# 基于Pymysql的飞机订票系统设计

## 实验内容、步骤以及结果

### 开发环境与平台

环境：PyCharm anaconda

约束：pymysql tkinter 等python类库

语言：python 3.7

数据库：MySQL

### 用户需求分析

本实验为模拟飞机订票系统，是根据实验题目3——“车站售票管理系统”修改得来。在售票的基础上，将其面向于大众购票，仿照类似于携程网购买飞机票（购买飞机票软件等等）的流程，制作类似软件。实验在背景上与实际有所偏差，部分假设依照本人主观意愿设计，但均源于实际情况。本次实验设计的飞机订票系统，可以方便的满足乘客在出行中的购票、查询等需求，拥有广泛的应用领域和丰富的应用价值。

**此系统需要完成的主要目的有：**

1. 乘客查询、购买机票、修改个人信息

乘客用户修改用户信息。

乘客用户可以创建新用户。

乘客用户可以修改个人密码。

乘客用户可以查看两城市间航班信息。

乘客用户可以查看某一航班具体信息。

乘客用户可以购买某天某一班次机票。

乘客用户可查看已购票信息

1. 管理员信息管理

管理员可以修改乘客购票信息。

管理员可以删除乘客购票信息。

管理员可以创建航班班次信息。

管理员可以修改航班班次信息。

管理员可以删除航班班次信息。

管理员可以创建机场与城市信息。

管理员可以修改机场与城市信息。

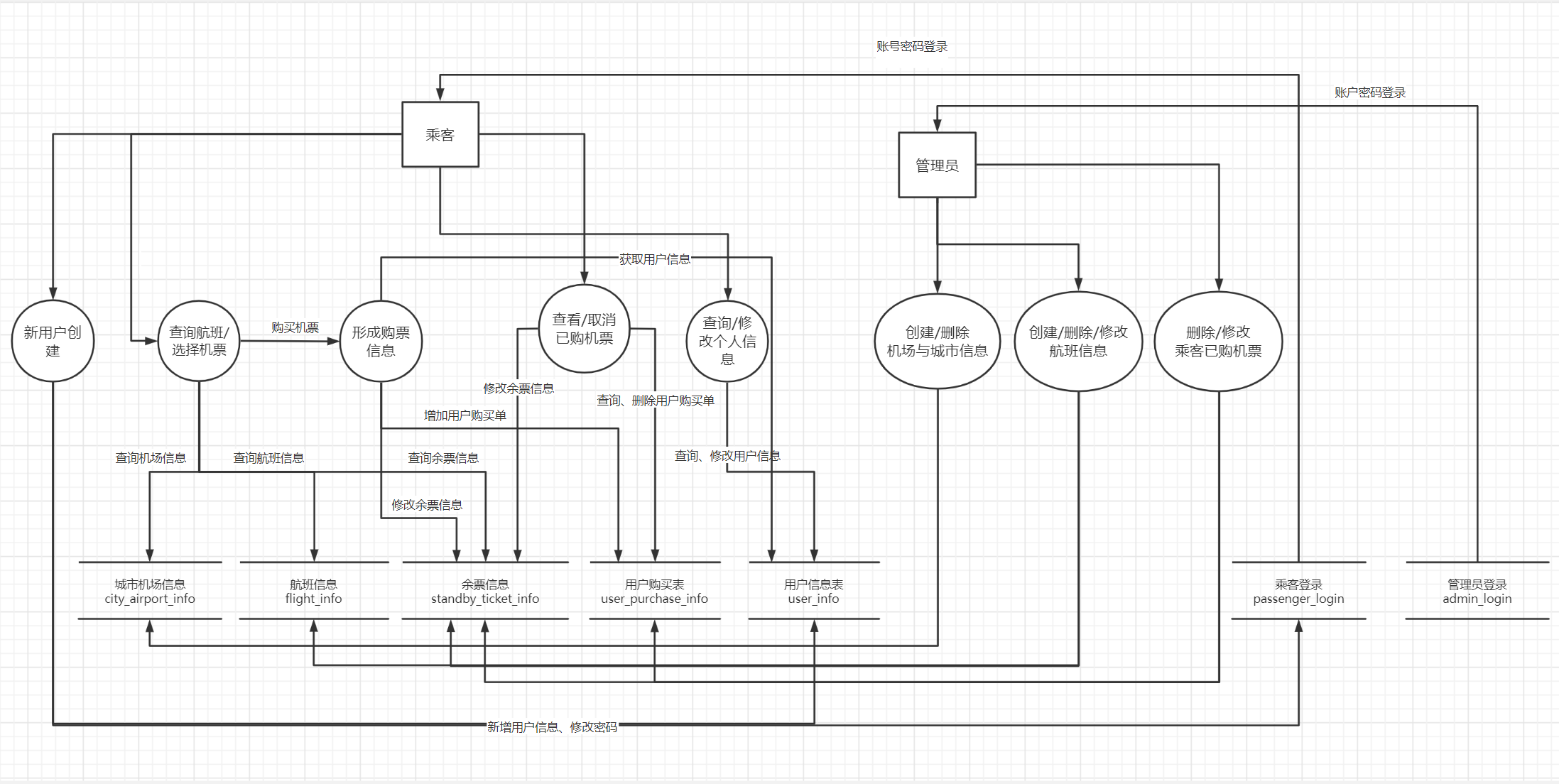
管理员可以删除机场与城市信息。

1. 软件功能查询

软件需要提供自身使用方式的说明，引导用户使用。

**数据流图：**

完整高清数据流图可查看报告中文件——数据流图.svg

****

**数据字典：**

1. 数据流的定义：
2. 乘客用户创建新用户、修改密码

数据流名：新用户创建

说明：当有乘客需要创建新用户并进行软件使用时，需要先创建一个新用户，将新用户的各种信息加入到用户信息表中，并且将用户的登录账号、密码加入到乘客登录表。如有需要也可从该处修改个人密码。

来源去向：乘客——用户信息表——乘客登录表

数据结构：passenger\_login user\_info

1. 乘客用户查询航班信息

数据流名：查询航班/选择机票

说明：当有乘客需要查询两座城市之间的航班信息，或者是查看某一具体航班信息时，需要从数据库中调用城市与机场信息、查询航班信息、从余票信息中查询航班的余票信息等。同时可以在此过程中为购买机票做选择

来源去向：乘客——航班信息——城市机场信息——余票信息

数据结构：city\_airport\_info、flight\_info、standby\_ticket\_info

1. 乘客用户购买某天某一班次机票

数据流名：形成购票信息

说明：当有乘客在选择好航班信息之后，购买机票时，需要获取用户信息，修改余票信息表中航班的余票，在用户购买表中新增购票信息。

来源去向：乘客——用户信息表——余票信息——用户购买表

数据结构： user\_info、standby\_ticket\_info、user\_purchase\_info

1. 乘客用户查询修改个人信息

数据流名：查询/修改个人信息

说明：当有乘客想要核实、修改个人信息时，需从用户信息表中查询、修改个人信息。

来源去向：乘客——用户信息表

数据结构： user\_info

1. 管理员创建/删除/修改航班班次信息

数据流名：创建/删除/修改航班信息

说明：当航班信息有所更替时，需要管理员创建、删除或修改对应的航班信息，管理员可在对应的航班信息表和余票信息中进行对应操作。

来源去向：管理员——航班信息表——余票信息

数据结构： standby\_ticket\_info、flight\_info

1. 管理员创建/删除/修改机场城市信息

数据流名：创建/删除机场与城市信息

说明：当一个城市中新建了一座机场、或旧的机场已经停止营运，需要管理员创建或删除机场信息，管理员可在对应的城市机场信息表进行对应操作。

来源去向：管理员——城市机场信息表

数据结构： city\_airport\_info

1. 管理员删除/修改乘客购票信息

数据流名：删除/修改乘客已购机票

说明：当乘客联系管理员改签机票、退票等操作时，需要管理员修改或删除乘客购票信息，管理员可在对应的用户购买表和余票信息进行对应操作。

来源去向：管理员——余票信息——用户购买表

数据结构：standby\_ticket\_info、user\_purchase\_info

1. 数据存储结构的定义：
2. admin\_login

数据存储名：管理员登录账户存储表

说明：管理员账户的登录信息存储

输入：无

输出：管理员登录页面

数据描述：管理员用户名 管理员密码

数据量：1-1000

存取频度：按照情况而定

存取方式：随机检索

1. passenger\_login

数据存储名：乘客登录账户存储表

说明：乘客账户的登录信息存储

输入：新用户创建

输出：乘客登录页面

数据描述：乘客用户名 乘客登录密码

数据量：1-100000+

存取频度：按照情况而定

存取方式：随机检索、更新

1. city\_airport\_info

数据存储名：城市机场信息

说明：记录了机场与城市之间的对应关系，包括城市名、机场名、机场代号等

输入：查询航班/选择机票、创建/删除机场与城市信息

输出：查询航班/选择机票

数据描述：城市名、机场名、机场代号和位置

数据量：1-100000

存取频度：按照情况而定，在节假日会显著上升

存取方式：随机检索、更新

1. flight\_info

数据存储名：航班信息

说明：记录了两个机场之间的航班信息，包括出发机场、到达机场、航班号等

输入：查询航班/选择机票、创建/删除/修改航班信息

输出：查询航班/选择机票

数据描述：航班号、航空公司、出发机场代号、到达机场代号、执飞机型、计划起飞时间、计划落地时间

数据量：1-100000

存取频度：按照情况而定，在节假日会显著上升

存取方式：随机检索、更新

1. standby\_ticket\_info

数据存储名：余票信息

说明：记录了航班某个日期、某个舱位等级的余票、价格等

输入：形成购票信息、查看/取消已购机票、删除/修改乘客已购机票

输出：查询航班/选择机票

数据描述：航班号、日期、F级舱位余票、F级舱位价格、C级舱位价格、Y级舱位余票、Y级舱位价格

数据量：1-100000

存取频度：按照情况而定，在节假日会显著上升

存取方式：随机检索、更新

1. user\_purchase\_info

数据存储名：用户购买表

说明：记录了用户购买某一日期的航班舱位等级等

输入：形成购票信息、查看/取消已购机票、删除/修改乘客已购机票

输出：查看/取消已购机票

数据描述：姓名、用户账号、用户身份证、航班号、日期、舱位、价格、手机号

数据量：1-100000

存取频度：按照情况而定，在节假日会显著上升

存取方式：随机检索、更新

1. user\_info

数据存储名：用户信息表

说明：记录了用户的相关等

输入：查询/修改个人信息、新用户创建

输出：查询/修改个人信息

数据描述：用户账号、姓名、用户身份证、手机号、性别

数据量：1-100000

存取频度：按照情况而定，在软甲运营、推广初期会显著上升

存取方式：随机检索、更新

1. 数据处理过程：
2. 处理过程名：新用户创建

说明：当有乘客需要创建新用户并进行软件使用时，需要先创建一个新用户，将新用户的各种信息加入到用户信息表中，并且将用户的登录账号、密码加入到乘客登录表。如有需要也可从该处修改个人密码。

输入：乘客passenger\_login

输出：passenger\_login user\_info

1. 处理过程名：查询航班/选择机票

说明：当有乘客需要查询两座城市之间的航班信息，或者是查看某一具体航班信息时，需要从数据库中调用城市与机场信息、查询航班信息、从余票信息中查询航班的余票信息等。同时可以在此过程中为购买机票做选择

输入：乘客passenger\_login、city\_airport\_info、flight\_info、standby\_ticket\_info

输出：暂无

1. 处理过程名：形成购票信息

说明：当有乘客在选择好航班信息之后，购买机票时，需要获取用户信息，修改余票信息表中航班的余票，在用户购买表中新增购票信息。

输入：查询航班/选择机票、user\_info、standby\_ticket\_info

输出：user\_purchase\_info

1. 处理过程名：查询/修改个人信息

说明：当有乘客想要核实、修改个人信息时，需从用户信息表中查询、修改个人信息。

来源去向：乘客——用户信息表

输入：乘客passenger\_login、user\_info

输出：user\_info

1. 处理过程名：创建/删除/修改航班信息

说明：当航班信息有所更替时，需要管理员创建、删除或修改对应的航班信息，管理员可在对应的航班信息表和余票信息中进行对应操作。如比较频繁的修改航班的价格信息等等。

输入：管理员登录admin\_login、standby\_ticket\_info、flight\_info

输出：standby\_ticket\_info、flight\_info

1. 处理过程名：创建/删除机场与城市信息

说明：当一个城市中新建了一座机场、或旧的机场已经停止营运，需要管理员创建或删除机场信息，管理员可在对应的城市机场信息表进行对应操作。

输入：管理员登录admin\_login

输出：city\_airport\_info

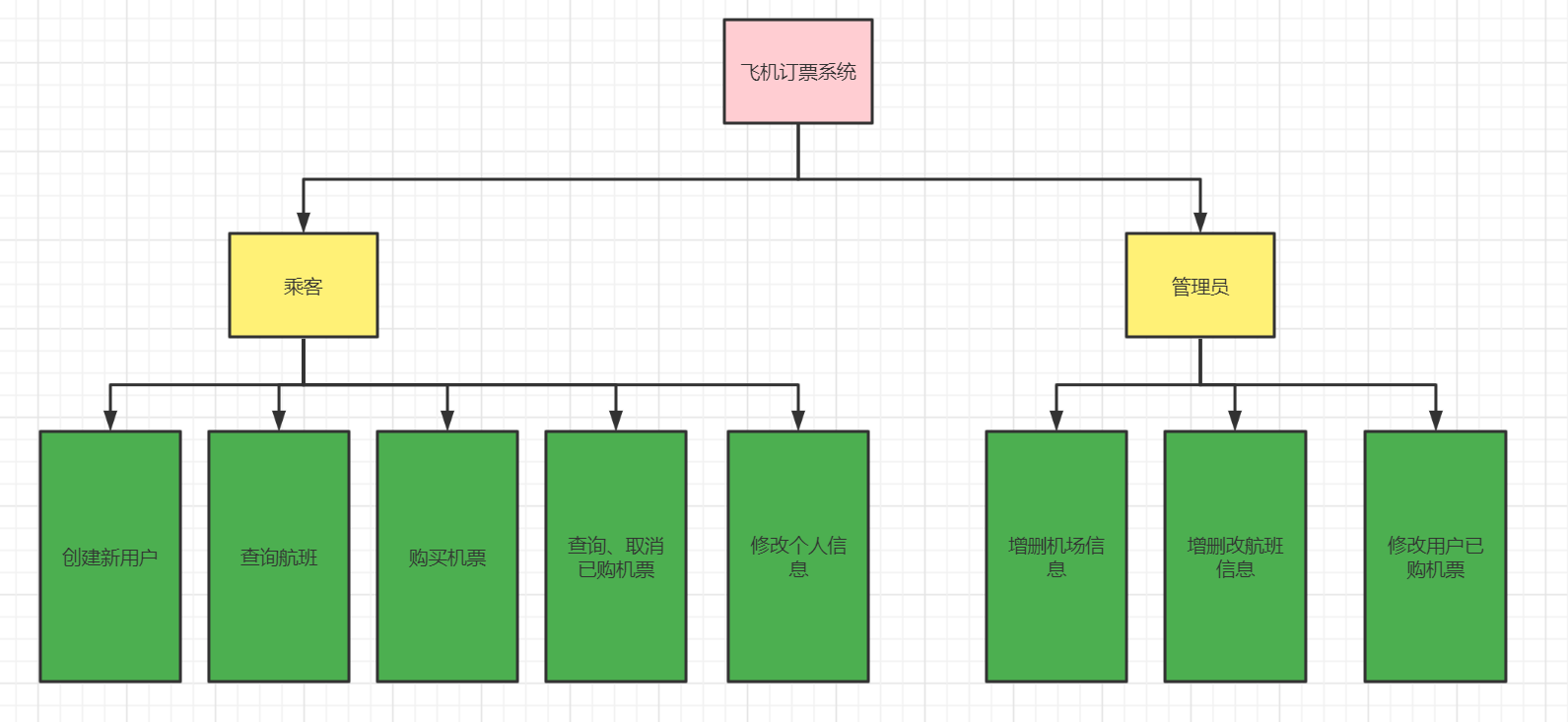
1. 处理过程名：删除/修改乘客已购机票

说明：当乘客联系管理员改签机票、退票等操作时，需要管理员修改或删除乘客购票信息，管理员可在对应的用户购买表和余票信息进行对应操作。

输入：管理员登录admin\_login

输出：standby\_ticket\_info、user\_purchase\_info

系统设计与模块划分：

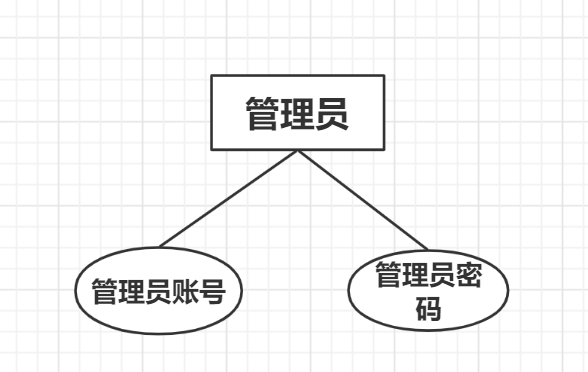


### 概念结构设计

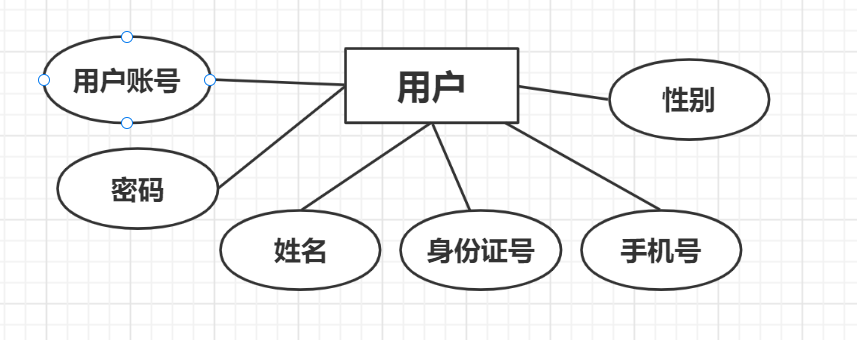
通过概要结构设计，设计系统的分ER图和整体ER图。

**分ER图：**

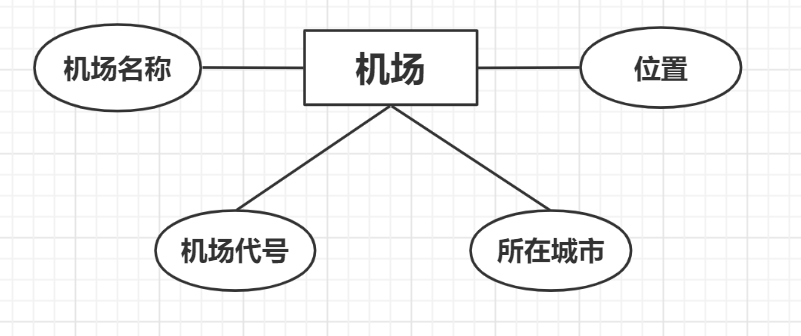
管理员：



乘客用户：



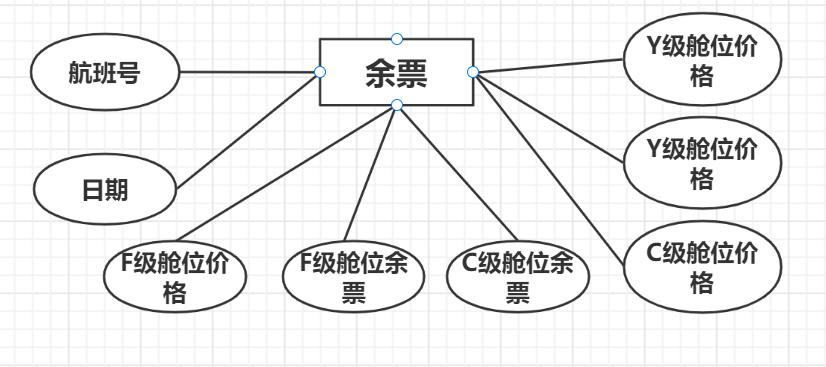
机场：



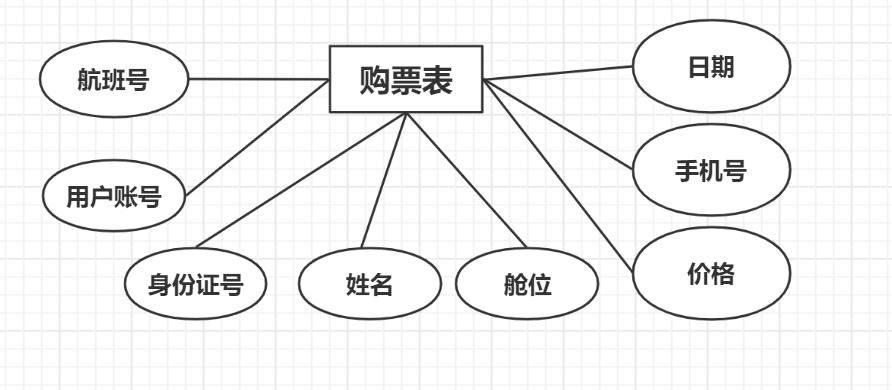
航班：



余票：

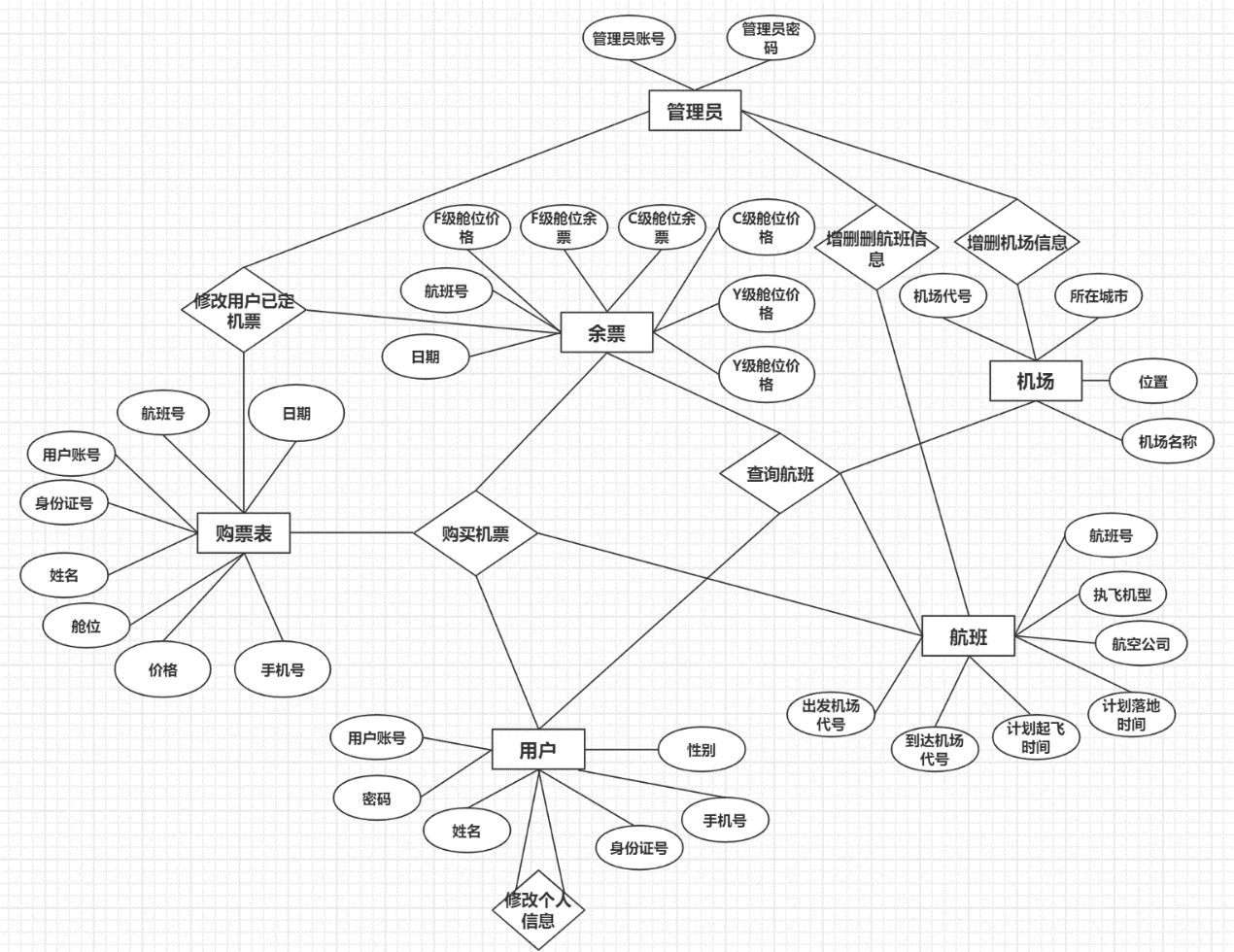


购买表：



**总ER图：**

完整高清的总ER图可查看报告中文件——er图.svg



### 逻辑结构设计

在ER图的基础上，进行**关系模式**的设计。

passenger\_login、city\_airport\_info、flight\_info、standby\_ticket\_info

用户登录（**用户ID**，用户登录密码）

F用户登录 = {用户ID→用户登录密码}

管理员登录（**管理员ID**，管理员登录密码）

F管理员登录 = {管理员ID→管理员登录密码}

用户信息（**用户账号**，姓名，性别，身份证，手机号）

F用户信息 = {用户账号→姓名，用户账号→性别，用户账号→身份证，用户账号→手机号}

机场（**机场代号**，机场名，所在城市，地理位置）

F机场 = {机场代号→机场名，机场代号→所在城市，机场代号→地理位置}

航班（**航班号**，出发机场代号，到达机场代号，计划起飞时间，计划到达时间，航空公司，执飞机型）

F航班 = {航班号→出发机场代号，航班号→到达机场代号，航班号→计划起飞时间，航班号→计划到达时间，航班号→航空公司，航班号→执飞机型}

余票（**航班号，日期**，C级舱位余票，C级舱位价格，F级舱位余票，F级舱位价格，Y级舱位余票，Y级舱位价格）

F航班 = {(航班号，日期) →C级舱位余票，(航班号，日期) →C级舱位价格，(航班号，日期) →Y级舱位余票，(航班号，日期) →Y级舱位价格，(航班号，日期) →F级舱位余票，(航班号，日期) →F级舱位价格}

购票（**航班号，日期，用户账号**，购买舱位，购买价格）

F航班 = {**(**航班号，日期，用户账号) →购买舱位，**(**航班号，日期，用户账号) →购买价格}

**此设计满足3NF。**

用户账号与用户ID的区分说明：用户ID是用来登录系统的时候使用的，而用户账号是在购买飞机票时所需要的用户信息。

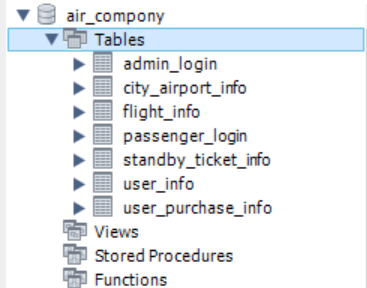
### 物理结构设计

索引设计与约束描述：

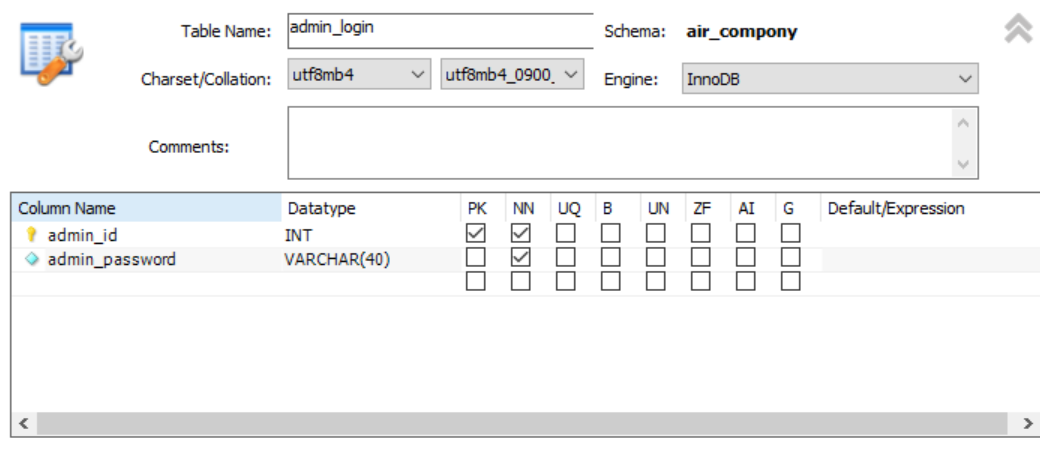
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 属性名 | 数据类型 | 长度 | 允许空 | 主码或索引 | 约束条件 | 备注 |
| 用户ID | char | 20 | × | √ |  |  |
| 用户登录密码 | Varchar | 40 | × |  |  |  |
| 管理员ID | Int | 20 | × | √ |  |  |
| 管理员登录密码 | Varchar | 40 | × |  |  |  |
| 用户账号 | Varchar | 20 | × | √ |  |  |
| 姓名 | Varchar | 45 | √ |  |  | 允许为空情况：  仅创建账号不填写个人信息 |
| 性别 | Varchar | 2 | √ |  |  |
| 年龄 | Int | 3 | √ |  |  |
| 身份证 | char | 18 | √ |  |  |
| 手机号 | char | 11 | √ |  |  |
| 机场代号 | char | 3 | × | √ |  |  |
| 机场名 | Varchar | 8 | × |  |  |  |
| 所在城市 | Varchar | 2 | × |  |  |  |
| 地理位置 | Varchar | 20 | × |  |  |  |
| 航班号 | Varchar | 6 | × | √ |  |  |
| 出发机场代号 | char | 3 | × |  |  |  |
| 到达机场代号 | char | 3 | × |  |  |  |
| 计划起飞时间 | Time | 5 | × |  |  |  |
| 计划到达时间 | Time | 5 | × |  |  |  |
| 航空公司 | Varchar | 8 | × |  |  |  |
| 执飞机型 | Varchar | 8 | × |  |  |  |
| 日期 | Date | 8 | × | √ |  |  |
| C级舱位余票 | Int | 3 | √ |  |  | 允许为空情况：  某些航班没有头等舱与商务仓 |
| C级舱位价格 | Int | 5 | √ |  |  |
| F级舱位余票 | Int | 3 | √ |  |  |
| F级舱位价格 | Int | 4 | √ |  |  |
| Y级舱位余票 | Int | 3 | × |  |  |  |
| Y级舱位价格 | Int | 4 | × |  |  |  |
| 购买舱位 | Varchar | 5 | × |  |  |  |
| 购买价格 | Int | 5 | × |  |  |  |

物理结构设计主要结构为在数据库air\_ compony下的数据库表

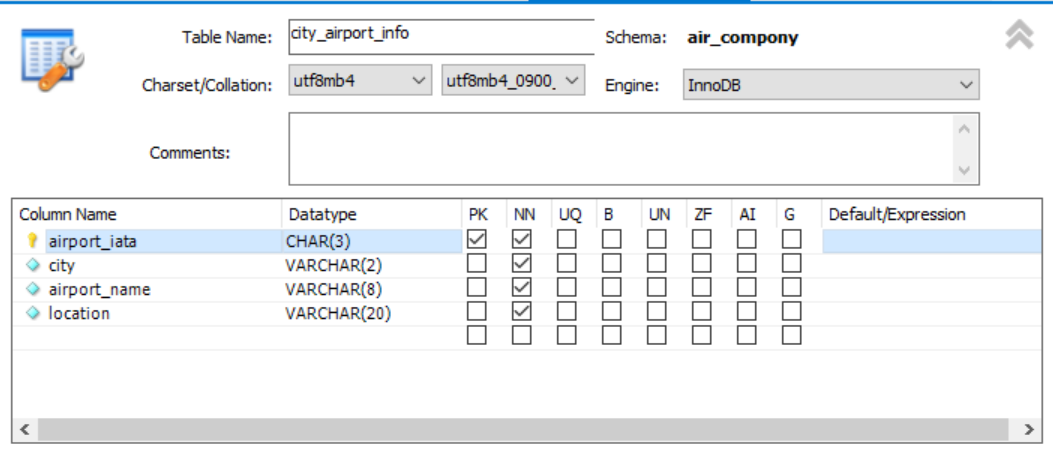
目录结构如下：



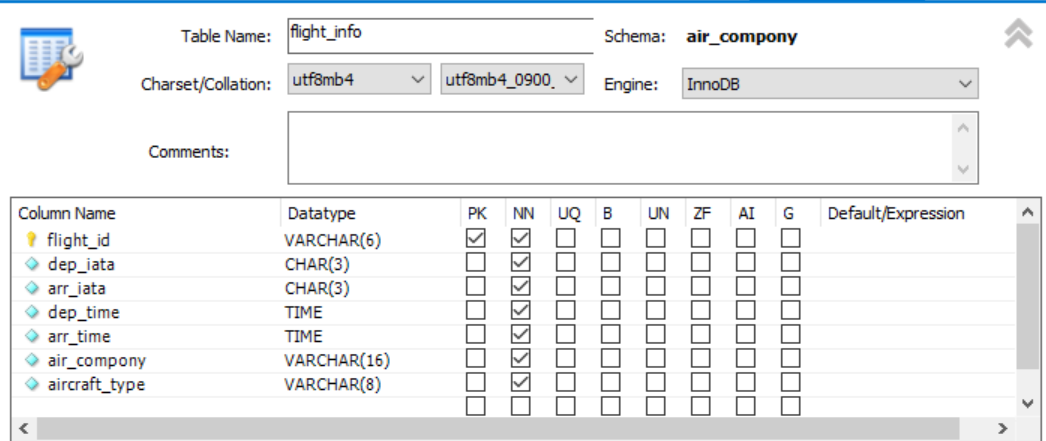
在admin\_login表中，存储结构为：



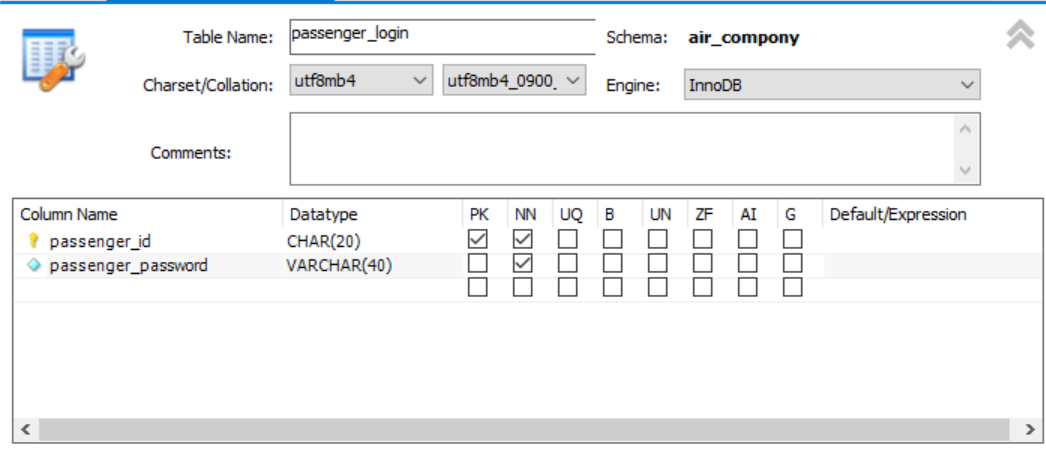
在city\_airport\_info表中，存储结构为：



在flight\_info表中，存储结构为：



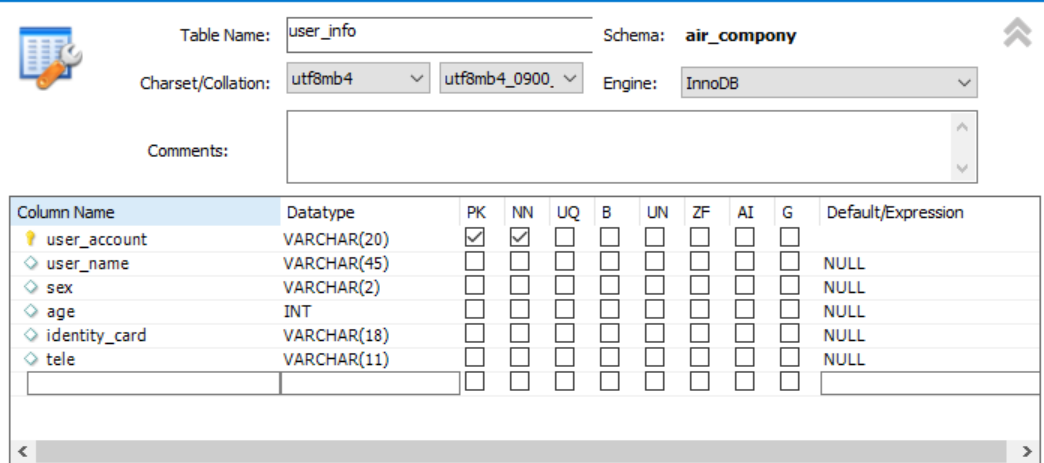
在passenger\_login表中，存储结构为：



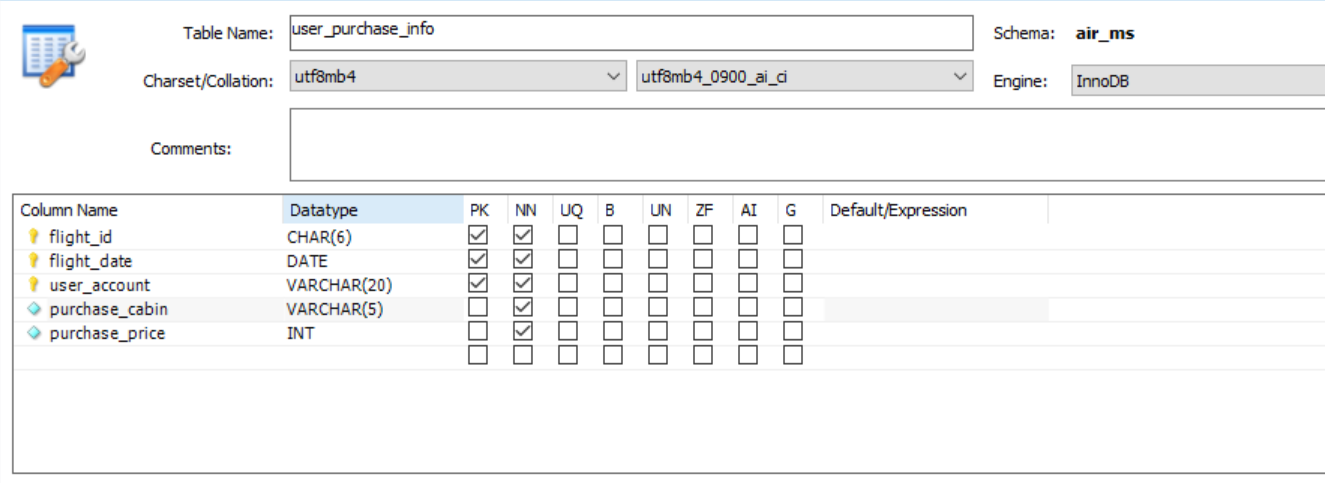
在standby\_ticket\_info表中，存储结构为：



在user\_info表中，存储结构为：

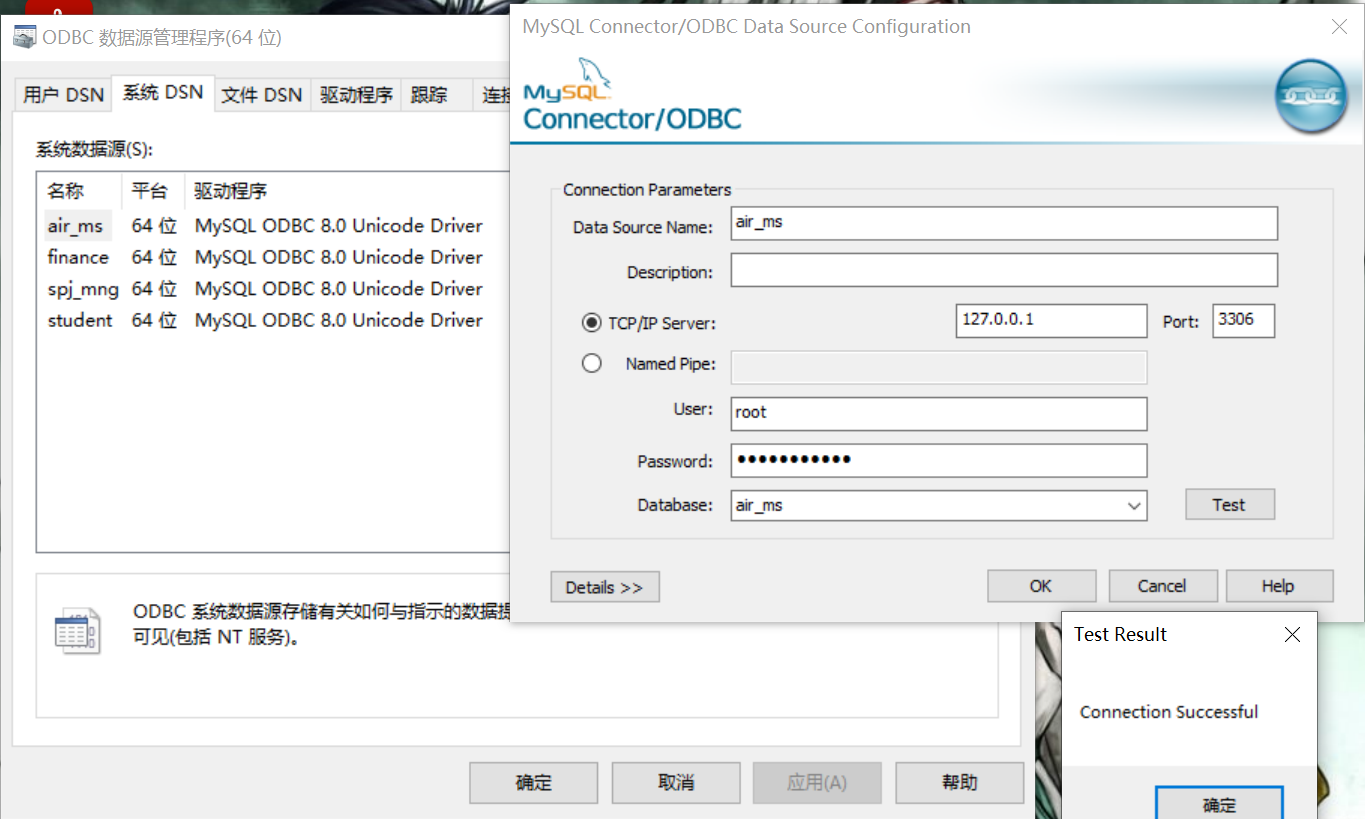


在user\_purchase\_info表中，存储结构为：



### 数据库实施与维护

**完成以上数据库的物理设计，并初步创建一些简单的数据之后，创建一个air\_ms的ODBC数据源：**



**Pymysql是基于python的一种可以直接操作MySQL的API，因为其使用方便，所以应用前景广泛，具有很高的使用价值，因此被广泛的应用于对MySQL的开发中。**

**本次实验使用pymysql完成。**

1. 安装Pymysql模块：

直接使用pip源可以直接安装pymysql。

pip3 install pymysql

因为本次实验使用的是conda环境，所以很容易对包进行管理。

1. Pymysql的使用

Pymysql需要创建链接才能使用，首先登陆数据库，然后在数据库中创建一个执行的游标，就可以通过sql语句对数据库进行操作了。

# 导入pymysql模块

import pymysql

# 连接database

conn = pymysql.connect(

host= “你的数据库地址”,

user= “用户名”, password= “密码”,

database= “数据库名”,

charset= “utf8”)

# 得到一个可以执行SQL语句的游标对象

cursor = conn.cursor() # 执行完毕返回的结果集默认以元组显示

# 得到一个可以执行SQL语句并且将结果作为字典返回的游标

#cursor = conn.cursor(cursor=pymysql.cursors.DictCursor)

# 定义要执行的SQL语句

sql = """ """

# 执行SQL语句

cursor.execute(sql)

results = cursor.fetchall()

for row in results:

# 关闭游标对象

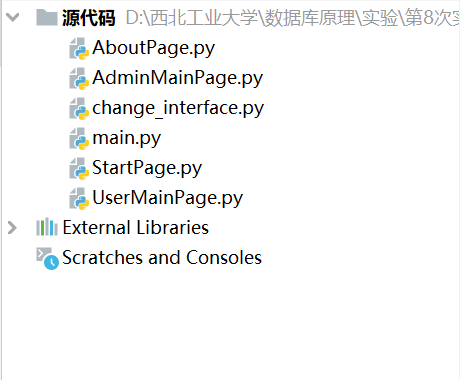
cursor.close()

# 关闭数据库连接

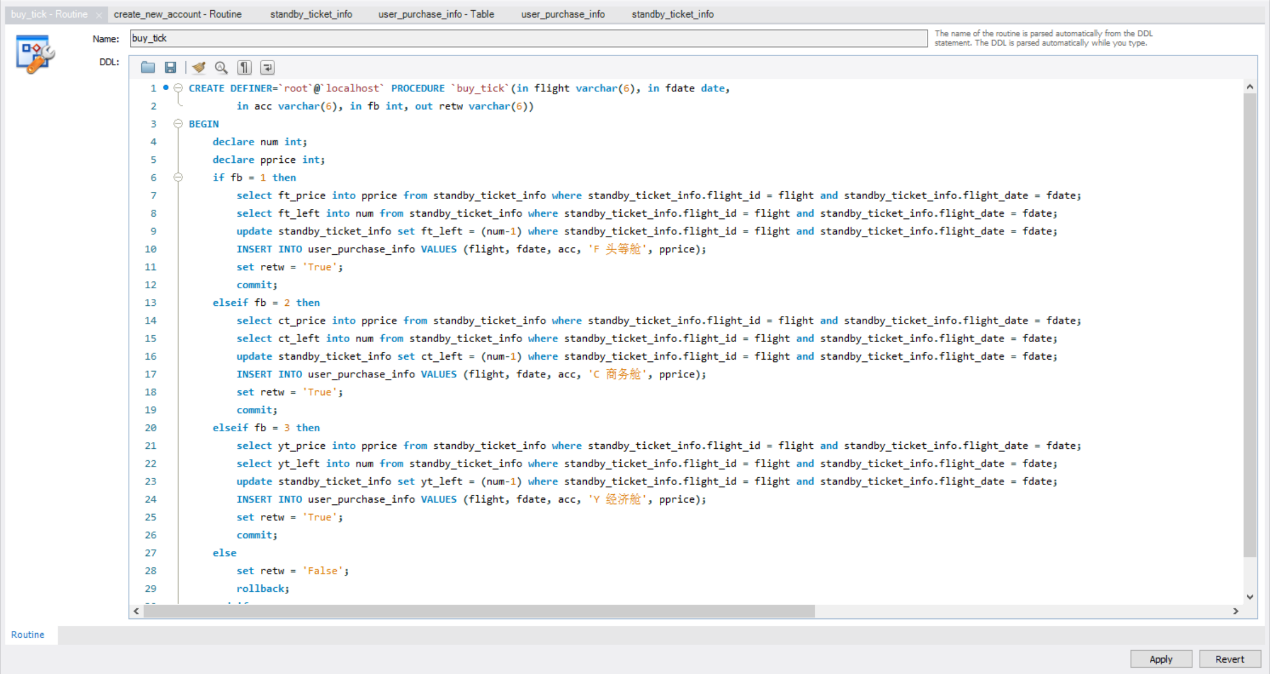
conn.close()

**同时，我们使用Tkinter，作为开发python的GUI库，它是Python的一个标准包，并且已经在conda中内置，因此，我们需要做的操作是在其中创建一个飞机订票系统的图形用户界面。**

**代码部分及代码注释已经在大作业中附有，具体可以查阅程序代码。**



实验过程中用到的购买过程体如下：



CREATE DEFINER=`root`@`localhost` PROCEDURE `buy\_tick`(in flight varchar(6), in fdate date,

in acc varchar(6), in fb int, out retw varchar(6))

BEGIN

declare num int;

declare pprice int;

declare pprice2 int;

declare pprice3 int;

if fb = 1 then

select ft\_price into pprice from standby\_ticket\_info where standby\_ticket\_info.flight\_id = flight and standby\_ticket\_info.flight\_date = fdate;

select ft\_left into num from standby\_ticket\_info where standby\_ticket\_info.flight\_id = flight and standby\_ticket\_info.flight\_date = fdate;

update standby\_ticket\_info set ft\_left = (num-1) where standby\_ticket\_info.flight\_id = flight and standby\_ticket\_info.flight\_date = fdate;

INSERT INTO user\_purchase\_info VALUES (flight, fdate, acc, 'F 头等舱', pprice);

set retw = 'True';

commit;

elseif fb = 2 then

select ct\_price into pprice2 from standby\_ticket\_info where standby\_ticket\_info.flight\_id = flight and standby\_ticket\_info.flight\_date = fdate;

select ct\_left into num from standby\_ticket\_info where standby\_ticket\_info.flight\_id = flight and standby\_ticket\_info.flight\_date = fdate;

update standby\_ticket\_info set ct\_left = (num-1) where standby\_ticket\_info.flight\_id = flight and standby\_ticket\_info.flight\_date = fdate;

INSERT INTO user\_purchase\_info VALUES (flight, fdate, acc, 'C 商务舱', pprice2);

set retw = 'True';

commit;

elseif fb = 3 then

select yt\_price into pprice3 from standby\_ticket\_info where standby\_ticket\_info.flight\_id = flight and standby\_ticket\_info.flight\_date = fdate;

select yt\_left into num from standby\_ticket\_info where standby\_ticket\_info.flight\_id = flight and standby\_ticket\_info.flight\_date = fdate;

update standby\_ticket\_info set yt\_left = (num-1) where standby\_ticket\_info.flight\_id = flight and standby\_ticket\_info.flight\_date = fdate;

INSERT INTO user\_purchase\_info VALUES (flight, fdate, acc, 'Y 经济舱', pprice3);

set retw = 'True';

commit;

else

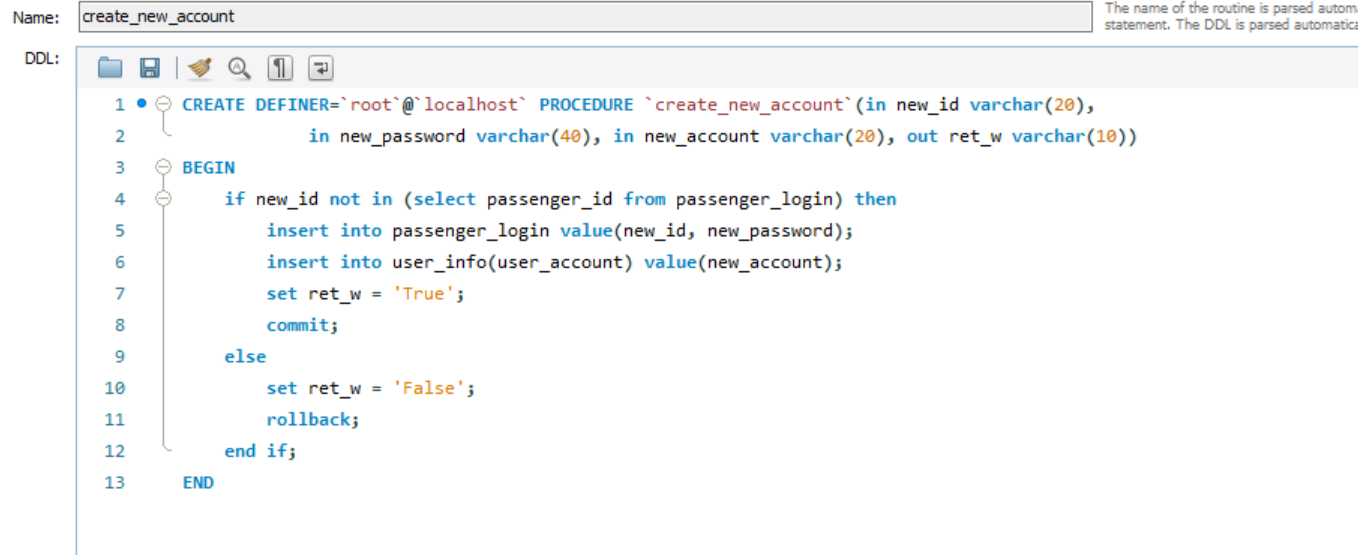
set retw = 'False';

rollback;

end if;

END

创建新用户过程体如下：



CREATE DEFINER=`root`@`localhost` PROCEDURE `create\_new\_account`(in new\_id varchar(20),

in new\_password varchar(40), in new\_account varchar(20), out ret\_w varchar(10))

BEGIN

if new\_id not in (select passenger\_id from passenger\_login) then

insert into passenger\_login value(new\_id, new\_password);

insert into user\_info(user\_account) value(new\_account);

set ret\_w = 'True';

commit;

else

set ret\_w = 'False';

rollback;

end if;

END

### 程序运行过程

首先，我们可以运行开启并运行main.py程序，开启主界面：

主界面上显示为飞机订票系统，由四个功能按钮，分别为：管理员登录，用户登录，使用开发说明和退出四个操作。



我们首先使用管理员身份进行登陆，点击管理员登录按钮，弹出管理员登录界面：



输入管理员用户名和密码之后，即可进行登录操作：

目前设置的管理员登录用户名为：20220204，密码为：123456

点击登录按钮后，提示正在登录管理员界面，点击确定，即可进入管理员界面。



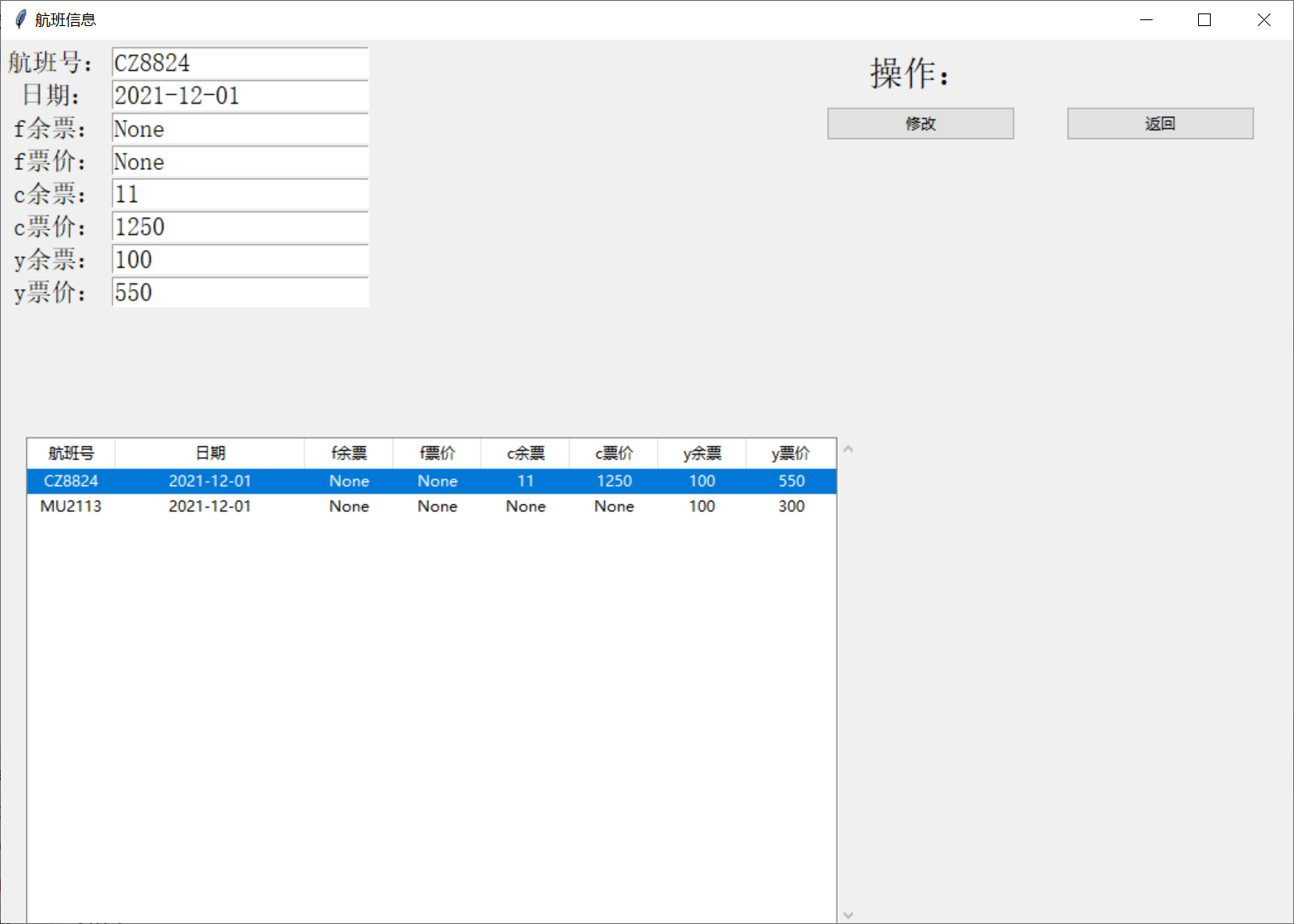


随后进入管理员操作界面，这里包含了管理员应有的三个功能：

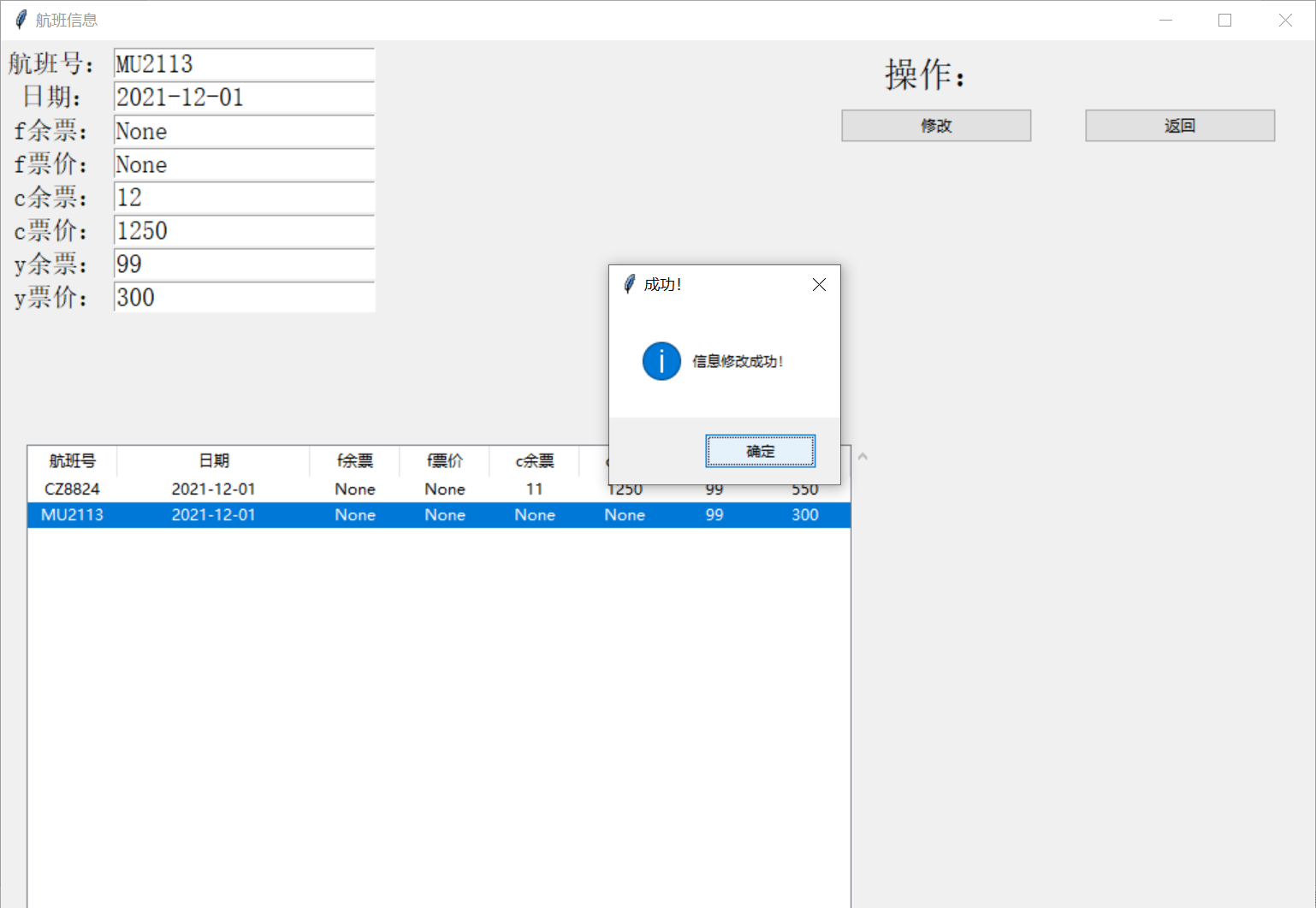


首先点击第一个功能：航班信息，这里用来航班信息的修改：

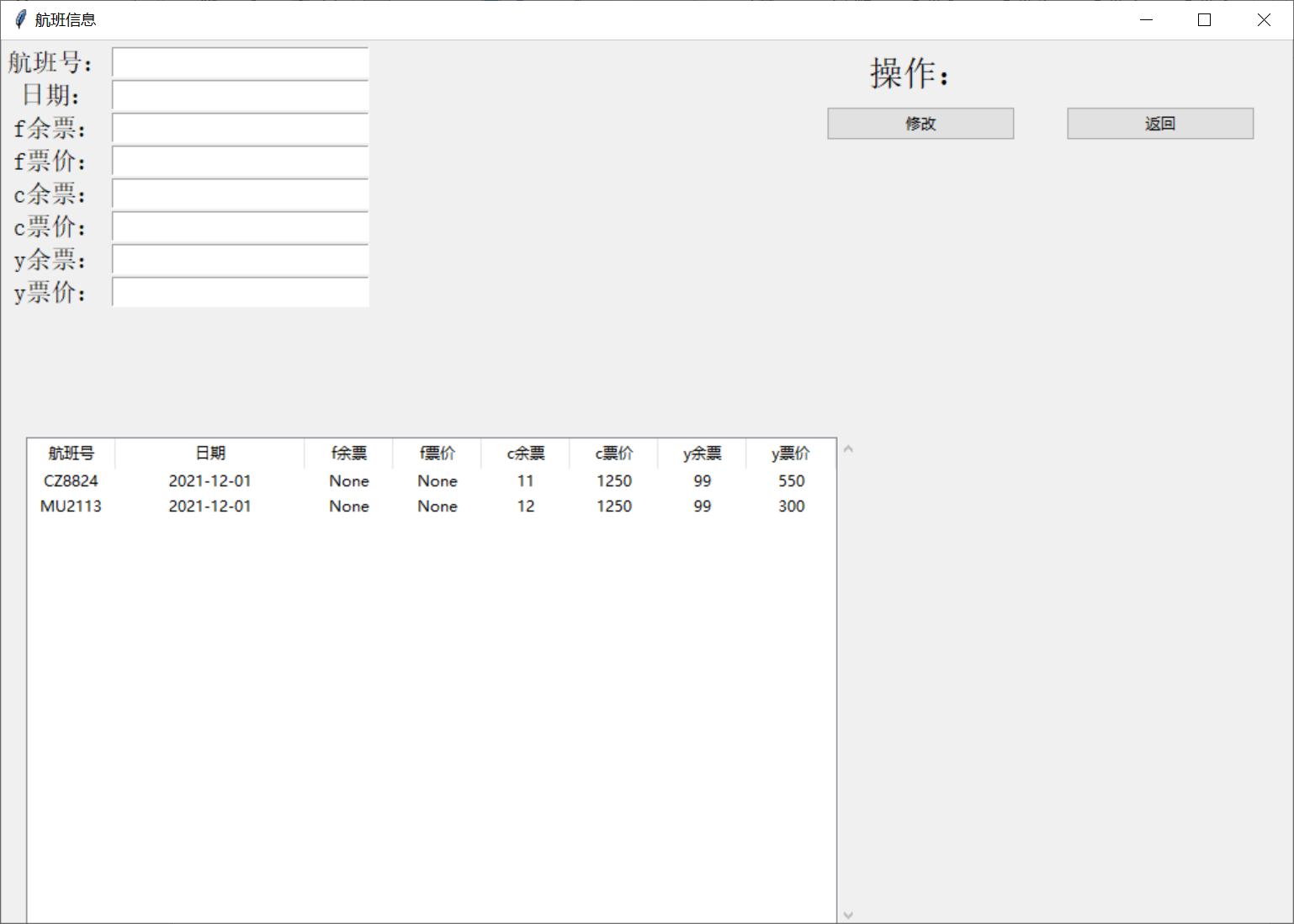
由三部分组成，分别是录入信息，操作和候选信息组成



当我们需要更新信息时，首先选择对应日期和对应航班号的信息，点击其需要更新的信息，这是发现其在信息框中显示，我们在信息框中完成对信息的更新操作，然后点击修改操作即可

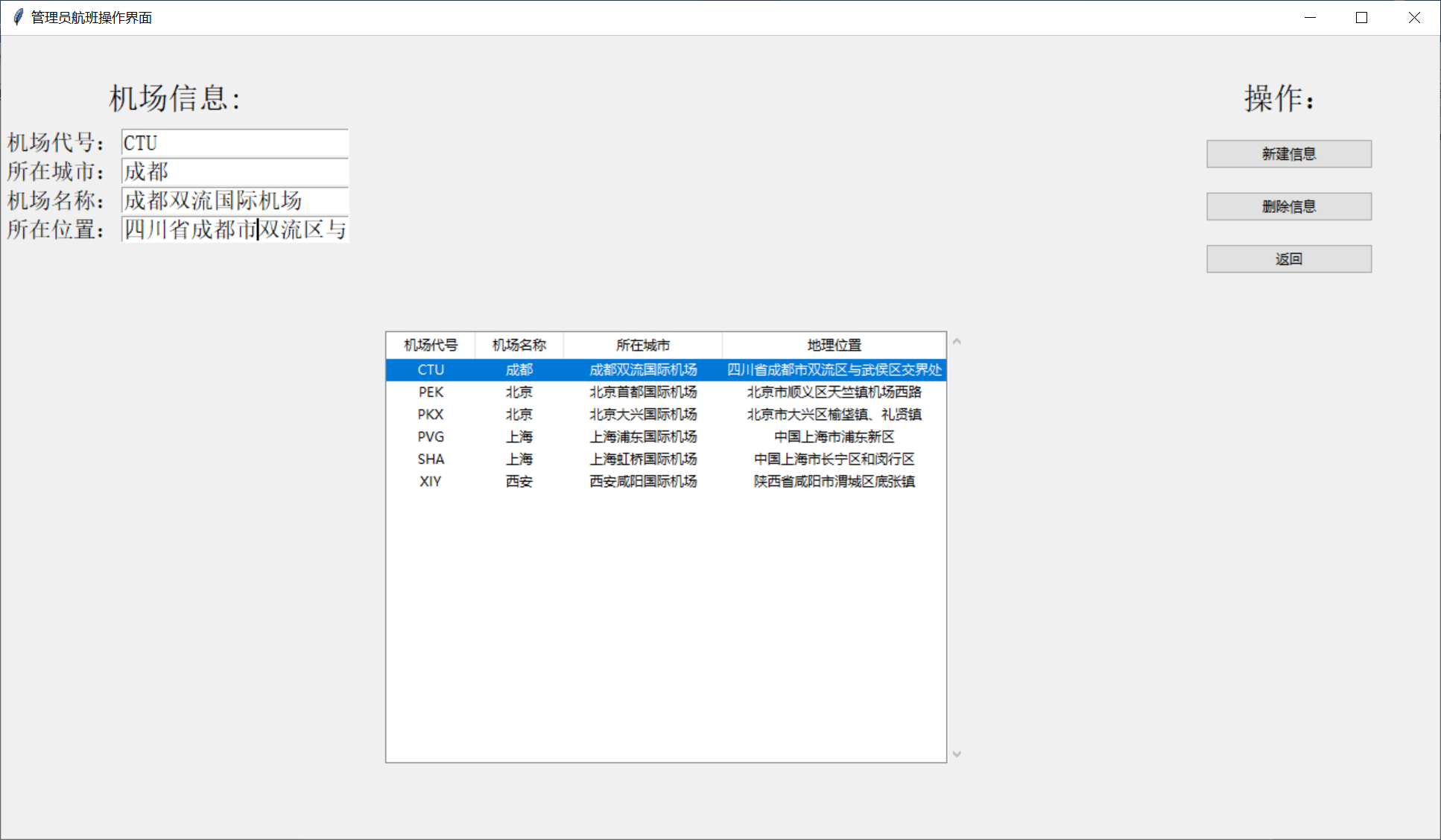


修改后再次查看如下：

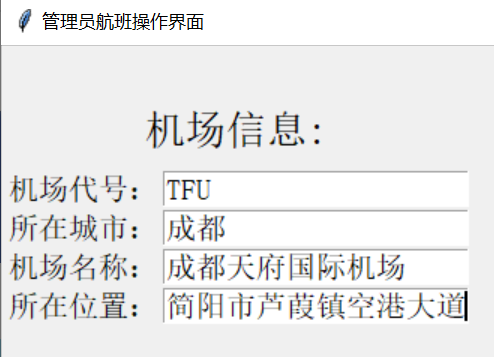


可点击返回，退回到之前的管理员主界面。

接下来是机场信息的设置，由三部分组成，分别是录入信息，操作和候选信息组成。

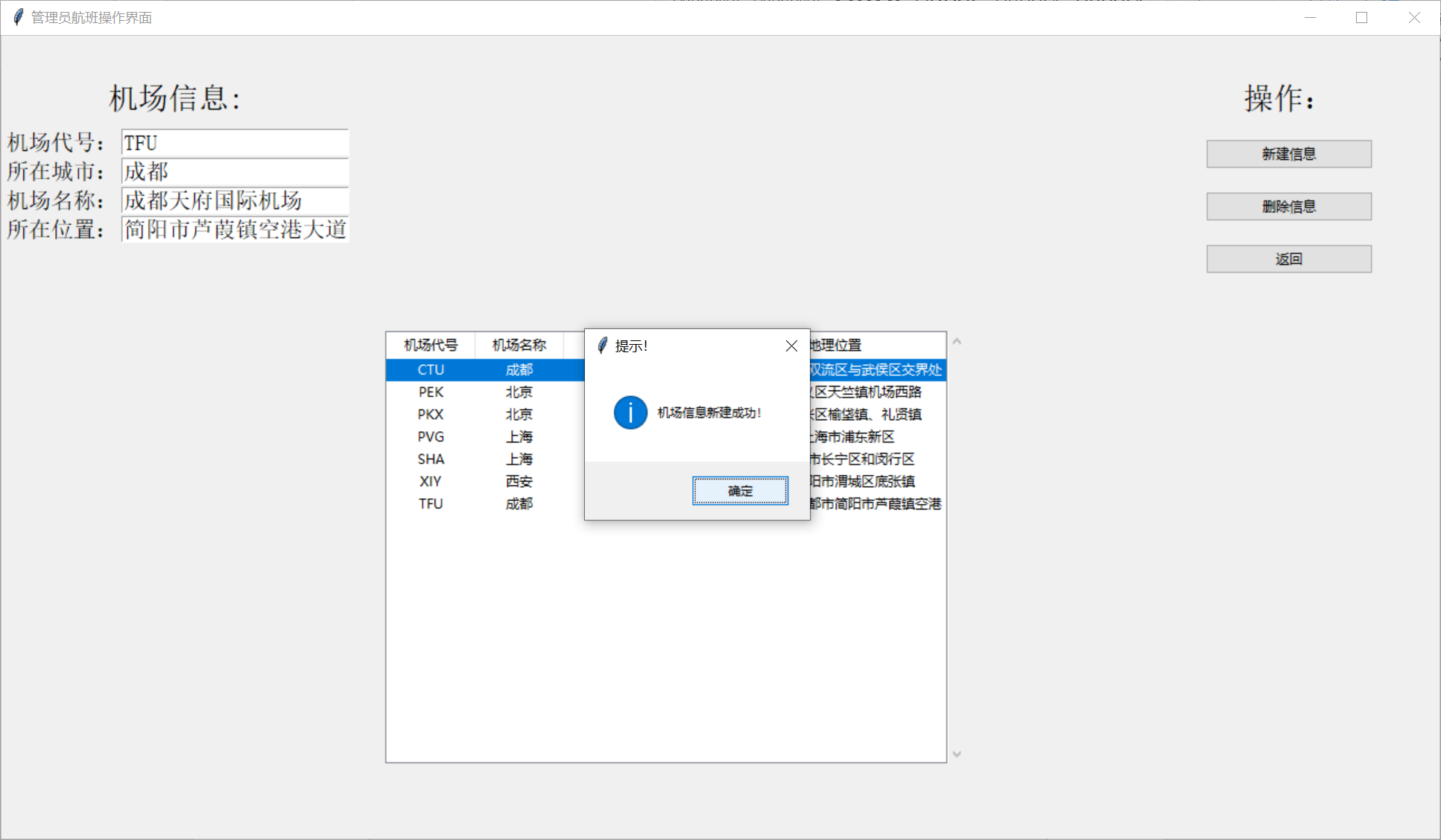


当我们需要插入信息时，首先我们应该在信息框中输入需要插入的值。

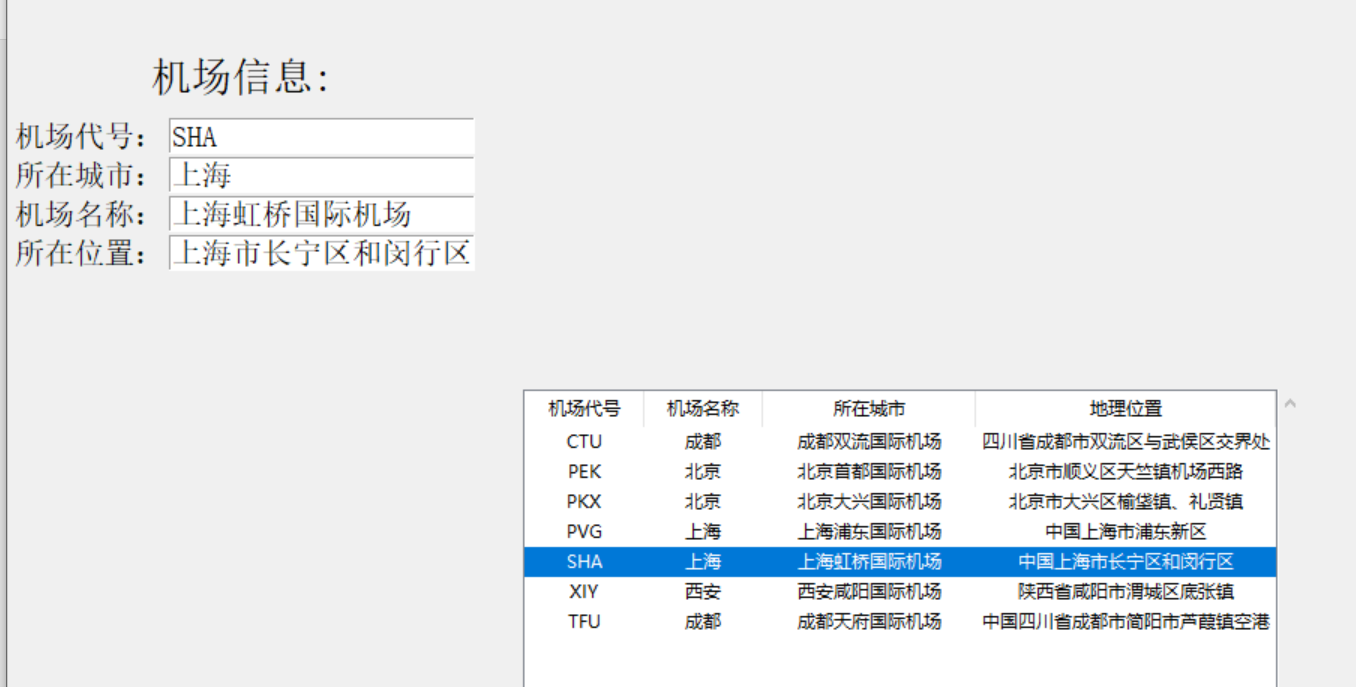


然后点击新建信息，出现提示：

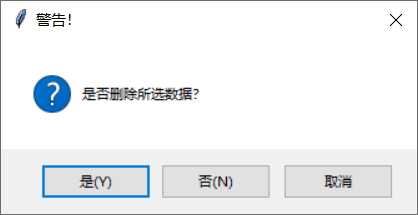
观察到插入成功，并且实时更新



当我们需要删除用户时，同样的，首先选择需要被删除的机场。

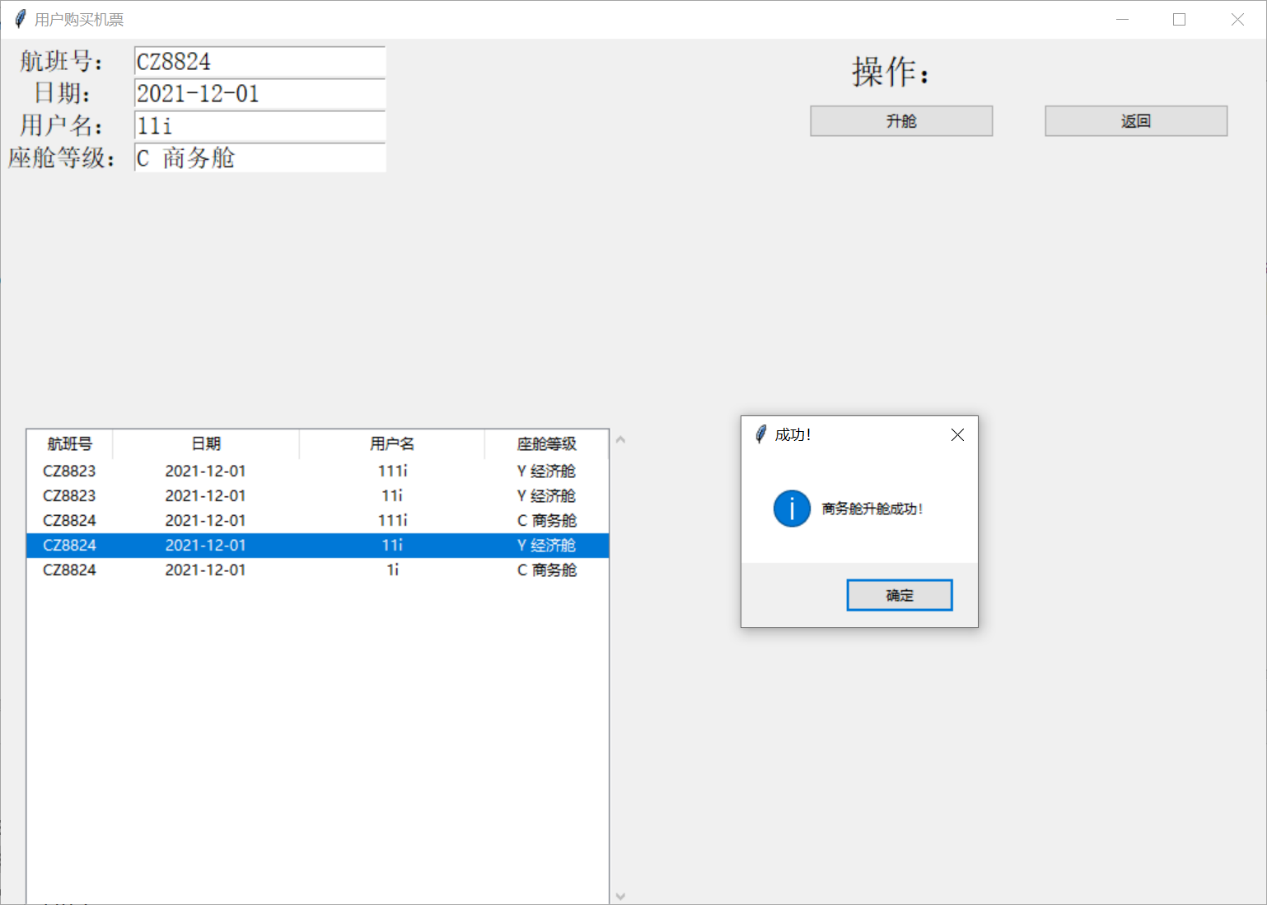


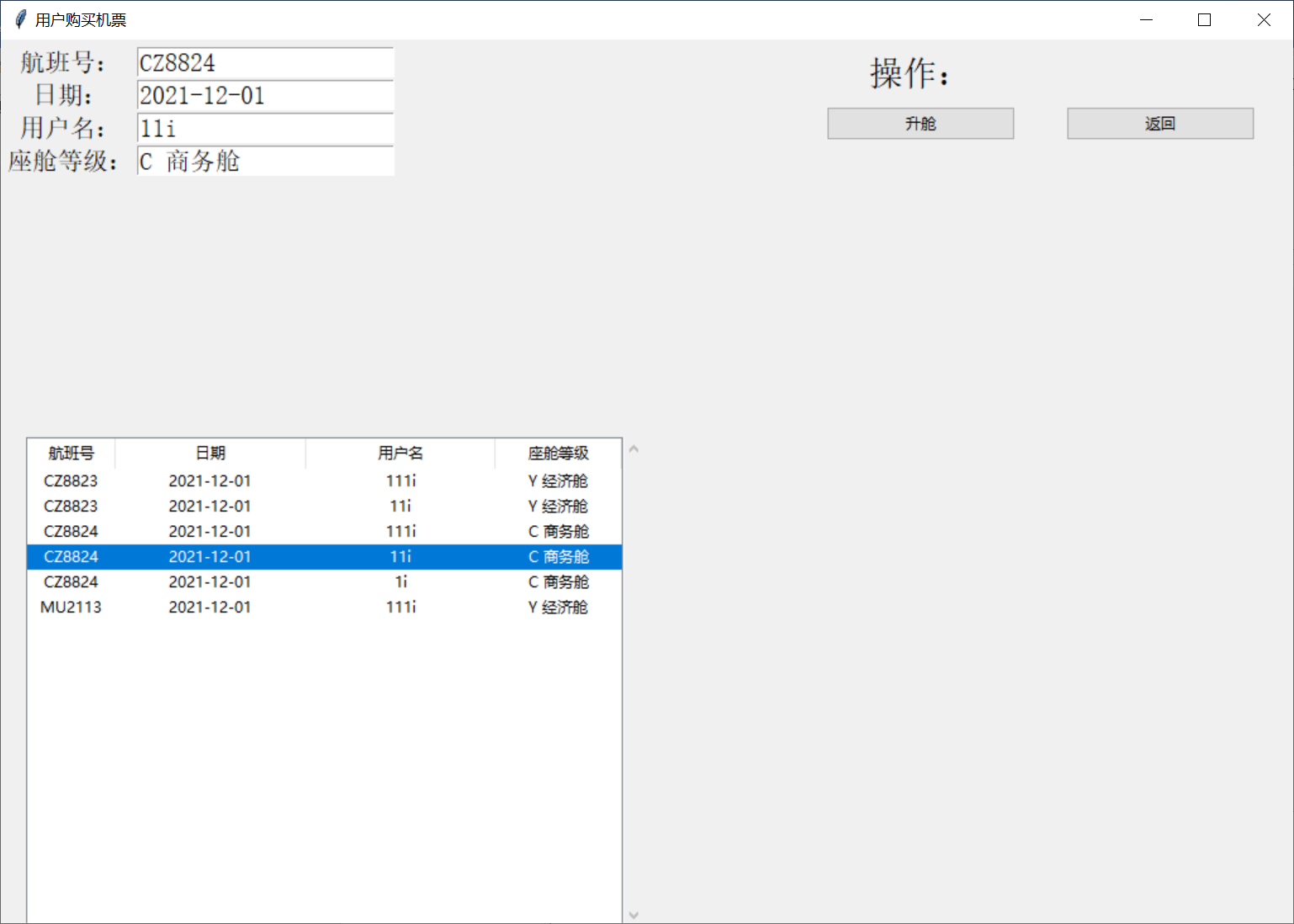
然后点击操作中的删除信息，出现选择框，可以选择删除，不删除，或者取消此次操作。



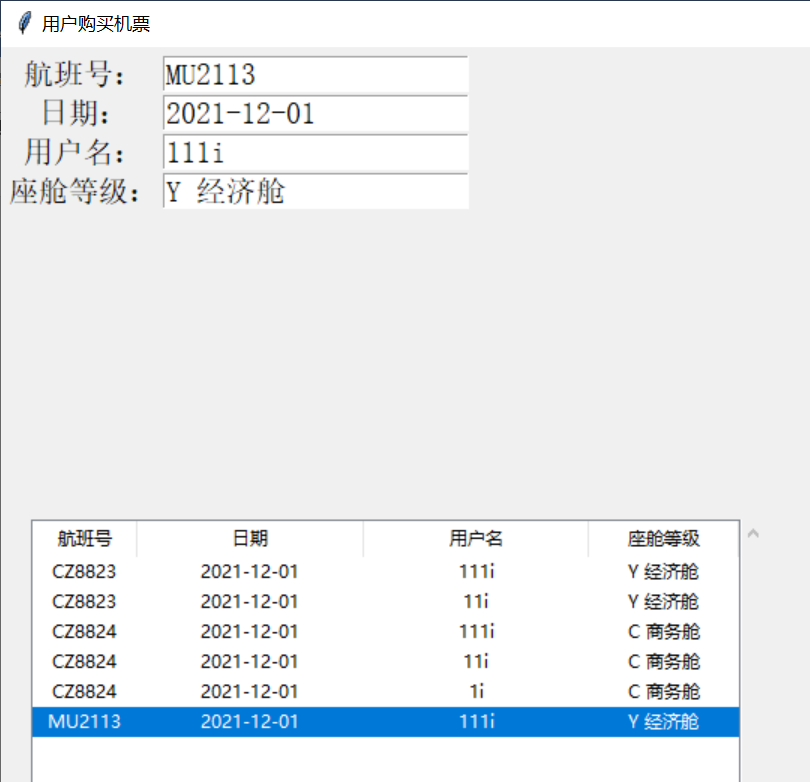
点击确定即可删除。

最后是修改用户购买的机票，可以为其进行修改舱位的操作：

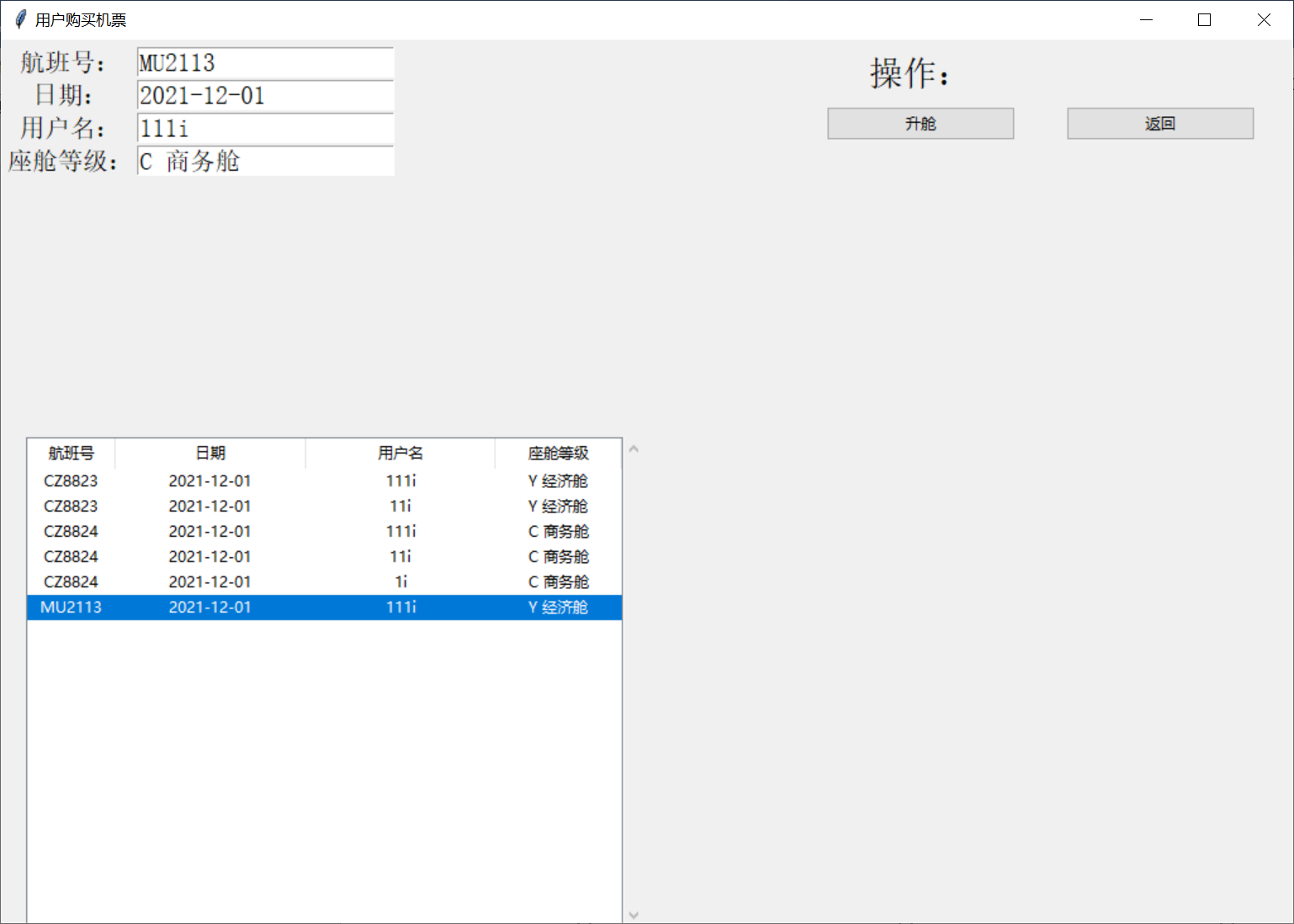




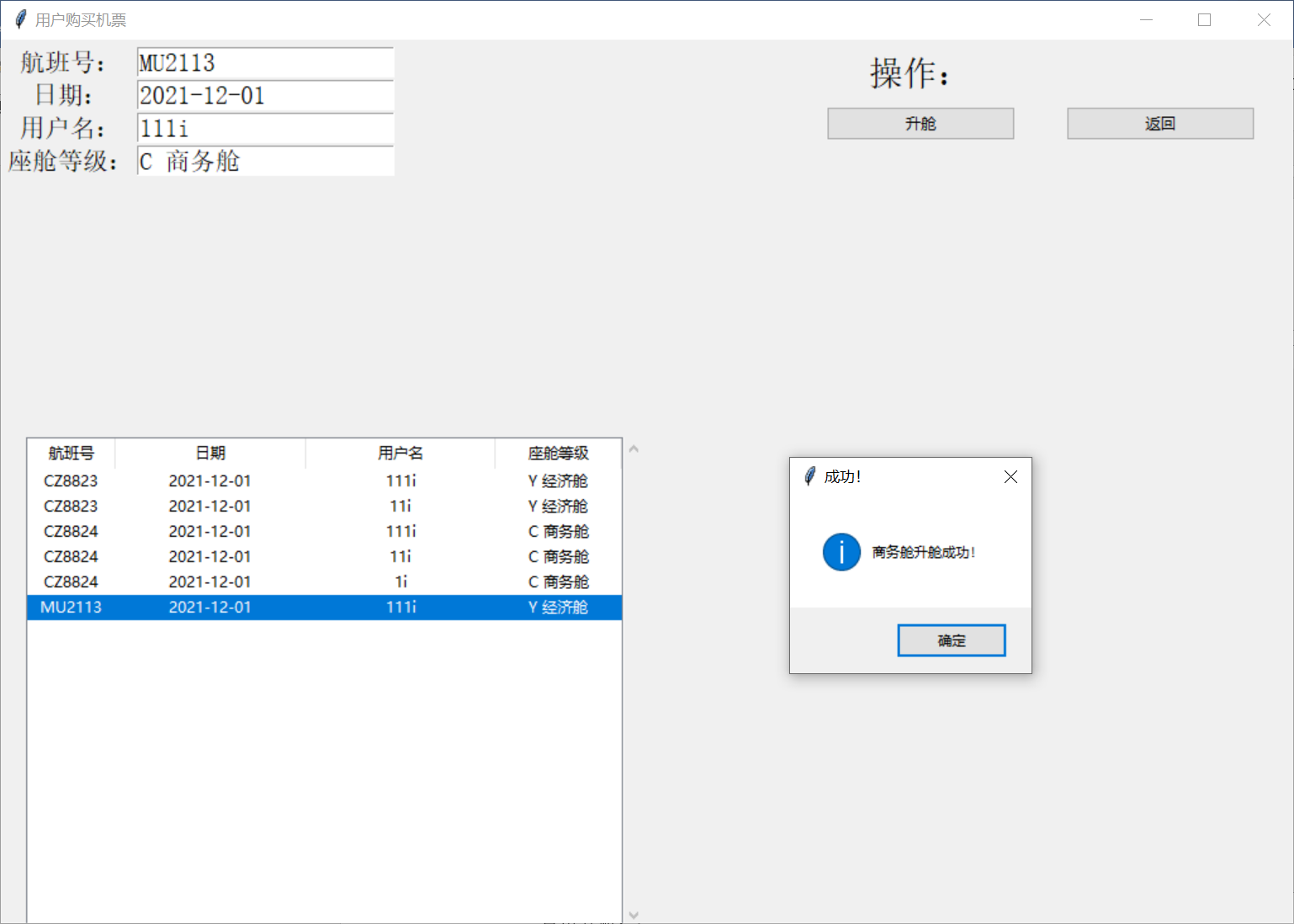
先选择一个信息：



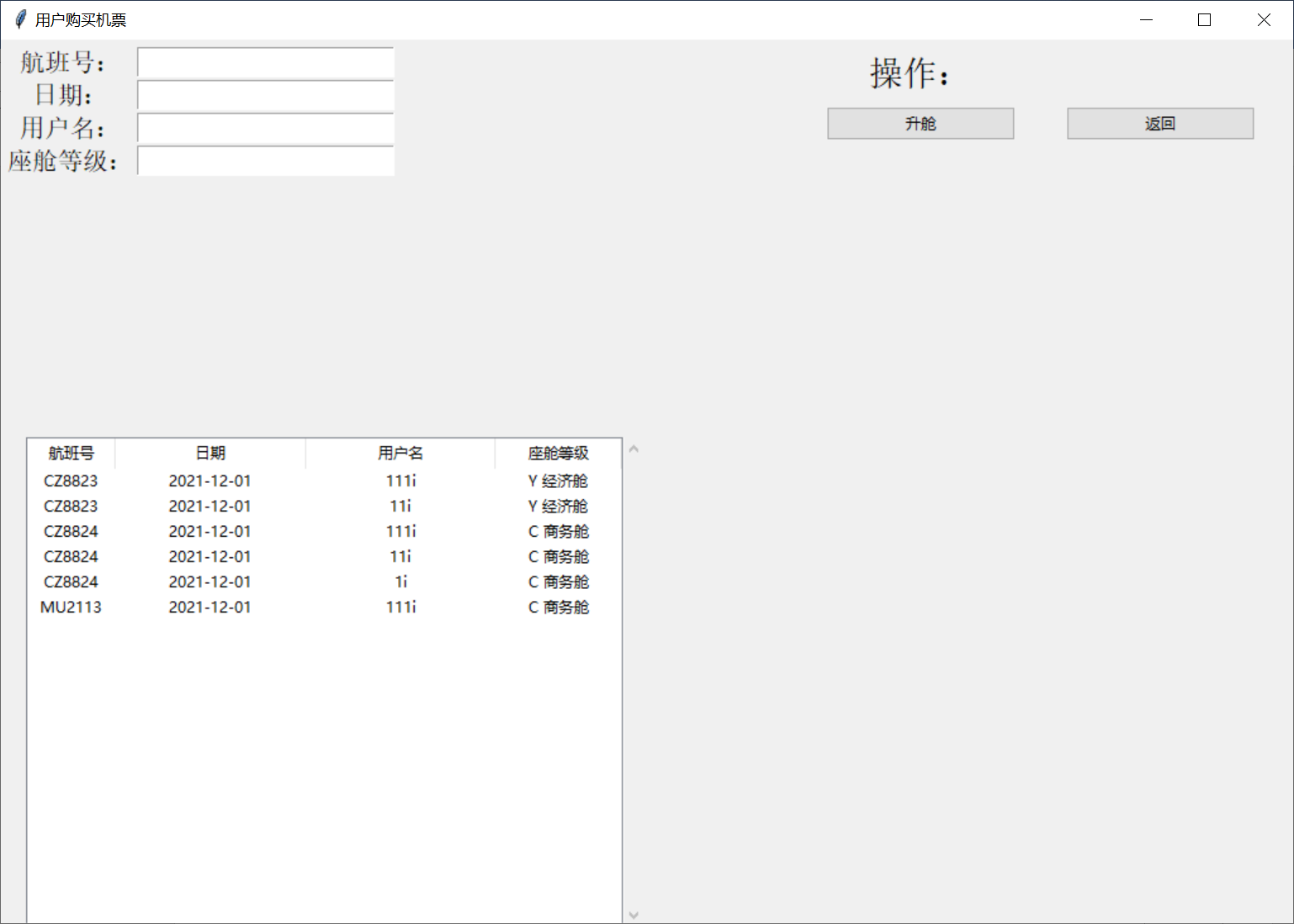
然后填写修改信息：



点击升舱：



再次查看可见：



**当我们需要退出管理员用户程序时，我们可以点击右上角的关闭，或点击退出则退出程序。**



当我们需要使用用户账户进行操作时，在主界面点击用户登录按钮，即可进入用户登录界面。



在用户登录界面中，我们可以使用用户账号进行登录。

用户账号革新新建账号，可自行设置密码。

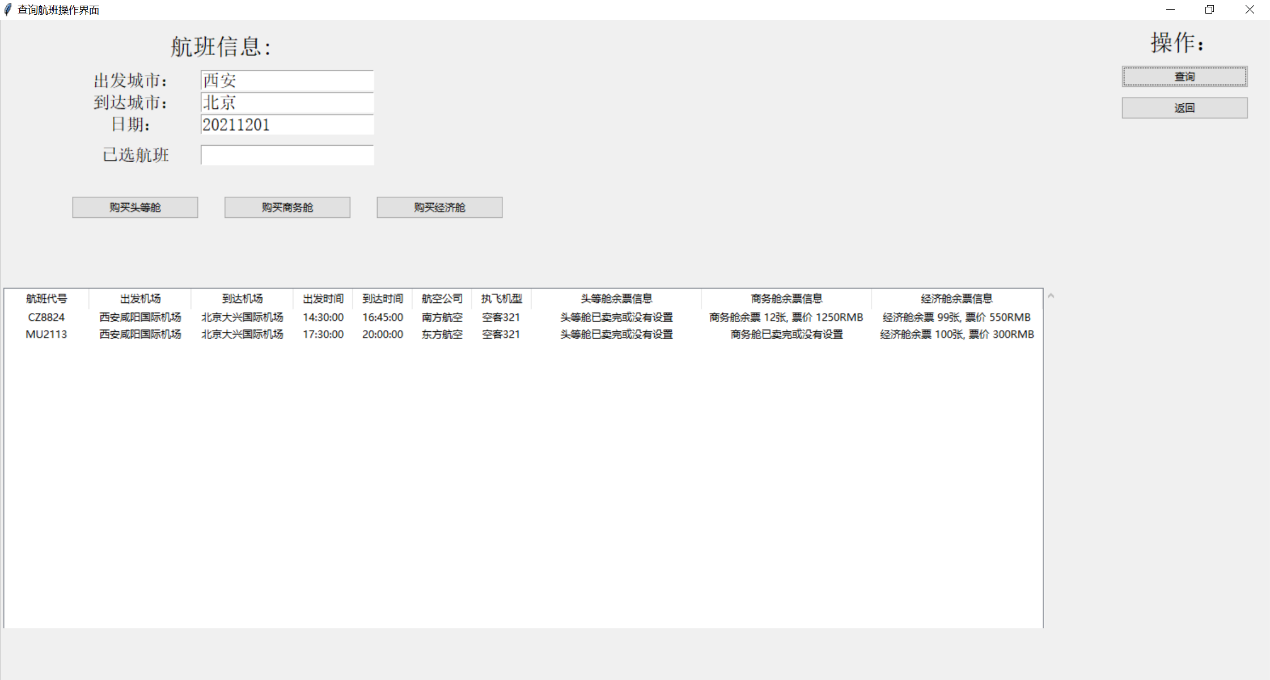
也可在当前窗口创建新用户：



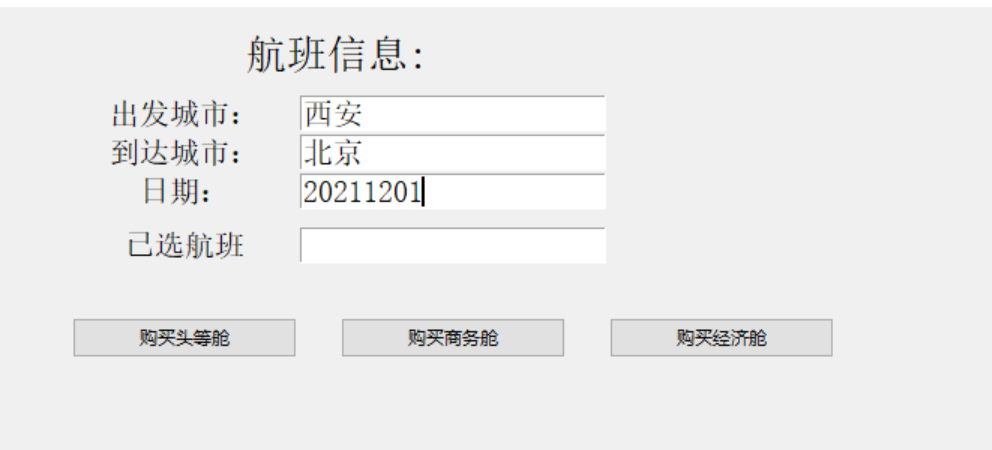
点击还没有账号？使用当前账户名密码来注册！，即可进行注册，如果已注册成功则可使用用户密码直接登录。进入用户主界面，可以在此页面中查看信息。



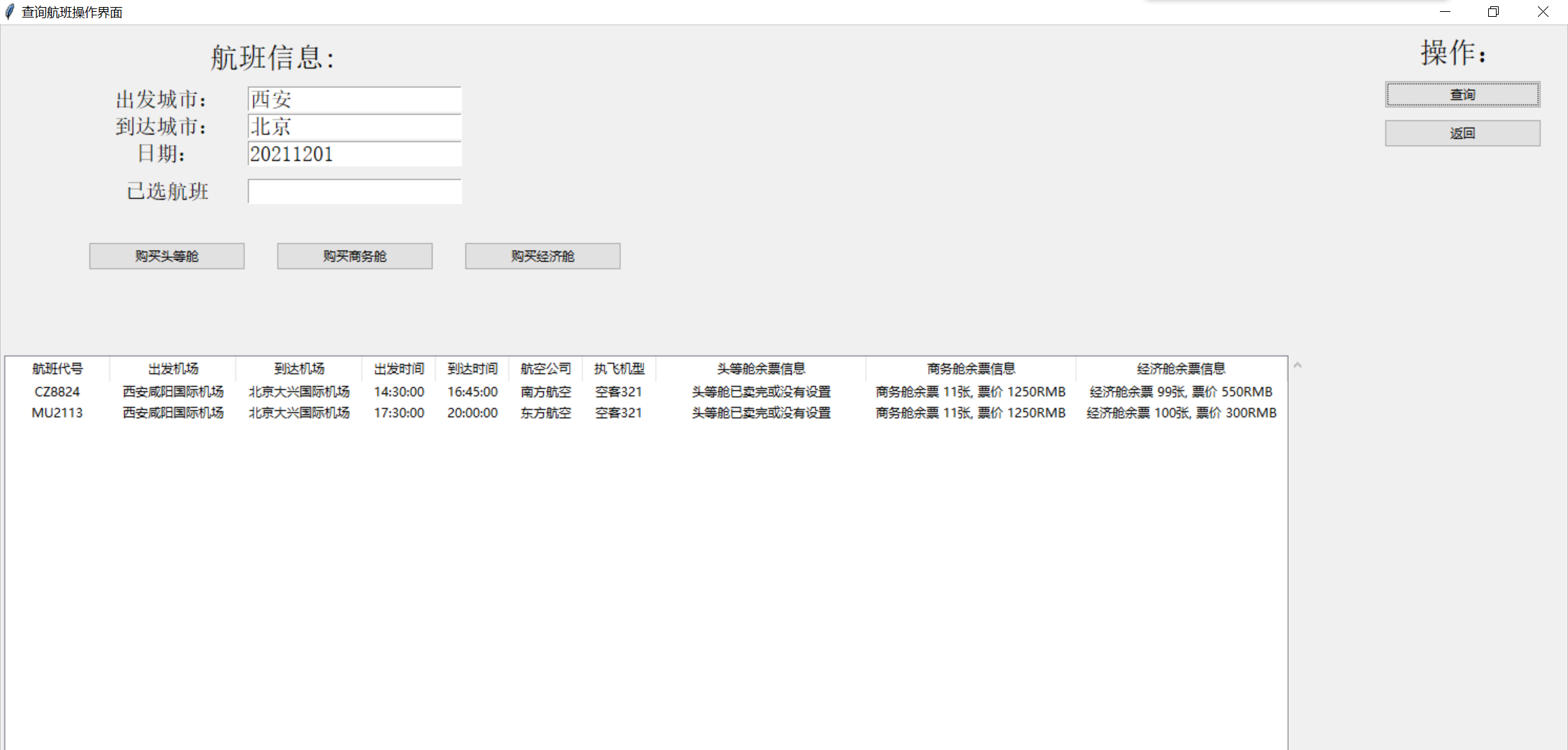
首先是查询购买机票



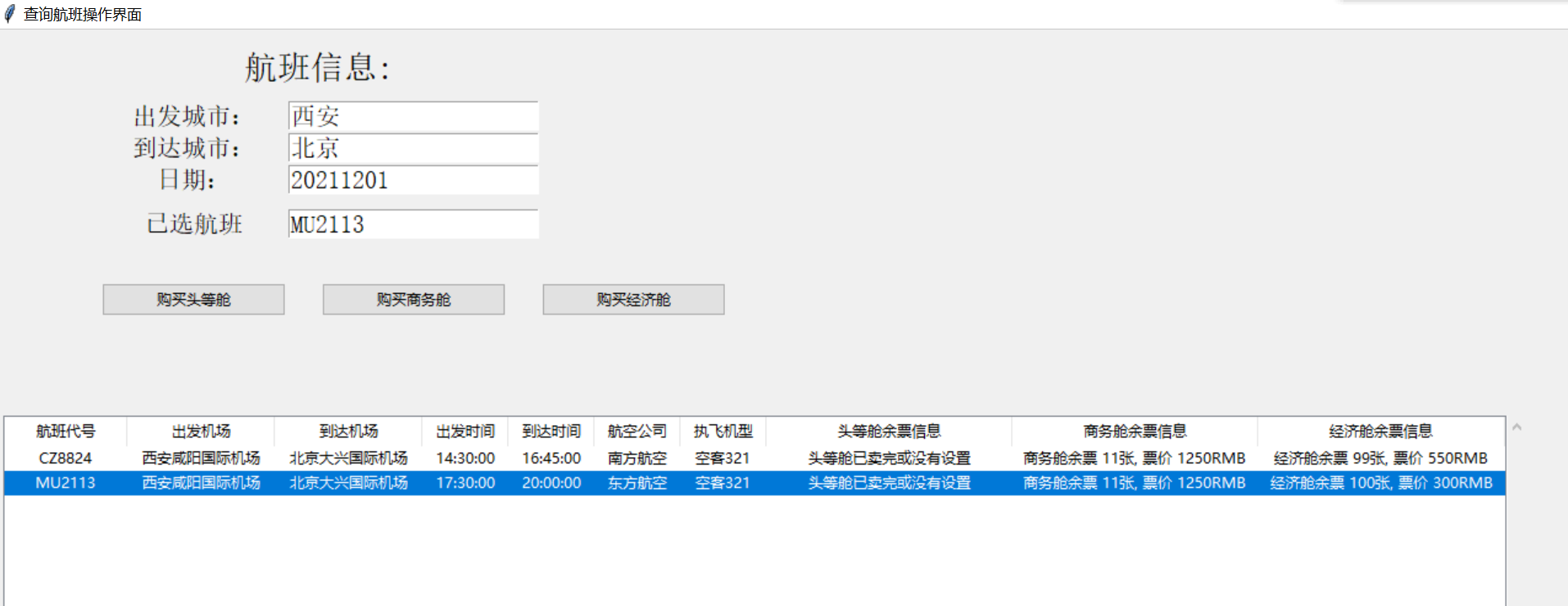
当我们需要查询时，我们需要将相关信息输入到框中。



点击查询即可

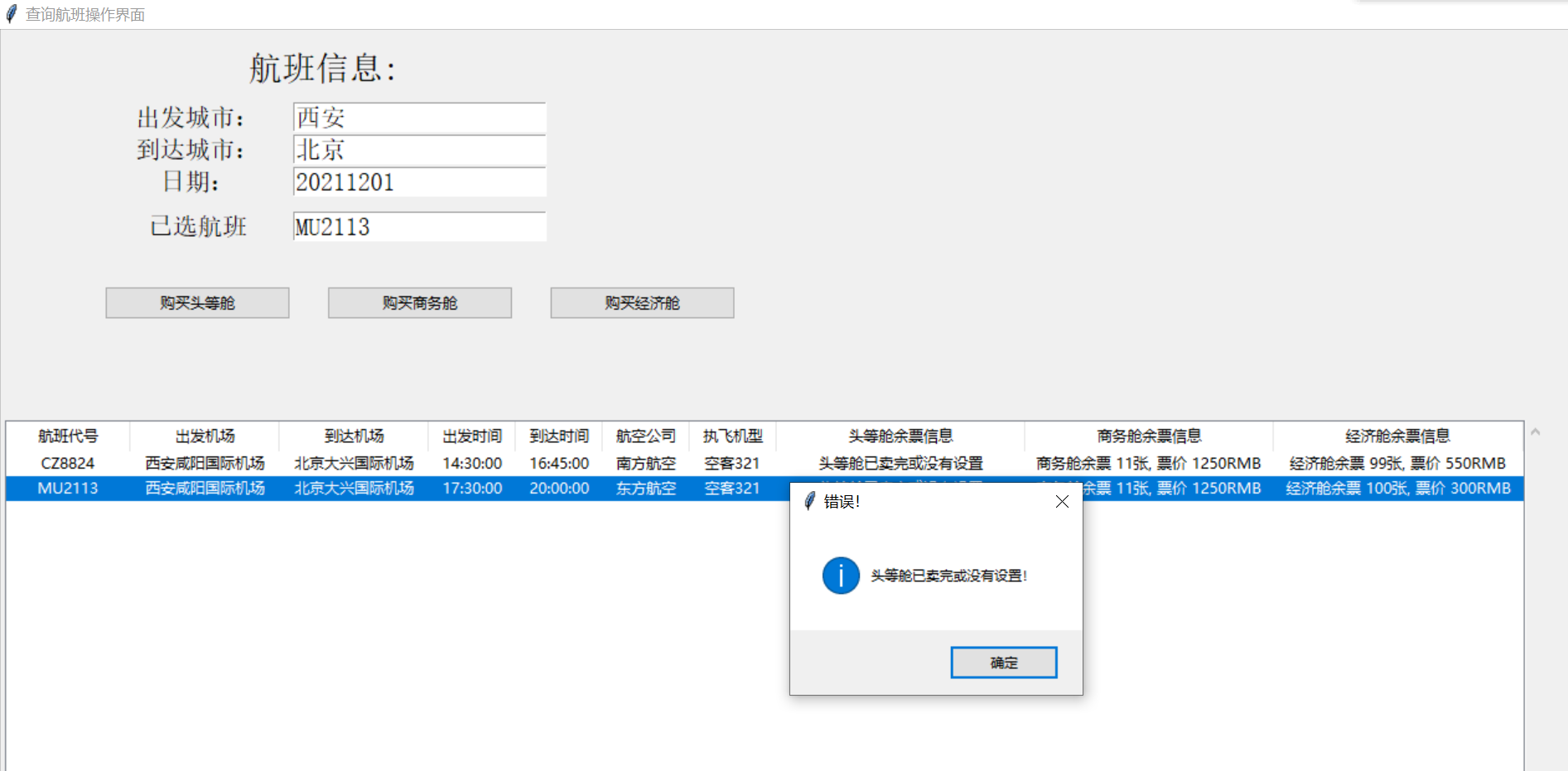


选中需要购买的航班

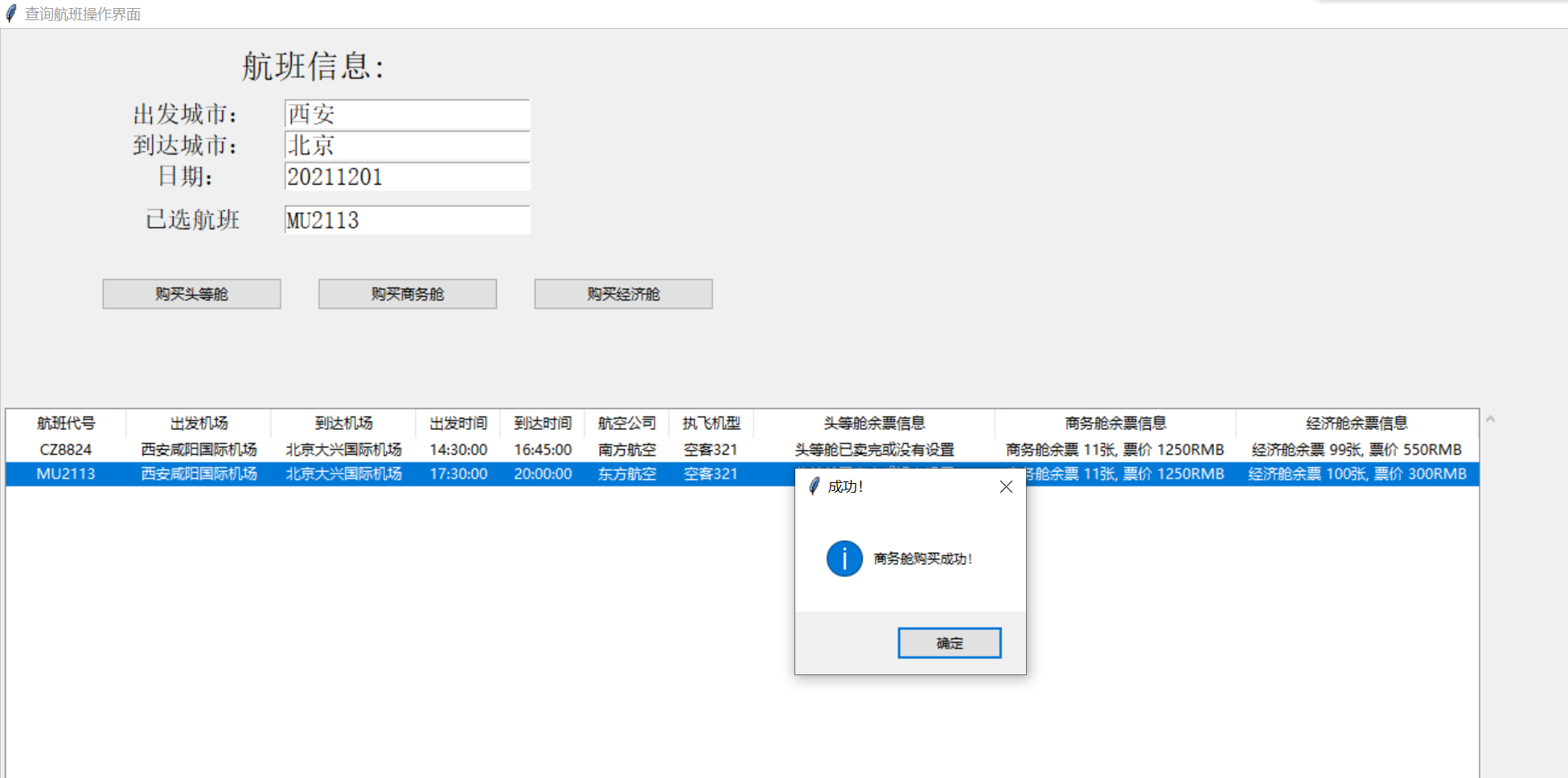


在点击准备购买的舱位：

需要注意，如果没有对应舱位或者已经售空则无法购买：

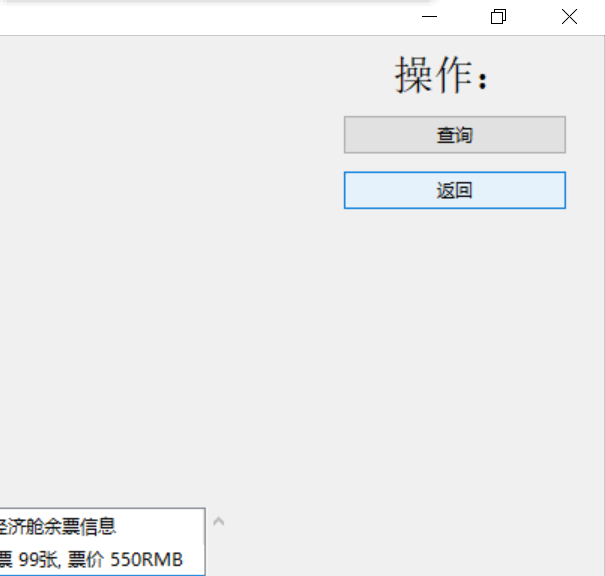


我们点击购买商务舱：



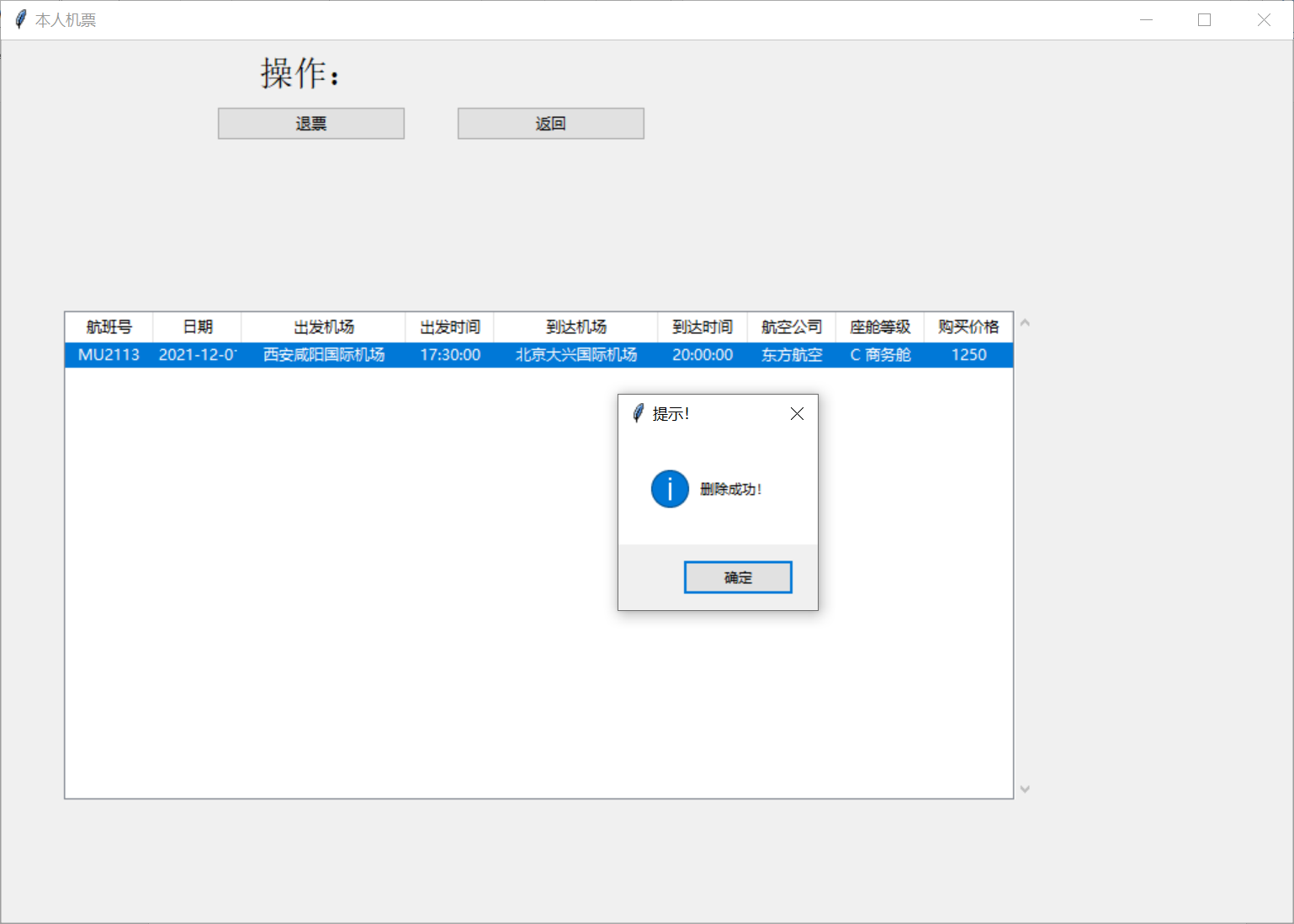
即可购买成功。

点击返回按钮返回上一界面：



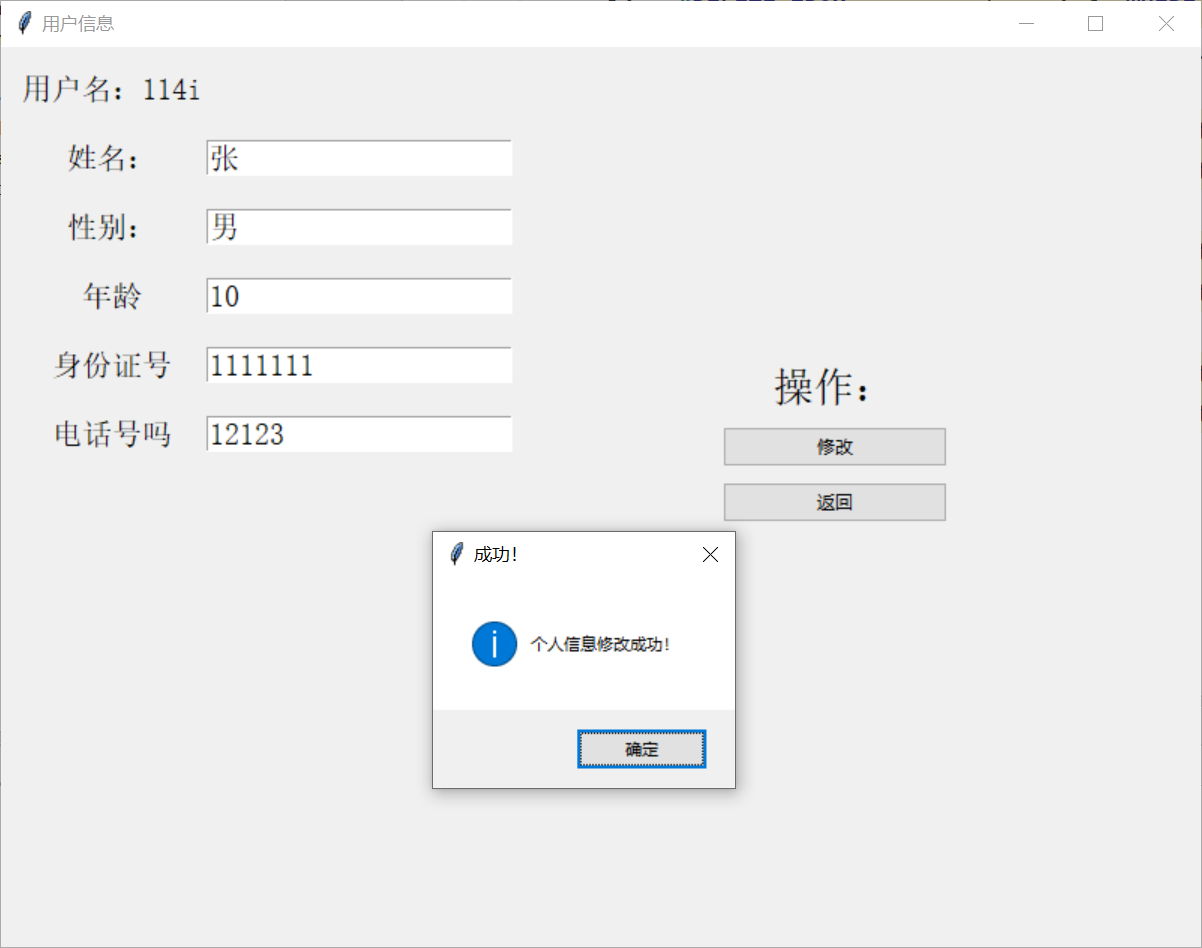
接下来点击已购买机票即可查看刚刚购买的机票



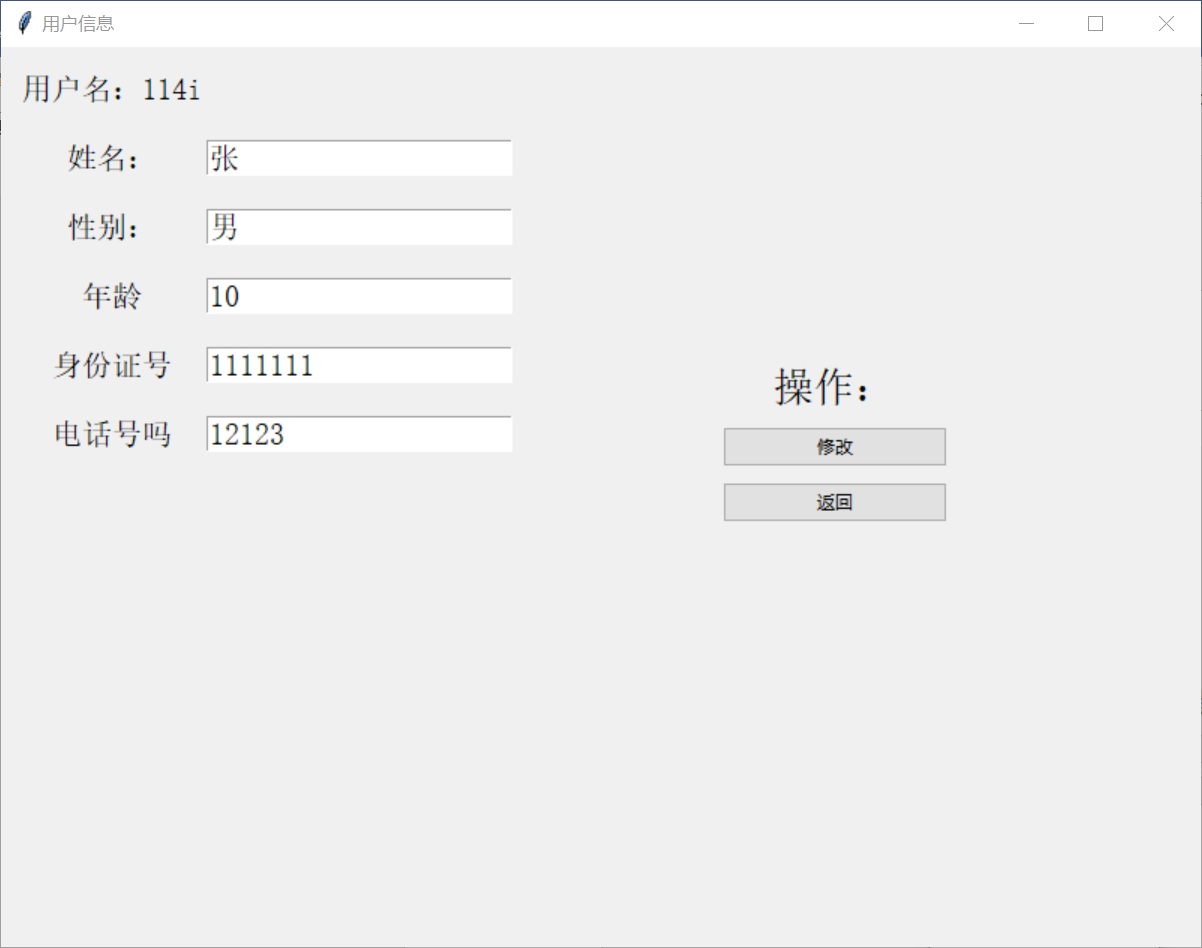
点击退票则可成功退票：  


最后是个人信息的修改：

填入相关个人信息之后，点击修改之后即修改成功

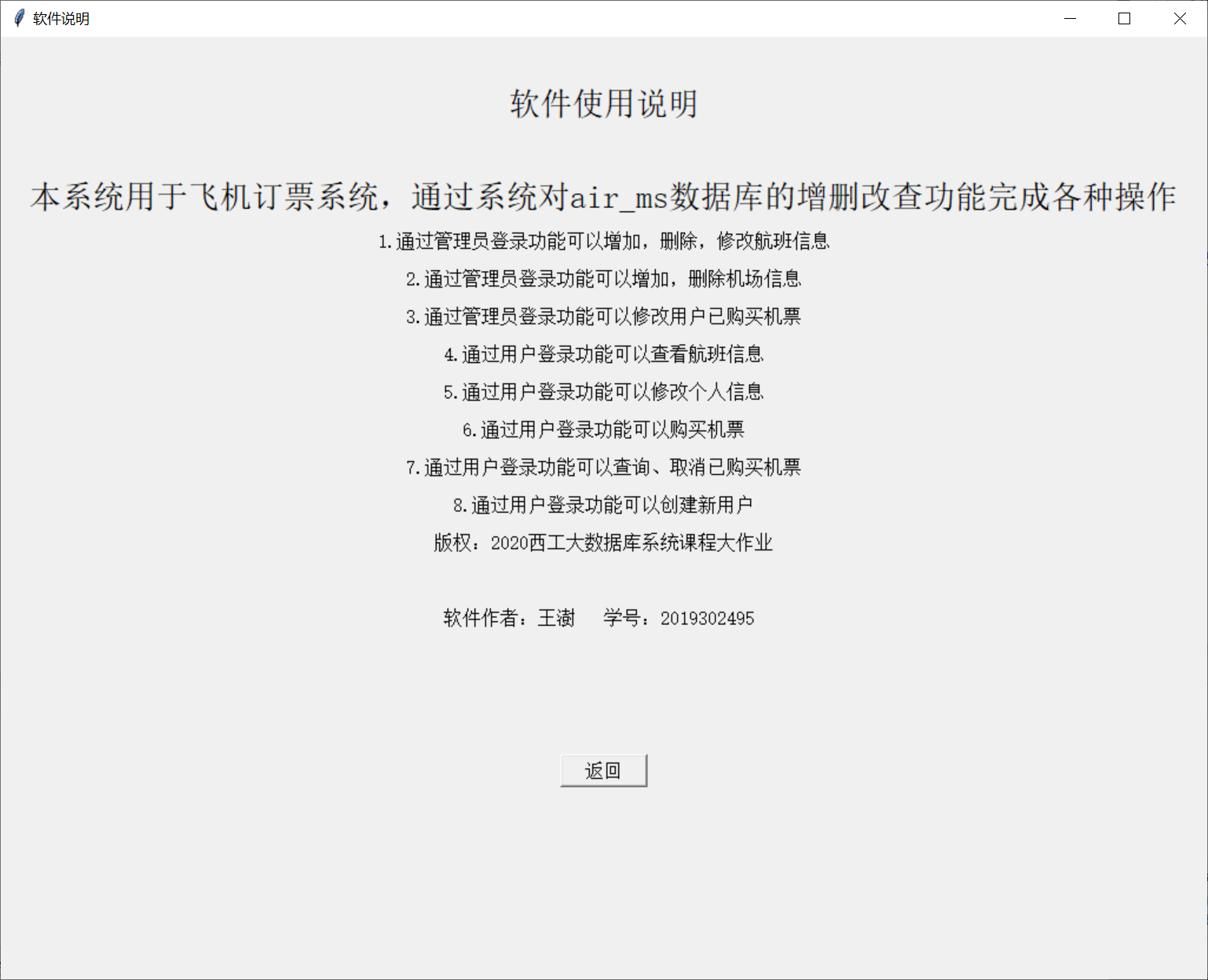


再次查看为已修改的信息



**当我们需要退出用户界面是，电机返回首页即可关闭用户菜单并返回到首页。**

此程序还提供了使用说明，用户可以通过点击使用说明和开发说明按钮进行查看。

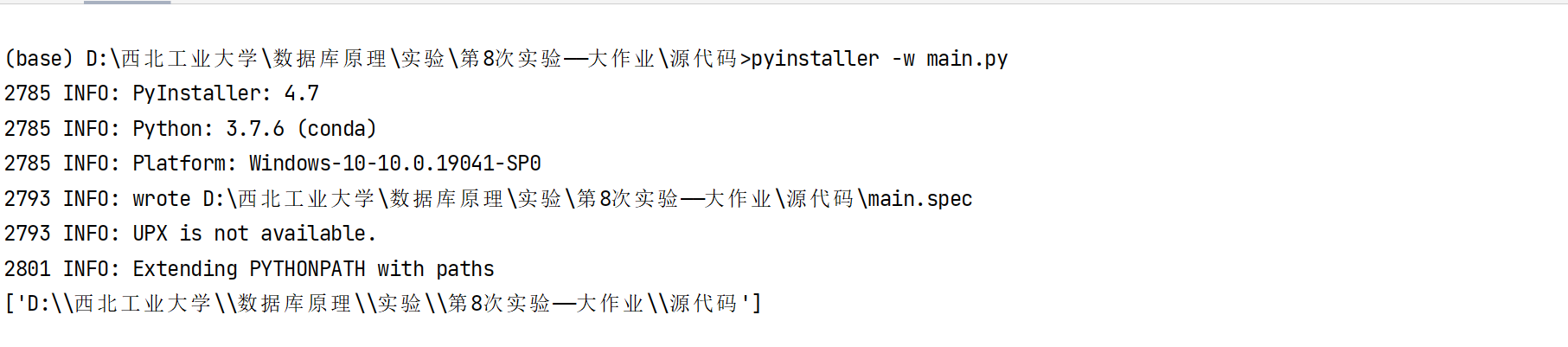


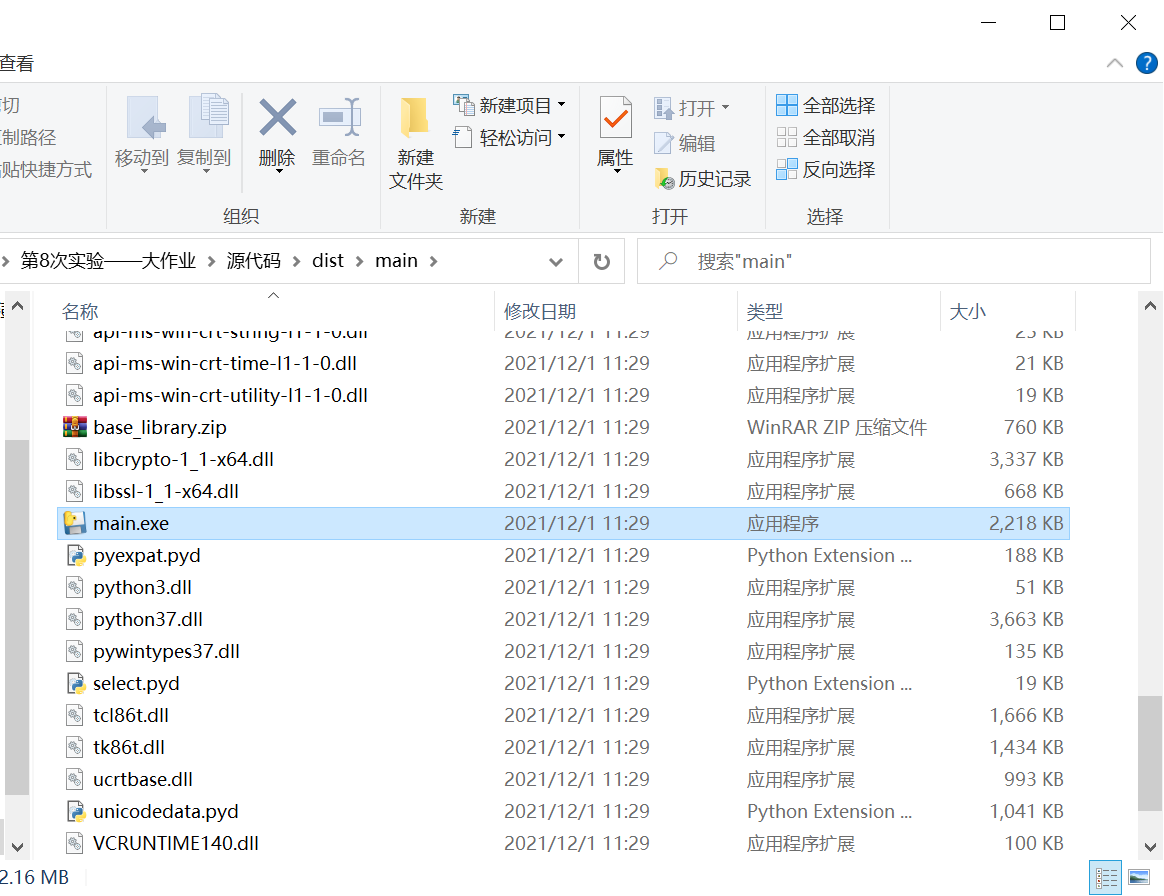
同样只需要点击返回按钮，即可退出返回到首页。

在首页，点击退出程序按钮，即可退出程序。



最后生成 可执行程序





## 实验中出现的问题以及解决方案

使用Tkinter包编写图形用户界面程序:

Tkinter 是使用 python 进行窗口视窗设计的模块。Tkinter模块 ("Tk 接口”) 是Python的标准Tk GUI工具包的接口。作为 python 特定的GUI界面，是一个图像的窗口，tkinter是python 自带的，可以编辑的GUI界面。

首先，我们导入Tkinter包，然后创建一个窗口，最后给这个窗口设置标题。

from tkinter import \*

window = Tk()

window.title("First Window")

window.mainloop()

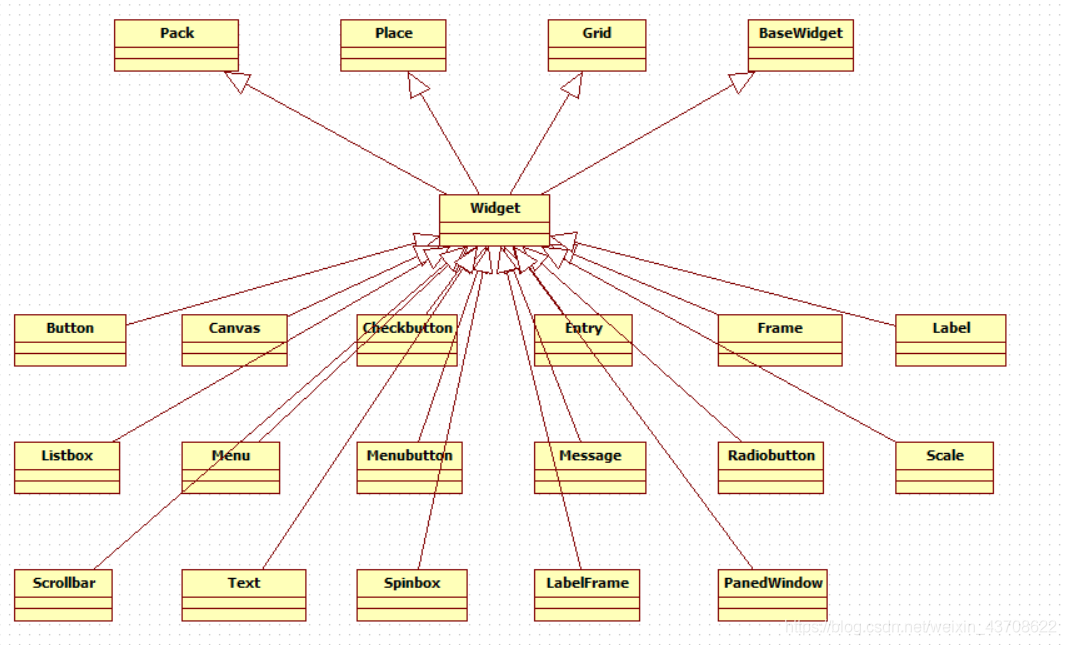
最后一行我们调用了mainloop函数，这个函数将让窗口等待用户与之交互，直到我们关闭它。如果忘记调用mainloop函数的话，将不会向用户显示任何内容（没有窗口）。

然后，通过调用各种控件，完成控制操作。



关于控件的位置摆放：

Pack方法：



print函数的打印信息如下：

{'in': <tkinter.Tk object .>, 'anchor': 'center', 'expand': 0, 'fill': 'none', 'ipadx': 0, 'ipady': 0, 'padx': 0, 'pady': 0, 'side': 'top'}

我们来看两个主要的属性anchor和side.

anchor为center,表示相对于父控件(这里为主窗口)button居中显示,显然button将占用这一整行的空间,下一个控件将从下一行开始布局.

side为top,表示向顶部对齐.

side 属性 决定将控件放在父对象的哪一边.TOP (default) 表示顶部, BOTTOM 表示底部, LEFT 表示靠左, RIGHT 表示靠右.

anchor 属性 决定控件的锚点，也就是控件的起始位置。可设置的值为’n’, ‘ne’, ‘e’, ‘se’, ‘s’, ‘sw’, ‘w’, ‘nw’; ‘e’、‘w’、‘s’、'n’分别表示东西南北。

pack布局中,控件是紧凑分布的，可以通过side和anchor属性的组合，来决定控件的实际布局。

expand 属性 设置为True,将填满控件的父对象的空间。

填充分布

fill 属性 决定填充方向.None (default)表示保持控件原尺寸.X 表示填充水平方向.Y 表示填充垂直方向.BOTH 表示水平和垂直方向都填充.

外边距

外边距通过两个属性来控制，padx和pady。

内边距

内边距通过两个属性来控制，ipadx和ipady。pack不同属性的组合可以形成非常多的布局效果，本文就不再一一列举。pack布局还支持不同父子控件的组合分布，但是不管怎样组合，pack的属性就是这几种，使用灵活。

**批阅者：**

**批阅日期：**

**实验成绩：**

**批注：**