

**数据库系统原理实践报告**

|  |  |  |
| --- | --- | --- |
| 项目名称： | 机票预订系统 | |
| 姓 名： | | 刘美 |
| 专 业： | 计算机科学与技术 | |
| 班 级： | 1806 | |
| 学 号： | U201814788 | |
| 指导教师： | 胡侃 | |

|  |  |
| --- | --- |
| 分数 |  |
| 教师签名 |  |

2021 年 07 月 07 日

**教师评分页**

|  |  |
| --- | --- |
| 子目标 | 子目标评分 |
| 1 |  |
| 2 |  |
| 3 |  |
| 4 |  |
| 5 |  |
| 6 |  |
|  |  |

目 录

[1 课程任务概述 1](#_Toc76583324)

[2 SQL练习部分 2](#_Toc76583325)

[2.1 任务要求 2](#_Toc76583326)

[2.2 完成过程 2](#_Toc76583327)

[2.3任务总结 12](#_Toc76583328)

[3实验任务二 软件功能学习 13](#_Toc76583329)

[3.1 任务要求 13](#_Toc76583330)

[3.2任务总结 15](#_Toc76583331)

[4 综合实践任务 17](#_Toc76583332)

[4.1系统设计目标 17](#_Toc76583333)

[4.2 需求分析 17](#_Toc76583334)

[4.3 总体设计 20](#_Toc76583335)

[4.4 数据库设计 22](#_Toc76583336)

[4.5 详细设计与实现 25](#_Toc76583337)

[4.6 系统测试 39](#_Toc76583338)

[4.7 系统设计与实现总结 46](#_Toc76583339)

[4 课程总结 48](#_Toc76583340)

# 1 课程任务概述

本实验主要分为三个部分，分别为软件功能学习，SQL练习和综合实践任务。

软件部分，学生需要练习数据库的软件功能备份和脱机备份两种备份方式以及在数据库上增加用户并配置权限的操作。

SQL练习，主要考察学生对SQL基本语言的熟练程度，主要包括——创建数据库、表和视图，并在相关数据库和表上进行增、删、改、查以及了解协同查询性能分析功能和函数及存储过程等一系列操作。实验通过在Educoder平台上编写好代码后直接进行评测。

综合实践部分，学生需要完成一个小型数据库应用系统的设计和实现，主要设计流程包括：需求分析、总体设计、数据库设计、详细设计与实现、实验测试等。实验要求能够体现数据库语言和编程语言的结合。

通过本实践课程，学生应能够更加熟练掌握数据库的有关知识，并能体会到数据库在具体应用中的作用，能够尽可能的实现一个完善的数据库和应用系统。

# SQL练习部分

## 2.1 任务要求

创建自己的地区抗议数据库，设计建立人员表、地点表、行程表、诊断表、密切接触表、隔离表和隔离地点表。然后在这些表的基础上完成一系列数据查询、数据更新、函数及存储过程和事务等相关任务，最终实现巩固数据库有关概念、较为熟练应用SQL语言的目标。

## 2.2 完成过程

### 2.2.1 创建数据库

使用Create语句进行创建。

 if exists (select \* from sys.databases where name = 'covid19mon')

    drop database covid19mon;

 create database covid19mon  COLLATE Chinese\_PRC\_CI\_AS;

使用if语句能够保证如果第一次创建数据库失败，下次执行新的创建语句能够被有效执行。

### 2.2.2建表

首先分析各个表列之间的参照约束关系，确定各个表的创建顺序。主要是：人员表和地点表必须作为前两个定义的表，因为其主码被后面表所参照；隔离地点表必须建立在隔离表之前，因为隔离表中列隔离地点编号参照了隔离地点表对应列。

然后使用Create table语句进行建表。

if exists (select \* from sys.databases where name = 'covid19mon')

    drop database covid19mon;

 create database covid19mon  COLLATE Chinese\_PRC\_CI\_AS;

 go

 use covid19mon;

 go

- 表1 人员表(person)

 create table person(

     id int,

     fullname char(20) not null,

     telephone char(11) not null,

     constraint pk\_person primary key(id)

 );

-- 表2 地点表(location)

 create table location(

     id int,

     location\_name char(20) not null,

     constraint pk\_location primary key(id)

 );

-- 表3 行程表（itinerary）

 create table itinerary(

     id int,

     p\_id int,

     loc\_id int,

     constraint fk\_itinerary\_pid foreign key(p\_id) references person(id),

     constraint fk\_itinerary\_lid foreign key(loc\_id) references location(id),

     s\_time datetime,

     e\_time datetime,

     constraint pk\_itinerary primary key(id)

 );

-- 表4 诊断表（diagnose\_record）

 create table diagnose\_record(

     id int,

     p\_id int,

     constraint fk\_diagnose\_pid foreign key (p\_id) references person(id),

     diagnose\_date datetime,

     result int,

     constraint pk\_diagnose\_record primary key(id)

 );

-- 表5 密切接触者表（close\_contact）

create table close\_contact(

    id int,

    p\_id int,

    contact\_date datetime,

    loc\_id int,

    case\_p\_id int,

    constraint fk\_contact\_pid foreign key (p\_id) references person(id),

    constraint fk\_contact\_lid foreign key (loc\_id) references location(id),

    constraint fk\_contact\_caseid foreign key(case\_p\_id) references person(id),

    constraint pk\_close\_contact primary key(id)

);

-- 表6 隔离地点表（isolation\_location）

create table isolation\_location(

    id int,

    location\_name char(20),

    capacity int,

    constraint pk\_isolation\_loc primary key(id)

);

-- 表7 隔离表（isolation\_record）

create table isolation\_record(

    id int,

    p\_id int,

    s\_date datetime,

    e\_date datetime,

    isol\_loc\_id int,

    constraint fk\_isolation\_pid foreign key(p\_id) references person(id),

    constraint fk\_isolation\_lid foreign key(isol\_loc\_id) references isolation\_location(id),

    state int,

    constraint pk\_isolation primary key(id)

);

### 2.2.3数据更新

1. 插入数据

insert into person values(1,N'张小敏',N'13907110001');

insert into person values(2,N'李大锤',N'18907110002');

insert into person values(3,N'孙二娘',N'13307100003');

1. 删除数据

delete from person where id=2;

1. 修改数据

update person set telephone =N'13607176668' where id=1;

### 2.2.4 数据查询

1. 查询累计人流量大于30的地点名称和累计人流量，请用visitors作累积人流量的标题名称。查询结果按照人流量从高到低排序,人流量相同时，依地点名称的字典顺序排序。

将行程表和地点表连接，因为最终查询结果需要根据地点信息统计，所以按照地点名进行分组，并按照题意指定count的结果大于30

select location.location\_name,count(p\_id) as visitors

from location,itinerary

where location.id=itinerary.loc\_id

group by location.location\_name

having count(p\_id)>30

order by count(p\_id) desc,location.location\_name;

1. 查询每个隔离地及该地正在进行隔离的人数，以number为隔离人数的标题.查询结果依隔离人数由多到少排序，隔离人数相同时，再依隔离点名称字典顺序排序。

select location\_name,count(p\_id) as number

from isolation\_record,isolation\_location

where isolation\_location.id=isolation\_record.isol\_loc\_id and state=1

group by location\_name

order by count(p\_id) desc,location\_name;

1. 查询人员编号大于30的接续行程，输出信息包括：人员编号,姓名,重合时间,起始地点id,起始地点,结束地点id,结束地点。查询结果依人员编号排序。

接续行程指的是前一趟行程的结束时间恰好为下一趟行程的开始时间，由于设计到两趟行程之间的比较，所以不难想到需要两张行程表以及两张地点表和人员表进行连接。

select p.id, fullname, telephone, i1.e\_time as reclosing\_time ,i1.loc\_id as loc1,l1.location\_name as address1,i2.loc\_id as loc2,l2.location\_name as address2

from person p, location l1,location l2,itinerary i1,itinerary i2

where p.id=i1.p\_id and l1.id=i1.loc\_id and l2.id=i2.loc\_id and

    i1.p\_id=i2.p\_id and i1.e\_time=i2.s\_time and i1.p\_id >30

order by p.id;

1. 查询充珉瑶和贾涵山的行程情况。查询结果包括：姓名、电话、到过什么地方（地名），何时到达，何时离开，查询结果依人员编号降序排序，同一人员行程依行程开始时间顺序排列。

根据题目，我们需要注意要查询的两个人可能并不存在行程记录，所以from实现连接时，先进行行程表和地点表的连接，然后在和人员表进行右外连接，这样如果某人不存在行程则人员信息被保存，而行程信息均为NULL。

select fullname,telephone,location\_name,(select convert(char(19),s\_time,20)) as s\_time, (select convert(char(19),e\_time,20)) as e\_time

from itinerary

    join location on itinerary.loc\_id=location.id

    right join person on p\_id=person.id

where fullname=N'充珉瑶' or fullname=N'贾涵山'

order by person.id desc;

1. 查询地名中带有‘店’字的地点编号和名称。查询结果按地点编号排序。

select id,location\_name

from location

where location\_name like N'%店%'

order by id asc;

1. 新发现一位确诊者，已知他在2021.2.2日20:05:40到21:25:40之间在“活动中心”逗留，凡在此间在同一地点逗留过的，视为接触者，请查询接触者的姓名和电话。查询结果按姓名排序。

只要人员行程的的时间段和接触者的时间段有交集并且为同一个地点则可被视为接触者。

use covid19mon;

select fullname,telephone

from person ,itinerary

where person.id=itinerary.p\_id and

itinerary.loc\_id=(select id from location where location.location\_name='活动中心') and

itinerary.s\_time<= '2021-02-02 21:25:40' and

itinerary.e\_time >='2021-02-02 20:05:40'

order by fullname asc;

1. 查询仍在使用的隔离点名称。注意，隔离记录里如果只有隔离结束或确诊转院的记录，表明该隔离点已暂时停用，只要还有一个人在此处隔离，表明该隔离点仍在使用。查询结果按隔离点编号排序。

select location\_name

from isolation\_location

Where id in(

select distinct isol\_loc\_id

from isolation\_record

where isolation\_record.state=1);

1. 查询前30位有出行记录的人员姓名和电话。查询结果按照人员编号排序。

select top 30 fullname,telephone

from person

where exists(

select \*

from itinerary

where person.id=itinerary.p\_id)

order by id asc;

1. 查询人员表中没有去过地点“Today便利店”的人数。请给统计出的人数命名为number。

select count(id) as number

from person

where not exists(

select \* from itinerary,location

where itinerary.p\_id=person.id and location.id=itinerary.loc\_id and location.location\_name=N'Today便利店');

1. 查询人员表去过所有地点的人员姓名。查询结果依人员姓名的字典顺序排序。

select fullname

from person

where not exists(

    select \* from location

    where not exists(

        select \* from itinerary

        where p\_id=person.id and loc\_id=location.id))

order by fullname asc;

1. 建立隔离点现状的视图，视图命名为isolation\_location\_status，内容包括：地点编号，隔离地点名，房间容量，已占用量。请保持原列名不变，已占用量由统计函数计算得出，该计算列命名为occupied。

create view isolation\_location\_status as

select id,location\_name,capacity,(select count(\*) from isolation\_record where isol\_loc\_id =i.id and state = 1) as occupied

from isolation\_location i;

1. 从视图isolation\_location\_status中查询各隔离点的剩余空房间的数目。需要列出的数据项为：隔离点名称，剩余房间数。其中剩余房间数为计算得出，请给该列命名为available\_rooms。查询结果依隔离点编号排序。

select location\_name,capacity-occupied as available\_rooms

from isolation\_location\_status

where capacity-occupied>0

order by id;

1. 查询靳宛儿接触者的姓名和电话。与靳宛儿在同一地点逗留时间有交集的均为其接触者。查询结果按照人员姓名排序。

select fullname,telephone

from person

where fullname <>N'靳宛儿' and id in(

    select  p\_id

    from itinerary i1

    where  exists

        (select \* from itinerary i2

         where i1.loc\_id=i2.loc\_id and i1.s\_time <= i2.e\_time and i1.e\_time >=i2.s\_time and

                i2.p\_id in (select id from person where fullname =N'靳宛儿' )

        ))

order by fullname;

1. 依据密切接触表的内容查询每个地点的密切接触者的数量，列出内容包括：地点名称，密接者人数。人数由统计获得，列名命名为close\_contact\_number.查询结果依密接者人数降序排列。密接者人数相同时，依地点名称排序。

select location\_name,count(close\_contact.id) as close\_contact\_number

from location,close\_contact

where location.id=close\_contact.loc\_id

group by location\_name

order by count(close\_contact.id) desc,location\_name;

1. 查询感染人数最多的人员编号，姓名，和被其感染的人数。感染人数由统计所得，命名为infected\_number。

select top 1 person.id as case\_p\_id,person.fullname,count(close\_contact.p\_id) as infected\_number

from person,close\_contact

where person.id=close\_contact.case\_p\_id

group by person.id,person.fullname

order by count(close\_contact.p\_id) desc;

1. 查询2021-02-02 10:00:00到14:00:00期间，行程记录最频繁的三个人的姓名及行程记录条数。记录条数命名为record\_number。记录数并列的，按姓名顺序排列。

select top 3 fullname,count(itinerary.id) as record\_number

from person,itinerary

where person.id=itinerary.p\_id and s\_time<='2021-02-02 14:00:00' and e\_time >= '2021-02-02 10:00:00'

group by p\_id,fullname

order by count(itinerary.id) desc,fullname;

1. 从隔离点中，查询房间数(capacity)居第二多的隔离点名称及其房间数。

第二多意味着该元素为所有房间数集合中去掉最多后的新集合中取值最大的元素。

select top 1 location\_name,capacity

from isolation\_location

where capacity not in (

    select max(capacity)

    from isolation\_location

    )

group by location\_name,capacity

order by capacity desc;

### 2.2.5 创建触发器

编写一个触发器，用于实现以下完整性控制规则： 当隔离表（isolation\_record）中的某位隔离人员在诊断表（diagnose\_record）中的诊断结果（result）为1(新冠确诊)”时，自动将隔离表中的隔离状态（state)改成3（转入医院）。

CREATE TRIGGER trigger\_tpl

ON diagnose\_record

AFTER INSERT,UPDATE

AS

BEGIN

    declare @diag\_result int;

    select @diag\_result = result from inserted;

    if @diag\_result = 1

        begin

            update isolation\_record set state = 3

            from isolation\_record,inserted

            where isolation\_record.p\_id = inserted.p\_id

        end

END;

### 2.2.6 创建函数并在语句中使用它

1. 编写一个依据人员编号计算其到达所有地点的次数(即行程表中的记录数)的自定义函数，同一人员到达同一地点多次，去几次算几次。

函数传入的参数为地点的编号，返回值为该地点在行程表中出现的次数

create function Count\_Records(@a int)

returns int

as

begin

    declare @b int;

    select @b = count(loc\_id) from itinerary where itinerary.p\_id=@a;

    return @b;

end;

1. 并利用其查询至少有3条行程记录的人员。

select \* from person

where dbo.Count\_Records(person.id)>=3

order by person.id;

### 2.2.7 了解系统的查询性能分析功能

选择2.2.4节中某些较为复杂的SQL语句，查询其执行之前系统给出的分析计划和实际的执行计划，记录观察的结果，并对其进行简单的分析。

1. 使用关键字EXPALIN查询2.2.4节中第17题的语句，修改最终的查询结果为全部的属性。

select \*

from isolation\_location

where capacity not in (

    select max(capacity)

    from isolation\_location

    )

group by location\_name,capacity

order by capacity desc

limit 1；

1. 查询结果如表2.1所示。

表 2.1 系统查询行分析表

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| id | select\_type | table | type | possible\_keys | key\_len | ref | rows | Extra |
| 1 | PRIMARY | isolatin\_location | ALL | null | null | null | null | Using where; Using temporary; Using filesort |
| 2 | SUBQUERY | isolation\_location | ALL | null | null | null | 6 | null |

select\_type中，查询中若包含任何复杂的子部分，最外层查询则被标记为Primary

SUBQUERY指出在WHERE列表中包含了子查询

table代表执行的表格，两次都是对isolation\_location表进行操作

type代表查询的结果，ALL是全表扫描，ref是对于每个来自前面的表的记录，所有匹配的行从这张表中取出。

possible\_key：查询可能用到的索引，两次都不需要用到索引

key：查询真正用到的索引，两次都没有用到索引

ref：连接查询时显示驱动表的关联字段。显然这里并没有用到连接

rows：执行计划中估算的扫描行数

Extra：Using temporary代表使用了临时表存储中间的结构，Using where代表使用了where筛选，Using filesort意味着MYSQL需要多做一次额外的快排。所以看出第一次查询的优化并不是很好。

### 2.2.8 DBMS存储过程

尝试编写DBMS的存储过程，建立每个隔离点的人数统计表，并通过存储过程更新该表。

1. 首先建立每个隔离点的人数的统计表

create table isolation\_people\_num(

loc\_id int,

num int,

constraint ck\_loc\_id foreign key(loc\_id) references itinerary\_location(id)

);

1. 创建存储过程

create procedure count\_people(in loc\_id int)

begin

declare num int;

set num = (select count(p\_id) from isolation\_record where isol\_loc\_id=loc\_id)

insert into isolation\_people\_num values(loc\_id,num);

end

1. 调用该存储过程，向建立的表中插入数据

call count\_people(1);

call count\_people(2);

call count\_people(3);

call count\_people(4);

call count\_people(5);

call count\_people(6);

1. 最终结果如图2.1所示



图2.1 基于存储过程向表中插入数据结果显示

## 2.3任务总结

1. 粗心问题

在本次实验中出现了不少由于粗心带来的问题，而且这些问题检查的时候很耗时，比如创建表时忘记写关键字“table”、使用表时表名写错等。写代码时一定要细致。

1. 中英文编码导致的错误

在2.2.4节数据查询中的第6题评测时，平台多次报出Conversion failed when converting date and/or time from character string的错误，仔细检查代码感觉并没有什么问题，后来询问老师之后才发现我的时间字符串中含有中文字符，导致转换为date类型失败。后来直接在英文环境下输入才通过。

1. 派生表和嵌套

在首次通关全部关卡时，我很多任务的代码设计都是基于派生表写的，后来老师指出派生表本来是效率很低下的一种方法，实际场景中很少用到这种方法，初学者在觉得无法表示时常采用这种方式，但实际上是对SQL的join和嵌套方法掌握不透。后来我尝试将所有的派生表都改为了嵌套的方式，发现其实改起来一点也不麻烦，而且发现Educoder上显示的运行时间确实有所减少。所以在之后的SQL设计中需要多多注意这方面的问题。

# 3实验任务二 软件功能学习

## 任务要求

完成下列1~2题

1. 练习SQL Server或其他某个主流关系数据库管理系统软件的备份方式：数据和日志文件的脱机备份、系统的备份功能。
2. 由于阿里云的备份操作较为复杂，最终使用Navicat进行对数据的备份。但又因为Navicat在导入中文数据时发生乱码，最终使用其它的数据库完成数据备份。
3. 在Navicat主页点击双击对应的数据库打开，然后就可以看到下面出现备份的选项，点击，即可对数据进行备份。操作结果如图3.1所示

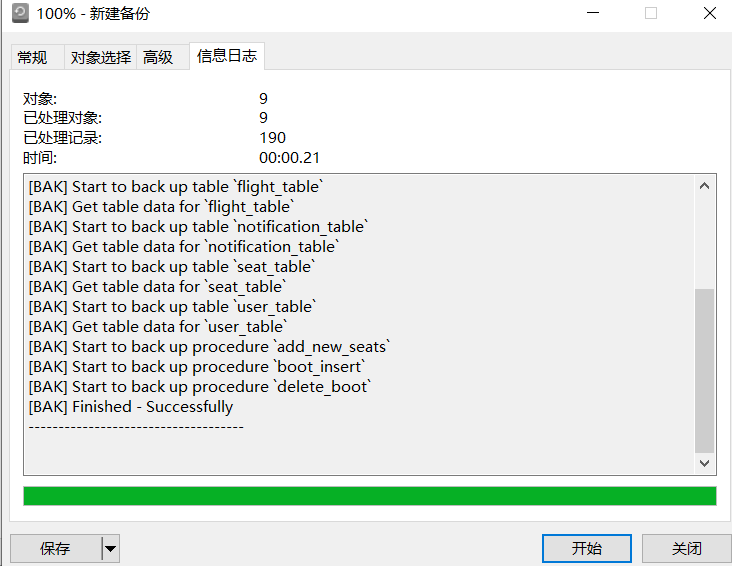


图3.1 利用navicat实现数据备份

1. 还原备份并验证备份的正确性。

修改该数据库的seat表的第一项记录的S\_STATUS列为1，然后点击想要备份的数据版本即可完成数据还原，操作结果前后如图3.2所示。

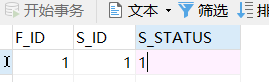
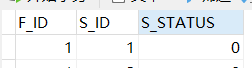
 

图3.2 还原备份操作前后对比（左前右后）

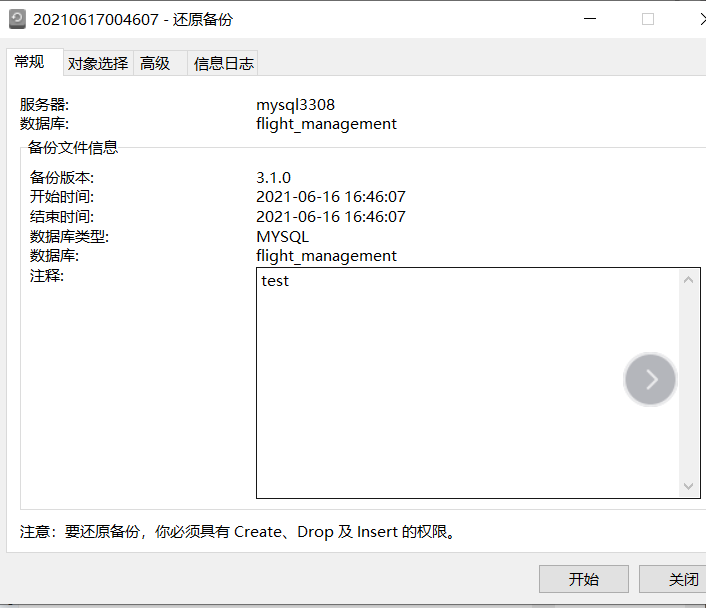


图3.3还原备份操作

1. 练习在新增的数据库上增加用户并配置权限的操作，通过用创建的用户登录数据库并且执行未经授权的SQL语句验证自己的权限配置是否成功。
2. 阿里云平台提供了非常便捷的创建用户的功能，只需要按照图形化界面的显示操作，就可以创建拥有不同权限的用户，而不需要使用SQL指令。分别创建超级用户super\_user和普通用户normal\_user，其创建用户如图3.4所示。其所具有的权限分别如图3.5所示。

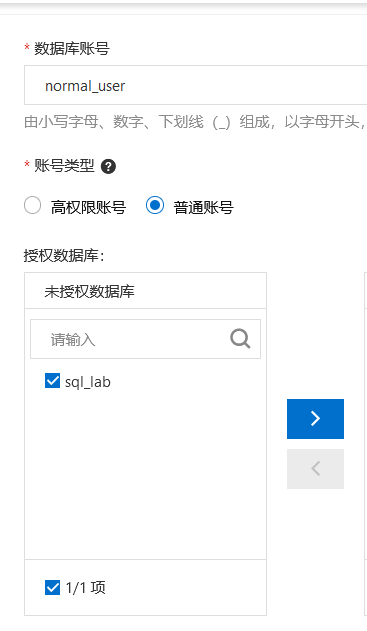
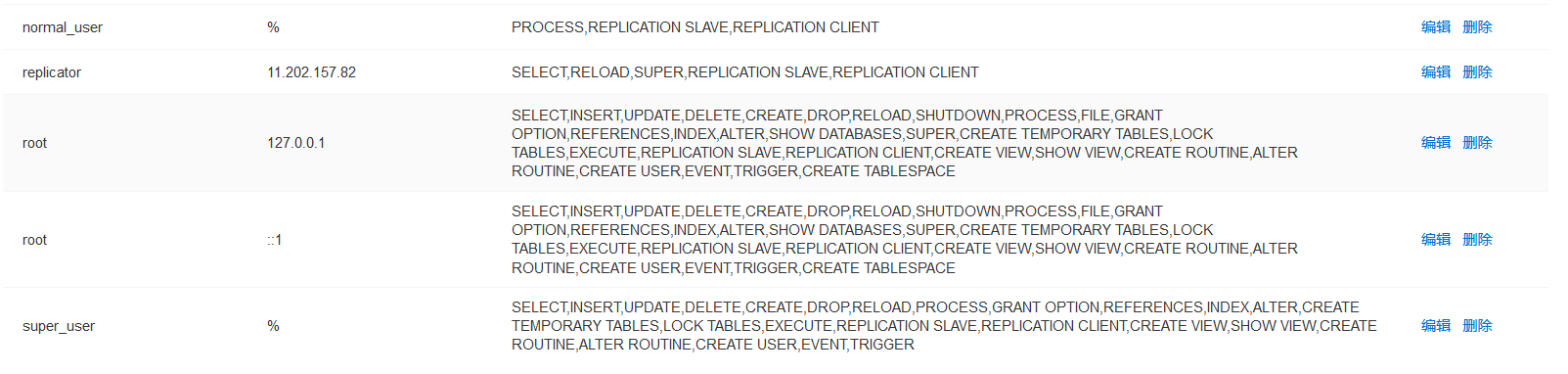


图3.4 分别创建高权限用户super\_user和普通用户normal\_user



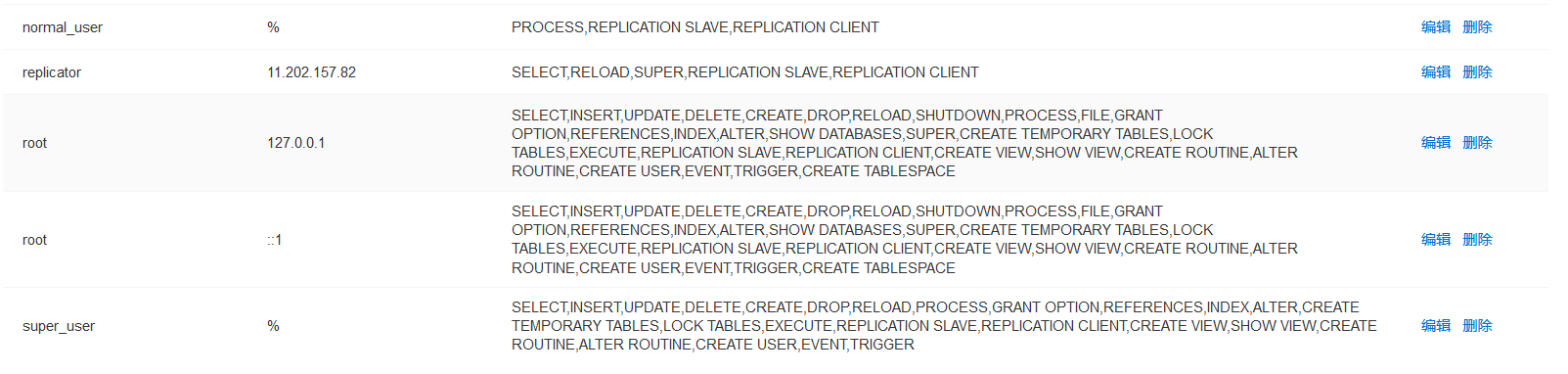


图 3.5 两种用户分别具有的权限

1. 从图3.5可以看出用户”super\_user”具有grant权限但是用户”normal\_user”不具有，所以分别测试在”super\_user”用户环境下下为”normal\_user”赋予select权限和在”normal\_user”用户环境下为super\_user赋予相同权限。

执行结果分别如图3.6和图3.7所示。

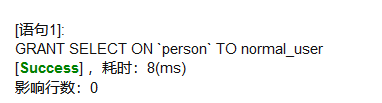


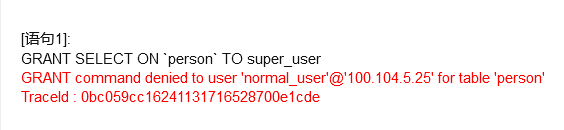
图 3.6 在super\_user用户环境下下为normal\_user赋予select权限

图3.7 在normal\_user用户环境下为super\_user赋予select权限

从上两张图可以看出超级用户想普通用户授权成功，但普通用户向超级用户授权失败，验证两个不同权限账户所能拥有的执行权限不同。

## 3.2任务总结

本次软件功能实现是基于阿里云平台完成的。阿里云的DMS数据库是一个很好的数据库开发软件，图形化的界面很方便对SQL语言了解不多的人使用。在创建函数或触发器或存储过程等直接点击创建，会自动生成固定的代码，只需要在此基础上进行修改即可。并且它还有配套的教程，非常助于使用。在本次实验中，我基于阿里云的DMS数据库完成了数据库和表的创建和查询，并由此完成了不同权限用户的实验。在实验时遇到一个问题就是关于备份的问题。教程上说使用任务编排进行周期性备份，但在实际操作时，点击备份任务节点，页面右侧并未出现内容设置（添加数据库），操作时相较与navicate操作较为复杂，但其功能十分完善。

# 4 综合实践任务

## 4.1系统设计目标

互联网技术的发展为航空公司带来了全新的售票方式，并随着电子机票的普及，网上订票比传统售票方式有更多的好处：随时随地查询和预订机票，航班出发提醒等。本实验希望参照现有的机票预订系统并能够较为完善的还原机票预订管理以及用户在线预订平台的全过程，并且实现的系统操作简单便捷且实用安全。实现的过程包括航班信息管理、用户信息管理、订单信息管理、机票信息管理等，主要解决为用户提供航班信息、航班查询、预订机票和退票等服务项目管理。

## 4.2 需求分析

### 4.2.1 功能需求

1. 用户界面主要实现的功能包括

* 查询：用户输入出发地点和目的地点以及出发日期，能够显示当天的航班情况。
* 个人信息显示及修改：显示个人信息并提供修改功能
* 订票：对用户指定的航班航次，输入用户的相关信息完成预订
* 查询个人所预订的机票和退票：在用户所订有的票界面，根据用户的退票需求在数据库中删除或更新有关机票的记录
* 打印取票通知：在行程出发的前一天用户可以对有关的机票进行打印
* 登录与注册：进入系统预订机票前用户根据账户类型登录自己的系统账号，如果第一次使用，需要先注册相关账号

1. 后台功能包括：

* 航班信息管理：能新增航班和删除已有的航班
* 用户信息管理：能对用户的信息（除密码和用户名以外）进行修改
* 订单管理：能对所有用户预订的订单进行删除管理

### 4.2.2 性能需求

1. 可操作性

进行机票服务之前，应将数据库的原始信息准备好。如软件系统的用户名和初始密码等信息、客机信息和航班信息等。数据输入的格式应该符合业务习惯，并且直观、方便。要求系统处理的数据准确且输出信息直观简洁。

1. 可靠性

系统运行具有较高的可靠性，提供较为严格的并发控制，确保数据的一致性和正确性，防止脏数据的读入和数据的丢失，

1. 实用性

从用户的实际需要出发，不盲目追求高新技术和花里胡哨的功能。系统界面要求友好，易于使用。

### 4.2.3 数据完整性需求

1. 实体完整性

用户表主码：用户ID

航班信息主码：航班ID

旅客订票信息主码：机票ID

1. 参照完整性
2. 账单表中参照了航班表的航班ID和用户表中的用户ID，通过这两个外码可以锁定账单对应的航班信息和用户信息。
3. 旅客订票信息表参照了航班表的航班ID和用户表的用户ID，通过这两个外码可以锁定订单对应的航班和用户。
4. 取票通知表参照了航班表的航班ID和用户表的用户ID，通过这两个外码可以锁定取票通知对应的航班和用户。
5. 座位表参照了航班表的航班ID，通过这个外码可以将某些座位锁定某一航班。
6. 用户自定义完整性

唯一且不为空：用户表中的用户名和身份证号。这样可以方便后台数据操作且身份证唯一和现实相对应。

### 4.2.4 数据流图

这里给出主要流程的数据流图。

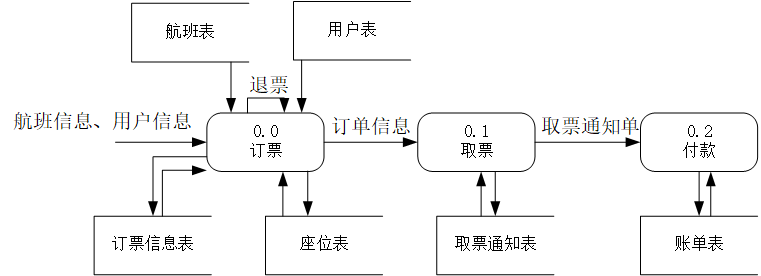


图4.1 机票预订系统主要流程的数据流程图显示

### 4.2.5 数据字典

1. 主要的存储定义
2. 数据存储名称：航班表

数据结构：航班信息=航班编号+航班号+出发时间+到达时间+出发站+目的站+航班容量+航班票价

说明：对航班进行维护

1. 数据存储名称：用户表

数据结构：用户信息=用户编号+用户名+用户密码 +用户真实姓名+身份证号+用户类型

说明：对用户信息进行维护

1. 数据存储名称：账单信息

数据结构：用户编号+航班编号+座位编号+座位票价+操作类型

说明：对订单信息进行维护

1. 数据存储名称：订票信息表

数据结构：机票信息=机票编号+用户编号+航班编号+座位编号+座位票价

说明：对机票信息进行维护

1. 数据存储名称：取票通知表

数据结构：取票信息=用户编号+航班编号+座位编号+座位票价+出发时间+到达时间

说明：对取票信息进行维护

1. 数据存储名称：座位表

数据结构：航班编号+座位编号+座位状态

1. 主要的处理过程
2. 处理过程编号：0.0

处理过程名称：订票

输入：航班信息、用户信息

输出：订票信息

说明：用户对想要预订的航班进行预订机票

1. 处理过程编号：0.1

处理过程名称：取票

输入：订票信息

输出：取票通知单

说明：用户完成订票后，通过订票信息，在飞机出发的前一天打印取票通知单

1. 处理过程编号：0.2

处理过程名称：付款

输入：取票通知单

输出：账单

说明：用户凭借取票通知单完成付款后可以生成账单

## 4.3 总体设计

### 4.3.1 系统体系架构图

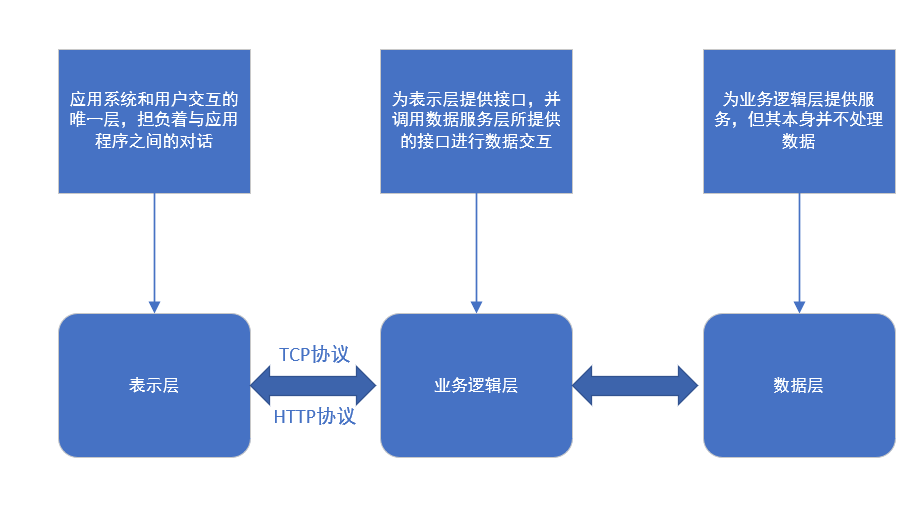


图4.2 系统体系架构图

系统前端由JSP实现，后台使用Java实现。

用户在前端操纵网页上的按钮，同时按钮绑定jsp文件或者servlet事件，后台执行相应的文件或函数完成对数据库的操作，更新数据。

### 4.3.2 功能模块组成及其说明

本次实验设计的系统一共有八个模块。

1. 登录模块。前端用户输入自己的用户名和密码，客户端将用户名和密码传给后台服务器，服务器将通过查询数据库验证输入的用户名和密码是否正确和匹配。首次使用该系统的用户首先需要注册一个账号，注册时会在相应的数据库表中插入有关记录。用户进行登录时，后台会查询数据库该账号属于管理员账号还是普通用户账号，从而进入不同的界面。
2. 主页。登录到系统后，直接显示个人主页。对于用户，客户端显示用户的用户名、真实姓名、身份证号码，并提供修改功能，用户点击然后输入相关信息即可完成修改。对于管理员，仅仅只输出“你好，xx”的信息。
3. 查询和订票模块。用户输入行程的出发城市和目的城市以及出发时间，后端将通过查询数据库将满足条件的航班信传递给前端，并以表单的形式向用户展示，最后的表单项目包括航班信息：航班号，出发时间，到达时间，始发站，目的站，航班容量，票价和满座率。 此界面提供预订的功能，用户点击某趟航班的预订按钮，输入个人信息，如果满足要求即可完成预订。
4. 订单模块。在此界面，客户端将用户的所有已经预订的机票以表单的形式显示出来，并提供退票的功能。
5. 取票模块。在此界面，客户端只会显示当天是用户所定的航班的出发日期的前一天的航班，并提供打印功能，用户点击后可以实现打印取票信息。
6. 用户信息管理模块。此模块只有管理员能够进入，管理员可以修改用户的部分信息。
7. 航班管理模块。此模块只有管理员能够进入，客户端会显示目前拥有的所有航班的信息，并提供删除功能供管理员删除航班。同时另外有一个界面可以让管理员录入新的航班，管理员只需要提交航班的具体信息即可。
8. 订单管理模块。此模块只有管理员能够进入。客户端会将目前未被退票的所有订单显示，并提供删除功能供管理员删除订单。

### 4.3.3 系统总体业务流程图及其说明

系统总体业务主要为订票。

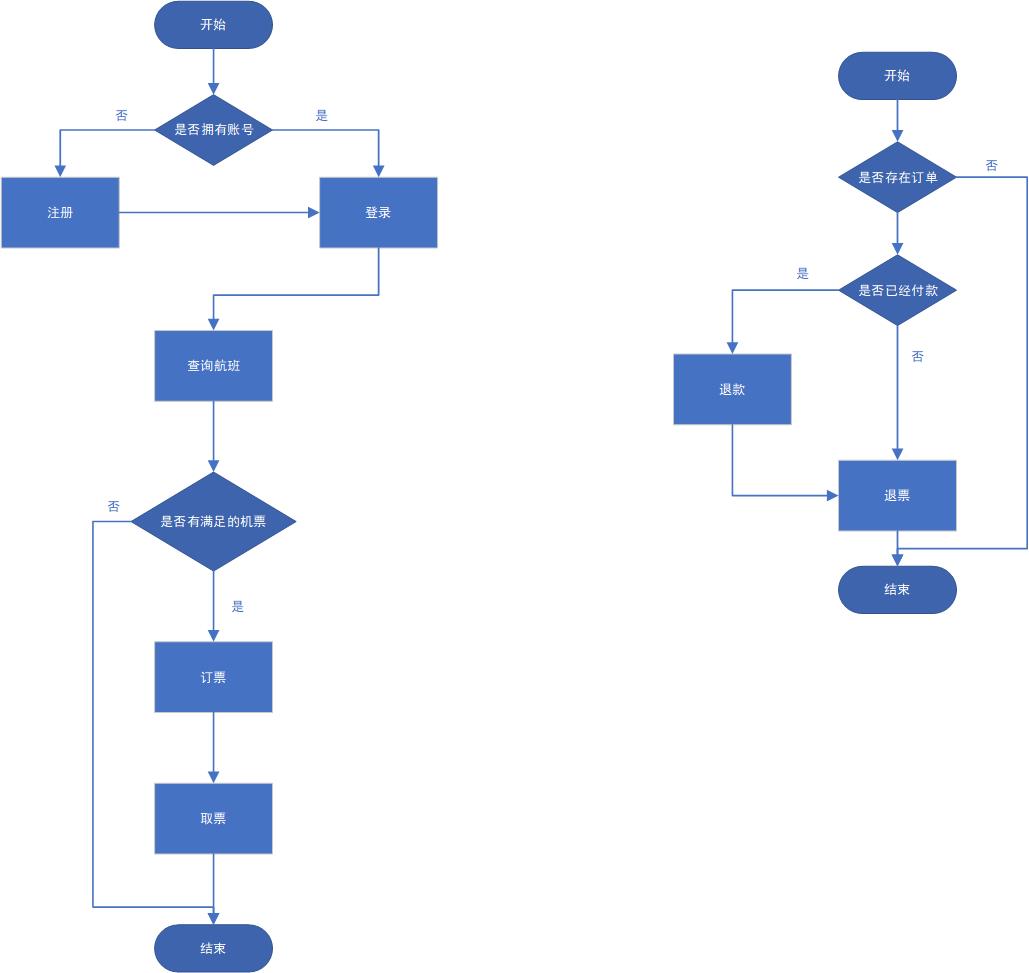


图4.3 系统取票/退票流程图

## 4.4 数据库设计

### 4.4.1 ER图设计及说明

一个用户可以预定多趟航班，一趟航班承载了多个用户。用户预订一趟航班后形成一份订单。一趟航班包含多个座位。一张机票对应一份取票通知、一份账单和一个座位。

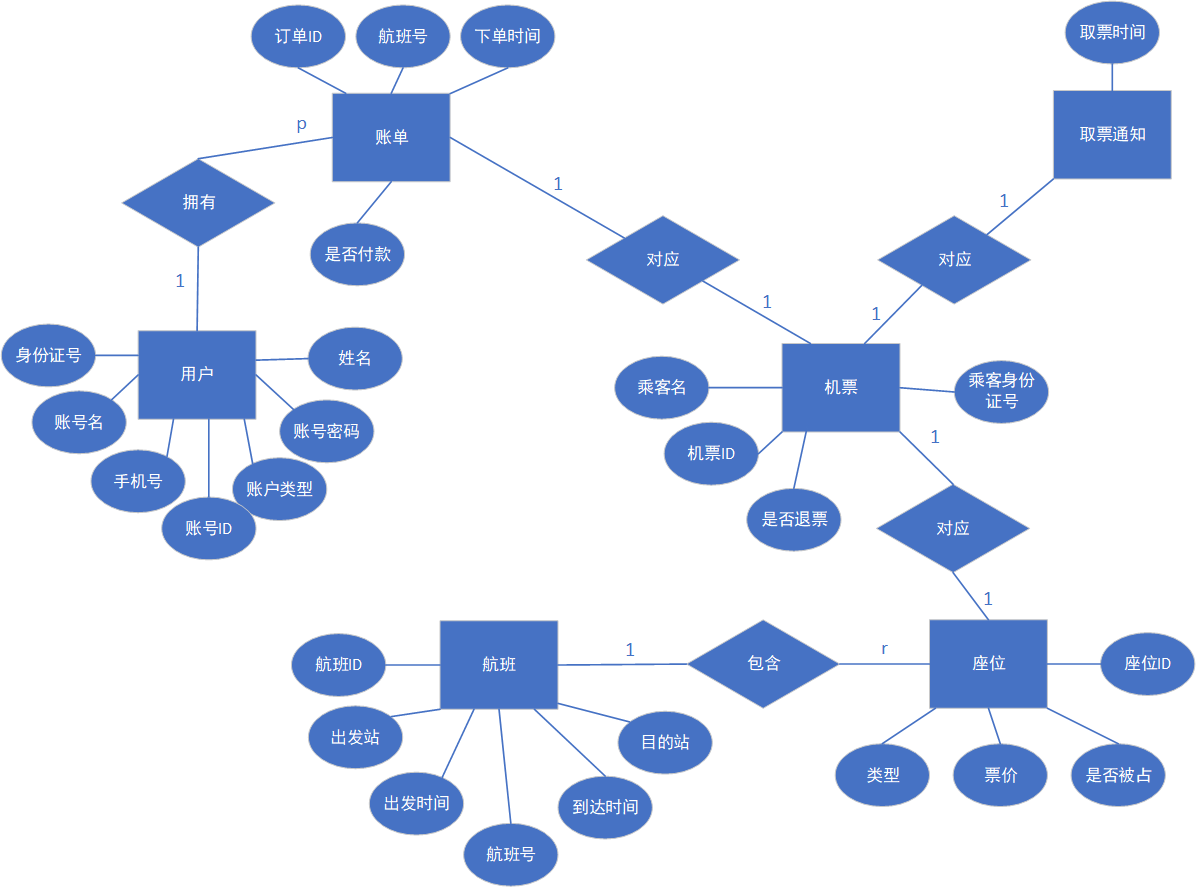


图4.4 系统总体ER图

### 4.4.2 数据库逻辑结构设计

1. 航班信息表

表 4.1 航班信息表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 字段名 | 描述 | 类型 | 约束 | 备注 |
| F\_ID | 航班编号，标识种子，自动增长 | Int(11) | PK |  |
| F\_NAME | 航班号 | Varchar(255) |  |  |
| S\_TIME | 出发时间 | Varchar(255) |  |  |
| E\_TIME | 到达时间 | Varchar(255) |  |  |
| S\_CITY | 出发城市 | Varchar(255) |  |  |
| E\_CITY | 到达城市 | Varchar(255) |  |  |

1. 航班座位信息表

表 4.2 航班座位信息表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 字段名 | 描述 | 类型 | 约束 | 备注 |
| F\_ID | 航班编号 | Int(11) | FK | 参照航班表的 |
| S\_ID | 座位编号 | Int(11) | PK |  |
| S\_STATUS | 座位类型 | Varchar(255) |  | 0: 被预订  1：未被预订 |

1. 旅客订票信息表

表 4.3 旅客订票信息表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 字段名 | 描述 | 类型 | 约束 | 备注 |
| T\_ID | 机票编号，标识种子，自增长 | Int(11) | PK |  |
| F\_ID | 航班编号 | Int(11) | FK | 参照航班表 |
| S\_ID | 座位编号 | Int(11) | FK | 参照座位表 |
| U\_ID | 订单编号 | Int(11) | FK | 参照用户表 |
| S\_PRICE | 出发时间 | Varchar(255) |  |  |

1. 取票通知表

表 4.4 取票通知表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 字段名 | 描述 | 类型 | 约束 | 备注 |
| U\_ID | 用户编号 | Int(11) | FK | 参照用户表 |
| F\_ID | 航班编号 | Int(11) | FK | 参照航班表 |
| S\_ID | 座位编号 | Int(11) | FK | 参照座位表 |
| S\_PRICE | 座位票价 | Int(11) |  |  |
| S\_TIME | 出发时间 | Varchar(255) |  |  |
| L\_TIME | 到达时间 | Varchar(255) |  |  |

1. 账单表

表 4.5 账单表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 字段名 | 描述 | 类型 | 约束 | 备注 |
| U\_ID | 用户编号 | Int(11) | FK | 参照用户表 |
| F\_ID | 航班编号 | Int(11) | FK | 参照航班表 |
| S\_ID | 座位编号 | Int(11) | FK | 参照座位表 |
| S\_PRICE | 座位票价 | Int(11) |  |  |
| OP | 操作 | Int(11) |  | 0:已订票  1：退票 |

1. 用户表

表 4.6 用户表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 字段名 | 描述 | 类型 | 约束 | 备注 |
| U\_ID | 用户编号，标识种子，自增长 | Int(11) | PK |  |
| USRENAME | 用户名 | Varchar(255) | Unique  Not Null |  |
| PASSWORD | 用户密码 | Varchar(255) | Not Null |  |
| U\_NAME | 用户真实姓名 | Varchar(255) |  |  |
| ID\_CARD | 身份证号 | Varchar(255) | Unique |  |
| U\_TYPE | 用户类型 | Int(11) | Not Null | 0: 普通用户  1：管理员 |

### 4.4.3 数据库物理设计

由于本系统规模较小，查询可以在极短时间内完成，数据占用空间也很小，效率较高，所以并没有专门为数据库的物理结构增加额外的设计。

## 4.5 详细设计与实现

### 4.5.1 开发环境

开发语言：JAVA + JSP/CSS

数据库：mysql-connector-java-5.1.39

### 4.5.2 系统业务流程图

根据用户权限类型，下图分别为普通用户业务流程图和管理员业务流程图。

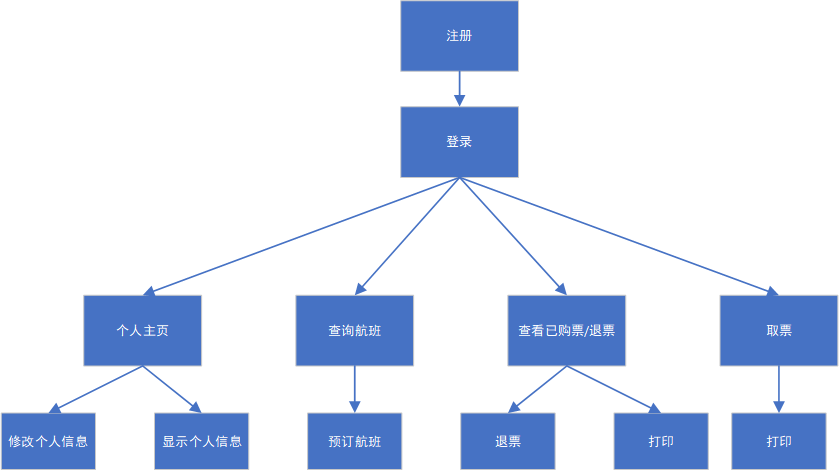


图 4.5 普通用户业务流程图

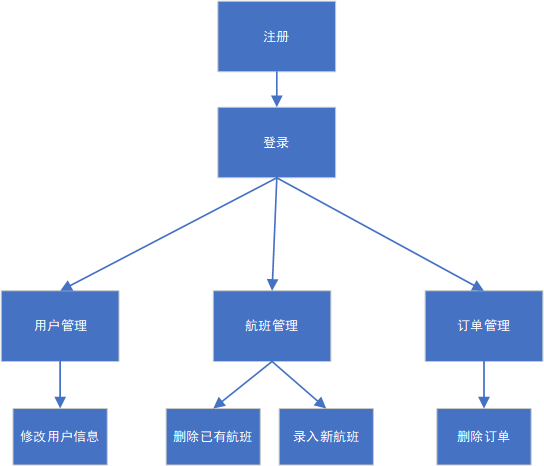


图 4.6 管理员业务流程图

### 4.5.3 关键技术和算法说明

下面列举一些系统核心的重要操作及其算法进行说明。

1. 注册

算法设计：

1. 用户输入用户名、密码、用户真实姓名、身份证号信息然后提交
2. 检查输入是否有空，如果有空则退回并显示错误信息
3. 执行SQL查询，向用户表中插入新的用户，由于用户表中用户名被设置为唯一约束，所以执行时如果检查到用户输入的用户名已经在在系统中存在，会返回错误信息，插入失败，注册失败

|  |
| --- |
| “INSERT INTO USER\_TABLE(USERNAME,PASSWORD,U\_NAME,ID\_CARD,U\_TYPE) VALUES(?,?,?,?,0)",user.getName(),user.getPassword(),  user.getRealName(),user.getIdCard() |

1. 登录

算法设计：

1. 用户输入用户名和密码并点击登录
2. 检查输入是否有空，若有空则返回并显示错误信息。
3. 执行SQL语句，查询用户表中是否存在该用户名，如果不存在则返回错误信息。

|  |
| --- |
| “SELECT \* FROM USER\_TABLE WHERE USERNAME=? AND PASSWORD=?",**new** UserResultSetHandler(),username,password |

1. 客户端根据查询到的用户元组中记录的用户类型的属性，如果该值为0则跳转到用户界面，如果为1则跳转到管理员界面
2. 航班查询实现

算法设计：

1. 用户输入想要出发地点、目的地点和出发时间
2. 判断用户提交数据是否有空，若有则提示错误信息
3. 执行SQL查询，获取满足的航班信息

|  |
| --- |
| “SELECT \* FROM flight\_table WHERE START\_CITY=? AND END\_CITY=? AND SUBSTRING\_INDEX(START\_TIME,\' \',1) =? ORDER BY END\_TIME ASC",**new** FlightResultSetHandler(),start,destination,s\_time |

1. 计算当前航班的满座率，加入到要显示的航班信息中。
2. 订票

算法设计：

1. 用户查询完航班后，点击对应航班的预订按钮
2. 通过航班号查询该航班的所有座位中为空的座位且编号最小的作为预订的座位，这里直接使用java语言设计完成
3. 由上一步得到座位号，更新座位表中对应的元组记录，设置该元组的S\_STAUS=1表示已经有人预订
4. 在订单表、取票通知表和账单表中插入相应的信息

3和4通过定义了存储事务的过程实现，保证了数据库内容的正确性。

|  |
| --- |
| “CALL boot\_insert(?,?,?,?)",userID,flightID,seatID,seatPrice |

1. 退票

算法设计：

1. 用户点击对应的订单，客户端向后端传递对应机票信息
2. 更新座位表，将对应的座位的状态置为0表示无人预订
3. 在订单表和通知表中删除相应的记录
4. 在账单表中插入相应的信息，其中操作置为1表示退票

2，3，4使用定义了存储事务的过程完成，保证数据库内容的正确性。

|  |
| --- |
| "CALL delete\_boot(?,?,?,?,?)",ticketID,userID,flightID,seatID,seatPrice |

1. 取票

算法设计：

1. 用户点击取票界面，后端执行SQL语句查询该用户所有预订的机票

|  |
| --- |
| “SELECT \* FROM FLIGHT\_RESERVE\_TABLE WHERE U\_ID=?",**new** FlightReserveResultSetHandler(),user.getUserID() |

1. 遍历所有这些机票，如果这些机票对应的航班的出发日期为当前日期的后一天，则加入到取票通知单中，否则忽略。这样就确保了只有在出发的前一天用户才可以打印取票通知单
2. 为每一个待取票的订单设置一个按钮，该按钮绑定浏览器的打印接口
3. 点击打印后，后端会提供机票的详细信息进行预览，点击确认即可完成打印。
4. 管理员用户管理

算法设计：

1. 在用户表中查询所有的用户

|  |
| --- |
| "SELECT \* FROM USER\_TABLE",**new** UserResultSetHandler() |

1. 前端提供第一步提供的表单信息，并为每个用户提供修改信息按钮
2. 点击修改按钮，输入要修改的信息——仅包括用户真实姓名和身份证号码，点击提交
3. 执行SQL查询，更新对应用户的信息

|  |
| --- |
| “UPDATE USER\_TABLE SET U\_NAME=?,ID\_CARD=?,U\_TYPE=? WHERE U\_ID = ?",user.getRealName(),user.getIdCard(),user.getUserType(),user.getUserID() |

1. 管理员航班管理

算法设计：

1. 在航班表中查询所有航班信息，按照航班编号升序排列

|  |
| --- |
| "SELECT \* FROM FLIGHT\_TABLE ORDER BY F\_ID ASC",**new** FlightResultSetHandler() |

1. 前端显示上面查询到的航班表单信息，并为每一个航班添加一个“删除”按钮
2. 若用户点击“删除”按钮，则前端向后端提供航班编号信息，后端执行SQL查询，依次删除订票信息表、账单表、取票通知表、座位表和航班表中有关该航班的元组

|  |
| --- |
| "DELETE FROM FLIGHT\_RESERVE\_TABLE WHERE F\_ID=?",flightID  "DELETE FROM BILL\_TABLE WHERE F\_ID=?",flightID  "DELETE FROM NOTIFICATION\_TABLE WHERE F\_ID = ?",flightID  "DELETE FROM SEAT\_TABLE WHERE F\_ID = ? ",flightID  "DELETE FROM FLIGHT\_TABLE WHERE F\_ID=?",flightID |

1. 删除成功后返回到原来界面，并显示删除成功的提示信息。
2. 录入新航班

算法设计：

1. 管理员输入航班信息
2. 判断输入的航班信息是否为空，若为空退回并显示错误信息
3. 判断输入的航班信息是否规范，若不规范退回并显示错误信息
4. 对航班执行SQL插入

|  |
| --- |
| "INSERT INTO FLIGHT\_TABLE (F\_NAME,START\_TIME,END\_TIME,START\_CITY,END\_CITY,PRICE,CAPACITY) VALUES(?,?,?,?,?,?,?)", flight.getName(),flight.getS\_time(),flight.getE\_time(),flight.getS\_city(),flight.getE\_city(), flight.getPrice(),flight.getCapacity() |

1. 执行带有存储事务的过程，在座位表中生成座位信息

|  |
| --- |
| "CALL add\_new\_seats(?,?)",flight.getFlightID(),flight.getCapacity() |

1. 管理员订单管理

算法设计：

1. 查询订单表显示所有的订单信息

|  |
| --- |
| "SELECT \* FROM FLIGHT\_RESERVE\_TABLE",**new**FlightReserveResultSetHandler() |

1. 前端显示1提供的订票信息，并为每一个订票提供“删除”按钮
2. 如果点击“删除”，和用户退票执行相同的过程。

### 4.5.4 数据库事务定义和实现

在本次实验中，一共用到了三个数据库过程，其中两个同时还设计了存储事务。目的是保证一系列操作的原子性和持续性，防止在发生意外情况时导致的数据库数据错误。

1. 生成航班座位信息：add\_new\_seats(flighted, capacity)

输入参数：

flighted：航班ID

capacity: 航班容量

作用：每一次新建新的航班时，向座位表中插入该航班的所有座位的信息，用于初始化座位。

具体实现：

|  |
| --- |
| CREATE PROCEDURE `add\_new\_seats`(IN flightid INT(11), IN capacity INT(11))  BEGIN  DECLARE num INT(11);  SET num=1;  WHILE num<capacity+1 DO  INSERT INTO `seat\_table` VALUES(flightid,num,0);  SET num=num+1;  END WHILE;  END |

在使用时执行CALL add\_new\_seats()即可。

1. 订票：boot\_insert(userid, flightid, seatid, price)

输入参数：

userid: 订票用户的ID

flightid：订票航班ID

seatid: 订票座位ID

price：座位票价

作用：在预订机票时，向订单表、通知表和账单表中插入本次订票信息，并更新座位表信息。

具体实现：

|  |
| --- |
| CREATE PROCEDURE `boot\_insert`(IN userid INT(11),IN flightid INT(11),IN seatid INT(11),IN PRICE INT(11))  BEGIN  DECLARE EXIT HANDLER FOR SQLEXCEPTION,SQLWARNING ROLLBACK ;  START TRANSACTION;  UPDATE `seat\_table` SET `S\_STATUS` = 1  WHERE `F\_ID` = flightid and `S\_ID` =seatid;  INSERT INTO `flight\_reserve\_table`(`U\_ID`, `F\_ID` , `S\_ID` , `S\_PRICE` )VALUES(userid,flightid,seatid,price);  INSERT INTO `notification\_table` (`U\_ID`,`F\_ID`,`S\_ID`,`S\_PRICE`)VALUES(userid,flightid,seatid,price);  INSERT INTO `bill\_table` VALUES(userid,flightid,seatid,price,0);  COMMIT;  END |

在使用时执行CALL boot\_insert()即可。

1. 退票：delete\_boot(userid, flightid, seatid, ticketed, price)

输入参数：

userid：订单用户ID

flightid：订单航班ID

seatid：订单座位ID

ticketid: 订单ID

price: 订单价格

作用：在用户退票或者管理员删除订票时，在订单表和通知表中删除记录，在账单表中插入本次操作的内容。

具体实现：

|  |
| --- |
| CREATE PROCEDURE `delete\_boot`(IN Reserveid INT(11),IN Userid INT(11), IN Flightid INT(11), IN Seatid INT(11), IN Price INT(11))  BEGIN  DECLARE EXIT HANDLER FOR SQLEXCEPTION,SQLWARNING ROLLBACK ;  START TRANSACTION;  UPDATE `seat\_table` SET `S\_STATUS` = 0 WHERE `F\_ID` = Flightid and `S\_ID` = Seatid;  DELETE FROM `flight\_reserve\_table` WHERE `T\_ID` = Reserveid;  DELETE FROM `notification\_table` WHERE `F\_ID` = Flightid and `S\_ID` = Seatid;  INSERT INTO `bill\_table` VALUES (Userid, Flightid, Seatid, Price, 1);  COMMIT;  END |

在使用时执行CALL delete\_boot()即可。

### 4.5.5 界面设计

各个主页的截图如下所示：

1. 登录界面

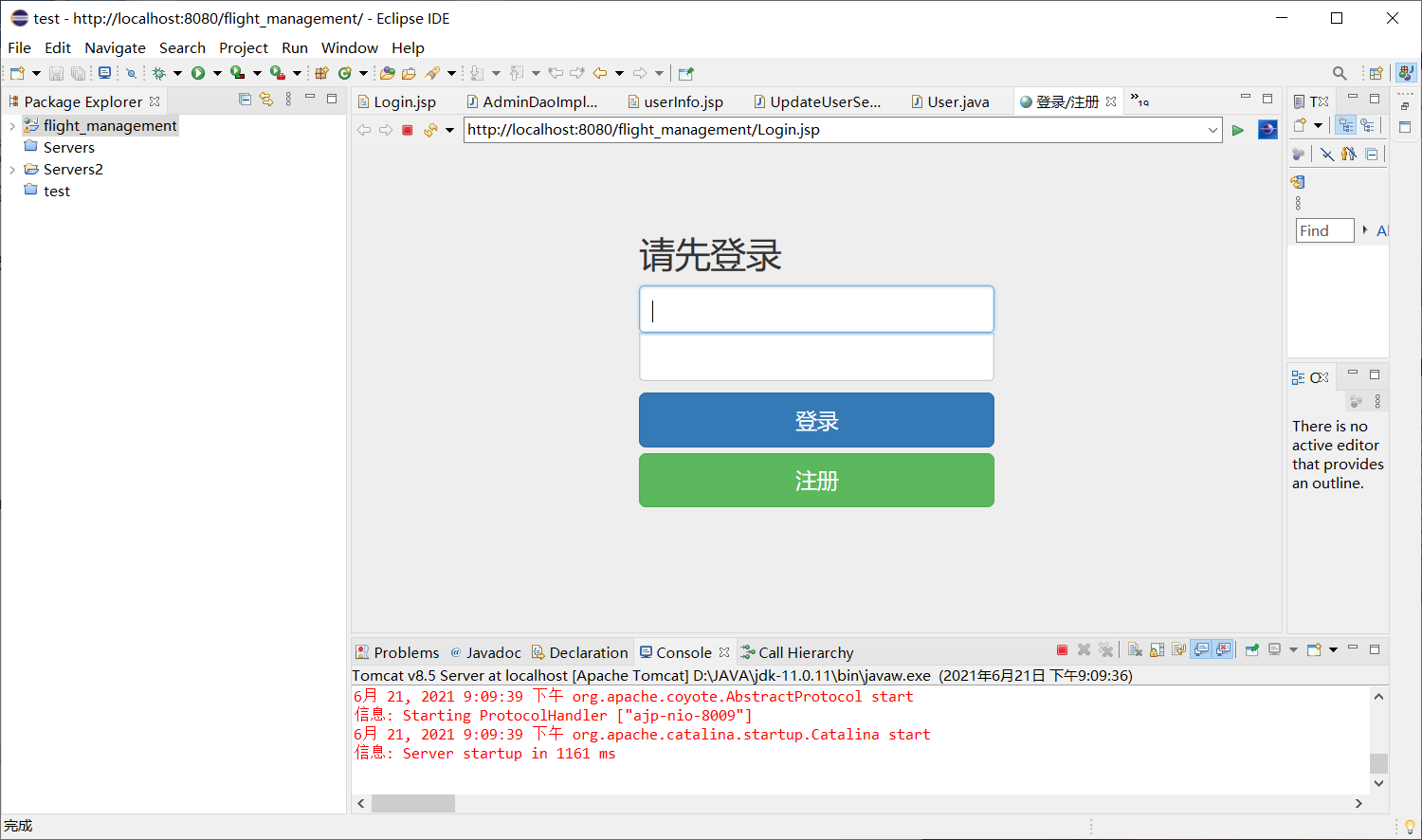


图 4.7 登录界面

1. 注册界面



图 4.8 注册界面

1. 普通用户主页



图 4.9 普通用户主页

1. 管理员主页

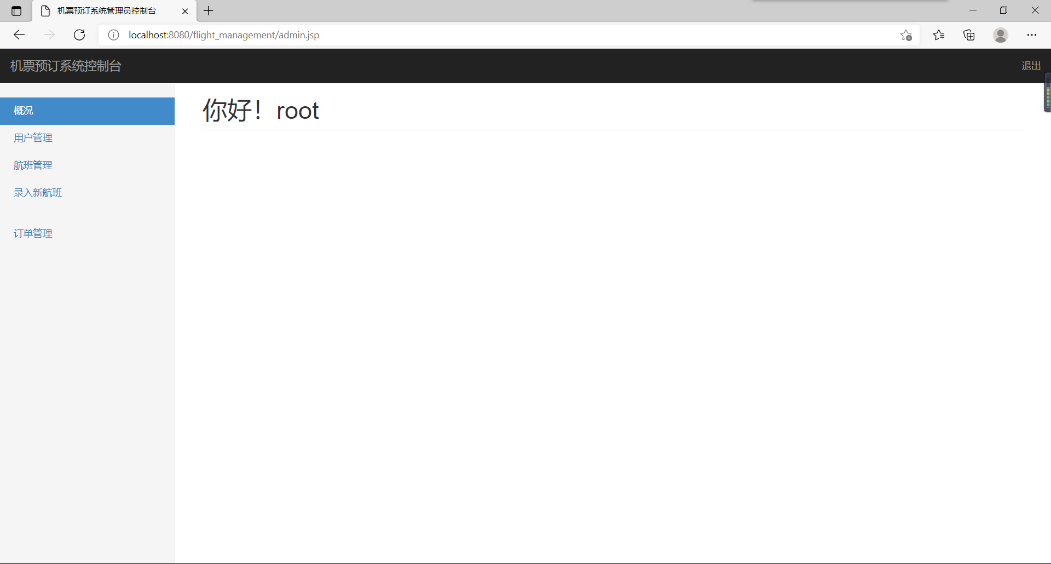


图 4.10 管理员主页

1. 用户修改个人信息



图 4.11 普通用户修改个人信息

1. 查询航班

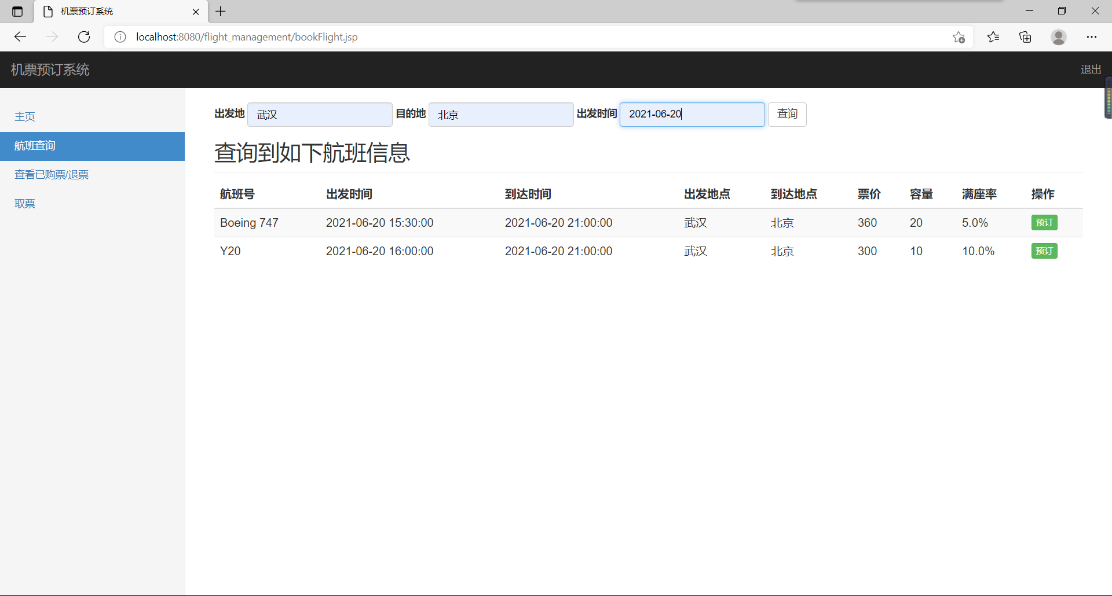


图 4.12 用户查询航班

1. 预订并填写乘客信息

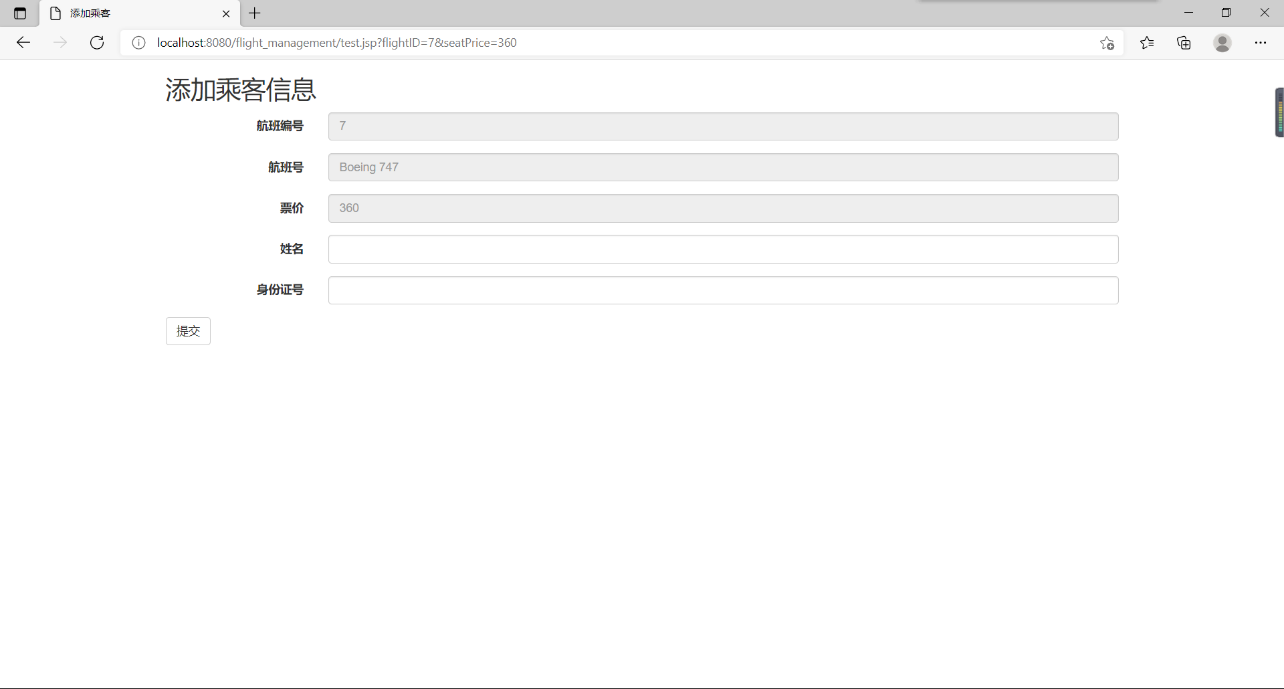


图 4.13 预订航班后添加乘客信息

1. 查看已订票/退票界面

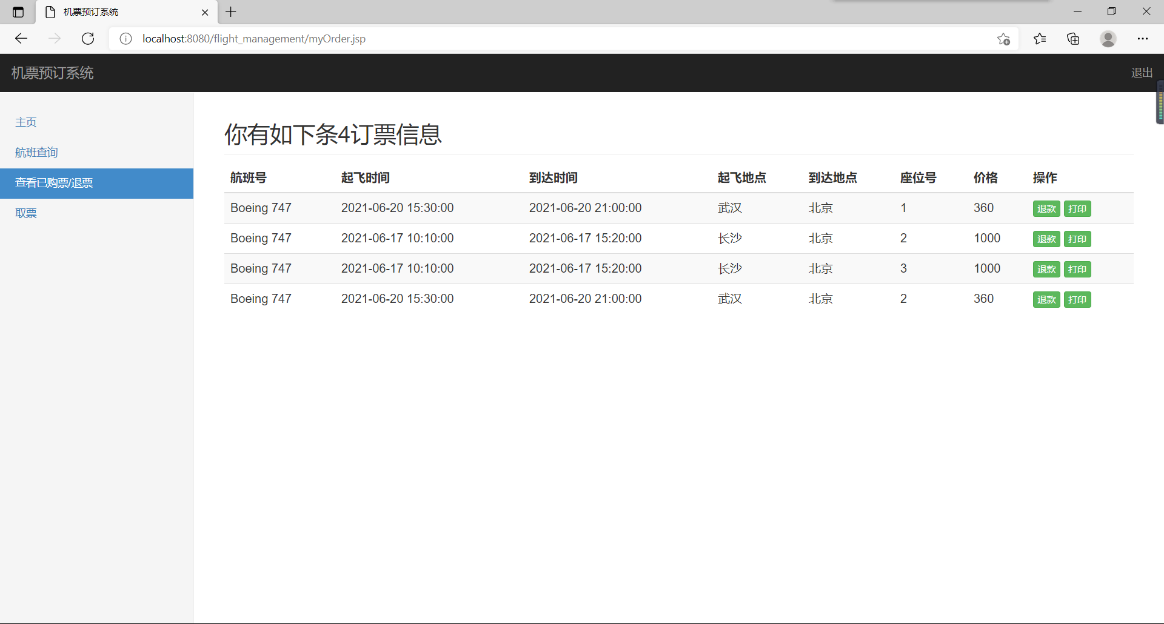


图 4.14 查看已订票/退票

1. 订单打印界面



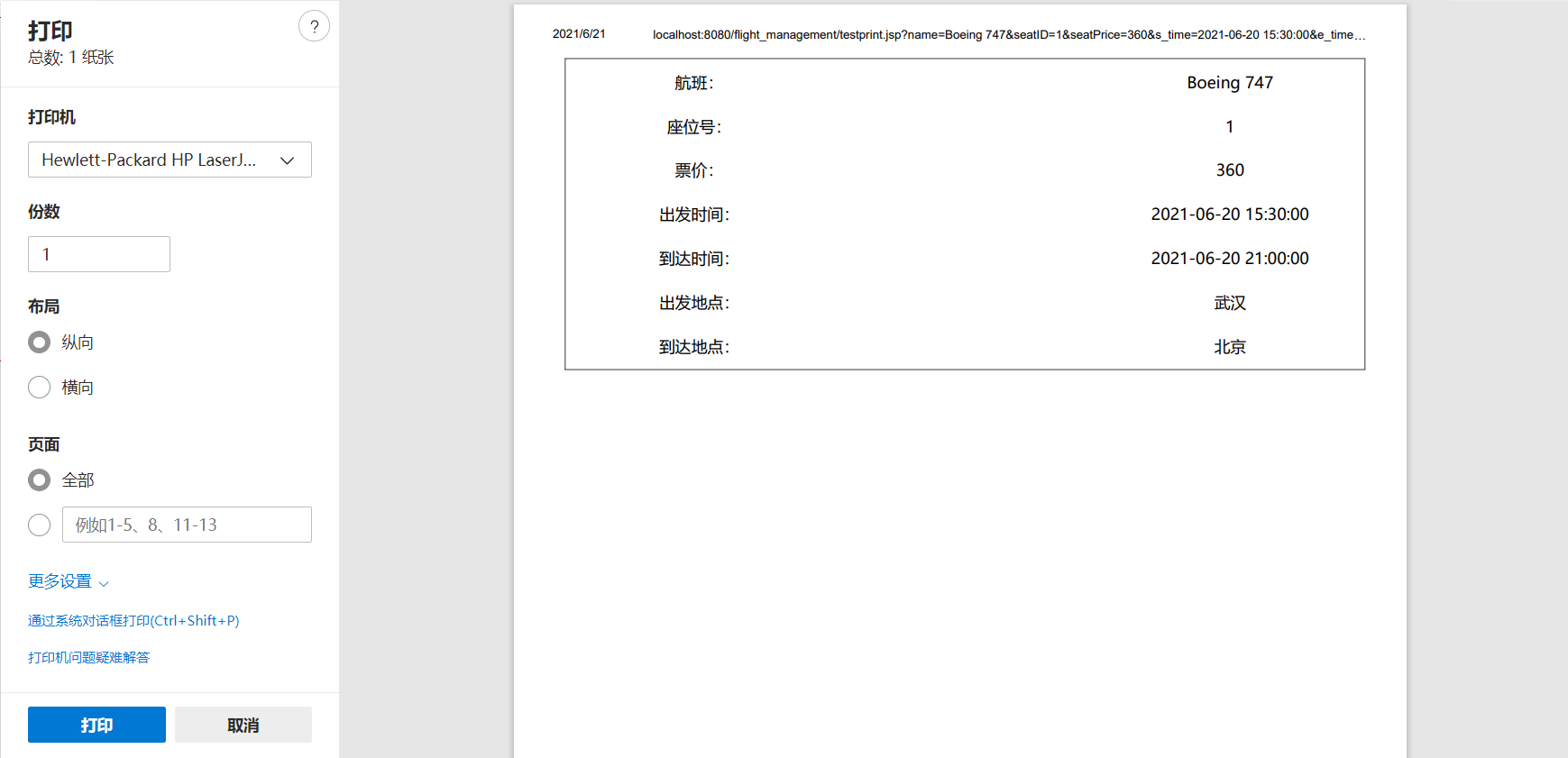


图 4.15 确认打印订单

1. 取票界面

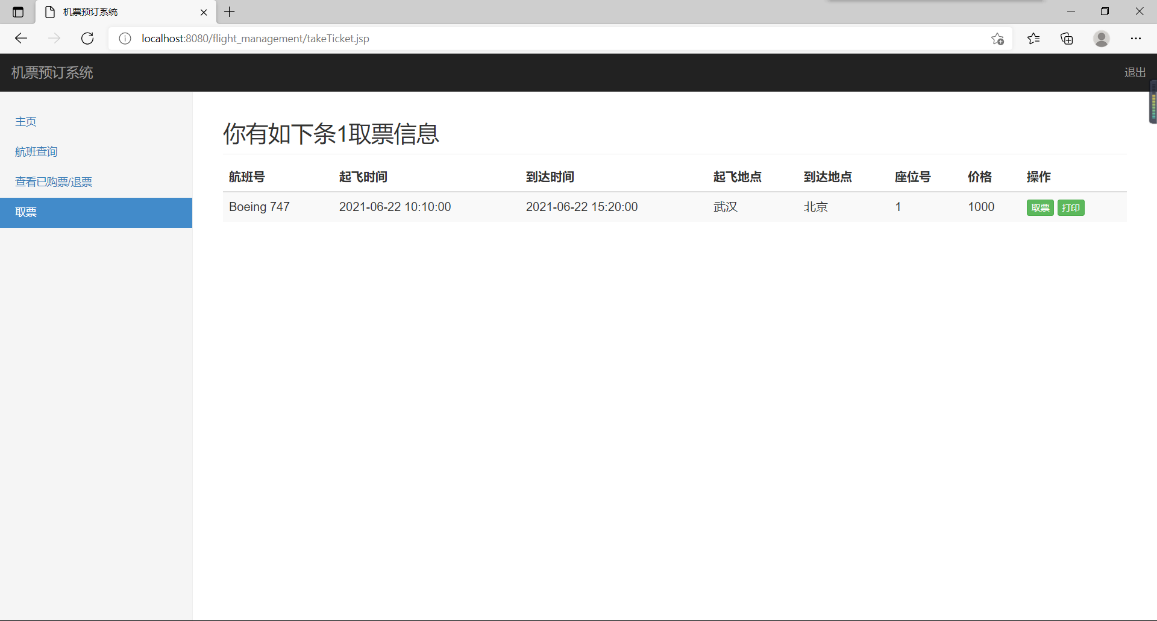


图 4.16 取票界面

1. 打印取票通知单

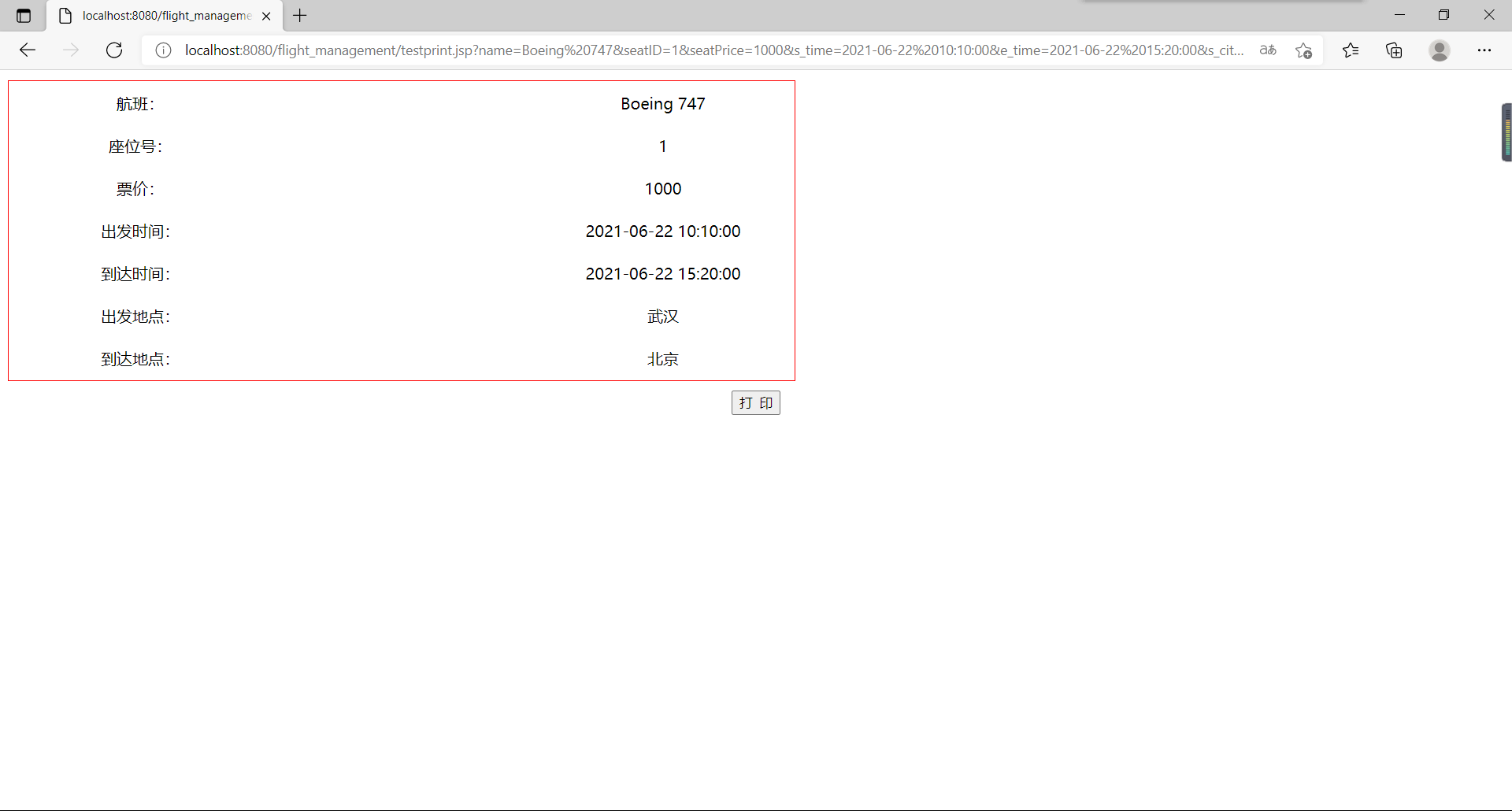




图 4.17 确认打印取票通知单

1. 管理员用户管理界面

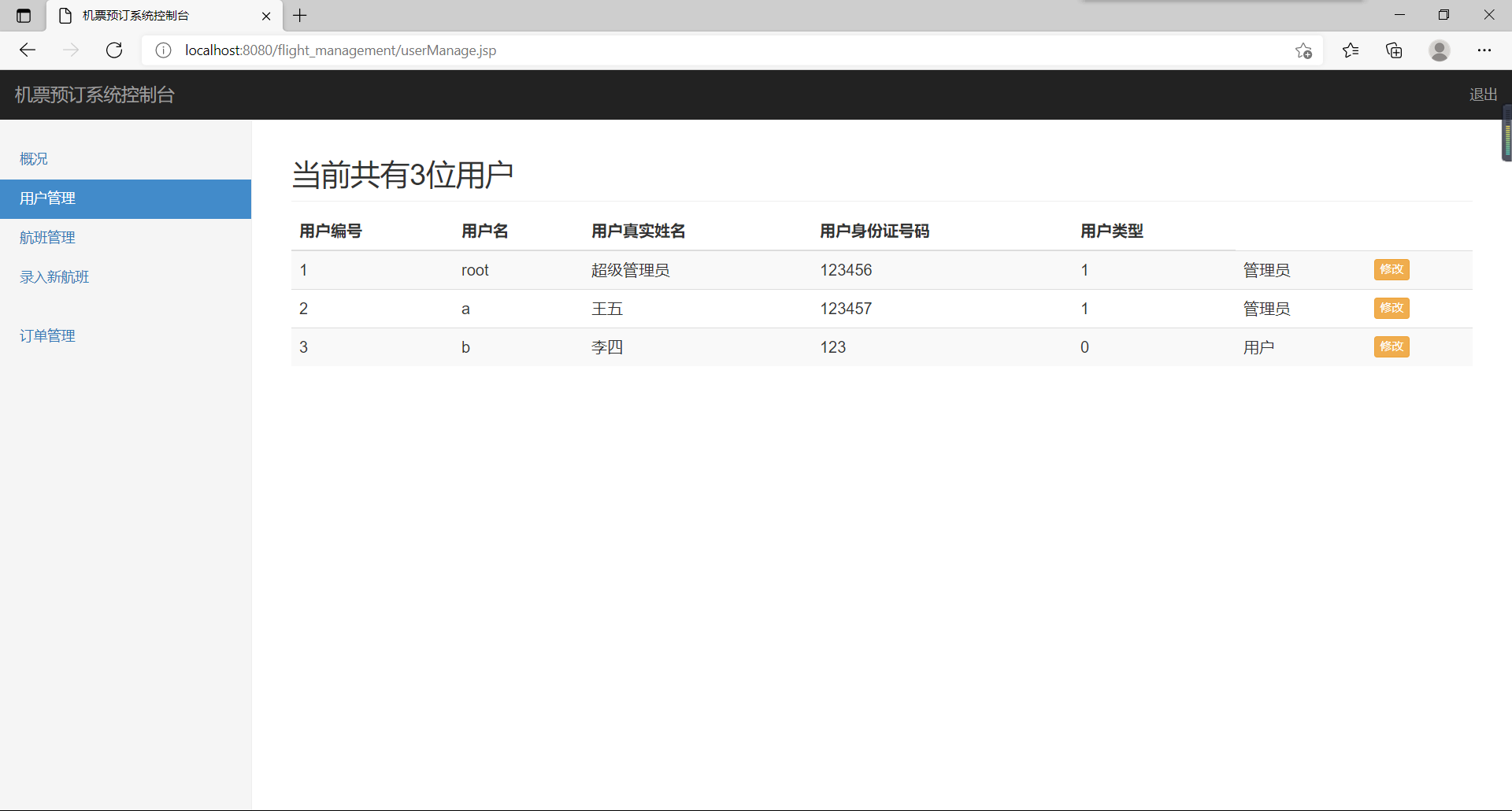


图 4.18 用户管理界面

1. 管理员修改用户信息界面

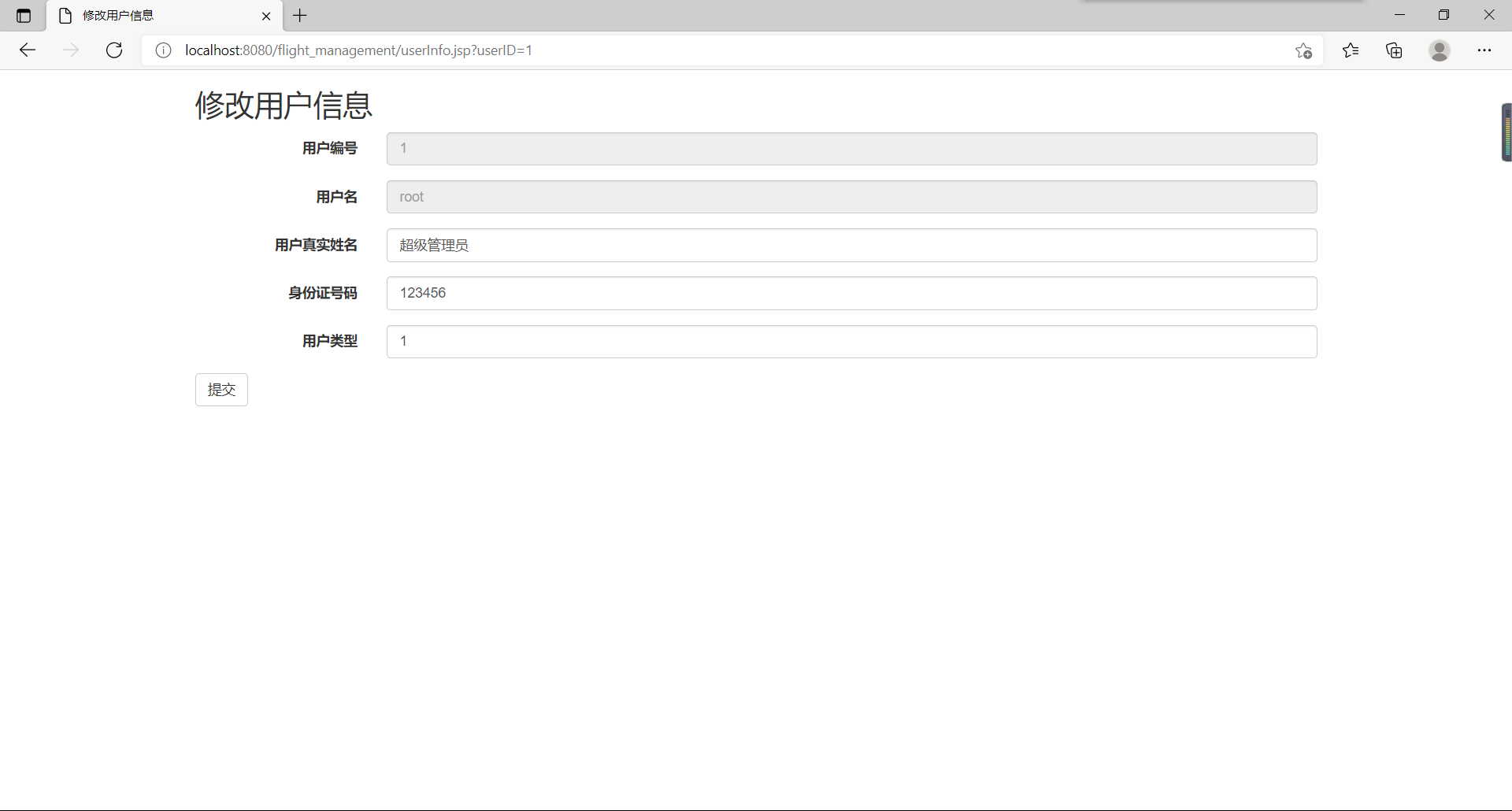


图 4.19 管理员修改用户信息界面

1. 管理员航班管理界面

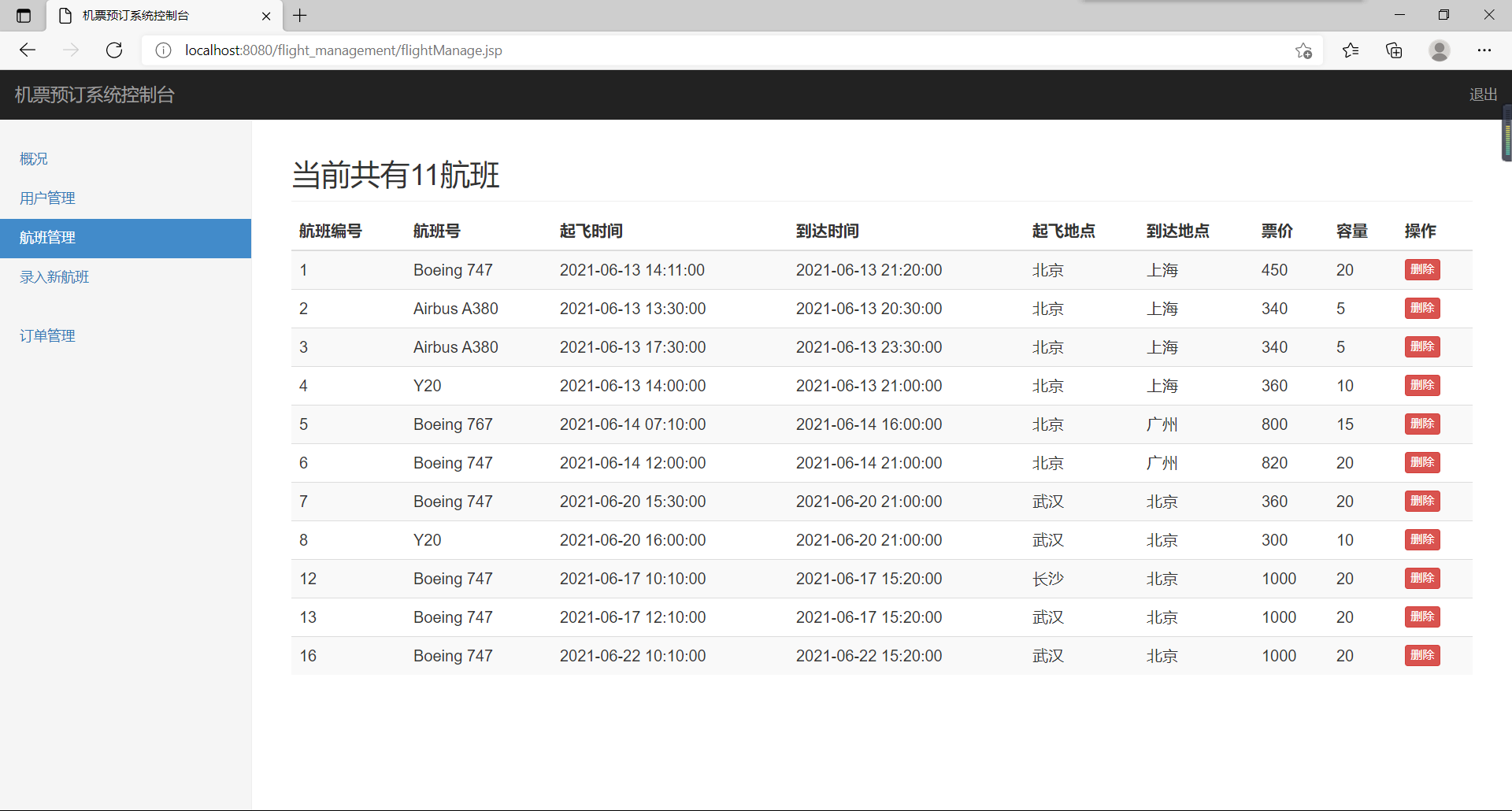


图 4.20 航班管理界面

1. 管理员录入新航班界面



图 4.21 管理员录入新航班界面

1. 管理员订单管理界面

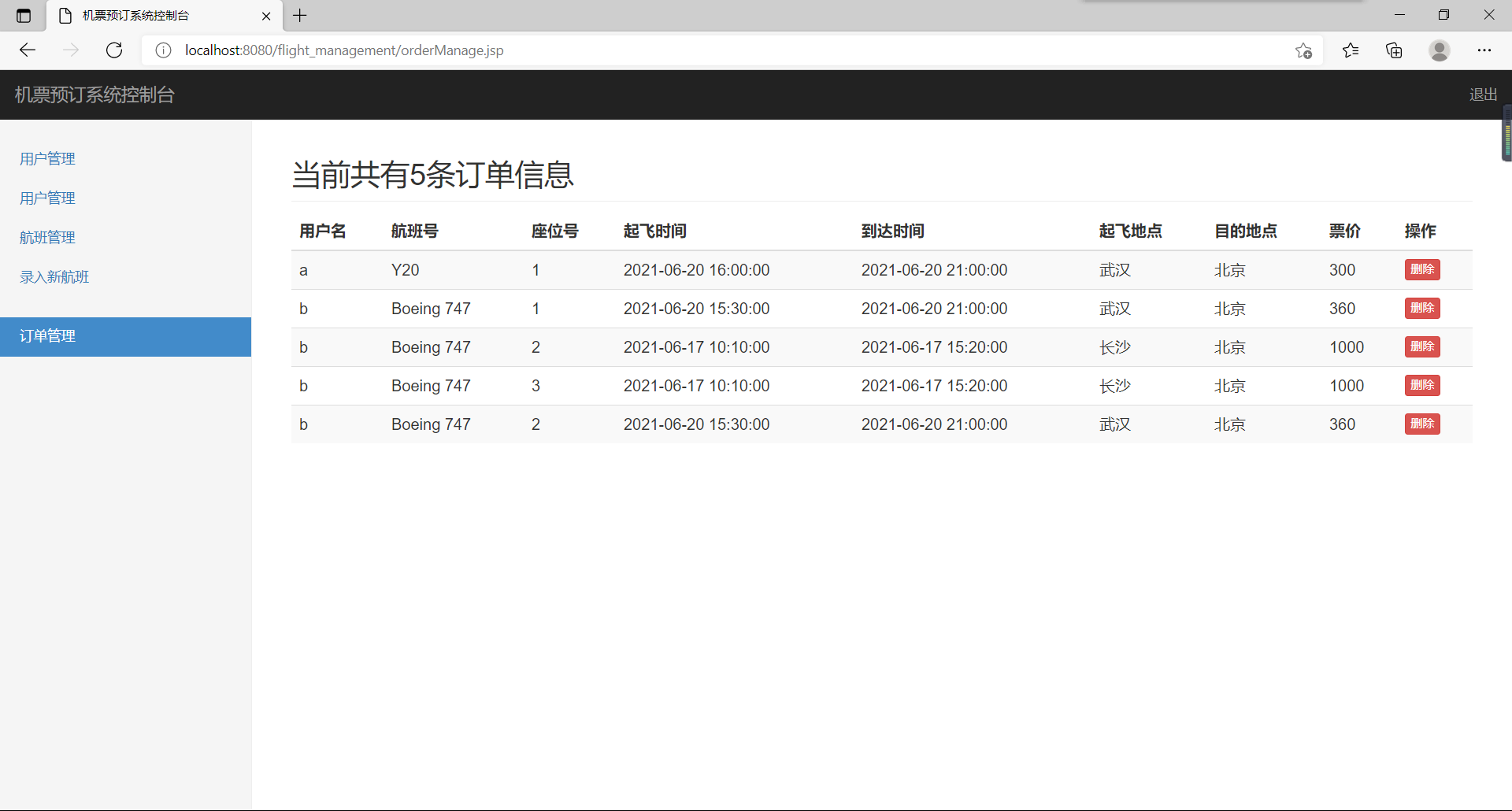


图 4.22 订单管理界面

## 4.6 系统测试

### 4.6.1 功能测试

1. 登录和注册的数据验证测试
2. 用户名或密码错误

当输入错误的用户名和密码时，显示如图4.23的错误提示：



图 4.23 输入错误的用户名或密码时的报错显示

1. 注册时，输入已经存在的用户名

在注册界面，注册的用户名和数据库中已经存在的用户名重名，点击注册会看到如图4.24的错误提示：



图 4.24 注册时输入的用户名重名报错

1. 当输入的信息不全时

当漏输入某项信息时，会得到如图4.25的提示：

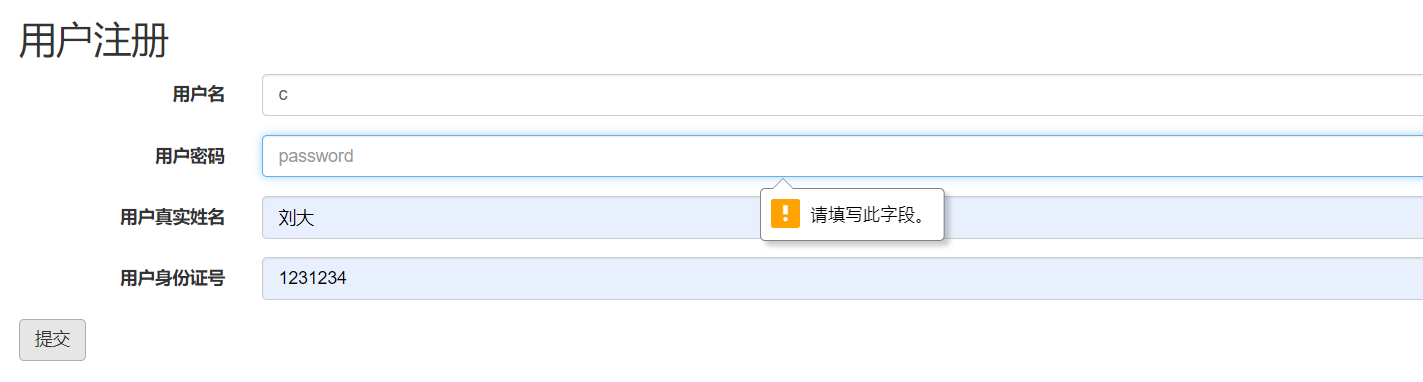


图 4.25 提交信息时，漏选提示

通过以上测试，验证系统在登录和注册时的数据验证是完备的

1. 用户功能测试
2. 用户查询功能

例如，用户查询从武汉到北京，2021-06-21出发的航班，查询前后的结果如图4.26和4.27所示：



图 4.26 航班查询



图 4.27 航班查询结果

1. 用户预订功能

接上，尝试将两趟航班都预订一次，并填写乘客信息，信息如图4.28所示，结果如图4.29所示。



图 4.28 添加乘客信息



图 4.29 查看订票结果

1. 用户退票功能

当用户点击Boeing747航班、座位编号为7的机票的退款按钮时，结果如图4.30所示：



图 4.30 用户退票

1. 用户取票

设定只有在航班出发的前一天，系统才会显示取票信息提供取票功能，由上图可知航班出发日期为6月20，而目前测试日期为6月21，所以取票界面不会显示任何信息。如图4.31所示。然后我们预订6月22的票，取票界面则会显示该机票的信息，提示可以取票和打印。



图 4.31 预订6月22日机票前的取票界面



图 4.32 预订6月22日机票后的取票界面

1. 打印功能

用户在订单界面打印订单，也可以当有取票通知单时打印取票通知单。打印效果分别如图4.17所示。

1. 用户修改个人信息

修改前用户的个人信息如图4.33：

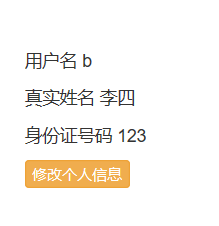


图 4.33 修改前的个人信息

在用户修改界面，可以对用户的用户名、用户密码、用户真实姓名以及身份证号码进行修改。

测试将真实姓名改为“刘大”，并修改密码从“bb”变为“b”，修改界面如图4.34所示。



图 4.34 填写的修改信息

修改后点击提交，自动跳转到用户主页，可以看到用户的信息已经修改。如图4.35所示

如果此时退出系统以密码“bb”重新登录“b”的账号会报出错误提示，如图4.36所示，改为密码“b”之后可成功登录。

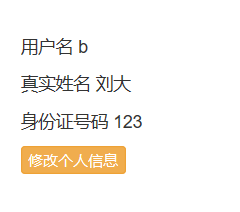


图 4.35 修改密码后，输入原密码显示错误 图 4.36 修改后的个人信息展示

综上，通过以上的测试，验证系统用户功能完善，达到了预期的目标。

1. 管理员功能测试
2. 管理员管理用户

点击用户管理，可以看到如图4.37的用户信息。



图 4.37 管理员用户界面显示

管理员可以对管理员以及用户的信息进行修改，但和用户修改个人信息相比仅仅只包括用户真实姓名、用户身份证号以及用户类型。测试将用户王五的用户类型由“管理员”修改为“用户”，并将用户真实姓名改为“李四”，将身份证号码改为“0000”。

修改信息如图4.38所示。修改后的结果如图4.39所示。



图 4.38 修改时填写的信息



图 4.39 修改后的用户信息

1. 管理员航班管理

在航班管理界面，管理员可以对所有航班进行删除操作。我们测试删除武汉到北京6月22号出发的16号航班，测试前后结果分别如图4.40和图4.41所示。



图 4.40 删除航班前的航班情况



图 4.41 删除编号为16的航班后的航班情况

1. 管理员录入新航班

尝试添加一个新的航班，添加的新航班信息如图4.42所示。



图 4.42 添加的航班的信息

添加后，用户航班管理界面出现新添加的航班。



图 4.43 添加航班后，该航班显示在航班管理界面中

1. 管理员订单管理

在此界面，管理员可以对订单进行删除，下面测试删除用户b的订单，测试前后结果如图4.44所示。





图 4.44 删除用户b的订单前后的情况

删除后，切换到b用户，原订单情况如图4.30所示，经过管理员的删除操作后，其界面如图4.45所示。



图 4.45 删除用户b的订单后，用户b的订票情况

综合以上几点，可以验证系统的管理员功能是完备的，达到了预期的要求。

## 4.7 系统设计与实现总结

本次实验设计并实现了简易版的“机票预订系统”，实现了用户订票、取票、退票等流程，并设置有管理员管理信息的部分，功能上较为完善。整个过程分为以下几个阶段。

1. 完成数据库设计

撰写中期报告，设计数据库ER图、关键事务——订票的数据流图和流程图，管理员以及用户的事务流程图以及数据字典等，并根据ER图设计数据库的表结构，并在navicat平台下建立相关的六个表和建立完整性约束，并完成部分数据的初始化方便测试使用。

1. 系统开发架构设计

大体设计开发系统的整个流程，规划需要写哪些模块以及文件。

1. 界面设计

利用Dreamweaver等工具半设计半编写html代码完成主要网页的界面设计，包括用户网页和管理员网页。

1. 前端开发

为系统最终需要实现的模块建立jsp文件，利用Dreamweaver最后自动生成网页代码的功能，简化提供给jsp文件使用。并利用HttpServlet等知识实现网页和后台数据的联系。

1. 完善后台

基于创建的数据库，实现DBMS的核心操作，完善整个系统的核心功能。

1. 测试、debug和功能完善

通过不断在网页上删除、加入、更新数据对数据库进行测试，不断寻找系统中存在的问题，解决并完善功能。最终得到现在较为完善的机票预订系统。

# 4 课程总结

本次实验，主要做了三个工作——练习SQL语句的使用、练习SQL有关软件以及利用SQL和所学的编程语言完成一个综合的系统设计。

我认为本次实验是目前为止对学生综合能力考察最高的一个实验。不仅涉及到了后端还有前端和数据库。之前的实验也大多是基于C或C++实现的，也是第一次使用Java语言和JSP来完成一个系统设计，需要学的知识很多，一开始了解到整个实验设计的时候感觉压力非常之大。

再参考了大量的资料、论文以及代码之后，我初步设计出整个系统的模型。但是后来上手后发现，要完整的实现我想设计出来的系统，凭借我目前贫瘠的语言知识实在是难。我一开始设定的机票还分了三种不同的类型，用户可以根据需求进行选票，然而前端网页的开发实在是有点困难，一方面是网页的设计，另外一方面就是我知道了这个网页最终模拟运行起来是怎样一个效果，设计网页时是需要添加哪些组件，但是却不知道具体该如何编写jsp文件的代码，能够让前端提的有效数据供后端使用。并且我一开始试图首先利用Java语言设计好所有的类和自认为需要用到的函数，结果在最后去设计前端时，发现很多需要的函数没有设计，不需要设计的反而设计了一大堆。主要原因就是我首先考虑后端，在编写函数时传递的参数就考虑的是类/对象，然而在实际操作过程中，从网页传递过来的参数往往只是一个航班编号或者一些字符串而已。而对象的获得可以通过传递过来的参数结合SQL查询就可以获得。所以最后好不容易写到前端发现需要大改特改，结合之前总想着要实现的完善一点再完善一点，结果后来完全不会写了，只好放弃，重新设计新的数据结构和新的开发流程。

虽然说本次实验难度确实大且综合性高，但我也因此学到了很多知识，包括网页设计、如何利用java开发一个网页项目以及数据库在一个系统中是怎样体现其功能等等。在系统开发过程中，虽然代码量繁琐但实际上很多都是有相似之处的，在学会了一些基本函数的使用之后，运用起来就很好上手了，尤其是在编写JSP文件和使用Java对数据库进行操作时。数据库方面除了学习如何设计表使得既尽快可能减少数据的冗余，又方便后端编程使用以外，对于数据库应用逻辑和事务之间的关系、ER图设计和数据库模式设计，完整性如何利用等的实践应用有了更深的理解。

最终实现的机票预订系统，仍然存在一定的不足。主要体现在安全性检查和异常处理，例如查询航班时没有设定日期的格式，如果用户输入错误的日期格式，只会不会显示航班信息，并不会弹出具体的错误警告等等。并且实现的系统和现实的设计存在一定的偏差和简化，例如存储的用户信息不够完善、并不是管理员就可以修改管理员的信息，应该存在超级管理员，只有超级管理员才可以修改管理员信息，而普通的管理员只可以对用户和航班进行管理等。

最终实验检查时，通过和老师的讨论，我发现数据库表的设计存在一定的数据冗余。主要体现在航班表的设计上。

如果针对要实现我的系统功能，理想的、好的数据库设计中，航班表只需要包括航班号、出发地点、目的地点、票价和容量。航班号是唯一的，它的编号就已经标识了出发地点和目的地点以及航空公司，而我设计的航班表中，航班号变成了飞机的型号，并使用了一个航班编号来区分不同的航班，并且不同时间出发和到达的航班算一个元组。这样就导致，如果应用到现实中，一年365天，航班总数会非常之大，导致航班表很大，而且这个大是很冗余的。因为如果是理想的航班表，航班号为M1401，我只需要在表中存储一个元组，但如果是我实际设计的表，这趟航班每天都会飞，出发时间和到达时间对同一个航班号来说每天都不同，导致表中会存在365条这样的元组。

如果之后有时间，希望能够进一步完善数据库的设计和系统功能的实现，做出一个尽可能贴近现实的订票系统，并对数据库的其他方面如：数据库安全授权、触发器、批量生成数据、连接池、多用户并发操作性能问题研究都能有所涉及和应用。

最后十分感谢老师的指导，让我对我的数据库设计有了更深的理解，也感谢助教，仔细检查我的系统设计。谢谢！