|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | | | | |
|  |  |  |  |  |
|  |  |  |  |

**题目五：车站售票管理系统**

软件/运行环境：

MS SQL SERVER2016，Eclipse

JDBC6.0，外部jar包：sqljdbc41.jar

jar包运行环境：jre7

附件：可执行程序文件：TicketMS.jar

**一、需求分析阶段：**

**1. 支持普通用户/管理员注册/登录，普通用户可以查询列车信息，购买车票/团购，管理（创建/删除）自己的订单，管理员可以对列车信息进行修改（增删改查）。**

**2.列车信息包含一个始发站，多个途经站和一个终点站，相关费用以及到站/出站时间，可以方便的查询时刻表或者是车次，票价等信息。**

**3.管理员可以更改系统中存储的各种信息。**

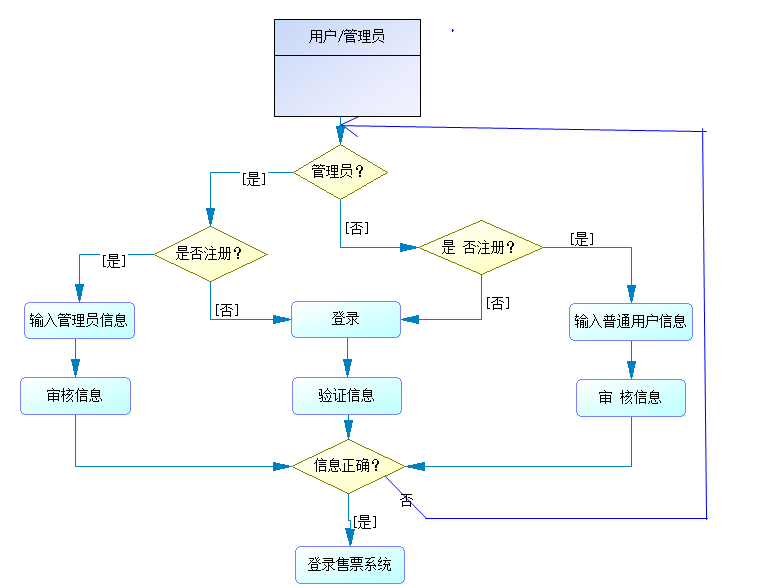
**4. 具有方便，快速的售票功能，包括车票的预定和退票功能，支持团体的预定和退票。**

**需求分析结果：**

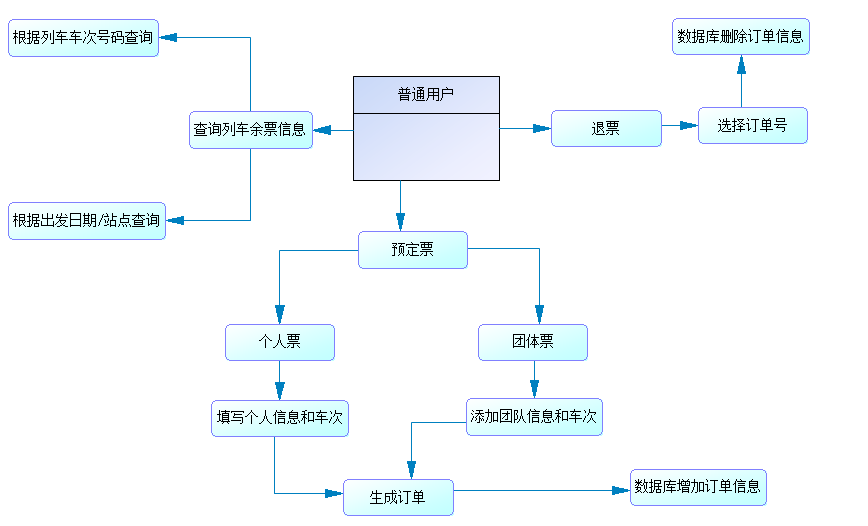
**1.业务流程图：**

为了比较清晰的说明设计思路，我使用了PowerDesigner软件绘制流程图和E-R图

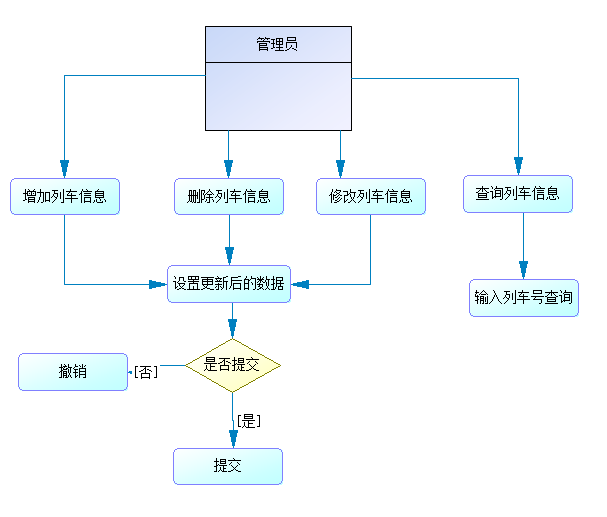
用户/管理员注册/登录流程：



普通用户操作流程：



管理员操作数据库流程：



**2.数据字典**

（1）数据库中用到的表

|  |  |
| --- | --- |
| 数据库表名 | 关系模式名称 |
| Customer | 客户 |
| Administrator | 管理员 |
| TrainInfo | 列车信息 |
| UpdateLog | 更新日志 |
| Ticket | 火车票 |
| CustOrder | 客户订单 |
| MakeUp | 组成（一张订单对应多个火车票） |

（2）Customer表的数据项

|  |  |  |
| --- | --- | --- |
| 属性名 | 数据类型 | 说明 |
| Cust\_id | char(20) | 客户ID（主键） |
| Cust\_pwd | char(20) | 登录密码 |
| ID\_no | char(20) | 身份证号 |

（3）Administrator表的数据项

|  |  |  |
| --- | --- | --- |
| 属性名 | 数据类型 | 说明 |
| Admi\_id | char(20) | 管理员ID(主键） |
| Admi\_pwd | char(20) | 管理员密码 |
| Admi\_no | char(20) | 管理员号 |
| Admi\_phone | char(20) | 管理员电话 |

（4）TrainInfo表的数据项

|  |  |  |
| --- | --- | --- |
| 属性名 | 数据类型 | 说明 |
| Train\_no | char(10) | 列车号（主键） |
| Pass\_sta | char(20) | 途经站点 |
| Till\_pri | int | 始发站到本站费用 |
| Come\_time | date | 到达本站的时间 |
| Go\_time | date | 离开本站的时间 |

（5）UpdateLog表的数据项

|  |  |  |
| --- | --- | --- |
| 属性名 | 数据类型 | 说明 |
| Cust\_id | char(20) | 管理员ID（主/外键） |
| Train\_no | char(10) | 列车号（主/外键） |
| Update\_time | date | 更新时间 |

（6）Ticket表的数据项

|  |  |  |
| --- | --- | --- |
| 属性名 | 数据类型 | 说明 |
| Tick\_no | char(10) | 车票号（主键） |
| Seat\_no | char(10) | 座位号 |
| Depa\_sta | char(20) | 出发站 |
| Dest\_sta | char(20) | 目的站 |
| Depa\_time | date | 出发时间 |
| Dest\_time | date | 到达时间 |
| Train\_no | char(10) | 列车号（外键） |
| Price | int | 票价 |

（7）CustOrder表的数据项

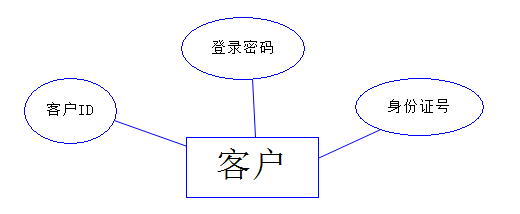
|  |  |  |
| --- | --- | --- |
| 属性名 | 数据类型 | 说明 |
| Order\_no | char(20) | 订单号（主键） |
| Cust\_id | char(20) | 客户ID（外键） |
| Create\_time | date | 创建时间 |
| Price | int | 价格 |

**二、概念设计**

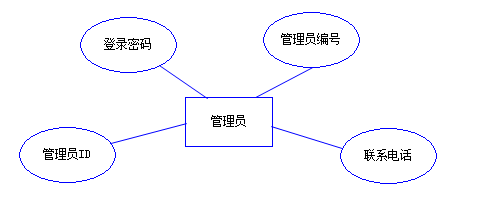
**包括分E-R图和全局E-R图：**

（1）实体及其属性的分E-R图：

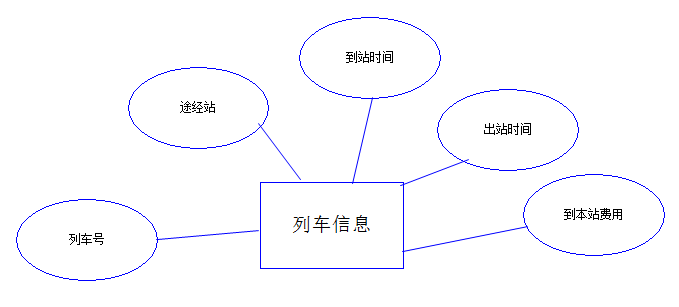
1.普通用户：



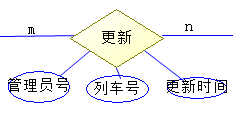
2.管理员：



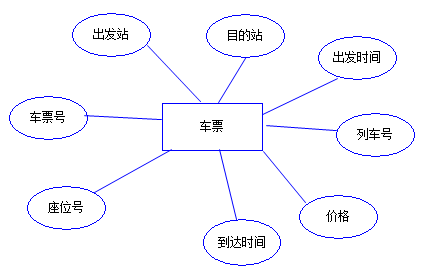
3.列车信息：



**注：管理员和列车信息之间的关系是多对多（m:n）联系，联系名定义为“更新”，联系的属性有 “列车号”、“管理员号”、 “更新日期”。**

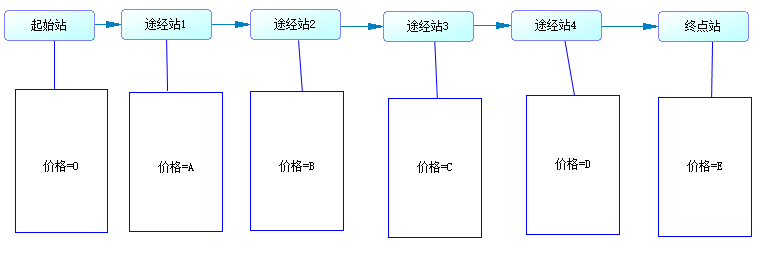


3.车票：



**注：**

**1.实体“列车信息”的途经站是列车中途经过的站点，到站时间是到达该途经站的时间，出站时间是离开该途经站的时间，价格是从全程的起始站到该途经站的价格。**

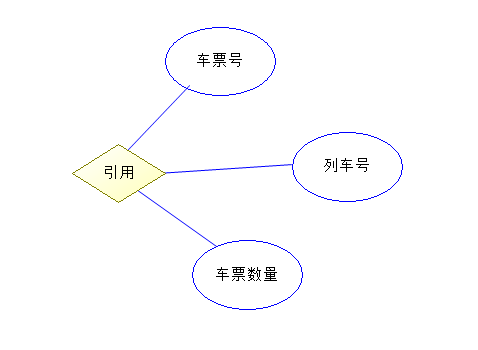


**实体“车票”的价格是根据实体“列车信息”的属性“到本站费用”计算出来的**

**例如：车票的出发站是列车信息的途经站2，目的站是列车信息的途经站4，**

**则车票的价格=D-B （元）**

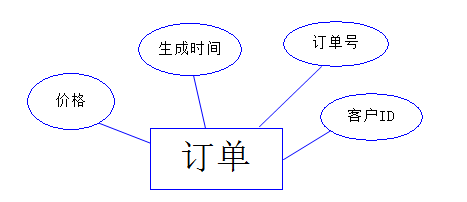
**2.实体“车票”与实体“列车信息”之间的关系是多对一关系，即一个列车信息对应多个车票，因为一个列车信息包含所有列车的途经站，而车票的出发站和目的站是列车的途经站的区间内的所有可能组合情况。**



**车票数量：剩余的车票的数量**

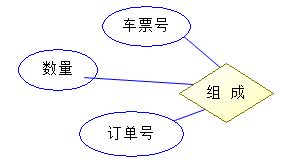
**在后面的全局E-R图设计中，将“车票”与 “列车信息”之间的联系称为“引用”。**

4.订单：

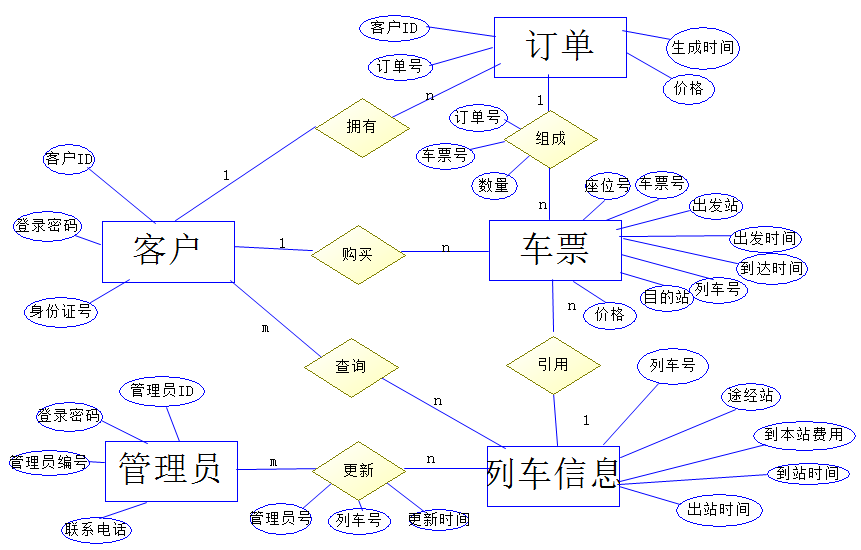


订单与车票是“组成的联系”，一个订单对应n个车票（包括团体票）

关系“组成”图如下：



（2）全局E-R图：



**三、逻辑设计**

1.将‘二’中的概念设计的全局E-R图模型转换为关系模式：

注：蓝色表示主键，斜体字表示外键。

客户（客户ID，登录密码，身份证号）

管理员（管理员ID，登录密码，管理员编号，联系电话）

列车信息（列车号，途经站，到本站费用，到站时间，出站时间）

更新日志（*管理员ID，列车号*，更新时间）

车票（车票号，座位号，出发站，出发时间，到达时间，*列车号*，目的站，价格）

订单（*订单号，客户ID*，生成时间，价格）

组成（*订单号，车票号*，数量）

引用（*车票号，列车号*，车票数量）

2.在MS SQL SERVER中定义对应的数据库表格英语名称：

表格名称（属性列1,2，…）

Customer(Cust\_id,Cust\_pwd,ID\_no)

Administrator(Admi\_id,Admi\_pwd,Admi\_no,Admi\_phone)

TrainInfo(Train\_no,Pass\_sta,Till\_pri,Come\_time,Go\_time)

UpdateLog(*Cust\_id,Train\_no*,Update\_time)

Ticket(Tick\_no,Seat\_no,Depa\_sta,Dest\_sta,Depa\_time,Dest\_time,*Train\_no*,Price)

CustOrder(Order\_no,*Cust\_id*,Create\_time,Price)

MakeUp(*Order\_no,Tick\_no,*Amou)

TicketRefer(*Tick\_no,Train\_no*,Amou)

3.创建表格的SQL语句：

创建Customer表：

create table Customer

(

Cust\_id char(20) primary key,

Cust\_pwd char(20) not null,

ID\_no char(20)

)

创建Administrator表：

create table Administrator

(

Admi\_id char(20) primary key,

Admi\_pwd char(20) not null,

Admi\_no char(20),

Admi\_phone char(20)

)

创建TrainInfo表：

create table TrainInfo

(

Train\_no char(10) primary key,

Pass\_sta char(20),

Till\_pri int,

Come\_time date,

Go\_time date

)

创建UpdateLog表：

create table UpdateLog

(

Cust\_id char(20),

Train\_no char(10),

Update\_time date,

)

创建Ticket表：

create table Ticket

(

Tick\_no char(10) primary key,

Seat\_no char(10),

Depa\_sta char(20),

Dest\_sta char(20),

Depa\_time date,

Dest\_time date,

Train\_no char(10),

Price int,

constraint ticket\_foreign\_trainno foreign key (Train\_no) references TrainInfo(Train\_no)

on delete cascade on update cascade

)

增加车票的存储过程：

create procedure add\_ticket(@trainno char(10))

as

begin

declare @ticketnum int

declare @ticketno char(10)

select @ticketnum=ticket\_code from SerialNumCode

set @ticketno='ticket-'+cast(@ticketnum as char(10))

update SerialNumCode

set ticket\_code=@ticketnum+1

declare @startsta char(20)

declare @endsta char(20)

select @startsta=Start\_sta ,@endsta=End\_sta from TrainInfo

where Train\_no=@trainno

insert Ticket values

(@ticketno,'a001',@startsta,@endsta,getdate(),getdate()+1,@trainno,150)

insert TicketRefer values

(@ticketno,@trainno,100)

end

创建CustOrder表：

create table CustOrder

(

Order\_no char(20) primary key,

Cust\_id char(20),

Create\_time date,

constraint custorder\_foreign\_custid foreign key(Cust\_id) references Customer(Cust\_id)

on delete cascade on update cascade

)

创建MakeUp表：

create table MakeUp

(

Order\_no char(20),

Tick\_no char(20),

Amou int,

primary key(Order\_no,Tick\_no)

)

创建TicketRefer表：

create table TicketRefer

(

Tick\_no char(10),

Train\_no char(10),

Amou int,

primary key(Tick\_no,Train\_no)

)

同时在列车信息TrainInfo表上创建触发器,当删除或更新TrainInfo上的一个列车信息时，自动删除或更新对应的列车号的TicketRefer表上的数据记录。

create trigger dele\_ticketrefer

on TrainInfo

after delete

as

begin

declare @deletrainno char(10)

select @deletrainno=Train\_no

from deleted

delete TicketRefer

where Train\_no=@deletrainno

end

create trigger upda\_ticketrefer

on TrainInfo

after update

as

begin

declare @oldno char(10)

declare @updatrainno char(10)

select @oldno=Train\_no

from deleted

select @updatrainno=Train\_no

from inserted

update TicketRefer

set Train\_no=@updatrainno

where Train\_no=@oldno

end

4.考虑到车票编号、列车编号、订单编号的唯一性，在TicketMS数据库中定义SerialNumCode表，数据项有order\_code(int),ticket\_code(int),train\_code(int),初始值均为0，每次需要创建新的订单号时，就从SerialNumCode中取出order\_code数据，装换为字符型char,增加一些英文前缀"order",作为新创建的订单的订单号，之后再将SerialNumCode中的order\_code加1，其他数据项对应的新建列车/车票信息类似处理。

create table SerialNumCode

(

order\_code int,

ticket\_code int,

train\_code int

)

insert SerialNumCode

values(0,0,0)

5.创建TrainInfo表上的触发器，当管理员对TrainInfo更新时，更新记录写到UpdateLog表上：

create trigger trutraininfo

on TrainInfo

after update

as

begin

declare @newtrainno char(10)

select @newtrainno=Train\_no

from inserted

insert UpdateLog

values('0001',@newtrainno,getdate())

end

create trigger tritraninfo

on TrainInfo

after insert

as

begin

declare @newtrainno char(10)

select @newtrainno=Train\_no

from inserted

insert UpdateLog

values('0001',@newtrainno,getdate())

end

6.考虑到实际应用过程中，客户买票和管理员更新列车信息是并发执行的事务，为了保证这两个事务的可串行化调度和数据的一致性，防止“丢失修改”、“读脏数据”这些问题，建立客户买票的存储过程，在执行客户买票事务时对列车和车票信息加xlock锁，在执行管理员修改列车信息时对列车信息和车票引用信息加xlock锁。

存储过程1（客户买票）：**其中用到了4中定义的SerialNumCode表中order\_code数据项。**

create procedure purchase\_ticket(@ticketno char(10),@customer char(20),@amount int)

as

begin

begin transaction

declare @tnum int

declare @ordernum int

declare @orderno char(20)

select @tnum=Amou from TicketRefer with(xlock)

where Tick\_no=@ticketno

if (@tnum<@amount)

rollback

update TicketRefer with(xlock)

set Amou=@tnum-@amount

where Tick\_no=@ticketno

select @ordernum=order\_code from SerialNumCode

set @orderno='order-'+cast(@ordernum as char(10))

update SerialNumCode

set order\_code=@ordernum+1

insert CustOrder

values(@orderno,@customer,getdate())

insert MakeUp

values(@orderno,@ticketno,@amount)

commit

end

存储过程2（管理员增加列车信息）：

create procedure add\_traininfo(@startsta char(20),@endsta char(20),@price int)

as

begin

declare @trainnum int

declare @trainno char(20)

select @trainnum=train\_code from SerialNumCode

set @trainno='train-'+cast(@trainnum as char(20))

update SerialNumCode

set train\_code=@trainnum+1

insert TrainInfo values

(@trainno,@startsta,@endsta,getdate(),getdate()+1,@price)

end

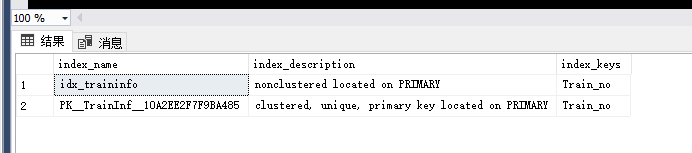
**四、物理设计**

1.创建索引

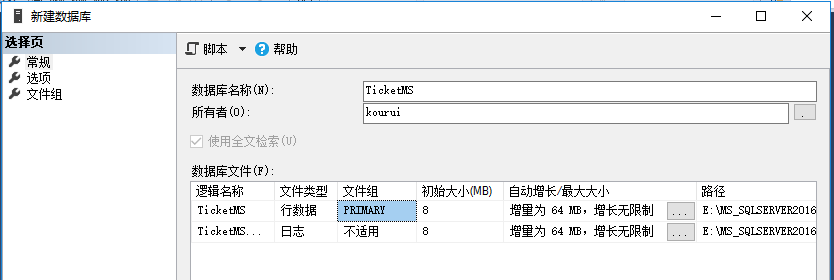
由于列车信息往往是由大量的数据组成，为了提高客户的查询效率，在列车信息TrainInfo表上创建非聚集索引。

create index idx\_traininfo

on TrainInfo(Train\_no)



2.创建TicketMS数据库，“行数据”初始大小为8mb，自动增长，增量为64mb,增长无限制。



**五、图形界面程序设计(java swing)**

**附件：**

1.**java项目名**：TicketMS

2.**图片资源包image:**admi.png,cust.png,myicon.png,train.png

3.**用户信息处理包Info:**AccountInfo.java

4.**SQL过程/事务处理包：**SqlTran.java

5.**UI界面设计包：**StartSys

登录窗口：MainUI.java

管理员窗口：AdministratorUI.java

客户窗口：NormalUI.java

窗口调度管理：FrameManage.java

SQL语句/结果封装类：SqlWork.java

6.**可执行jar包：**TicketMS.jar

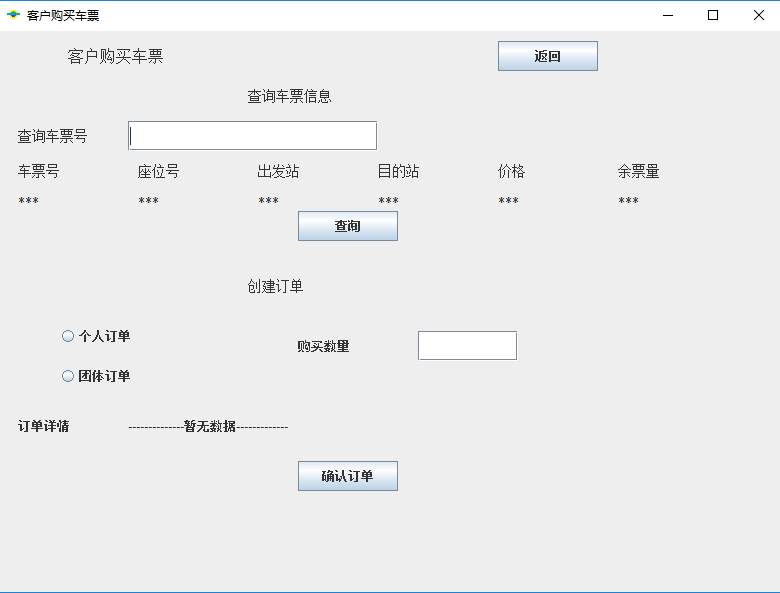
程序运行截图：

1.注册用户账号：





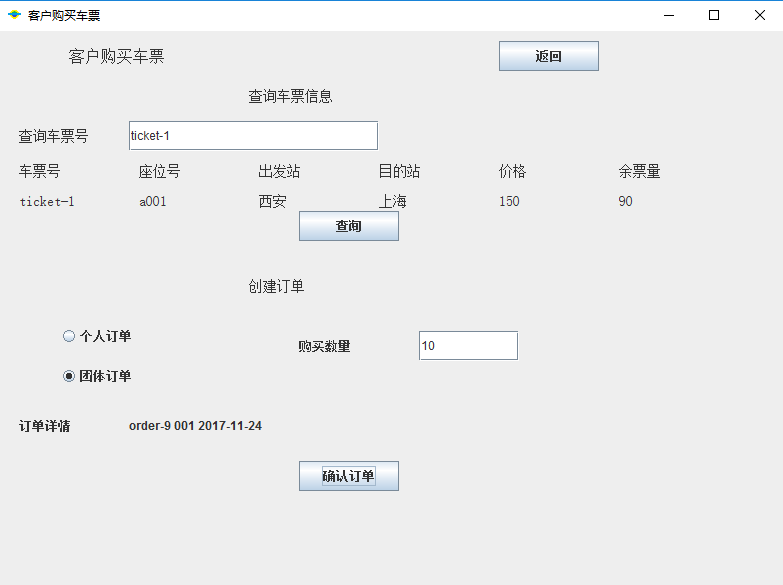
2.普通用户登录界面：



3.查询车票（输入车票号ticket-1，点击查询）



4.创建订单：选择团体订单->购买10张车票->确认订单





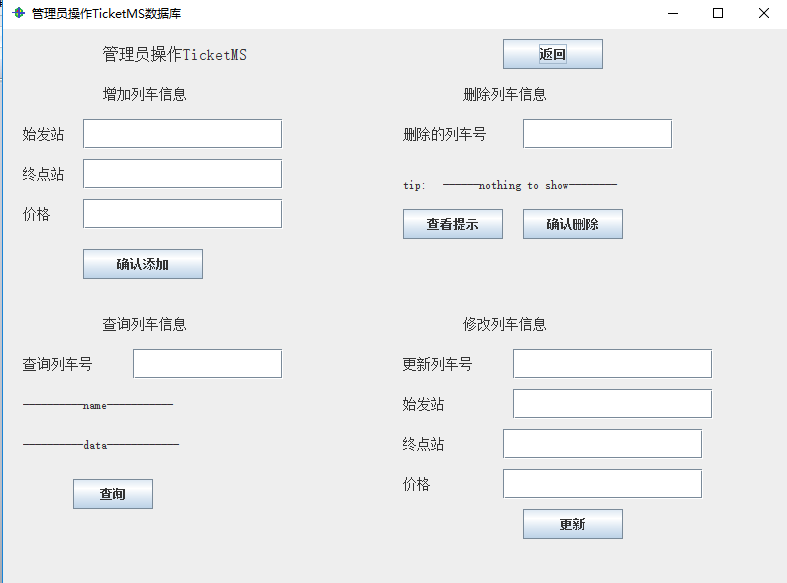
5.点击返回，到达登录界面，注册一个管理员账号：



6.选择功能（管理列车信息，管理车票信息）

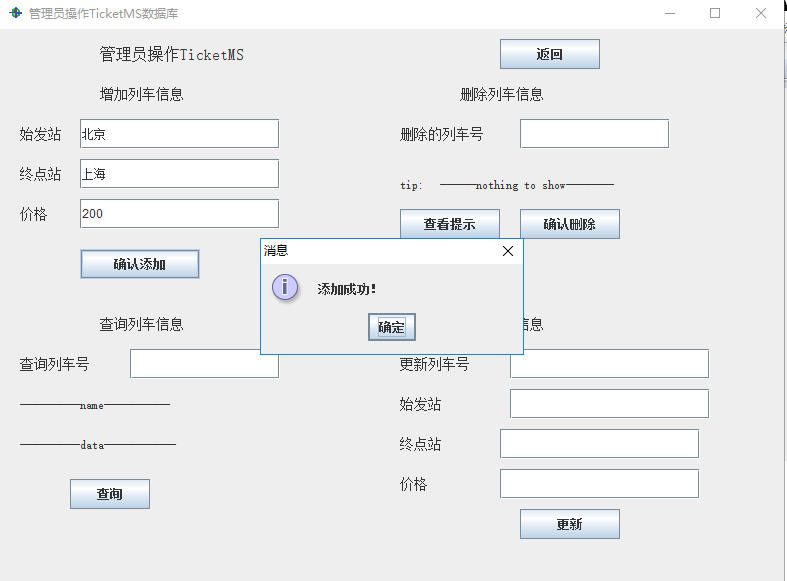


7.选择管理列车信息：

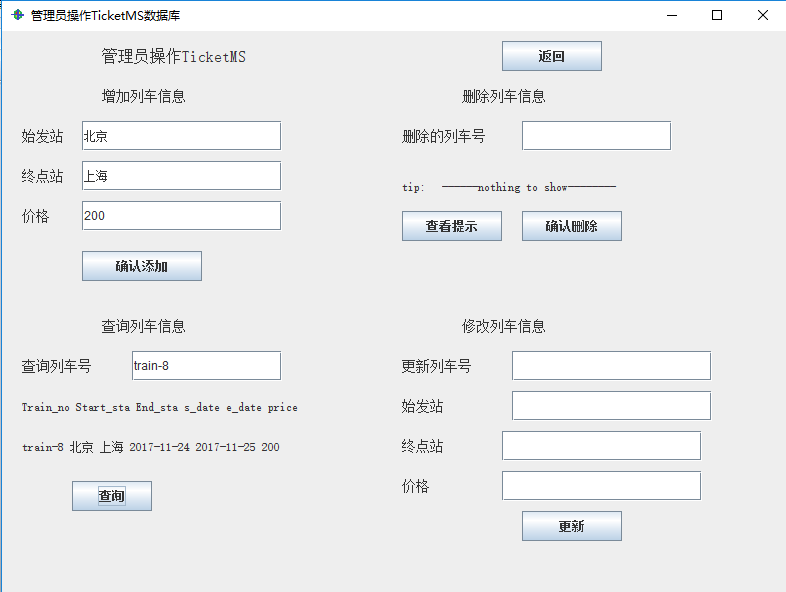


8.左上角增加列车信息（始发站：北京 ，终点站：上海 ，价格：200）->确认添加

系统自动分配唯一的列车号：train-8



9.左下角查询列车信息（输入列车号：train-8）->显示结果如左下角：



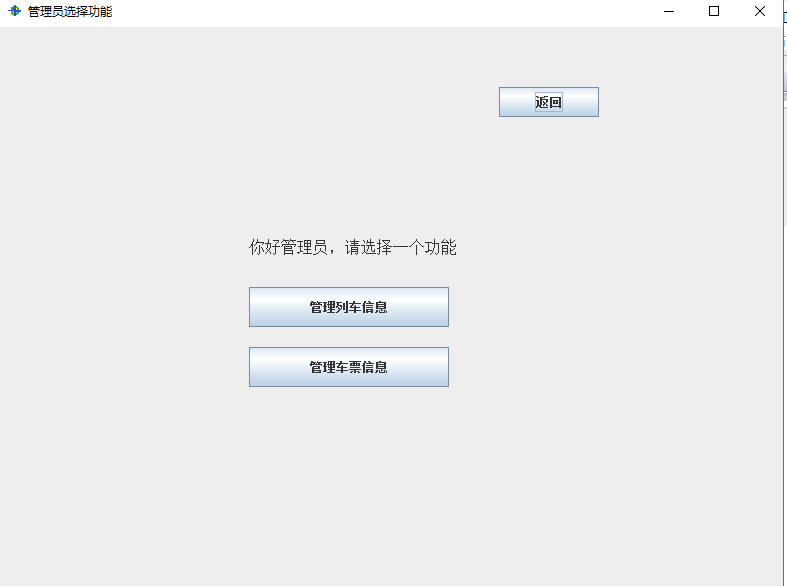
10.右上角删除列车信息（输入列车号：train-8）->查看提示->确认删除

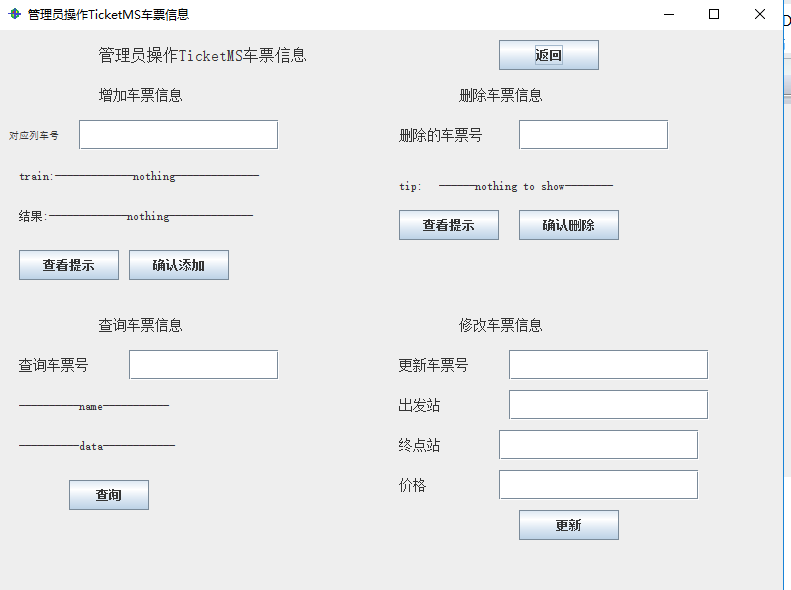


11.右下角更新列车信息（更新列车号：train-7,更新始发站终点站价格）



12.点击返回，到选择功能界面->选择管理车票信息：





操作方式和管理列车信息类似

**具体情况，运行可执行jar包：**TicketMS.jar

**批阅者：**

**批阅日期：**

**实验成绩：**

**批注：**