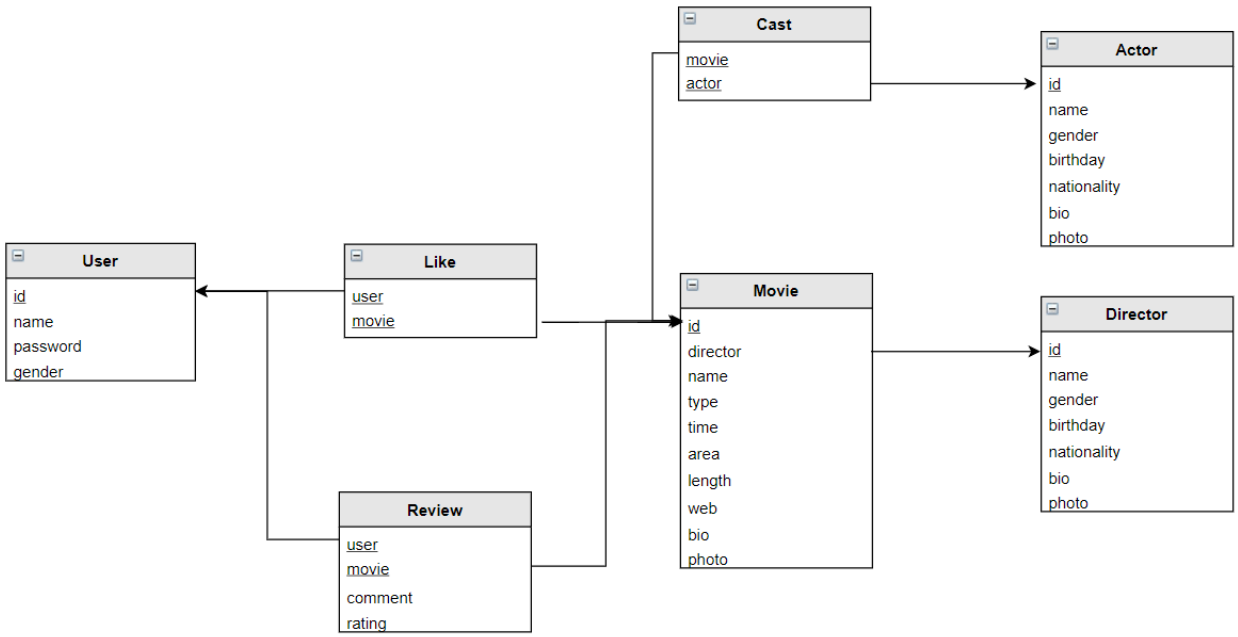
系统的ER图设计如下，有四个强实体集Movie, User, Actor, Director分别表示电影，用户，演员，导演，每个强实体集中都包含了该实体所需的属性。User和Movie间通过联系集like连接，表示某用户喜欢某电影。Actor和Movie间通过联系集cast连接，表示某演员出演某电影。Director和Movie间通过联系集direct连接，表示某电影是由某导演制作。此外User和Movie间还通过联系集review连接，表示某用户对某电影的评论，该联系集有两个额外属性comment和rating表示评论内容和评分。



根据上面的ER图，设计出该系统的数据库架构图如下，共有七个数据库表格：

1. Movie：包含属性电影id，导演，电影名称，类型，上映时间，制片地区，片长，观看网址，电影简介，电影海报。director是指向Director表的外键

2. Director：包含属性导演id，导演姓名，导演性别，导演生日，导演国籍，导演简介，导演照片
3. Actor：包含属性演员id，演员姓名，演员性别，演员生日，演员国籍，演员简介，演员照片
4. Cast：包含指向Movie的外键和指向Actor的外键
5. User：包含属性用户id，用户姓名，密码，用户性别
6. Like：包含指向Movie的外键和指向User的外键
7. Review：包含指向Movie的外键和指向User的外键，以及属性评论内容和评分



三. 影评系统实现

本次作业的影评系统是通过Django项目连接OpenGauss数据库实现的，下面分为“项目架构”，"数据库的连接与构建"，“后端构建”，前端构建“”四个部分来介绍。

（一）项目框架

本次项目框架中核心部件如下：

```
- MyProject
  - .venv          项目使用的python3.9
  - app1
    - migrations   存放数据库表格迁移的相关函数和信息
    - static       存放前端相关组件和照片
    - templates
      - manager    存放和管理员界面相关的html文件
      - user       存放和用户界面相关的html文件
    - templatetags
      - models.py  编写项目数据库表格
      - views.py   编写后端函数
  - DataBaseProject
    - settings.py  项目各项属性配置
    - urls.py      存放url网址和后端函数间的映射
  - media          存放数据库中的图片资源
```

其中数据库主要在models.py中构建，后端代码主要在view.py中构建，前端代码主要在templates和static文件中构建。

后端：接收来自前端的信息，与数据库交互实现相关功能（如增删查改等），将数据传递给前端

前端：接收来自后端的数据，渲染出网页的界面，将信息传给后端

在django中通过 “数据库 - 后端 - 前端” 的相互协调与配合，实现了本次作业中的影评系统

（二）数据库的连接与构建

首先在虚拟机配置好opengauss，更改配置文件，将其**加密方式改为md5**（很重要！！如果使用sha256将无法与django正常连接），关闭防火墙，然后创建用户testuser，设置密码为gauss@123

```
openGauss=# CREATE USER testuser PASSWORD 'gauss@123';
NOTICE:  The encrypted password contains MD5 ciphertext, which is not secure.
CREATE ROLE
```

然后授予该用户所有的数据库权限：

```
openGauss=# GRANT ALL  PRIVILEGES TO testuser;
ALTER ROLE
```

opengauss配置好后，在django的setting文件中输入数据库信息以连接opengauss，若程序运行没有报错则说明成功连接了opengauss。若不能连接则可能和数据库配置或者django工具版本有关，下面为我在连接数据库过程中出现的问题：

1. opengauss数据库的加密方式必须更改为MD5。更改加密方式后原来的用户不能使用，需要更改管理员用户密码然后创建新的用户。
2. python使用的版本不能过高，本次项目使用的是3.9版本
3. 可能需要根据报错信息降低django的版本

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql_psycopg2',
        'NAME': 'postgres',
        'USER': 'testuser',
        'PASSWORD': 'gauss@123',
        'HOST': '192.168.56.101',
        'PORT': 7654
    }
}
```

成功连接上opengauss数据库后，就可以根据上一部分的ER图和架构图在model文件中编写本次项目的数据库表了，代码如下：

```
from django.core.validators import MinValueValidator, MaxValueValidator
from django.db import models
# 用户表----------------------------
class User(models.Model):
    username = models.CharField(max_length=100, unique=True) #姓名
    password = models.CharField(max_length=32, unique=False) #密码
    gender = models.CharField(max_length=10) #性别
    class Meta: #指定表格迁移后在数据库的名称，若不指定django会自动添加些前缀
        db_table = "User"
```

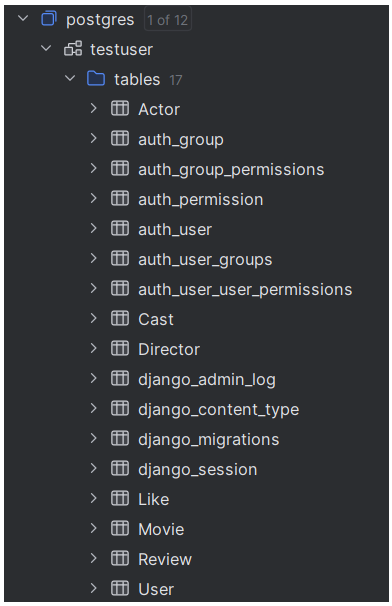
```python
# 导演表--------------------------
class Director(models.Model):
    directorname = models.CharField(max_length=100, unique=True) #姓名
    gender = models.CharField(max_length=10) #性别
    birthday = models.CharField(max_length=20) #生日
    nationality = models.CharField(max_length=20) #国籍
    bio = models.TextField() #简介，TextField类型的
    #照片，ImageField类型，照片内容存储在media/photos中
    photo = models.ImageField(upload_to='photos/', blank=True, null=True),
    class Meta:
        db_table = "Director"
# 电影表--------------------------
class Movie(models.Model):
    moviename = models.CharField(max_length=100)
    director = models.ForeignKey(to="Director", to_field="id",
on_delete=models.CASCADE)
    type = models.CharField(max_length=40)
    time = models.CharField(max_length=20)
    area = models.CharField(max_length=20)
    length = models.IntegerField()
    web = models.URLField(max_length=100)
    bio = models.TextField()
    photo = models.ImageField(upload_to='photos/', blank=True, null=True)
    class Meta:
        db_table = "Movie"
# 收藏表--------------------------
class Like(models.Model):
    #user和movie是级联删除的外键
    user = models.ForeignKey(User, to_field="id", on_delete=models.CASCADE)
    movie = models.ForeignKey(Movie, to_field="id", on_delete=models.CASCADE)
    class Meta:
        db_table = "Like"
        unique_together = ('user', 'movie') #user和movie的组合是唯一的
# 评论表--------------------------
class Review(models.Model):
    user = models.ForeignKey(User, to_field="id", on_delete=models.CASCADE)
    movie = models.ForeignKey(Movie, to_field="id", on_delete=models.CASCADE)
    comment = models.TextField()
    rating = models.IntegerField(
        validators=[MinValueValidator(1), MaxValueValidator(5)],  # 限制评分在1到5之间
        null=True, blank=True)
    class Meta:
        db_table = "Review"
        unique_together = ('user', 'movie')
# 演员表--------------------------
class Actor(models.Model):
    actorname = models.CharField(max_length=100)
    gender = models.CharField(max_length=10)
    birthday = models.CharField(max_length=20)
    nationality = models.CharField(max_length=50)
    bio = models.TextField(blank=True, null=True)
    photo = models.ImageField(upload_to='photos/', blank=True, null=True)
```

```
    class Meta:
        db_table = "Actor"
# 出演表------------------------------
class Cast(models.Model):
    movie = models.ForeignKey(Movie, on_delete=models.CASCADE)
    actor = models.ForeignKey(Actor, on_delete=models.CASCADE)
    class Meta:
        db_table = "Cast"
        unique_together = ('movie', 'actor')
```

然后在项目的终端输入下面的指令将model中的表格迁移到opengauss数据库，在数据库中完成表的构建

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

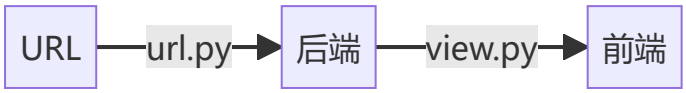
查看数据库内容，表格已经在数据库中创建，其余表格是django项目自动创建的



（三）后端构建

主要负责接收来自前端的信息，与数据库交互实现相关功能（如增删查改等），将数据传递给前端

后端既是前端与数据库连接的桥梁，也是url与前端连接的桥梁



url.py中存储的URL网址和对应后端函数如下：

```
    #初始网址，无内容，将自动导入到登陆界面
    path('', views.initial),
    #登录界面
    path('user/login/', views.user_login),
    #注册界面
    path('user/signup/', views.user_signup),
```

```
    #首页界面
    path('user/home/',views.user_home,name='user/home'),
    #电影详细信息界面
    path('user/movie/', views.user_movie, name='user_movie'),
    #增添收藏，只有后端函数实现该功能，没有单独的html界面
    path('add_to_favorites/', views.add_to_favorites, name='add_to_favorites'),
    #增添评论，只有后端函数实现该功能，没有单独的html界面
    path('add_comment/', views.add_comment, name='add_comment'),
    #我的主页界面
    path('myspace/',views.myspace, name='myspace'),
    #管理员界面，由七个表格的管理组成------------------------------
    # director----
    #导演列表界面
    path('manager/director/', views.manager_director),
    #导演增添界面
    path('manager/director_add/', views.manager_director_add),
    #导演删除，只有后端函数实现该功能，没有单独的html界面
    path('manager/director_delete/', views.manager_director_delete),
    #导演编辑修改界面
    path('manager/director_update/', views.manager_director_update),
    # movie---- ...和director类似
    # user ---- ...和director类似
    # review---- ...和director类似
    # like---- ...和director类似
    # actor---- ...和director类似
    # cast---- ...和director类似
```

下面展示view.py中各个后端函数的具体实现

1.登录

当从前端获取到GET请求时，则渲染登陆界面前端user/login.html到当前网页上

当从前端获取到POST请求（用户点击登录）时，则接收来自前端的用户名和密码，将用户名和加密后的密码与数据库用户表的内容比对：

(1)如果用户名不在数据库表中则显示用户名不存在

(2)如果用户名是manager则该用户为管理员，将当前网址重定位为/manager/director/进入管理员界面

(3)如果用户名和密码正确则重定位到/user/home/进入用户首页，并向该网址传入用户的id

(4)如果用户和密码与数据库表中内容不匹配则显示密码错误

```
def user_login(request):
    method = request.method
    if method == "GET":
        return render(request, "user/login.html")
    elif method == "POST":
        username = request.POST.get("username")
        password = request.POST.get("password")
        try:
            user = User.objects.get(username=username)
            user_id = user.id
```

```
            res = password + settings.SECRET_KEY
            password = hashlib.md5(res.encode("utf-8")).hexdigest()
            if password == user.password:
                if user.username == "manager":
                    return redirect("/manager/director/")
                else:
                    return redirect(f"/user/home/?user_id={user_id}")
            else:
                return render(request, "user/login.html", {"tip": "密码错误！"})
        except app1.models.User.DoesNotExist:
            return render(request, "user/login.html", {"tip": "用户名不存在！"})
```

2.注册

当从前端获取到GET请求时，则渲染注册界面前端user/signup.html到当前网页上

当从前端获取到POST请求（用户点击注册）时，则接收来自前端的用户名密码和性别，如果用户名在数据库中已经存在则显示"注册失败用户名已经存在"，否则将用户名，加密后的密码和性别作为新的表项存在用户数据表中并重定位到登陆界面

```
def user_signup(request):
    if request.method == "GET":
        return render(request, "user/signup.html")
    elif request.method == "POST":
        get_post = request.POST
        username = get_post.get("username")
        password = get_post.get("password")
        res = password + settings.SECRET_KEY
        password = hashlib.md5(res.encode("utf-8")).hexdigest()
        gender = get_post.get("gender")
        if User.objects.filter(username=username).exists():
            return render(request, "user/signup.html", {"msg": "注册失败，用户名已存
在!"})
        else:
            User.objects.create(username=username, password=password, gender=gender)
            return redirect("/user/login/", {"tip2": "注册成功！"})
```

3.用户首页

当从前端获取到GET请求（用户登录成功后）时，则渲染首页前端user/home.html到当前网页上，并将GET请求中包含的userid（在user_login函数中传递的）对应的数据库表项和电影列表传递给前端

当从前端获取到POST请求（用户输入查询内容并点击查询）时，则接收来自前端的查询信息和用户id，然后在数据库中模糊搜索出和查询信息有关的电影列表，最后把电影列表，用户，查询信息重新传给前端，首页就只会显示和查询内容相关的电影

```
def user_home(request):
    if request.method == "GET":
        movies = Movie.objects.all()
        user_id = request.GET.get('user_id')
        user = User.objects.get(id=user_id)
        return render(request, 'user/home.html', {'movies': movies, "user": user})
```

```
    if request.method == "POST":
        search_query = request.POST.get('search')
        user_id = request.POST.get('user_id')
        user = User.objects.get(id=user_id)
        movies = Movie.objects.filter(
            Q(moviename__icontains=search_query) |  # 电影名称模糊搜索
            Q(director__directorname__icontains=search_query) |  # 导演名称模糊搜索
            Q(type__icontains=search_query) |  # 类型模糊搜索
            Q(time__icontains=search_query) |  # 时间模糊搜索
            Q(area__icontains=search_query))  # 地区模糊搜索
        return render(request, 'user/home.html', {'movies': movies, "user": user,
"search_query": search_query})
```

4.电影详细信息

从GET请求（用户点击了某一部电影）中提取出电影和用户的id，在数据库中找到id对应的表项，找到电影外键的导演，和电影相关的评论，和电影相关的演员，再将上面所有内容传给user/movie.html前端，在前端显示电影详细信息

```
def user_movie(request):
    if request.method == "GET":
        movie_id = request.GET.get('movie_id')
        user_id = request.GET.get('user_id')
        user = User.objects.get(id=user_id)
        movie = Movie.objects.get(id=movie_id)
        director = movie.director
        reviews = Review.objects.filter(movie=movie)
        cast_list = Cast.objects.filter(movie=movie)
        actors = [cast.actor for cast in cast_list]
        context = {'movie': movie,'director': director,'user': user,
            'reviews': reviews,'actors': actors}
        return render(request, 'user/movie.html', context)
```

5.添加收藏

从前端获得ajax类型请求（用户点击收藏），并提取出在请求信息中的movie和user，然后在Like表中添加（user，movie）表项，最后将添加结果传递给前端，会在原网页显示"添加成功/失败"的浮窗

```
def add_to_favorites(request):
    if request.method == 'POST' and request.is_ajax():
        movie = request.POST.get('movie')
        user = request.POST.get('user')
        movie = Movie.objects.get(id=movie)
        user = User.objects.get(id=user)
        Like.objects.create(user=user, movie=movie)
        return JsonResponse({'success': True})
    else:
        return JsonResponse({'success': False})
```

6.添加评论

从前端获得ajax类型请求（用户点击我要评论），并提取出在请求信息中的movie,user,评论内容和打分，然后在Review表中添加相关表项，最后将添加结果传递给前端，会在原网页显示“添加成功/失败”的浮窗

```
def add_comment(request):
    if request.method == 'POST' and request.is_ajax():
        user_id = request.POST.get('user_id')
        movie_id = request.POST.get('movie_id')
        comment_text = request.POST.get('comment')
        rating = request.POST.get('rating')
        movie = Movie.objects.get(id=movie_id)
        user = User.objects.get(id=user_id)
        Review.objects.create(user=user, movie=movie, comment=comment_text,
rating=rating)
        return JsonResponse({'success': True})
    return JsonResponse({'success': False, 'message': '无效的请求'}, status=400)
```

7.我的主页

从前端获取GET请求（用户点击我的主页），接收GET请求中的用户id，然后在User中找到用户表项，Like中找到用户喜欢的电影，Review中找到用户的评论，最后把这些信息传给我的主页user/myspace.html前端，在前端显示和用户相关的评论和收藏

```
def myspace(request):
    user_id = request.GET.get('user_id')
    user = User.objects.get(id=user_id)
    likes = Like.objects.filter(user=user)
    liked_movies = [like.movie for like in likes]
    reviews = Review.objects.filter(user=user)
    context = {'user': user,'movies': liked_movies,'reviews': reviews}
    return render(request, 'user/myspace.html', context)
```

8.管理员界面：表项展示和查询

共有7个后端函数分别负责数据库中七个表的展示和查询，这里以review为例

当收到前端的GET请求则传递评论列表给前端manager/review.html，前端显示出相关的评论列表

当收到前端的POST请求（管理员点击查询）则提取出查询内容，并在数据库中的review表进行模糊搜索，最后返回搜索出的review列表和查询内容给前端，前端就会显示出和查询相关的列表。

```
def manager_review(request):
    if request.method == "GET":
        review_list = Review.objects.select_related('user', 'movie').all()
        return render(request, "manager/review.html", {"review_list": review_list})
    if request.method == "POST":
        search_query = request.POST.get('search')
        review_list = Review.objects.filter(
            Q(user__username__icontains=search_query)|
            Q(movie__moviename__icontains=search_query) |
            Q(comment__icontains=search_query) |
            Q(rating__icontains=search_query))
        return render(request, 'manager/review.html', {'review_list': review_list,
"search_query": search_query})
```

9.管理员界面：表项的添加

共有7个后端函数分别负责数据库中七个表的添加，这里以review为例

当收到前端的GET请求（管理员点击新建评论按钮）则传递movie和user给前端manager/review_add.html，前端显示出添加界面，并且对于外键会下拉显示出可选的内容（movie和user）

当收到前端的POST请求（管理员填写完添加内容并点击提交）则提取相关内容，并在数据库中的review表中添加相关表项，最终重定位到/manager/review/评论列表

```
def manager_review_add(request):
    if request.method == "GET":
        movies = Movie.objects.all()
        users = User.objects.all()
        return render(request, "manager/review_add.html", {"movies": movies,
"users": users})
    elif request.method == "POST":
        user_id = request.POST.get("user")
        movie_id = request.POST.get("movie")
        comment = request.POST.get("comment")
        rating = request.POST.get("rating")
        user = User.objects.get(id=user_id)
        movie = Movie.objects.get(id=movie_id)
        Review.objects.create(user=user, movie=movie, comment=comment,
rating=rating)
        return redirect("/manager/review/")
```

10.管理员界面：表项的删除

共有7个后端函数分别负责数据库中七个表的删除，这里以review为例

当收到前端的GET请求（管理员点击删除按钮）则获取到删除的评论id，然后在review表中删除相关表项，最终重定位到/manager/review/评论列表

```
def manager_review_delete(request):
    del_id = request.GET.get("del_id")
    Review.objects.filter(id=del_id).delete()
    return redirect("/manager/review/")
```

11.管理员界面：表项的修改

共有5个后端函数分别负责数据库中五个表的删除，这里以review为例

当收到前端的GET请求（管理员点击编辑按钮）则接收来自前端的修改评论的id，从数据库筛选出该评论，并选出所有的movie和用户，把这些信息一起传给前端修改界面manager/review_update.html，修改界面会显示出当前review并让用户修改

当收到前端的POST请求（管理员填写完修改内容并点击提交）则提取相关内容，并在数据库中的review表中更新相关表项，最终重定位到/manager/review/评论列表

```
def manager_review_update(request):
    if request.method == "GET":
        update_id = request.GET.get("update_id")
        review = Review.objects.get(id=update_id)
        movies = Movie.objects.all()
        users = User.objects.all()
        return render(request, "manager/review_update.html", {"review": review,
"movies": movies, "users": users})
    elif request.method == "POST":
        review_id = request.POST.get("review_id")
        user_id = request.POST.get("user")
        movie_id = request.POST.get("movie")
        comment = request.POST.get("comment")
        rating = request.POST.get("rating")
        Review.objects.filter(id=review_id).update(user_id=user_id,
movie_id=movie_id, comment=comment, rating=rating)
        return redirect("/manager/review/")
```

（四）前端构建

主要负责接收来自后端的数据，渲染出网页的界面，将信息传给后端

前端html代码存放在templates文件夹中，用到的css和js组件存放在static文件夹中

```
- templates
  - manager 存放和管理员界面相关的html文件------------------
    - actor.html         演员管理界面
    - actor_add.html     演员添加界面
    - actor_update.html 演员编辑界面
    - ......             每个表均有类似的三个界面，共21个
  - user    存放和用户界面相关的html文件--------------------
    - login.html         登陆界面
    - signup.html        注册界面
    - home.html          用户首页
    - movie.html         电影信息界面
    - myspace.html       用户主页界面
```

前端的实现并不需要向后端一样和数据库进行交互，只需要做到与后端间的信息传递和渲染页面即可。由于前端html文件较多且和数据库关系不大，因此这里只列举其中某一个页面的实现为例，其它前端页面的实现原理相同

```
{% load static %}      #在setting中配置static，从而使得页面可以调用项目中的图片
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>首页</title>
    <link rel="stylesheet" href="{% static 'plugins/bootstrap-3.4.1-
dist/css/bootstrap.css' %} ">  #调用css
    <style>
     # ......页面定义样式
    </style>
</head>

<body>
    <div class="hobbies">
    <div class="stars"></div>
    <h3 class="hobbies_tit">Love is the one thing that transcends time and
space</h3>
    <h3 class="titles">你好，{{ user.username }}! </h3>
    <img class="hobbies_bgc" src="../../static/img/3.webp" alt="">  #调用背景图片
    <div class='ribbon'>
        {% if username is None%}
            <a href="{% url 'myspace' %}?user_id={{ user.id }}"><span>我的主页</span>
</a>
            <a href="{% url 'user/home' %}?user_id={{ user.id }}"><span>首页</span>
</a>
            <a href='/user/login'><span>退出登录</span></a>
        {% endif %}
    </div>
    <div class="container">
    <!-- 搜索栏 -->
    <div class="row">
        <div class="col-md-12">
```

```
            <form action="/user/home/" method="post" class="text-center"
style="margin-bottom: 20px;">
                {% csrf_token %}
                <div class="input-group input-group-lg">
                    <input type="hidden" name="user_id" value="{{ user.id }}">
                    <input type="text" id="search" name="search" class="form-
control" value="{{ search_query }}" placeholder="搜索电影">
                    <span class="input-group-btn">
                        <button class="btn btn-default" type="submit">
                            <span class="glyphicon glyphicon-search" aria-
hidden="true"></span>
                        </button>
                    </span>
                </div>
            </form>
        </div>
    </div>
    <!-- 电影列表 -->
    <div class="row">
    {% for movie in movies %}
        <div class="col-sm-6 col-md-4 col-lg-3" style="margin-bottom: 20px;">
            <div class="movie-item" style="color: white;">
                <img src="{{ movie.photo.url }}" alt="{{ movie.moviename }}"
class="img-responsive">
                <h3><a href="/user/movie/?movie_id={{ movie.id }}&user_id={{ user.id
}}" style="color: white;">{{ movie.moviename }}</a></h3>
                <p>类型: {{ movie.type }}</p>
                <p>上映时间: {{ movie.time }}</p>
            </div>
        </div>
    {% endfor %}
    </div>
</div>
    <script rel="script" src="{% static 'js/jquery-3.6.0.min.js' %}"></script>
    <script rel="script" src="{% static 'plugins/bootstrap-3.4.1-
dist/js/bootstrap.js' %}"></script>
</body>
</html>
```

四. 总结与反思

在本次项目中，我独自一人完成了影评系统的全部搭建，这个过程虽然十分辛苦，利用数周时间完成，但是一步一步做下来自己确实有很大的收获。

在这个项目中我第一次使用国产数据库opengauss，我首先遇到的问题是opengauss数据库的配置与连接。我是在openEuler虚拟机上运行opengauss的，因此首先需要利用终端指令对数据库进行一系列的配置。在配置结束后发现windows中的django连接不上虚拟机的opengauss，搜索了很多资料发现是加密方式的问题，必须采用md5的加密方式才可以进行数据库的正常连接，在解决上面问题的过程中我学到了很多opengauss配置的相关知识。

接下来我花了很多时间对数据库的框架进行设计，我从用户需求的角度出发，首先设计好了该系统的ER图，最终将它转换为数据库中的七个表格，这些表格相互关联，构建出了总体的数据库架构，这个过程让我对ER图和数据库的应用有了更深的理解

在数据库构建完成后，我学会了使用django来编写项目，对于数据库，后端，前端的相互协调与配合有了较深的理解，可以独立搭建出一个网页，并学会了编写前端html代码。

当然，由于一个人的能力有限，我的项目中还存在着许多可以改进的地方，比如当前项目只能在我的主机上访问，没有尝试加入公网使得大家都可以访问我的网页，此外由于时间原因我没有花费很多的精力在前端网页的设计上，后续可以根据用户需求进一步改进前端代码使得网页更加精美。

opengauss下载与配置参考：b站up主“想要不愁”视频

django教程参考：[django教程 (csdn.net)]