# 《数据库系统原理》课程设计

# 系统实现报告

# 题目名称： 电影购票系统

## 学号及姓名：

## 1406XXXX

## 1406XXXX

2017年 1 月 5日

## 一、实现环境

本系统为基于web 的电影购票系统。本系统使用wampserver 集成开发环境(Windows Apache Mysql PHP)开发。系统前端主要采用html 与js完成。后端采用php 完成，数据库为mysql5.7.14。

## 二．系统功能需求分析

### 2.1 需求分析

本系统主要用于影院的在线售票，目的是方便用户购票和管理员管理。

电影购票系统的用户分为两类，一类是普通用户，另一类是管理员用户。普通用户的需求主要是注册和登录、购票、退票、个人信息查询、电影信息查询、电影评价六种；管理员用户的需求主要是登录、添加删除电影、增加排片、查看信息四种。

### 2 需求定义

2.2.1 普通用户需求：

1.用户实现基础的注册、登陆。

2.能够查询电影上映情况，并选择当天上映或即将上映且有余票的电影进行购票。

3.对未过期的影票，可以实现退票功能。

4.可以查询自己的未观看的和已购买的、已退票的电影信息

5.可以查询电影的基本信息(导演、主演、宣传片等)。

6.对网站上公布的电影，用户可以进行评分和评论，并随时查看排名前 10 的电影。另外用户可以对某部电影进行评论。

2.2.2 管理员用户需求：

1. 能登陆管理员账号。
2. 能在系统中新增、删除一部电影。
3. 能选择电影进行安排(放映时间、放映影厅等)。
4. 管理员能随时查看影票的销售情况、用户注册数等。

### 2.3 功能说明

经过分析，我们认为本系统需要完成的主要功能有如下部分：

2.3.1 注册及登录功能：

普通用户可以通过填写个人信息注册账号，并使用这个账号完成购票、退票等一系列操作，或者直接用已有的账号进行登录后再进行操作。

管理员用户只能登录，无法注册。如需新增管理员账户则需在数据库中直接添加管理员信息，这需要系统管理员的权限。

2.3.2 电影信息查询及更新功能：

普通用户仅能查询已有的电影的信息(导演、主演、简介、宣传片等)，管理员用户可以新增、删除电影。

2.3.3 购票、退票功能：

普通用户在登录后可以购买当天或未来的有余票的场次的电影票；如果电影还未放映，则可以退票。

管理员用户无法购票退票。

2.3.4 评论、打分、收藏功能：

普通用户可以对电影进行打分、评论、收藏。

管理员用户不可以对电影进行打分、评论、收藏。

游客(未登陆)用户可以对电影进行打分、收藏，但是不可以对电影进行评论。

## 三．系统功能结构设计

### 3.1 系统的功能分析

3.1.1 电影信息管理

电影信息：电影编号、电影名称、电影分类、简介、详介、导演、主演、收藏人数、打分人数、票价。

管理员可以添加、删除电影。

用户只可以查询电影信息，不可进行修改。

3.1.2 用户信息管理

1.普通用户信息：用户名、密码、昵称、联系电话。

管理员用户不可以对普通用户的个人信息做任何操作。

普通用户可以注册自己的账号，并保存到后台数据库中；普通用户还可以查阅自己的个人信息。

2.管理员信息：管理员编号，用户名，密码。

管理员用户不可以查询或修改管理员信息。

普通用用户不可以查询或修改管理员信息。

3.1.3售票情况管理

排片信息：电影名称，影厅编号、放映日期、放映时间、已销售的电影数量。

管理员对电影可以进行排片，排片情况包括电影的选择，所在影厅的选择，日期时间的选择。管理员可以看到目前已经销售的电影票数量。

用户可以看到的是电影排片的具体信息，包括电影名称，所在影厅，排片的日期和时间，并作出相应的选择。

3.1.5 评分评论管理

评分评论信息：电影的平均得分，用户对电影的评论。

管理员可以看到每一个电影所拥有的平均分以及用户所作出的评论，并不能作出修改。

用户可以看到每一个电影的平均分以及自己和他人作出的评论，并且可以自己作出对电影的评分以及作出相关的评论。

### 3.2 系统的结构设计

3.2.1 概念结构设计的方法

本系统采用自底向上的概念结构设计方法。即首先定义各局部应用的概念结构，然后将它们集成起来得到全局概念模式。

3.2.2 概念结构设计的步骤

概念结构设计可分为两个步骤：

1.抽象数据并设计局部视图。就是对需求分析阶段收集到的数据按照ER模型的要求进行分类、组织，形成实体的属性，标识实体的键，确定实体之间的联系类型，设计分ER图。

2.集成局部视图，得到全局的概念结构。就是要将所有的分ER图综合成一个系统的总ER图。

### 3.3 数据独立性

所有数据项的添加删除修改都是由网页端向用户提供的接口来实现对底层数据库的访问，不需要修改相应的应用程序。

### 3.4 数据安全性

数据库设计实现之时在电影，排片，用户，管理员，评论等数据实体及联系上对用户做了良好的操作分隔及相应操作的接口提供。可以有效的防止非法访问数据。

### 3.5数据一致性

同一事物的数据，不管出现在何时何处都是一致的。这是由数据库的触发器来实现所有相关数据发生增删改之后保持一致性。部分触发器功能概述如下：

1. 增加一条评论：评论表中增加一条评论；在电影表中相同电影名字那一项的评论人数自增1。
2. 退一张电影票：退票表中增加一行退票信息；在排片表中相同场次的电影的已售座位那一项自减一。

## 数据库基本表的定义

### 4.1 administrator表



### 4.2 arrangement表



### 4.3comments表



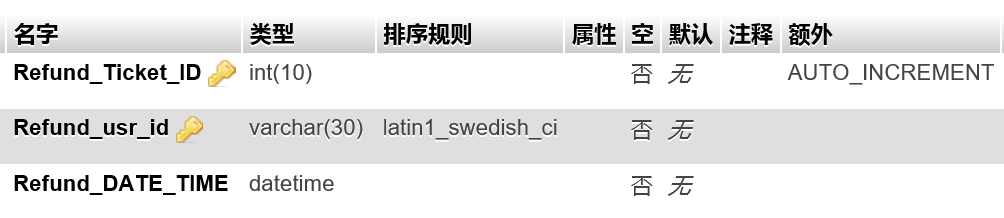
### 4.4 movie表



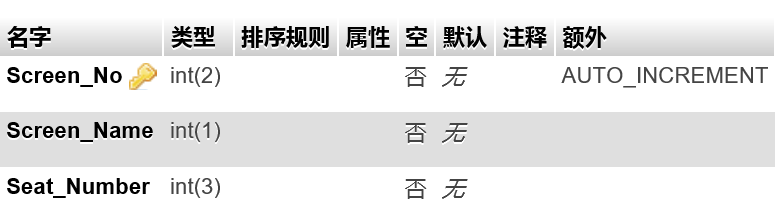
### 4.5 purchase表



### 4.6 refund表



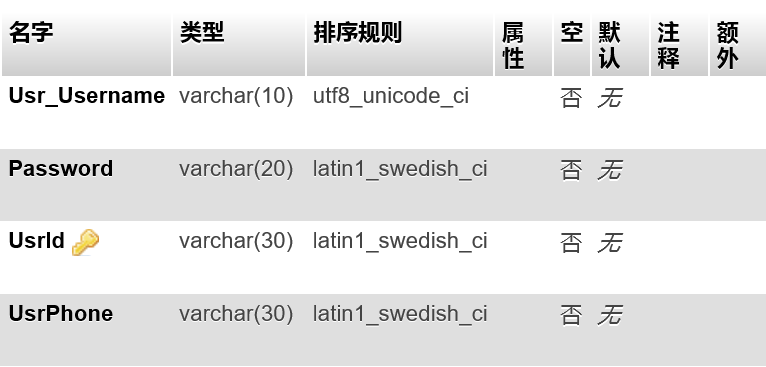
### 4.7 screen表



### 4.8 ticket表



### 4.9 usr表



## 五、触发器的定义与实现

定义：同一事物的数据，不管出现在何时何处都是一致的。这是由数据库的触发器来实现所有相关数据发生增删改之后保持一致性。

触发器如下：

### 5.1增加一条评论

评论表中增加一条评论；在电影表中相同电影名字那一项的评论人数自增1。

代码如下：

DELIMITER $

CREATE TRIGGER commender after INSERT ON commend FOR EACH ROW

begin

DECLARE c int;

set c = (select Comment\_of\_people from movie where Name = new.Movie\_Name);

update movie set Comment\_of\_people = c + 1 WHERE

Name = new.Movie\_Name;

End

$

### 5.2 退一张电影票：

代码如下：

DELIMITER $

CREATE TRIGGER refunder after INSERT ON refund FOR EACH ROW

BEGIN

DECLARE c int;

DECLARE a int;

set a = (select Seat from ticket where Ticket\_ID = new.Refund\_Ticket\_ID);

set c = (select Selled\_Num from arrangement where

Arrange\_Name = (select Movie\_Name from ticket where Ticket\_ID = new.Refund\_Ticket\_ID)

and Arrange\_Screen = (select Screen from ticket where Ticket\_ID = new.Refund\_Ticket\_ID)

and year(date\_format(Arrange\_Date,'%Y%m%d%H%i%s')) = year(date\_format((select Movie\_Time from ticket where Ticket\_ID = new.Refund\_Ticket\_ID),'%Y%m%d%H%i%s'))

and dayofyear(date\_format(Arrange\_Date,'%Y%m%d%H%i%s')) = dayofyear(date\_format((select Movie\_Time from ticket where Ticket\_ID = new.Refund\_Ticket\_ID),'%Y%m%d%H%i%s'))

and hour(date\_format(Arrange\_Time,'%Y%m%d%H%i%s')) = hour(date\_format((select Movie\_Time from ticket where Ticket\_ID = new.Refund\_Ticket\_ID),'%Y%m%d%H%i%s'))

and minute(date\_format(Arrange\_Time,'%Y%m%d%H%i%s')) = minute(date\_format((select Movie\_Time from ticket where Ticket\_ID = new.Refund\_Ticket\_ID),'%Y%m%d%H%i%s') )

and second(date\_format(Arrange\_Time,'%Y%m%d%H%i%s')) = second(date\_format((select Movie\_Time from ticket where Ticket\_ID = new.Refund\_Ticket\_ID),'%Y%m%d%H%i%s')));

if a = c then

UPDATE

arrangement SET Selled\_Num = c - 1 WHERE

Arrange\_Name = (select Movie\_Name from ticket where Ticket\_ID = new.Refund\_Ticket\_ID)

and Arrange\_Screen = (select Screen from ticket where Ticket\_ID = new.Refund\_Ticket\_ID)

and year(date\_format(Arrange\_Date,'%Y%m%d%H%i%s')) = year(date\_format((select Movie\_Time from ticket where Ticket\_ID = new.Refund\_Ticket\_ID),'%Y%m%d%H%i%s'))

and dayofyear(date\_format(Arrange\_Date,'%Y%m%d%H%i%s')) = dayofyear(date\_format((select Movie\_Time from ticket where Ticket\_ID = new.Refund\_Ticket\_ID),'%Y%m%d%H%i%s'))

and hour(date\_format(Arrange\_Time,'%Y%m%d%H%i%s')) = hour(date\_format((select Movie\_Time from ticket where Ticket\_ID = new.Refund\_Ticket\_ID),'%Y%m%d%H%i%s'))

and minute(date\_format(Arrange\_Time,'%Y%m%d%H%i%s')) = minute(date\_format((select Movie\_Time from ticket where Ticket\_ID = new.Refund\_Ticket\_ID),'%Y%m%d%H%i%s'))

and second(date\_format(Arrange\_Time,'%Y%m%d%H%i%s')) = second(date\_format((select Movie\_Time from ticket where Ticket\_ID = new.Refund\_Ticket\_ID),'%Y%m%d%H%i%s'));

end if;

END

$

## 六、存储过程的定义与实现

### 6.1 保存用户信息

create or replace procedure insertUsr

(Usr\_Username, Password, UsrId, UsrPhone)

as declare

getUsrName varchar;

begin

select getUsrName from usr where UsrId = Usr\_Username;

if getUsrName is not NULL then

rollback;

return;

end if;

insert into usr (Usr\_Username, Password, UsrId, UsrPhone)

values(Usr\_Username, Password, UsrId, UsrPhone);

commit;

end;

### 6.2 添加一条评论：

create or replace procedure addcomit

（Comment\_ID, Movie\_Name, Comments, Usr\_username）

begin

Insert into comments

(Comment\_ID, Movie\_Name, Comments, Usr\_username)

values

(Comment\_ID, Movie\_Name, Comments, Usr\_username);

commit;

end;

### 6.3 修改当前评分:

create or replace procedure updtscore(r, u, movieid)

begin

update movie set Score\_of\_people = r ,score = u where Movie\_No = movieid;

commit;

end

### 6.4 退票：

create or replace procedure refundtk

(temp, Refund\_Ticket\_ID,Refund\_usr\_id, Refund\_DATE\_TIME)

as declare

tkid int;

begin

select tkid from purchase where Purchase\_Ticket\_ID = tkid;

if tkid is NULL then

rollback;

return;

endif;

delete from purchase where Purchase\_Ticket\_ID = temp;

insert into

refund (Refund\_Ticket\_ID,Refund\_usr\_id, Refund\_DATE\_TIME)

values(Refund\_Ticket\_ID,Refund\_usr\_id, Refund\_DATE\_TIME)；

commit;

end;

### 6.5安排电影：

create or replace procedure refundtk

(pArrange\_Name, pArrange\_Date,pArrange\_Time, pArrange\_Screen)

as declare

Arrange\_Screen int;

begin

select

Arrange\_Screen

from arrangement where Arrange\_Date = pArrange\_Date and Arrange\_Time = pArrange\_Time and Arrange\_Screen= pArrange\_Screen;

if(Arrange\_Screen is NULL)

rollback;

return;

endif;

insert into arrangement

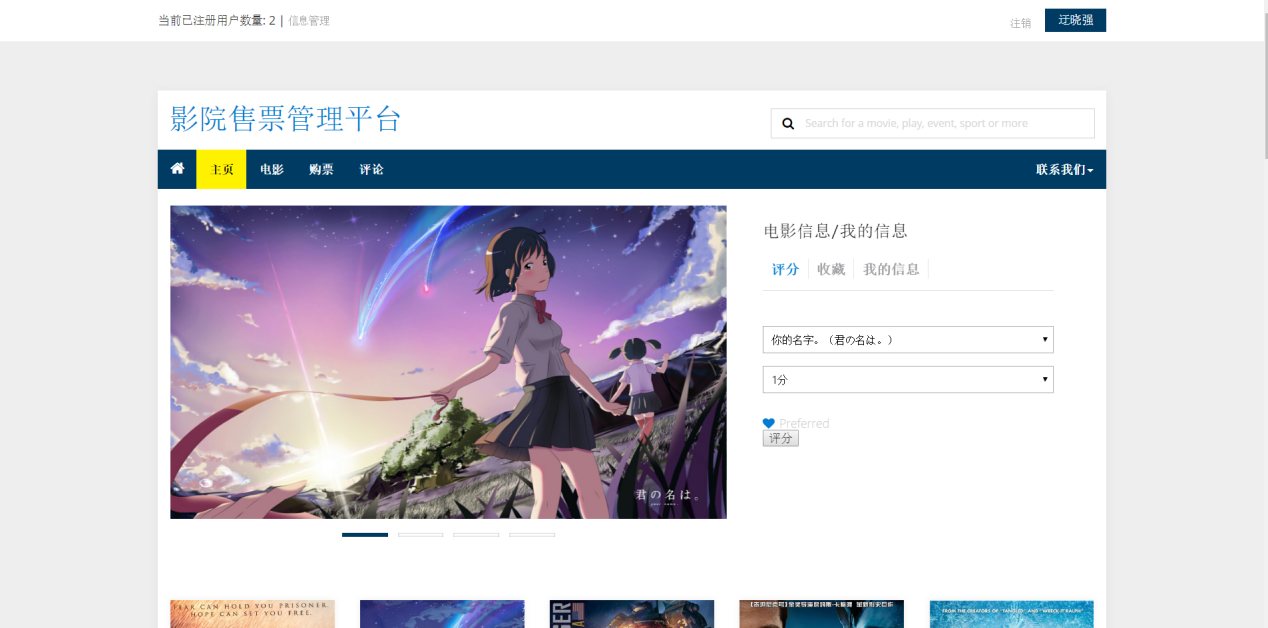
(Arrange\_Name,Arrange\_Screen,Arrange\_Time,Arrange\_Date) values(pArrange\_Name,pArrange\_Screen,pArrange\_Time,pArrange\_Date);

commit;

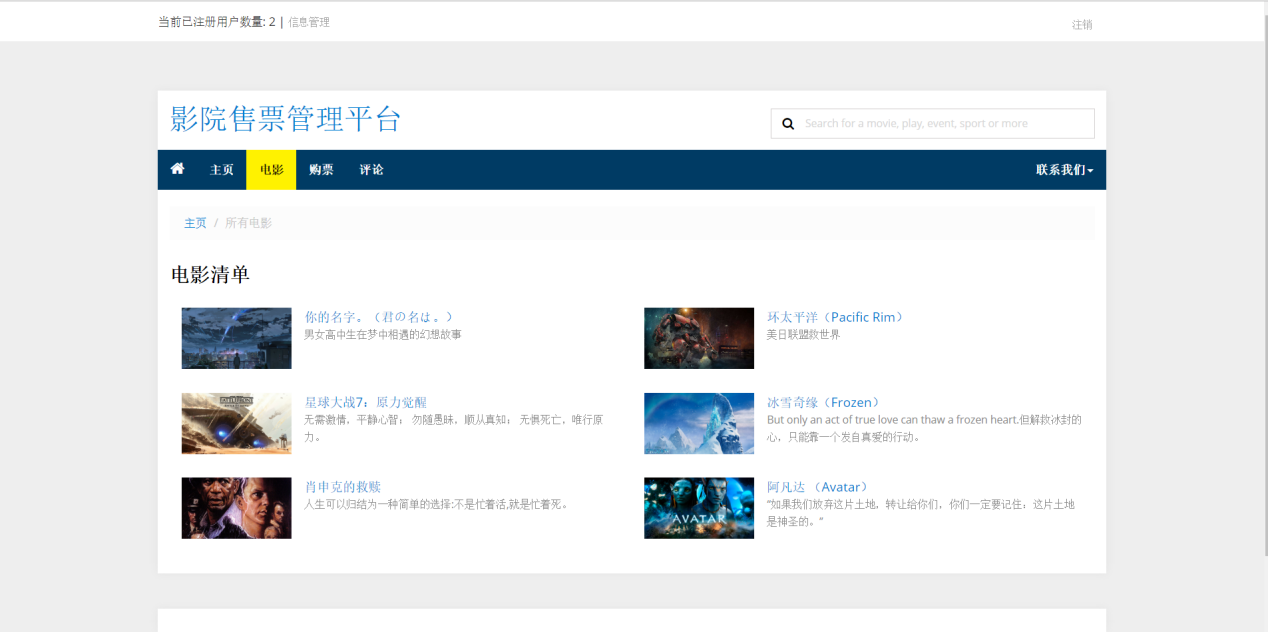
end;

## 系统实现结果

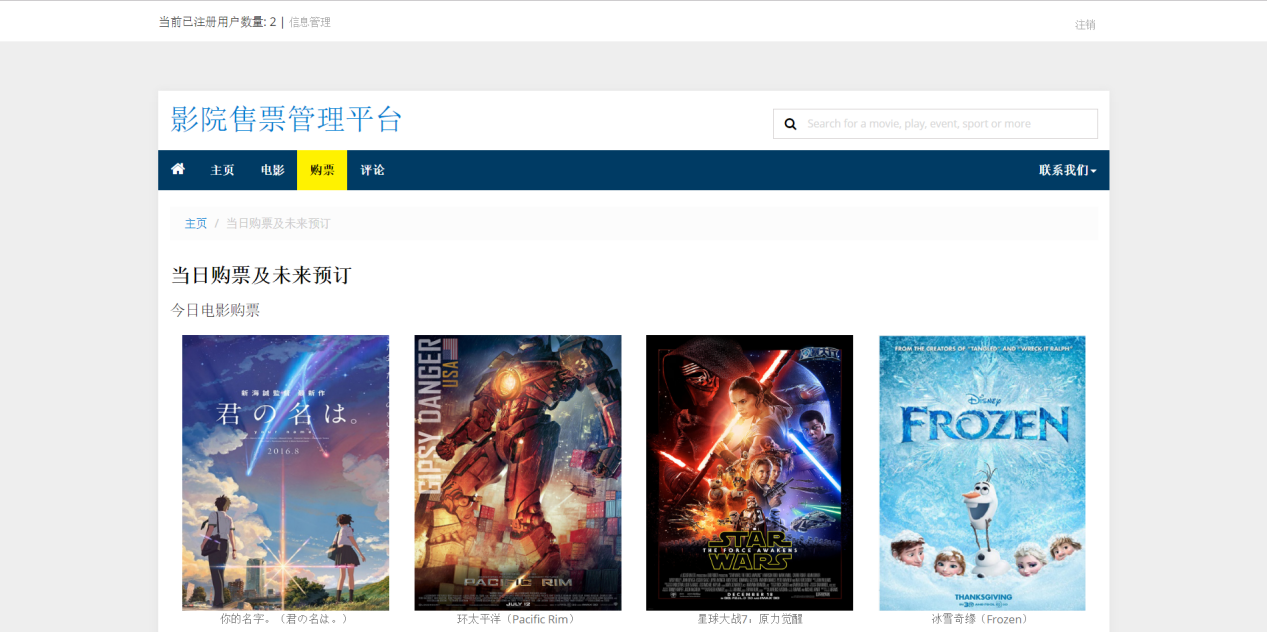
### 7.1 主页



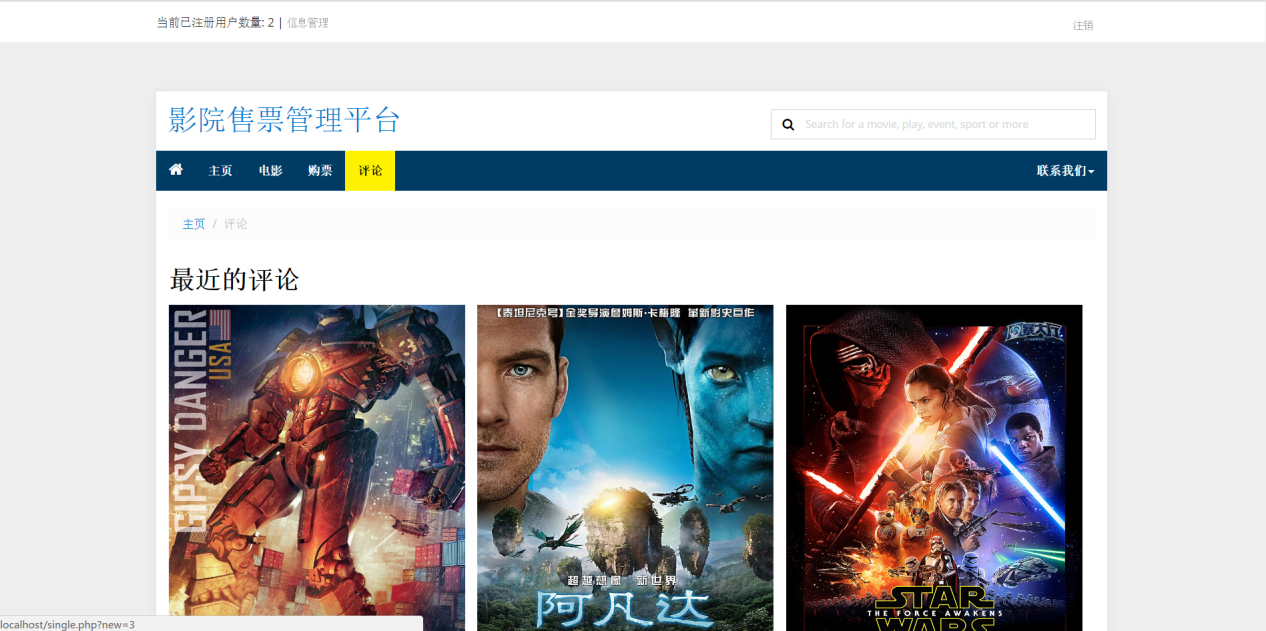
7.2 电影页面：



### 7.3 购票页面：

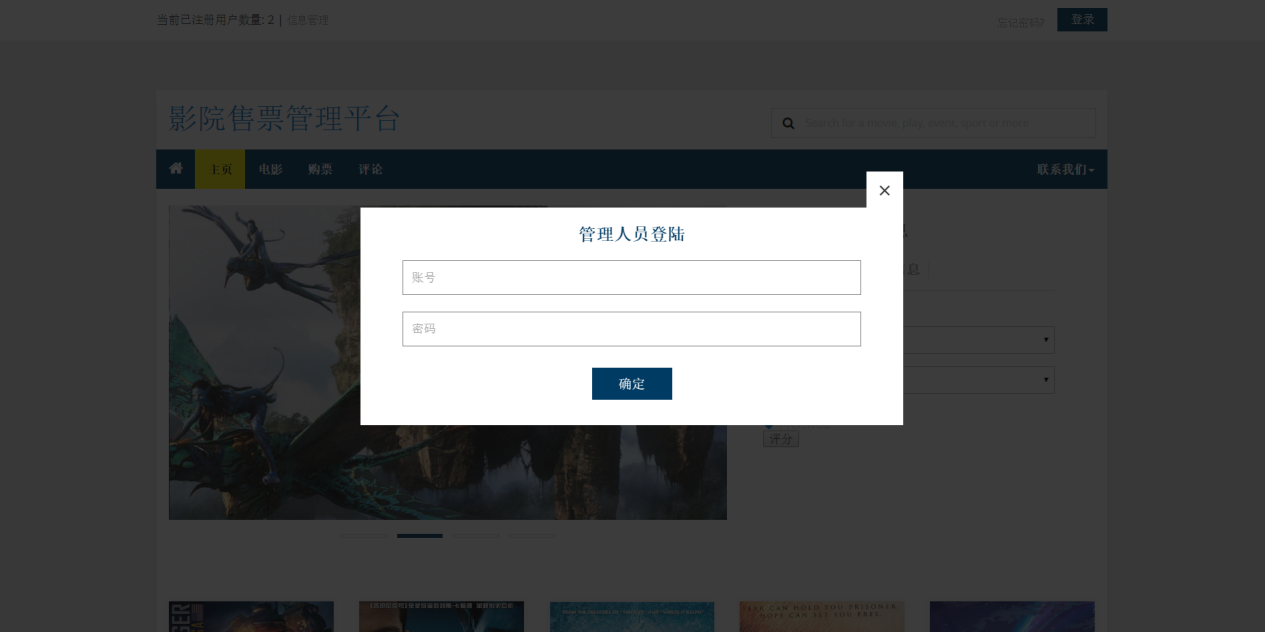


### 7.4 评论页面：

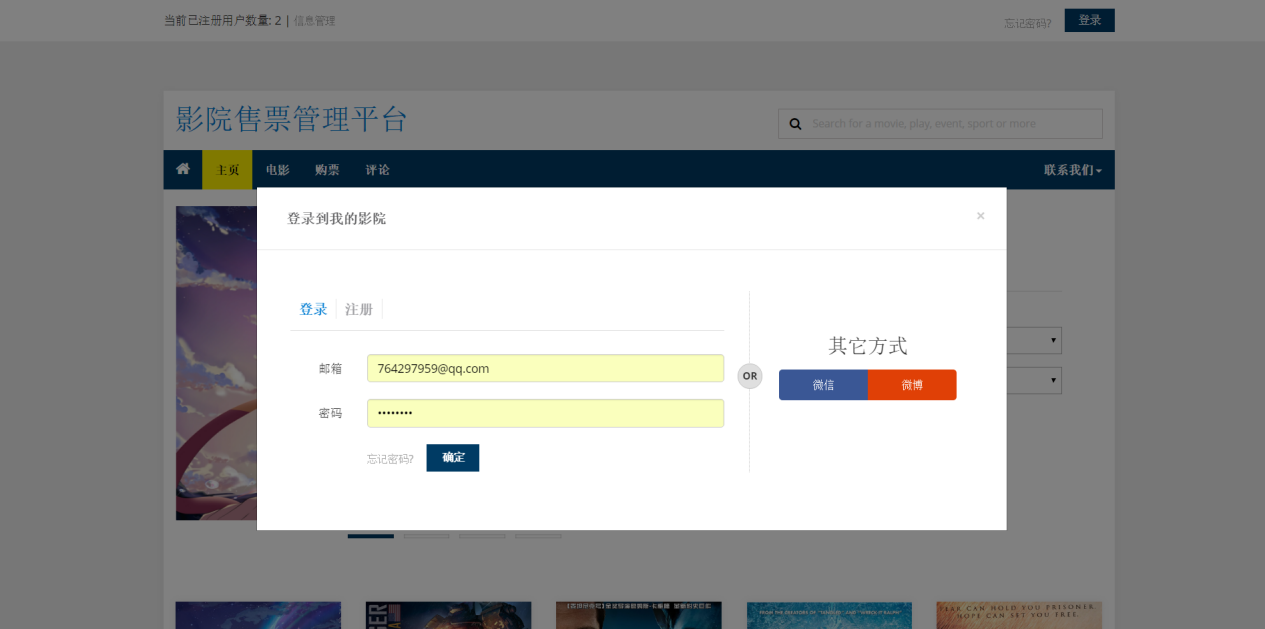


### 7.5 注册、登录页面：

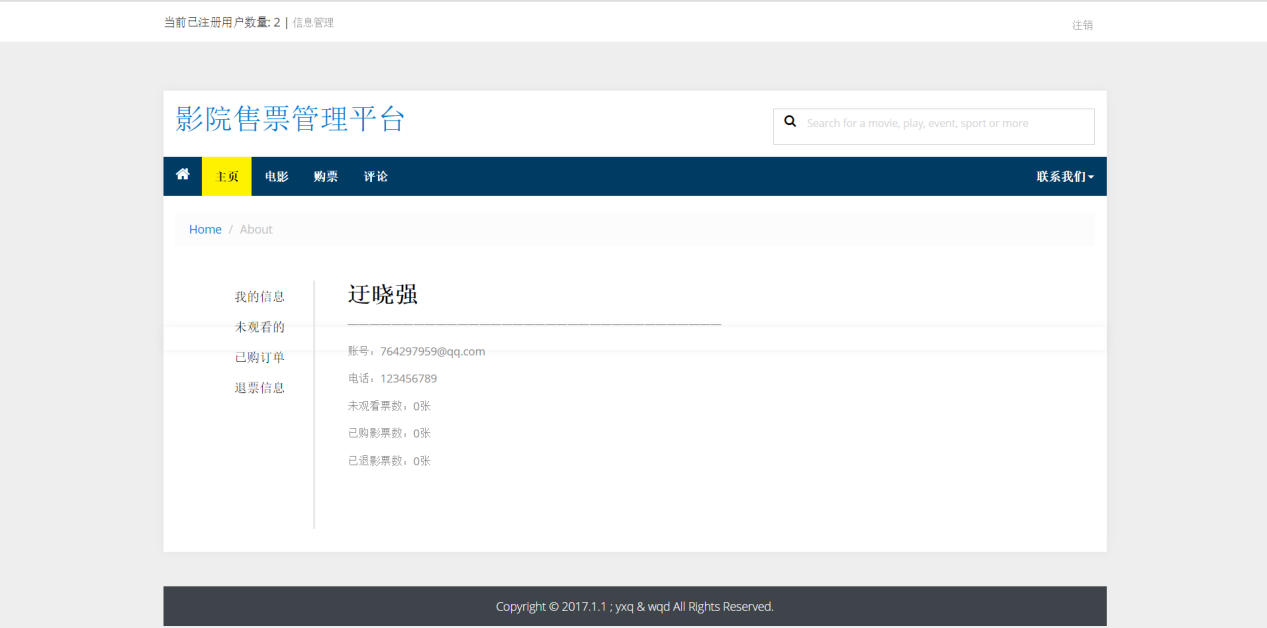
7.5.1 管理员用户登录页面：



7.5.2 普通用户注册登录页面：

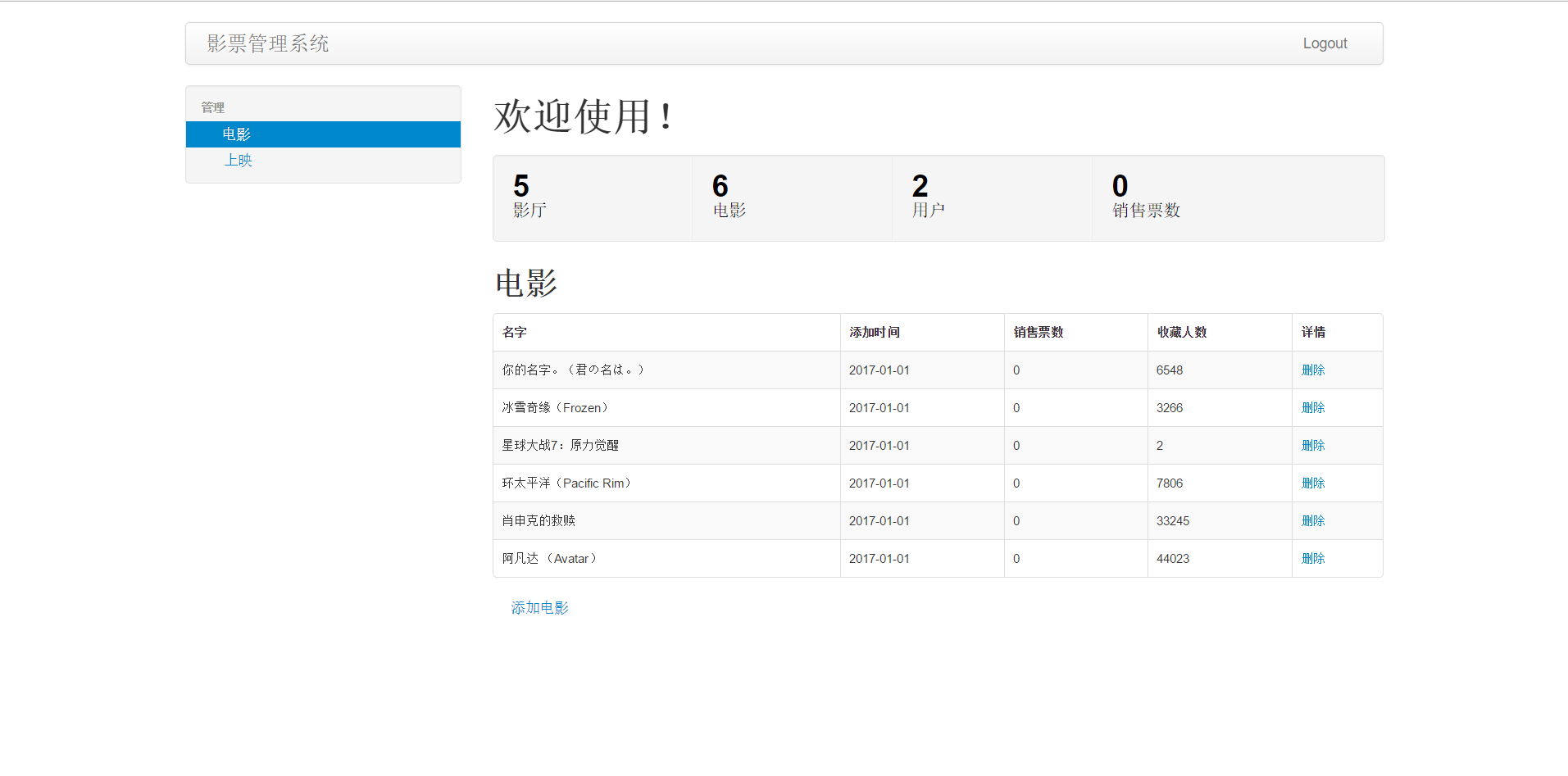


### 7.6 用户个人信息页面：

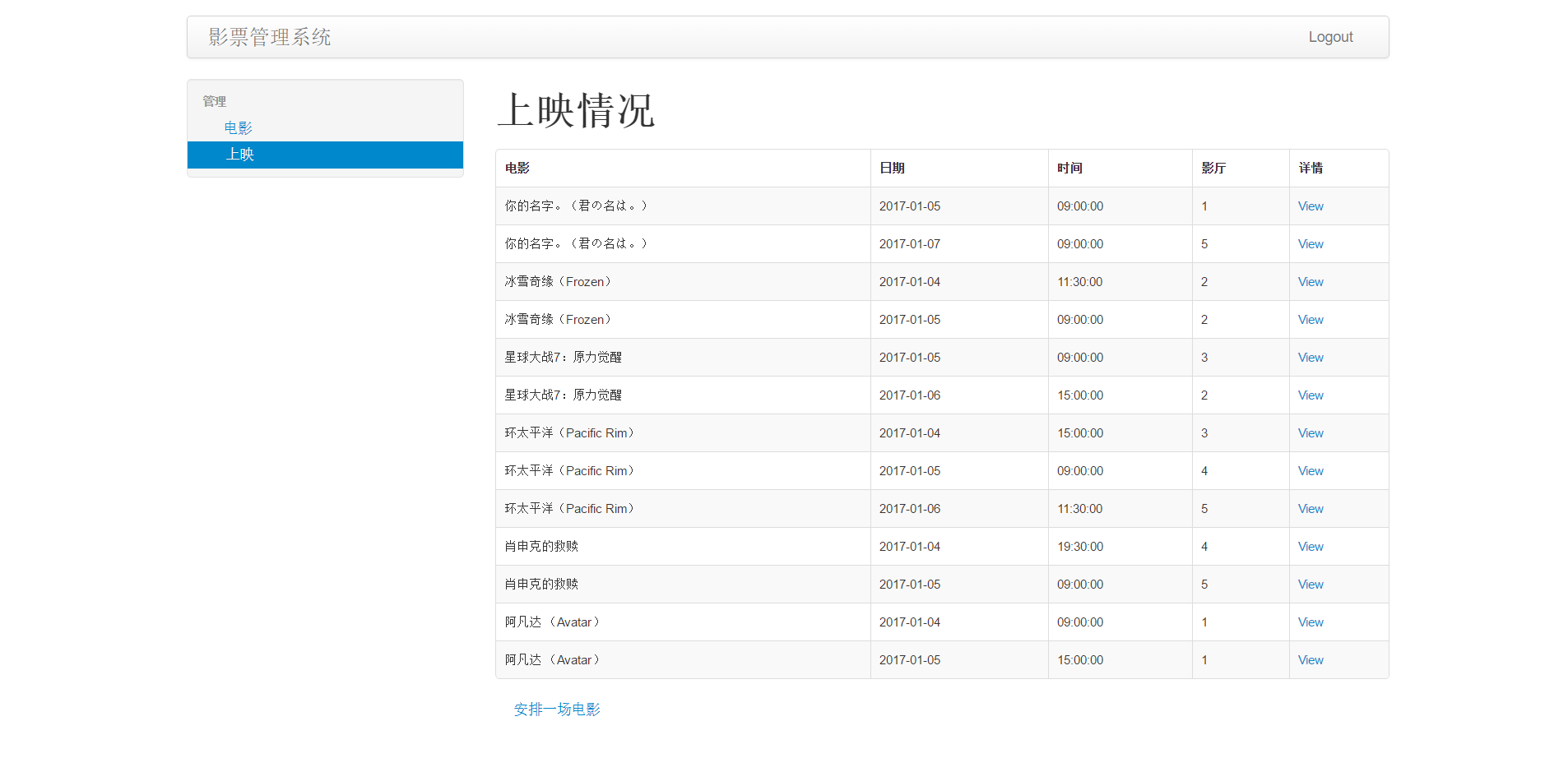


### 7.7 管理员页面：

7.1.1 电影管理界面：



7.1.2 排片管理界面：



其余实现的功能由于较难以截图形式给出，故在展示时给出系统的详细功能介绍

## 八、总结

这次数据库课程设计我们用了十天左右的时间完成，其中选题、数据库设计、E\_R图设计等总体问题用了一天的时间，编码用了五天左右，测试和修改用了两天，总结和撰写报告、PPT用了一天的时间。

在分工方面，我们一个人主要负责前端网站和后台数据库的开发，另一个人主要负责利用php语言将前端与后端连接起来。在这十天左右的时间里，我们主要学习并应用了HTML、php、mysql等知识，完成了一个相对功能比较全面的电影购票系统。

在学习相关知识的过程中我们也遇到了很多困难，例如如何保持用户的登录状态、页面之间的跳转的传参问题、数据库触发器的编写等，我们通过上网查阅相关资料将这些问题都一一解决了。最后的调试和修改阶段我们也找到了一些系统问题，并将其解决。不过这里还存有一个遗留的bug，就是在个人信息页面的已购订单和退票信息会出现偶尔显示不出来的问题，目前为止这个问题还没有解决。

经过这次数据库课程设计，我们学到了很多新的东西，虽然运用的还不算完美，但总归开了一个好头。而且通过对数据库的各种操作，我们也对建表、查询、更新、触发器的建立等数据库的相关知识有了更加深刻地认识。总之这次开发经历对我们来说很有意义。