



华中科技大学

数据库系统原理实践报告

项目名称：国内航班票务模拟系统

姓 名：王俪晔

专 业：计算机科学与技术

班 级：CS1805

学 号：U201814643

指导教师：左琼

分数	
教师签名	

2021 年 7 月 6 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
2 SQL 语言实践（SQL SERVER 版）	2
2.1 任务要求.....	2
2.2 完成过程.....	2
2.2.1 创建数据库	2
2.2.2 创建表 and 主外码约束	2
2.2.3 数据更新	2
2.2.4 查询人流量大于 30 的地点	2
2.2.5 接续行程	3
2.2.6 充珉瑶和贾涵山的行程情况	3
2.2.7 去过所有地点的人员	3
2.2.8 隔离点的视图现状	3
2.2.9 房间数第二多的隔离点	3
2.2.10 创建触发器.....	3
2.2.11 创建并使用自己创建的函数.....	3
2.3 任务总结.....	3
3 DBMS 软件功能学习（SQL SERVER 版）	4
3.1 任务要求.....	4
3.2 完成过程.....	4
3.2.1 验证被参考的一定是键	4
3.2.2 验证不设主码且插入重复元组.....	4
3.2.3 数据和日志文件的备份	4
3.2.4 用户与权限	5
3.3 任务总结.....	5

4 国内航班票务模拟系统	6
4.1 系统设计目标	6
4.2 需求分析	6
4.2.1 功能需求的五方面	6
4.2.2 静态实体信息	7
4.2.3 航司应用系统的功能需求	7
4.2.4 动态实体信息	8
4.2.5 乘客应用系统的功能需求	8
4.2.6 模拟时间和模拟实飞	10
4.2.7 乘客订票的性能需求	10
4.2.8 数据完整性需求	10
4.2.9 数据流图	11
4.3 总体设计	12
4.3.1 C/S 架构	12
4.3.2 功能模块	12
4.4 数据库设计	16
4.3.3 ER 图设计	16
4.3.4 基表	17
4.3.5 视图与授权	20
4.5 具体实现	20
4.4.1 实现工具	20
4.4.2 DBSQL: 连接池和事务	21
4.4.3 自动模拟: 触发器	21
4.5.4 票务小工具: 函数与存储过程	23
4.5.5 函数	错误!未定义书签。
4.5.6 存储过程	错误!未定义书签。
4.5.7 连接池	错误!未定义书签。

4.5.8	事务	错误!未定义书签。
4.5.9	XX	29
4.6	系统测试.....	38
4.7	系统设计与实现总结.....	55
4	课程总结	56
	附录.....	58

1 课程任务概述

1. 通过上机实践，熟悉常用大型数据库管理系统，如 SQL Server 和 MySql，了解 DBMS 体系结构。
2. 熟练掌握 SQL 语言的数据定义、数据操纵、数据查询和数据控制。
3. 熟悉数据库应用系统的设计方法和开发过程。

2 SQL 语言实践（SQL Server 版）

2.1 任务要求

1. 熟悉 Linux 系统下 SQL Server 的特点。
2. 学习如何创建数据库。
3. 学习如何创建表，并为表设置主码、外码，并定义主外码之间的参照完整性。
4. 学习使用 SQL 语言完成数据的增删改。
5. 练习单表查询、嵌套查询，完成多种情境下的数据查询任务。
6. 学习创建符合应用要求的触发器。
7. 学习创建并使用自定义函数。

2.2 完成过程

2.2.1 创建数据库

实现：仅需一句 “create database covid19mon;”

收获：第一句付诸实践的 SQL 语句，意识到每句需要 ‘;’ 作为结束符。

2.2.2 创建表和主外码约束

实现：数据定义语言。

收获：

1. 熟悉了数据库的实体完整性和参照完整性。
2. SQL 查询开始前，先用 “use covid19mon;” 确定所使用的数据库。
3. 对有参照关系的两表，被参照的表一定要先被创建。

2.2.3 数据更新

实现：使用 insert、delete、update。

收获：熟悉了 insert、delete、update 等数据操纵动词与句式。

2.2.4 查询人流量大于 30 的地点

实现：如图 2.1 所示。

```
select location_name, count(*) as visitors
from itinerary join location on location.id=itinerary.loc_id
group by loc_id, location_name
having count(*) > 30
order by count(*) desc, location_name asc;
```

图 2.1 查询人流量大于 30 的地点

收获：group by 后的关系必须包含 select 中的所有属性列，否则无法选择。

2.2.5 接续行程

实现：将两张同类表笛卡尔连接，再选择投影。

收获：当选择时的二元操作作用于同一实体域的二元，可采用同表连接，两表通过别名区分。

2.2.6 充珉瑶和贾涵山的行程情况

实现/收获：查询人员的行程时，采用左外连接，确保没有行程记录的人员不被丢失。

2.2.7 去过所有地点的人员

收获：SQL 语句的全称量词逻辑利用存在量词和非逻辑等价实现，体现为两重“not exists”。

2.2.8 隔离点的视图现状

收获：学习视图的建立，感受数据库三层模式架构中的外模式映射。

2.2.9 房间数第二多的隔离点

收获：

1. SQL Server 特有的“case when then”。
2. 派生表查询。

2.2.10 创建触发器

收获：

增进对触发器的理解，即当表 A（即将）执行更新时，将触发一系列指定操作，该操作不一定仍在表 A。

2.2.11 创建并使用自己创建的函数

收获：

1. 创建的 SQL 函数与过程性编程语言的函数类似，也有单个指定类型的返回值。
2. 使用函数时，可用在 select 后，也可用在 where 后。

2.3 任务总结

本次实验涵盖了 SQL 的数据定义、数据操作、数据查询，通过实践验证了理论课的所学，既加深了理论理解，又熟悉了具体 SQL 语法，收获颇丰。

3 DBMS 软件功能学习（SQL Server 版）

3.1 任务要求

1. 设计实验验证实体完整性和参考完整性。
2. 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。
3. 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。

3.2 完成过程

3.2.1 验证被参考的一定是键

实现：如图 3.1 所示，当尝试对表 `itinerary` 的非主码列建立起被参考关系，DBMS 会自动显示语义错误。

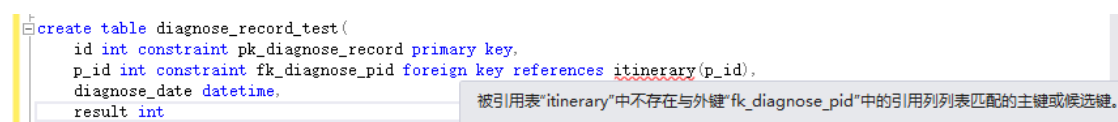


图 3.1 验证被参考的一定是键

收获：加深了对参考完整性的理解。

3.2.2 验证不设主码且插入重复元组

实现：建立一个不含主码的表成功后，插入两条完全一样的数据成功后，再尝试对该表指定数据 `update`，但结果如图 3.2 所示，两条数据完全无法区分开，只能同时被更新。

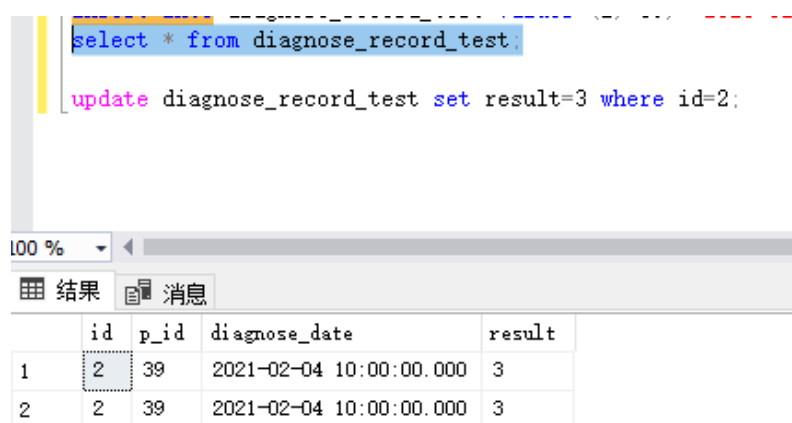


图 3.2 验证无主码时的插入与更新

3.2.3 数据和日志文件的备份

实现：如图 3.3 所示，SQL Server 有便捷的备份 GUI 选项，可以直接备份

和选择某一现有备份进行还原。



图 3.3 SQL Server 的备份 GUI 界面

3.2.4 用户与权限

实现：在新增的数据库上新建一个普通用户，通过 DBA 用户 revoke 该普通用户映射到“covid19”的权限。之后再以该普通用户身份登录 DBMS，尝试对 covid19 的某表进行查询，出现如图 3.4 所示的报错信息。

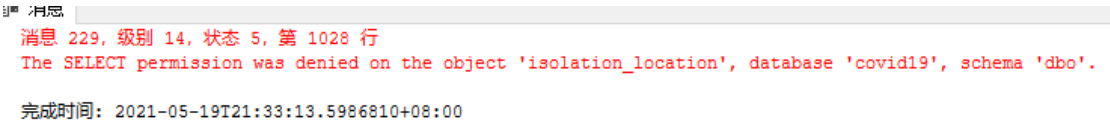


图 3.4 无权限导致的查询失败

3.3 任务总结

将任务一与任务进行对比，任务一专注于对数据库的应用层面，任务二更专注于对数据库的系统层面的使用，包括用备份机制完成系统故障恢复、用自主存取控制达到一定的数据库安全性。

两类先导实验结合起来，从外到里、由浅至深地引领我们去了解和使用大型 DBMS，为后续任务三设计与实现小型数据库应用系统打下基础。

4 国内航班票务模拟系统

4.1 系统设计目标

国内航班是一个广阔的、活跃的市场，涉及到省市、机场、航司和乘客等多个实体与实体间的联系，并且每时每刻都有大量的实时订单数据、实时航班飞行数据的产生与业务处理。

航班的核心问题之一是票务，相关操作既涉及到乘客方的航班筛选、订单创建、订单支付、订单取消等，又涉及到航司的航班提供、价格制定、值机选座等。

在这样的背景下，一个可用的国内航班票务系统需要与数据库密切、合理地结合起来，并且至少达到下列五点目标：涵盖静态实体、处理动态实体、乘客应用系统、航司应用系统和模拟实飞情况，具体阐释见 4.2.1。

4.2 需求分析

4.2.1 功能需求的五方面

1. 涵盖静态实体

如省份、城市、机场、航司、机型等。这些实体对航班系统必不可少，但同时又几乎固定不变。在系统建立之初将会这些静态实体数据直接导入系统，后期的少量增删改操作由应用系统管理员执行。

2. 处理动态实体

如订单、机票等实体。这些实体与票务紧密相关，随时有可能被增删改查。在系统内应有完善的、角色区分的业务逻辑去根据用户操作增删改这类实体数据。

3. 乘客应用系统

乘客是票务的需求方，主系统应当为乘客的购票业务需求专供一个乘客应用子系统。除购票业务外，子系统还涵盖了购票用户、乘客信息等实体数据，并交由乘客用户自身对其增删改管理。

4. 航司应用系统

各家航司是票务的提供方，具体提供形式是一定期限内的具体航班和航班在具体某天的实飞。为航司提供的应用子系统涵盖了航班、实飞等实体数据，并交由航司用户自身对其增删改管理。

5. 模拟实飞情况

真实的航班系统与具体的实飞情况密不可分。为了最大程度模拟实际，本系统被设计运行于模拟时间中，每当模拟时间后移，本系统需要自动修改已起飞的实飞数据。

除此之外，鉴于测试系统时，用户操作下单是唯一的订单和机票产生之源，

为了使查看具体某趟已飞实飞的座票数据时更丰富，系统会在每趟实飞起飞时随机产生一些虚拟票。

4.2.2 静态实体信息

1. 省份

对航空系统来说，省份实体仅需知晓省份名。

2. 城市

对航空系统来说，需要知晓城市名和城市所属省份。

3. 机场

对航空系统来说，需要知晓机场名、机场三字码和机场所属城市。

4. 航司

对航空系统来说，需要知晓航司名。

对乘客应用子系统来说，需要知晓起飞前 4 小时以内退订手续费比例和起飞前 24 小时以内退订手续费比例。

对于航司应用子系统来说，需要保管航司票务管理员登录系统的密码。

5. 机型

对航空系统来说，需要知晓机型名。

对于两个字系统的票务逻辑来说，需要知晓该机型的头等舱座位数、商务舱座位数和经济舱座位数。

4.2.3 航司应用系统的功能需求

1. 商业权限

航司的内部信息属于商业机密，不能对其他航司可见，所以每个航司票务管理员登录本系统后，仅能查看和管理属于本司的航班、实飞和机票信息。

2. 管理航班

航班作为实体，包含了唯一的航班名、所属航司、所用机型、出发机场、抵达机场、预计出发时间、预计抵达时间、准点率、有效期始、有效期末、头等舱平均价格、商务舱平均价格和经济舱平均价格等信息。

航司作为航班的提供方，航司应用子系统理应满足所有对航班的增删改需求。考虑到系统复杂性和数据库技术的重复涵盖，本系统仅提供新增航班和延长航班有效期等增改操作。

3. 管理实飞

实飞是本系统提出的界定名词，一条航班在有效期内的每一天都有一趟实飞，每趟实飞是独属于一条航班同时又能独立定价的实体，包含了所属航班、起飞日期、是否已起飞、实际起飞时间、实际降落时间、实际头等舱价格、实际商务舱价格、实际经济舱价格。

实飞的产生不可直接手动添加，而是跟随航班而生成。当有新增航班或原有航班有效期延长时，系统会自动生成数趟对应的实飞。同样出于复杂性考虑，本系统的修改操作仅提供调整实飞价格的功能。

4. 查看机票

选定航班后可查看所有实飞，相应的，选定实飞后可查看所有机票。由于机票的管理属于系统与乘客用户操作之间的自动逻辑，航司管理员只能查看不能操作。

综上，航司应用系统的实体之间联系如图 4.1 所示。



图 4.1 航司应用子系统的实体联系

4.2.4 动态实体信息

1. 订单

订单作为实体，包含所属实飞、舱位类型、下单用户、订单创建日期、订单创建时间、有效状态、所含机票数量等信息。其中有效状态有四类：待支付、超时未支付、已支付未取消、支付后已取消。订单在用户预定成功后自动创建。

2. 机票

机票作为实体，包含乘客身份证、所属订单、取票码、座位号等信息。机票在所属订单创建成功后自动创建，取票码在订单支付成功后自动生成，座位号在乘客值机成功后自动生成。

4.2.5 乘客应用系统的功能需求

1. 修改密码

登录用户作为实体，包含用户名、密码、银行卡号、剩余金额等信息。银行卡绑定前为空，剩余金额用非空绑定银行卡充值之前为 0。此外，能够修改密码是基本的账户使用需求。

2. 代购乘客

参考主流订票系统，本系统中登录到乘客应用系统的用户只是机票的代购人，而不是机票的最终乘客。所以乘客子系统的第一个功能是对代购乘客名单的管理，用户通过输入乘客的身份证号等基本的静态信息，添加代购乘客。一个系统用户可以添加多个代购乘客，但是当试图新增的代购乘客的身份证号已存在于本系统中，却与其他的已存基本信息不吻合时，会添加失败。

代购乘客作为实体，包含身份证号、姓名、出生年份和性别等信息。此外，

代购乘客与登录用户之间属于多对多关系。

3. 金额充值

为尽可能模拟现实情况，引入对金额的管理是必然的。用户绑定某张银行卡后（绑定过程略去验证），可充值金额到自己的购票系统账户。后续订单支付会扣除金额，订单取消可能会返回金额。

4. 行程查询

行程包含出发省市机场、到达省市机场、起飞时间，用户选定行程后，系统能提供符合行程要求的所有实飞信息。乘客应用系统仅能查询明天以后的实飞行程。

5. 机票预定

行程查询后，用户选定其中某趟实飞的某个舱位即可预定机票。机票预定前，该用户至少已有一个代购乘客；机票预定时，用户可在自己的代购乘客名单中多选，选中一名乘客即为其预定一张该趟实飞的该舱位的机票。考虑到经济能力与信用问题，剩余金额为 0 的用户不能预定机票。

机票预定成功，系统会为其创建一项订单和数张机票。之后，用户需要在 15 分钟内完成订单支付，否则订单会因超时失效。

6. 订单支付

机票预订，即订单和机票创建后，用户能通过该订单查询到包含的数张机票信息，但尚不能看到各机票的值机取票密码，因为尚且没有完成订单支付。用户选中一项待支付的订单，如果剩余金额足够，则可直接支付成功。支付成功后即可查询到各张机票的值机取票密码。

7. 值机取票

实际情况中，线下值机取票首先要抵达出发机场，再通过自己航班属于哪家航司来找到机场中对应的值机台，最后通过向值机人员提供乘客个人信息认证和机票信息认证完成值机，获得自己的座位信息。为模拟该过程，本系统的值机过程，需要用户输入出发机场名、航司名、乘客身份证号、值机取票密码，若四项信息符合，则值机成功，系统为其自动分配座位号。

8. 订单取消

在订单创建后，某张机票值机成功或该趟实飞起飞前，用户均能取消订单。综合考虑实际情况中高昂的机票退订代价，本系统中的每家航司均有各自的起飞前 4 小时内退订手续费和起飞前 24 小时内退订手续费。退订成功后，该订单和所含机票均失效，用户若已支付甚至已进入退订含手续费时限，账户余额会增加相应金额。

9. 航班查询

当我们获知亲朋好友乘坐的航班号后，输入航班号，即可查询到该航班的起

飞地点、降落地点、预计起飞降落时间和所属航司等基本信息，方便接机等。

综上所述，围绕票务这一核心功能，乘客应用子系统的业务流程图如图 4.2 所示。

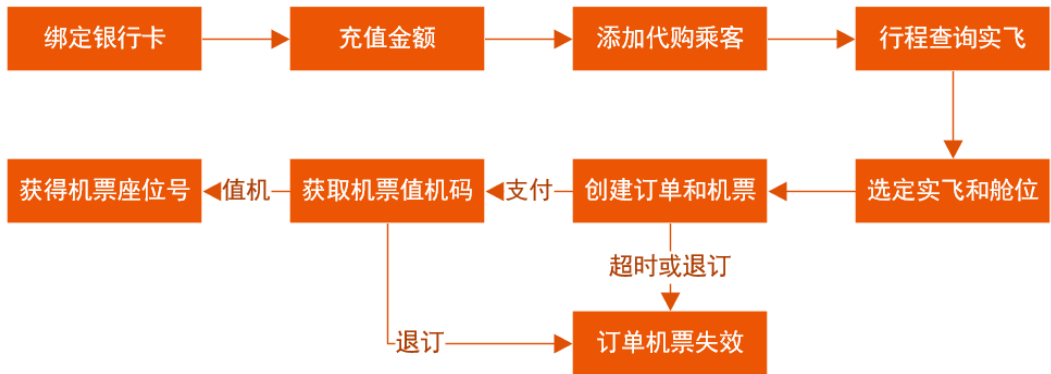


图 4.2 乘客应用子系统的票务流程图

4.2.6 模拟时间和模拟实飞

本系统的运行于模拟的日期和现实的时间，无论是未登录游客、系统管理员、航司票务管理员还是乘客用户，使用系统时均可正向调模拟日期。由于本系统在航班和订票两方面都有很多日期限制和时间限制，模拟日期是为了便于调试。

模拟日期具有持久性，再结合常见的系统日志功能，故每一次系统的模拟日期前调都会被记录入数据库。

4.2.7 乘客订票的性能需求

本系统在处理订票逻辑时，会面临多方用户在同一时刻对同一实飞的同一舱位的订票请求。其中单个用户的请求票数量均小于剩余票数，但多方总和数量超过了剩余票数。为了票务逻辑正确，本系统需要满足以下性能要求：在创订单对象相同时，系统会依靠数据库的锁将并行的请求串行化处理；锁要在可完成任务的前提下，尽可能设置在最小的范围里，以防订票延迟过长。

4.2.8 数据完整性需求

以上涉及到了省份、城市、机场、航司、机型、航班、实飞、用户、乘客、订单、机票等 11 个实体，以及用户与乘客间的多对多关系，均需要满足实体完整性。

参照完整性方面，存在以下参照关系：城市的所属省份参照省份；机场的所属城市参照城市；航班的所属航司参照航司；航班的所用机型参照机型；航班的起飞机场和抵达机场参照机场；实飞的所属航班参照航班；订单的所属实飞参照实飞；订单的下单用户参照用户；机票的所属订单参照订单；机票的所属乘客参照乘客；用户与乘客的多对多关系同时参照用户和乘客。

此外，为贴合现实情况和应用需求，存在下列对数据的完整性要求：航班的准点率是 0~0.99 的两位小数；用户的银行卡号为 16 位；乘客的身份证号为 18 位；订单的舱位类型为 1 或 2 或 3，依次对应头等舱、商务舱、经济舱；用户名用于登录，故需要非空且独一无二。

4.2.9 数据流图

整个系统的数据流图如图 4.3 所示。

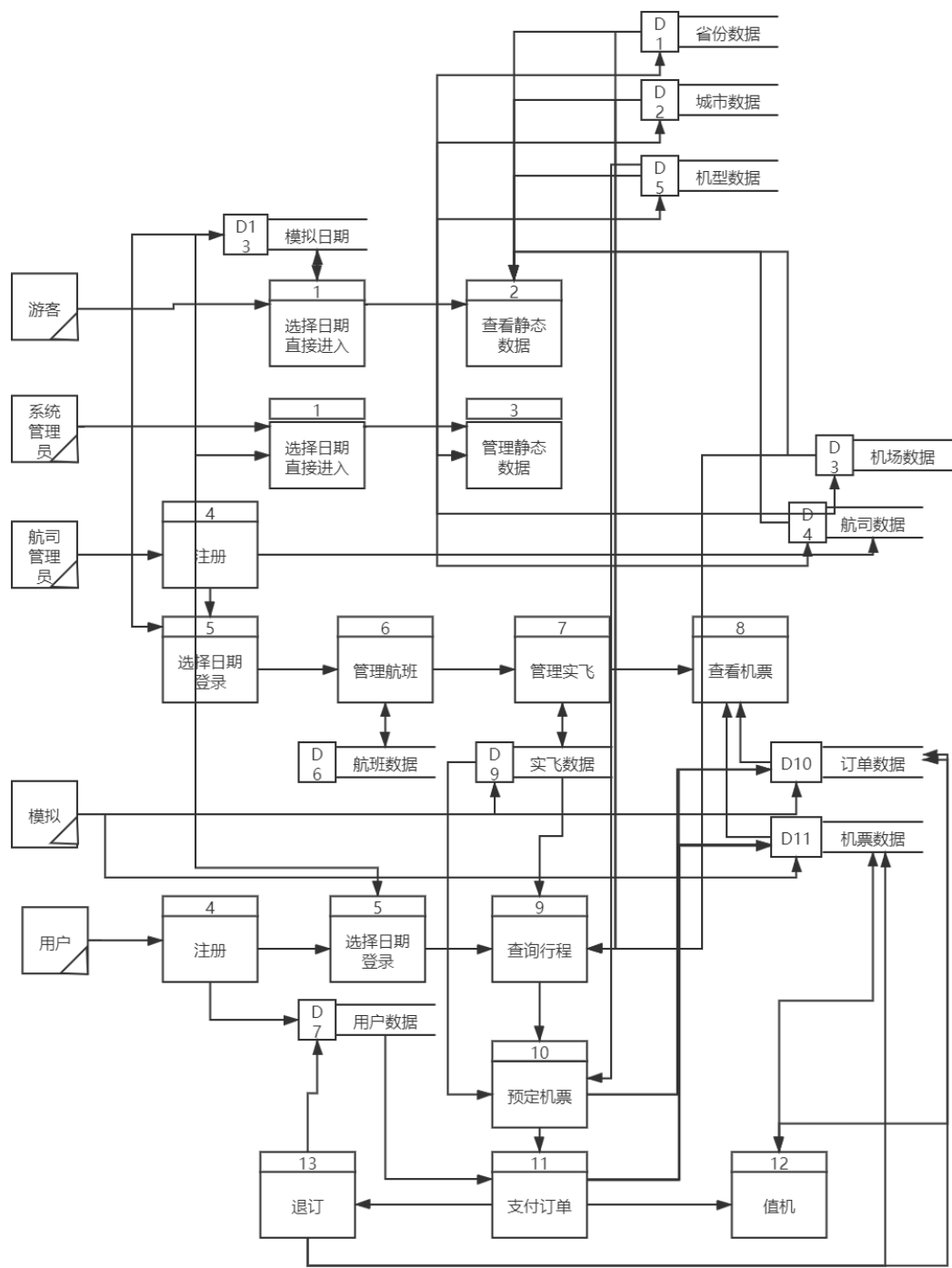


图 4.3 系统数据流图

4.3 总体设计

4.3.1 C/S 架构

本系统采用 Client/Server 架构，应用系统主体为 Client，本系统所用数据库 airdb 为 Server。系统支持多个 Client 同时连接，各 Client 独立地获取用户操作、向 Server 请求数据、处理基本业务逻辑和向用户展示输出。系统 C/S 架构图如图 4.4 所示。

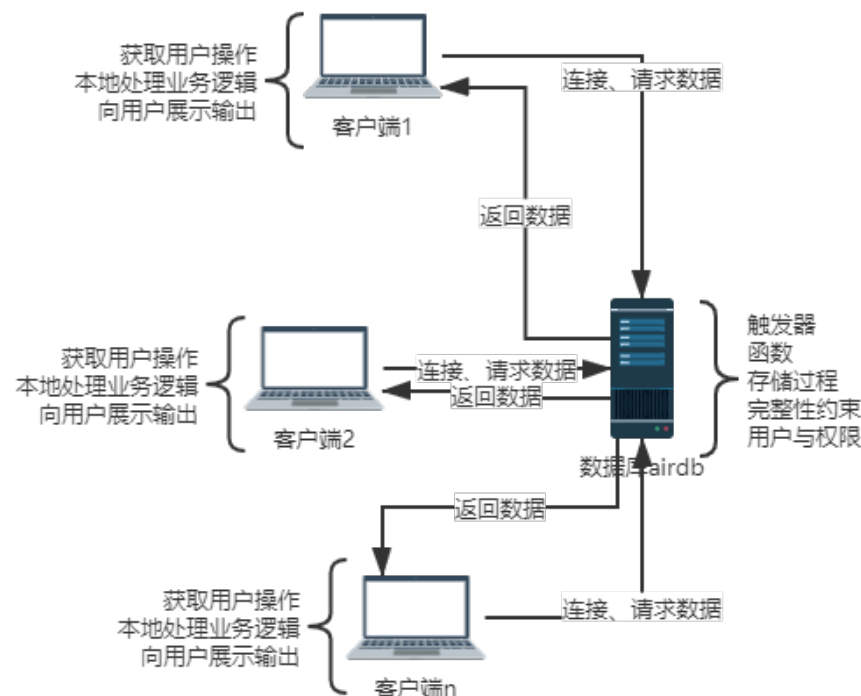


图 4.4 C/S 架构图

4.3.2 功能模块

此处介绍客户端的功能模块划分。功能模块首先包含以下六个个主页面功能模块：welcome、visitor、administor、login、user、company，依次为系统欢迎首页、游客模式主页面、系统管理员模式主页面、登录主页面、乘客应用子系统主页面和航司应用子系统主页面。

其次包含两个用于获取用户输入的对话框模块，inputdialog 和 selectdialog，分别能够获取数项文本输入和获取数项多选输入。此外，还包括专用于向数据库请求数据的功能模块 dbsql。以下是对上述模块的具体说明，包含 GUI 模块最终的成品效果展示。

1. welcome

welcome 是系统欢迎首页模块。首先，使用者能够选择模拟日期，或者保持当前模拟日期不变。之后，用户可选择“游客”、“系统管理员”和“登录”，分

别进入 visitor、administor、login 这三个页面。为便于单机模拟多客户端同时操作，welcome 界面不会在其他页面出现后消失，所以使用者可通过 welcome 同时打开多个系统使用页面。最终 welcome 效果图如图 4.5 所示。



图 4.5 welcome 成品图

2. visitor

visitor 是系统的游客页面，用于显示省份、城市、机场、航司、机型等静态数据。成品效果如图 4.6 所示。



图 4.6 visitor 成品图

3. administer

administer 是系统管理员页，包含对静态数据的增删改。成品如图 4.7 所示。



图 4.7 administer 成品图

4. login

login 是航司/乘客应用子系统的登陆界面，两个子系统有两套独立的账户密码数据库，能够分别注册无重名的账户。航司注册意味着在原有航司数据的基础上增加了新的航司。根据选项，登录成功后会进入 user 界面或 company 界面。最终成品效果如图 4.8 所示。

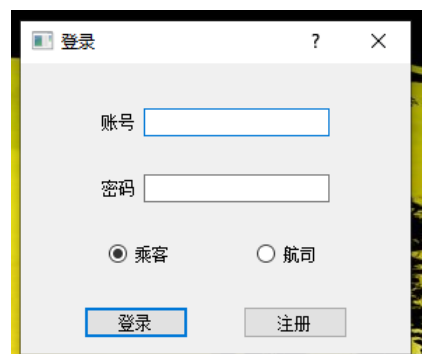


图 4.8 login 成品图

5. user

user 是乘客应用子系统界面，包含了 7 个 tab 页和两个功能按钮，具体包含如 4.2.5 所示的各项功能。最终成品图如图 4.9 所示。

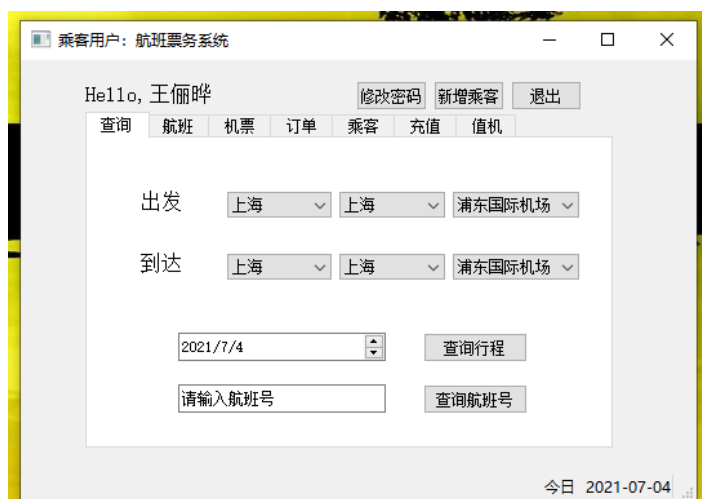


图 4.9 user 成品图

6. company

company 是航司应用子系统界面，包含了 3 个 tab 页和一个功能按钮，具体包含如 4.2.3 所示的各项功能。最终成品图如图 4.10 所示。



图 4.10 company 成品图

7. inputdialog

inputdialog 是一个用于获取多项文本输入的对话框功能模块，可向其传入两个长度相等的列表，分别是各项文本输入前的标签文字和文本输入框内的原有数据，其中原有数据可为空。最终效果图如图 4.11 所示。

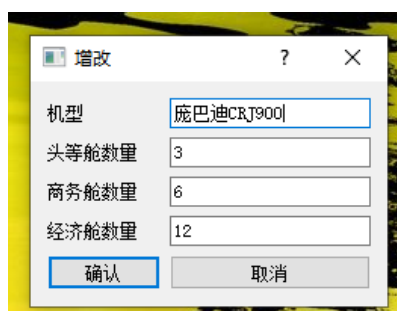


图 4.11 inputdialog 成品图

8. selectdialog

selectdialog 是一个用于获取多项选择的对话框功能模块，可向其传入存放各多选项的文本内容的列表。最终效果图如图 4.12 所示。

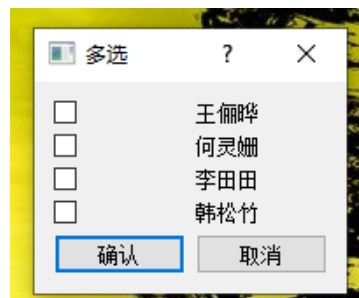


图 4.12 selectdialog 成品图

9. dbsql

dbsql 是唯一一个与 GUI 无关的功能模块，是对连接数据库、请求数据、事务执行和异常处理等功能的封装。

综合上述模块划分，模块之间的依赖关系如图 4.13 所示。

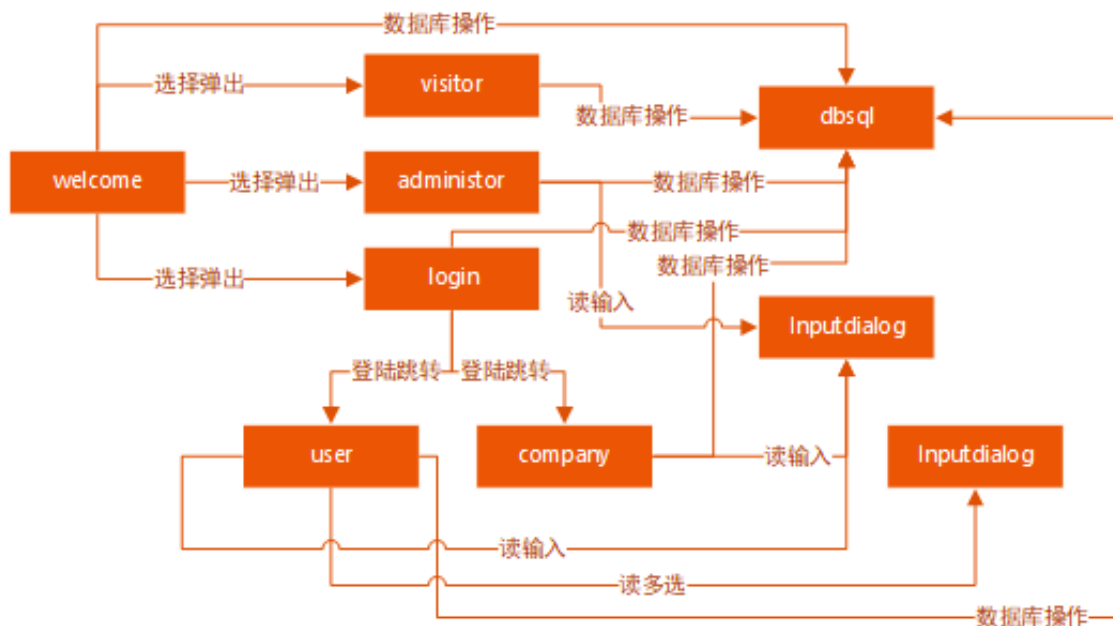


图 4.13 系统模块依赖图

4.4 数据库设计

4.3.3 ER 图设计

根据之前对实体和联系的分析，本系统的数据库 ER 图如图 4.14 所示。

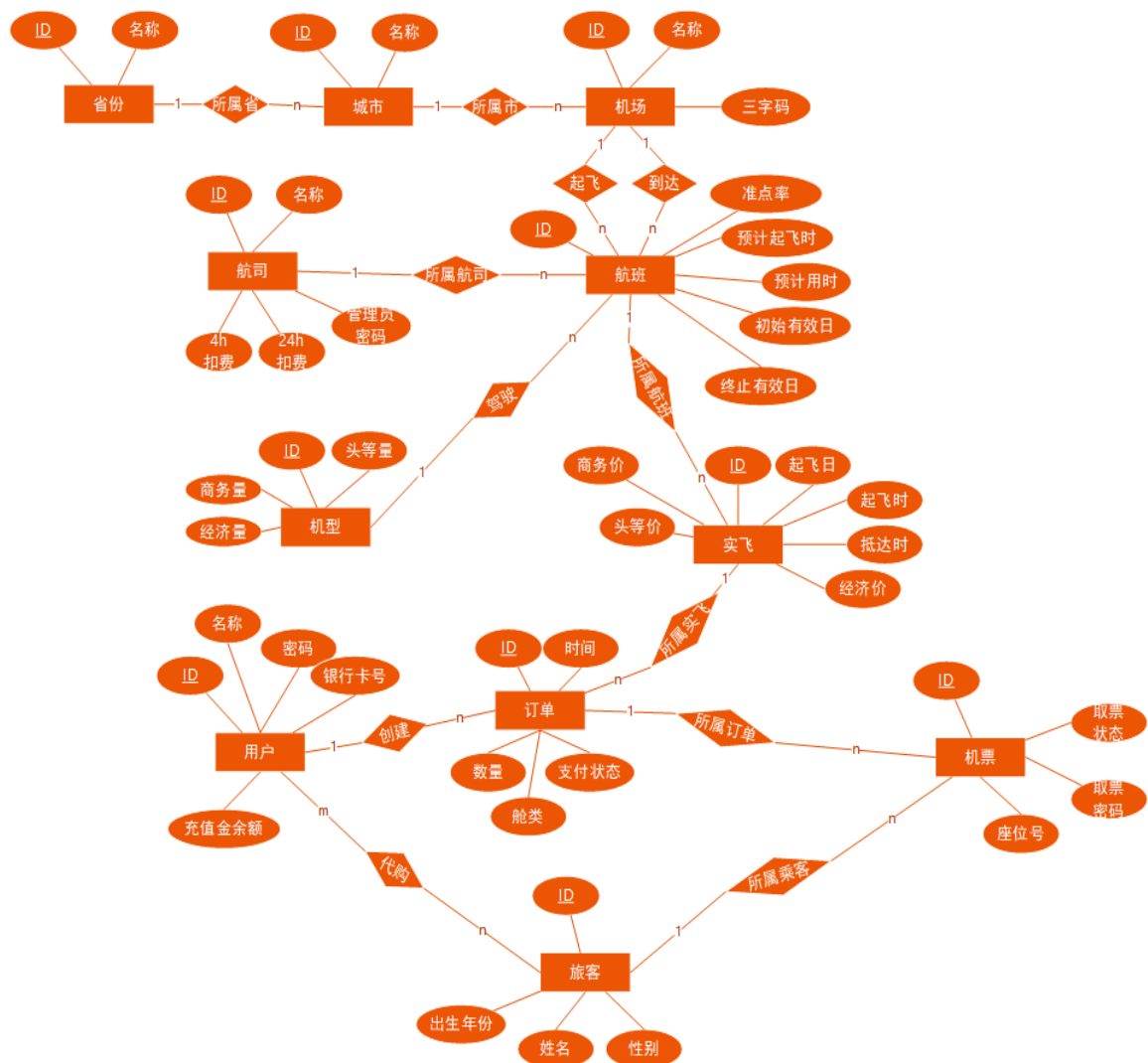


图 4.14 数据库 ER 图

4.3.4 基表

根据 ER 图，结合 4.2.8 完整性分析，总共建立 13 个基表，包含 11 个实体基表、1 个日志表和 1 个多对多联系表，具体信息如下。

1. 省份表 province

id int not null auto_increment primary key,
name char[20] unique not null --省份名

2. 城市表 city

id int not null auto_increment primary key,
pid int not null foreign key references province(id), --省份
编号
name char[30] not null --城市名

3. 机场表 airport

id int not null auto_increment primary key,

- | | | |
|------|---|--------|
| name | char[30] unique not null, | --机场名 |
| cid | int not null foreign key references city(id), | --城市编号 |
| code | char[10] unique not null | --三字码 |
4. 航司表 company
- | | | |
|--------|--|------------------------|
| id | int not null auto_increment primary key, | |
| name | char(20) not null, | --航司账户名 |
| passwd | char(20) not null, | --航司账户密码 |
| debit1 | float(2,2), | --4 小时以内退订收费比例 debit1 |
| debit2 | float(2,2) | --24 小时以内退订收费比例 debit1 |
5. 机型表 aircraft
- | | | |
|------|--|----------|
| id | int not null auto_increment primary key, | |
| name | char(20) not null, | --机型名 |
| numf | int | --头等舱座位数 |
| numc | int | --商务舱座位数 |
| numy | int | --经济舱座位数 |
6. 航班表 flight
- | | | |
|----------|---|----------|
| id | int not null auto_increment primary key, | |
| acid | int not null foreign key references aircraft(id), | -- 机型 |
| 编号 | | |
| cid | int not null foreign key references company(id), | -- |
| 航司编号 | | |
| aid_f | int not null foreign key references airport(id), | -- 起 飞 |
| 机场编号 | | |
| aid_t | int not null foreign key references airport(id), | -- 到 达 |
| 机场编号 | | |
| off_due | time | --预计起飞时刻 |
| land_due | time | --预计到达时刻 |
| begin | date | --执行首日 |
| end | date | --执行末日 |
7. 实飞表 fly
- | | | |
|----------|--|----------|
| id | int not null auto_increment primary key, | |
| fid | int foreign key references flight(id), | --航班号 |
| fdate | date, | |
| flew | bool, | |
| off_real | time, | --实际起飞时间 |

land_real	time,	--实际到达时间
pricef	int,	--头等舱价格
pricec	int,	--商务舱价格
pricey	int,	--经济舱价格
8. 用户表 user		
id	int not null auto_increment primary key,	
name	char(20) not null,	--账号名
passwd	char(20) not null,	--账户密码
bkid	char(16),	--十六位银行卡号
money	int	--充值金余额
9. 乘客表 person		
id	int primary key,	--身份证号
name	char(20) not null,	--真实姓名
birth	year	--出生年份
sex	int	--性别
10. 订单表 porder		
id	int not null auto_increment primary key,	--订单号
ffid	int foreign key references flight(id),	--实飞号
stype	int,	--舱类
uid	int foreign key references user(id),	--下单用户
号		
otime	datetime not null,	--创建时间
amount	int not null,	--数量
paid	bool	--支付状态
11. 机票表 ticket		
id	int not null auto_increment primary key,	--机票号
pid	int foreign key references person(id),	--乘客身份
证号		
oid	int foreign key references order(id),	--订单号
Seat	char(10),	--座位号
Passwd	char(16),	--十六位取票密码
Fetchd	bool	--取票状态
12. 代购表 agent		
pid	int foreign key references person(id),	--身份证号
uid	int foreign key references user(id),	--用户号

primary key(pid, uid)

13. 日志表 datelog

today date --最新虚拟日期

4.3.5 视图与授权

为了达到航司之间商业保密性的需求，我们需要用到数据库的视图和授权机制。

视图方面，需要建立三个视图：cad_flight、cad_fly 和 cad_ticket。上述视图映射与当前用户有关，将分别从基表 flight、fly、porder 和 ticket 中映射所属航名与当前数据库用户同名的数据项。

授权方面，首先需要创建数据库角色“company_ad”，将在视图 cad_flight、cad_fly 和 cad_ticket 上的所有权限赋予该角色；再为每个航司创建一个同名数据库用户，并将“company_ad”角色赋予所有航司用户，并设置为默认角色。

综上，本数据库的视图与授权机制如图 4.15 所示。

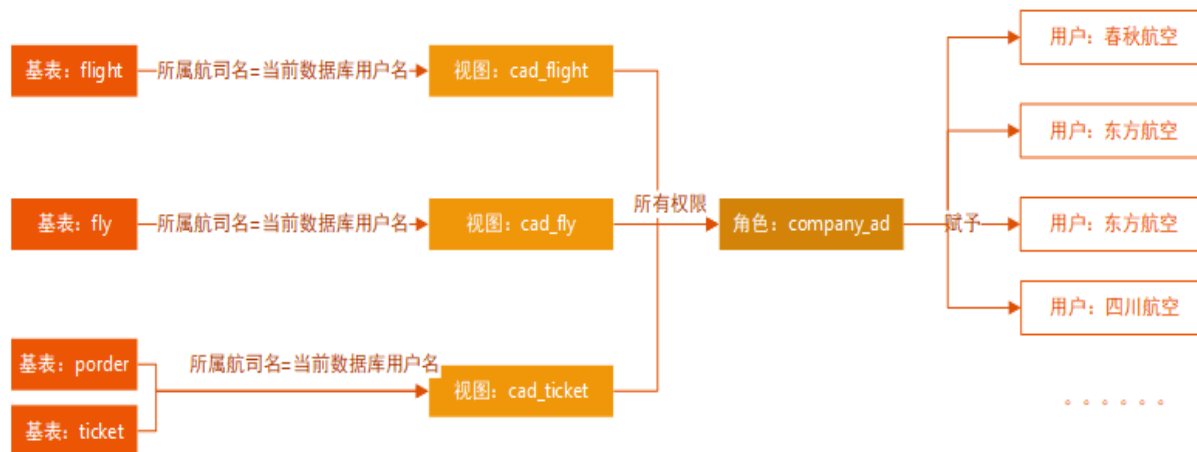


图 4.15 授权与视图映射示意图

由此，每个航司登录系统时，均使用同名用户，均只能查询和操作自家航班的航班数据、实飞数据和机票数据。

4.5 具体实现

4.5.1 实现工具

1. 编程语言：python 3.8
2. IDE：pycharm 社区版
3. 数据库：MySQL 8.0
4. 数据库编程模块：pymysql
5. GUI 编程模块：pyqt
6. 数据库连接模块：DBUtils.PersistentDB

4.5.2 dbsql 模块：连接池和事务

dbsql 模块是对连接池分配和事务操作的封装。初始化一个 dbsql 对象时，需要传入一个已设置好的 PersistentDB 类。数据库连接模块 DBUtils.PersistentDB 的 PersistentDB 类提供了连接池方式，能够连接指定类数据库服务器。该类与经典的 PooledDB 类似，均能够快捷地提供连接池，其优势在于多线程安全，即该模块不会将曾分配给某线程的连接再分配给其他线程。

execsql 是 dbsql 中唯一的方法，调用时需要传入一个存放 sql 语句的列表。execsql 在每次执行时，会先用 PersistentDB 类获取一个连接，再由该连接获取一个游标。之后，所有传入的 sql 语句按序在一个事务中执行，最终 execsql 返回一个列表，存放着每条 sql 语句的返回结果。

Execsql 中事务的具体实现结合了 python 的异常处理机制。首先将变量“autocommit”设为 0，防止每条语句自动提交；再依次执行每条 sql 语句；如果所有语句顺利执行完毕，则提交事务、关闭游标、关闭连接；如果中途出现异常，则事务回滚、对话框报错。

4.5.3 自动模拟：触发器

系统中存在四处自动模拟，非常适合分别用一个触发器完成，具体需求与设计如下。

1. 日期插入触发模拟实飞

每当系统模拟日期往后调，系统就会自动模拟实飞起飞，修改起飞日期在最新模拟日期之前的实飞，包括已飞状态、实际起飞时间和实际落地时间等数据项。

为此，设置触发器 datelog_insert，在基表 datelog 被 insert 后，对所有触发行依次执行以下三项操作：

对 fly 基表，若某元组的起飞状态为否，且起飞日期小于新插入行的日期，则将其起飞状态设置为是；

对 fly 基表，若某元组的起飞状态为是，且实际起飞时间为空，则结合随机数、该实飞所属航班的预计起飞时间和准点率，计算并设置该元组的实际起飞时间，具体计算公式为：

$$\text{实际起飞时间} = \text{预计起飞时间} +$$

$$\text{interval round}(50 * (1 + \text{rand}())) * (1 - \text{准点率}) \text{ minute}$$

对 fly 基表，若某元组的起飞状态为是，实际起飞时间非空，且实际抵达时间为空，则结合实际起飞时间、该实飞所属航班的预计起飞时间和预计抵达时间，计算并设置该元组的实际抵达时间，具体计算公式为：

$$\text{实际抵达时间} = \text{实际起飞时间} - \text{timediff}(\text{预计起飞时间} - \text{预计抵达时间})$$

2. 航班插入触发实飞生成

每当由航司票务管理员新增一条航班，系统会自动生成从有效日期始到有效日期止的每天一趟的实飞元组。所用触发器 `flight_insert` 的主体 sql 代码如下所示：

```
--计算第一条插入元组的起飞日期
set bdatetime = CONCAT(new.begin,' ', TIME(NOW()));
--计算总共插入(d+1)条实飞元组
set d = TO_DAYS(new.end)-TO_DAYS(new.begin);
--循环插入(d+1)条实飞元组
while d >= 0 do insert into fly value
(0,new.id,DATE(bdatetime),false,null,null,new.av_pf,new.av_pc,new.av_py);
--计算下一条插入元组的起飞日期
set bdatetime = bdatetime + interval 1 day;
set d = d-1;
end while;
```

3. 航班更新触发实飞生成

每当航司票务管理员延后某现有航班的有效期终止日期，系统就会自动生成从原有效终止日到现有效终止日的每天一趟的实飞元组。触发器 `flight_update` 与上述 `flight_insert` 主体 sql 代码相似，不再赘述。

4. 实飞更新触发机票生成

据 4.2.1 所述，为了便于展示票务，系统会在每趟实飞状态修改为已飞后，自动生成一些匿名经济舱机票。所用触发器 `generate_ticket` 主体 sql 代码如下：

```
--如果该实飞的起飞状态由否变为是，则执行触发器主体代码
if new.flew = true and old.flew=false then
--使用函数 count_ticket，计算该实飞现已售出的实名经济舱票数
set n=(select count_ticket(new.id,3));
--从机型数据和航班数据中，找出该实飞经济舱的总个数
set c=(select numy from aircraft where aircraft.id=(select acid from flight where
flight.id=new.fid));
--结合随机数，将 i 计算为 0.3*c~0.6*c 之间的某个整数，拟生成 i 张匿名票
set i=round(0.3*c*(1+rand()));
--检查再生成 i 张票是否会票余量不足；不足则将 i 降为一半，循环检查至余量满足
while (n+i)>c do set i = round(i/2+1); end while;
set n=n+i;
--先插入票量为 i 的一项订单
insert into porder value (0, new.id, 3, null, (select today from datelog order by today desc limit
```

```
1), time(now()), 2, i);
--找出刚插入的订单编号
set n=(select max(id) from porder where porder.valid=2);
--找出现有最后一条机票的编号
set c=(select max(id) from ticket);
while i>0 do
--计算下一条待插入的机票的编号
set c=c+1;
--插入所属订单为上述新建订单的匿名机票
insert into ticket value (c,null,i,null,null,null);
--调用函数 seat_ticket 为新建机票确定座位
update ticket set seat=(select seat_ticket(c)) where id=c;
set i=i-1;
end while;
end if;
```

除去以上四种典型的触发器应用场景，本系统中还有一处有明显的自动逻辑：每当有新的航司建立则自动为其创建一个数据库同名用户。但该需求不能用数据库触发器实现，因为创建数据库用户属于事务提交语句，而触发器中不允许包含任何显式或隐式事务提交行为。

综上所述，本数据库的四个触发器基本信息总结如表 4.1 所示。

表 4.1 触发器基本信息表

触发器名	触发事件	触发时机	关联基表
datelog_insert	insert	after	datelog
flight_insert	insert	after	flight
flight_update	update	after	flight
Generate_ticker	update	after	fly

4.5.4 票务小工具：函数与存储过程

在处理票务逻辑时，有一些逻辑代码，不仅使用频率高，而且代码内部多次请求数据库数据或数据库处理，非常适合用数据库的函数或存储过程实现。以下是对本系统两个函数和一个存储过程的具体阐述。

1. 函数 count_ticket

该函数功能简单，用于飞机起飞后统计对应实飞的对应舱票数。函数包含两

个传入参数：实飞主码编号和舱位编号，返回结果是该实飞当前在该舱位的已售票数。

本系统的有效订单状态有“15 分钟内待支付”和“已支付且未退票”两种，本系统的订票功能在飞机起飞日即关闭，所以使用 `count_ticket` 时，仅存在后者一种有效订单状态。故此处仅统计属于状态为“已支付且未退票”的订单的该实飞的该舱位的机票。

2. 函数 `seat_ticket`

该函数用于乘客值机成功后系统为其机票自动生成座位号，或系统生成匿名票时使用。函数传入参数是机票主码编号，返回值是生成的座位号字符串。在函数内部，首先确定该机票的舱位类型和所属实飞编号，再数出编号小于等于该机票编号的有效的同飞同舱机票数。

如以下 `sql` 代码所示，`type_p` 表示舱位类型（1 是头等舱 F，2 是商务舱 C，最后 3 是经济舱 Y），`num_p` 是数出的所排序号，舱位编号拼接上所排序号，即为函数返回的座位编号。

```
if type_p=1 then return concat('F', cast(num_p as char));
elseif type_p=2 then return concat('C', cast(num_p as char));
else return concat('Y', cast(num_p as char));
end if;
```

3. 存储过程 `create_order`

该存储过程功能如其名，在乘客用户创建订单时使用，包含四个 `in` 参数和 1 个 `inout` 参数。四个 `in` 参数分别是航班主码编号、实飞起飞日期、舱位类型、下单用户主码编号，一个 `inout` 参数传入时等于用户申请购票数量。

在存储过程内部，首先在状态为“15 分钟内待支付”和“已支付且未退订”两种有效订单里寻找同飞同舱的机票，统计已售票数，记为 `sold`。之后，根据舱位类型，在航班和机型数据中找到该舱位的座位总数，记为 `total`。最后，如果 `sold` 加上申请票数小于等于 `total`，则新建一条该票数的订单，并将 `inout` 参数设置为 0；若不满足则存储过程直接结束。

4.5.5 并发订票：排他锁与事务

据 4.2.7 所述，系统需要以较低的性能开销满足并发订票的正确性，所以本系统在具体实现时结合了 4.5.4 中的 `procedure create_order`、排他锁 `for update` 和 4.4.2 的事务。

在应用场景中，并发订票开始于某用户选定航班、日期、舱位和请求票数后。订票第一步，是将一个独属于该用户的变量的值设置为请求票数，如下述 `sql` 语句所示，该变量的命名以当前用户 `id` 为后缀，确保不会被其他并发连接更改：

```
'set @n_in_%s=%s;' % (str(self.uid), str(select_n))
```

第二步,调用 `create_order`,其中 `inout` 参数为第一步所设置的变量;第三步,查询该变量,若为 0,说明订单创建成功,否则失败;如果订单创建成功,接下来还有第四步,即建立申请数的数项机票数据,每张机票对应用户所勾选的一位乘客。

上述前三步一同传给 `dbsql` 对象,作为整个事务执行。但并发正确性的关键不仅在此,必须在第二步所使用的 `create_procedure` 存储过程中对订单、机票等基表用 `for update` 加上排他锁,如下述 `sql` 代码所示。

```
set sold=(select count(porder.num) from flight, fly, porder where flight.id=fly.fid and
porder.fid=fly.id and porder.stype=ftype_in and porder.valid<>0 and porder.valid <> 3 and
porder.uid=uid_in and flight.id=fid_in and fly.fdate=fdate_in for update);
```

被加上排他锁的基表,在所在事务提交之前,均不能被其他连接访问查询,由此才能保证并发购票的正确性。

此外,本系统仅对订单的创建过程加锁,将即机票的创建放在阻塞体之外,尽可能地降低性能开销。

4.5.6 GUI: PyQt5

本系统的大部分工作与 GUI 密不可分,故有必要在阐述各模块具体实现前,先介绍整个系统中主要的 GUI 功能。下述 GUI 均为 QT 相关的现成工具或以 PyQt5 为基础自实现的功能。

1. QT Designer 和 pyuic5

QT Designer 是 QT 的图形化设计工具,设计页面原型非常方便,如图 4.16 所示是系统 user 界面在 QT Designer 中的原型设计图。

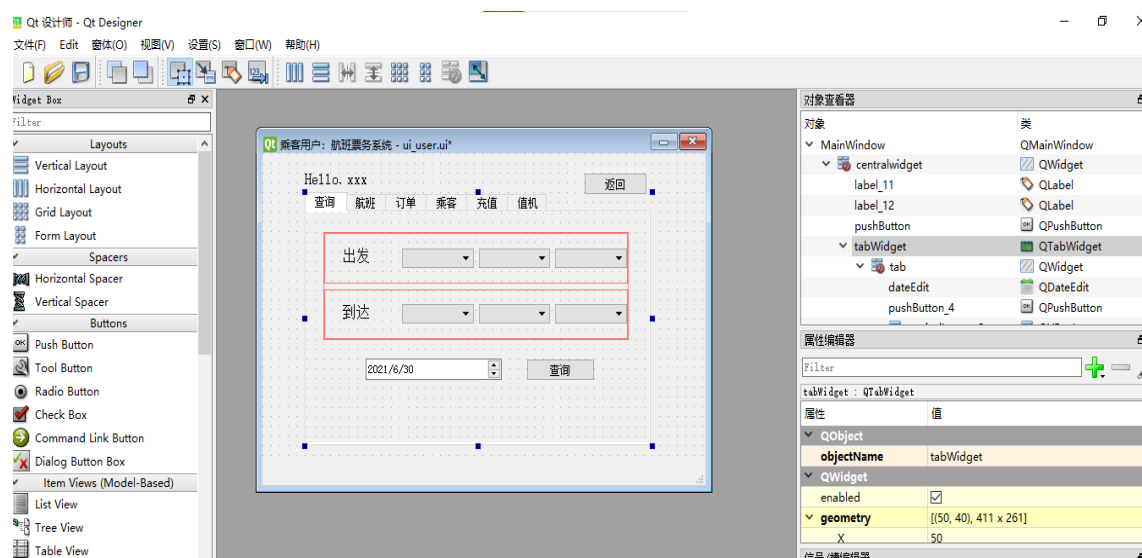


图 4.16 user 设计原型图

`pyuic5` 是 `pyqt` 针对 QT Designer 的设计文件所开发的命令行工具,能将单个设计文件转换成单个 `python` 模块的代码。虽然所得的 `python` 模块继承于 `Obeject`,

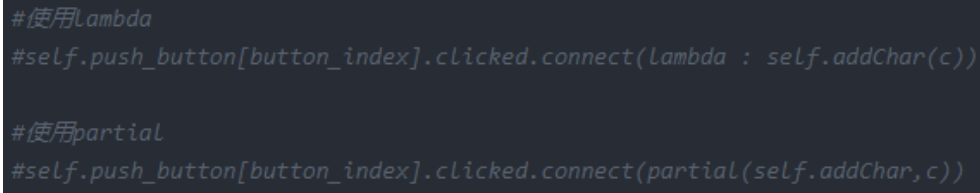
与 QT 的 QObject 无关，但是该模块的类可通过调用 setupUI 的方法，将传入的 QObject 或 QObject 子类进行 UI 设置。

在本系统实现中，六个主页面中，welcome、visitor、administor 和 login 四个类均直接继承于它们的原型设计转化得到的 python 类，通过对已建立的 QT 对象 setupUI 而得到对应界面；user 和 company 出于以下 4.5.7 对信号设置的考虑，与自动转换得到的 python 类仅存在代码参考关系，无继承和调用等直接关系。

2. 槽函数和 partial

信号和槽是 QT 的经典机制，能够通过 connect 将信号与槽函数绑定起来。但在 pyqt5 中，若被绑定的槽函数需要传入参数，则需要借助 partial，而不能像经典 QT 中一样借助 lambda。

两者用法形式上的差别如图 4. 17 所示。两者实际效果的区别在于，lambda 方式无论何时绑定，变量 c 为槽函数传入的值都是它的初始值；而 partial 方式下，变量 c 为槽函数传入的值是绑定时刻的值。



```
#使用lambda
#self.push_button[button_index].clicked.connect(lambda : self.addChar(c))

#使用partial
#self.push_button[button_index].clicked.connect(partial(self.addChar,c))
```

图 4. 17 partial 与 lambda 的使用区别

3. QTabWidget 和 QTableWidgetItem

两者均为 QT 中的常用组件，虽然名字相似，但不具有继承关系，具有各自不同的特性。

QTabWidget 提供一个 tab 容器，可通过 addTab 方式添加数个 QWidget 作为 tab 页，从而达到一个窗口可展示多个页内容的效果。QTableWidget 则可提供一表格，对该表格可以设置行数、列数和列标题，并且用另一种 QT 组件——QTableWidgetItem 去一一填入某行某列，非常适合关系的展示与操作。在实际应用中，常常在新建 QTableWidgetItem 对象时将其初始化到 QTabWidget 的某个 tab 页上，其效果即为一窗多页，一页一表。

本系统的六个主页面中，除了对话框形式的 welcome 和 login，其余四个页面均使用了上述 QTableWidgetItem 与 QTabWidget 结合使用的 GUI 方式。

4. CustomContextMenu 实现单击右键

上述的基本 QTableWidgetItem 仅提供基本的数据展示功能，若想对表中某行数据（也就是某个实体值）操作，可以通过 CustomContextMenu 实现单击右键、弹出临时操作菜单的效果。

首先调用 QTableWidgetItem 类的 setContextMenuPolicy 方法，设置为“Qt.CustomContextMenu”；之后，对 QTableWidgetItem 类的 customContextMenu-

Requested 信号绑定上一个自定义槽函数，该信号会在 table 被右键点击时出现；最后编写该槽函数。

在该槽函数内部，首先通过 QTableWidgetItem 类 selectionModel() 获取被选中的行号；之后用组件创建一个临时 QMenu 对象，并向该 menu 添加所需要的 action，之后执行该 menu，获取用户的点击选择，从而执行用户所选的逻辑操作。如图 4.18 所示，是用户对表中某行单击右键后跳出的临时菜单。



图 4.18 单击右键后的临时菜单

5. QComboBox

QComBox 能够提供一个下拉框，使用时仅需调用 addItem 方法向其传入一个字符串列表，该下拉框则能将各条字符串按序以一条选项的形式呈现。

在 user 主页面对行程的选择时，系统结合信号和槽，将 QComBox 的选项动态化：用户在选中某省后，城市的选择下拉框则只包含该省的城市；用户再选中某市后，机场的选择下拉框则只包含该市的机场选项，如所示是该成品图。

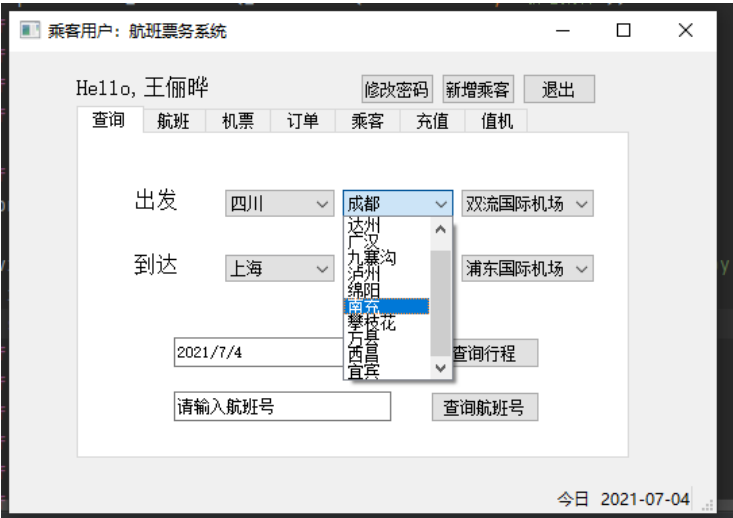


图 4.19 动态下拉框成品图

4.5.7 登陆跳转：信号与 event 重写

在 QT 中，可以很快捷地创建新的窗体，在本系统中，这种新窗体的出现有两种情景，第一种是从 welcome 直接弹出后续窗体，但原 welcome 窗体仍存在；第二种是从 login 跳转到子应用系统主页，login 原窗体会因此消失。

第二种情况下，由于 user 或 company 对象是在 login 对象中创建的，所以 login 对象不能真的直接关闭，否则子系统主页面也会随之关闭。为了达到这种登录跳转的效果，在子系统主页面出现后，先将 login 窗体隐藏。但隐藏只能满足视觉消失，在子系统主页面关闭后，无用的隐藏的 login 窗体也应该随即关闭。

此处系统用信号和重写 closeEvent 实现。例如从 login 登录到 user 时，首先定义信号，在 user 中用 pyqtSignal() 定义一个信号 close_signal；之后绑定槽函数，将 user 的 close_signal 信号绑定上 login 的 reject 函数；最后还需要解决信号发射的问题，即重写 user 的 closeEvent，该函数原本即存在，会在窗体关闭前自动执行，将其重写为先发射 close_signal 信号，再 accept 关闭事件。

4.5.8 welcome 模块具体实现

welcome 类包含四个方法：porder_check、date_get、date_update 和 entry，各自具体功能如下所述。

1. porder_check

专用于检查并更新已超过 15 分钟且未支付的订单，使用的 sql 语句如下。

```
'update porder set valid=0 where valid=1 and (odate<"%s" or (odate="%s" and
(otime+interval 15*60 second)<=TIME(NOW())));' % (self.user_date, self.user_date)
--valid 为 0 表示超时，为 1 表示待支付，需处理创建日期小于模拟日期，或创建日期等于模拟日期但创建时间据当前真实事件已超过 15 分钟的待支付订单。
```

2. date_get

用于获取系统模拟日期，具体方式是向基表 datelog 查询日期最晚的数据项，即为系统模拟时间。系统的数据库在初始化时，会自动插入一条“2021-7-1”的记录。

3. date_update

借助 QDateEdit 组件，获取用户对模拟日期的调整。若用户调至当前模拟日期之前，则会报错“日期不可回调”；若用户往后调模拟日期，则将其插入基表 datelog，作为当前的系统模拟日期；若用户没有调整，则系统模拟日期不变。

4. entry

entry 是 welcome 页面上三个入口按钮的点击槽函数，用参数区分不同的入口按钮。entry 函数为对应按钮创建对应的新窗体对象，并向其传入系统模拟日期、连接池对象等初始化数据，最后展示新窗体。

综上，welcome 内部流程图如图 4.20 所示。

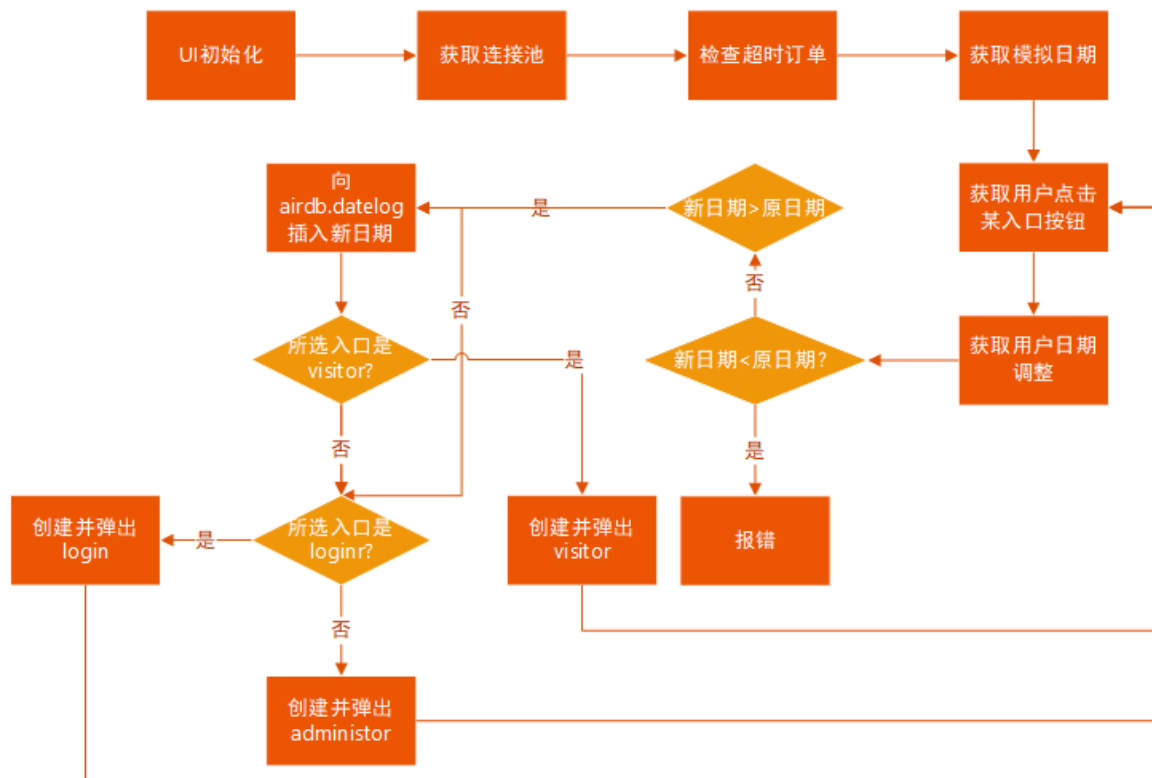


图 4.20 welcome 流程图

4.5.9 visitor 模块具体实现

visitor 只需要展示五个结构相似的查询表即可，除了初始化之外，只包含 setupTable 一个方法。

以第一个 tab 页中关于省份的查询表为例，需要先在初始化中依次创建省份表到 tab1、确定查询 sql 语句、确定展示表的列名（即表头），再用上述的表对象、sql 语句列表和表头列表调用 setupTable 方法。

在 setupTable 方法内部，首先设置展示表的图形位置大小，并设置为不可编辑，再用 sql 语句获取数据库返回的查询关系数据。在表中行数由查询数据有多少条决定，列数有表头有多少项决定，既然此时待展示的数据均已准备就绪，最后的工作就是依次设置列数、列名、行数和各行各列的文字内容。

4.5.10 administor 模块具体实现

administer 是对 visitor 的功能加强，需要在原展示表的基上增加对数据增删改的功能。每张表的增删改操作思想类似，通过单击表中某行弹出含“修改”和“删除”选项的临时菜单，从而操纵删改；而在 tabwidget 之外的部分有一个单独的“新增”按钮，点击后能够对当前窗体所展示的表插入新的实体值。

administer 具体包含 setupAllTable、fillTable、clickTable、addNew、getOldList、sqlDelete、sqlInsert、sqlUpdate、updateGUI 等方法。

1. setupAllTable

对页面中的五张表设置基本信息，不包含对行信息的设置。

2. fillTable

执行 sql 语句获取数据，填入对应表的具体行信息。

3. clickTable

处理对表的单击右键操作，包含对删除的处理和对修改的处理。删除时，获取该行中能够标识该实体值的一组候选码值，连同当前页编号，一起传给 sqlDelete 方法处理；修改时，获取表中该行的原始数据，连同各数据表名，一起传给 inputdialog 对象，获取用户输入的新值的列表后，再连同当前页标号，一起传给 sqlUpdate 方法处理。

在删除时，城市展示表的候选码较为特殊，因为存在同名城市，所以需要同时获取城市名和所属省份名作为一组候选码。

4. addNew

处理对“新增”按钮的点击操作，与修改操作类似，但不必获取某选中行，直接使用 inputdialog 对象，获取用户输入的新值的列表后，再连同当前页编号，一起传给 sqlInsert 方法处理。

5. getOldList

专用于获取某表某行的当前文字数据列表。但此处并不会获取该行所有的数据项，只获取该基表原有的数据项。以省份为例，虽然同时有“省份名”“城市个数”“机场个数”等数据项，但实际上只有“省份名”是属于原基表的，其余数据都是通过参考关系计算得到，所以 getOldList 对省份展示表只获取“省份名”一项旧值，其他展示表同理。

6. sqlDelete

对传入的表编号和候选码值构造 delete 的 sql 语句并执行即可。

7. sqlInsert

对传入的表编号和新值列表构造 insert 的 sql 语句并执行即可。

8. sqlUpdate

对传入的表编号、旧值列表和新值列表构造 insert 的 sql 语句并执行即可。

9. updateGUI

在增删改操作结束后，调用该方法更新相关展示表。以省份表为例，如果将“四川”修改为“川”，不仅省份展示表中的数据会有变化，城市展示表中原属于“四川”的城市的“所属城市名”也会发生变化，所以需要同时对两表调用 fillTable 方法来更新展示内容。

4.5.11 login 模块具体实现

login 以对话框形式执行两类子系统账户的注册与登录操作，具体包含 createUser、login、searchUser 三个方法。

1. createUser

createUser 是“注册”按钮点击后执行的槽函数，首先获取用户输入的账户名、账户密码和登陆选项；再根据登陆选项，分别向 user 基表和 company 基表查询该账户名对应的主码编号，若查询结果非空，报错“该用户已存在”，否则向各自基表插入一条新实体数据。

当注册选项为航司时，注册成功后，系统需要再为其创建一个同名数据库系统用户，并将 company_ad 角色授予该角色。

2. searchUser

searchUser 是“登录”按钮点击后执行的槽函数，依旧首先获取用户输入的账户名、账户密码和登陆选项；再根据登陆选项，分别向 user 基表和 company 基表查询该账户名对应的主码编号和密码，若查询结果为空，报错“该用户不存在”，非空则继续检查密码是否匹配，不匹配则报错“密码不正确”，匹配则调用 login 方法。

3. login

首先调用 hide 方法将 login 窗体自身隐藏，再根据登陆选项，创建 user 或 company 对象。创建时 user 时，需要传入 login 窗体自身、连接池对象、模拟日期和用户 id；创建 company 对象时，需要传入 login 窗体自身、模拟日期、用户 id 和航空公司名。之所以有区别，是因为在 company 中需要使用以自身公司名为名的数据库用户连接的连接池。

综上所述，login 内的流程图如图 4.21 所示。

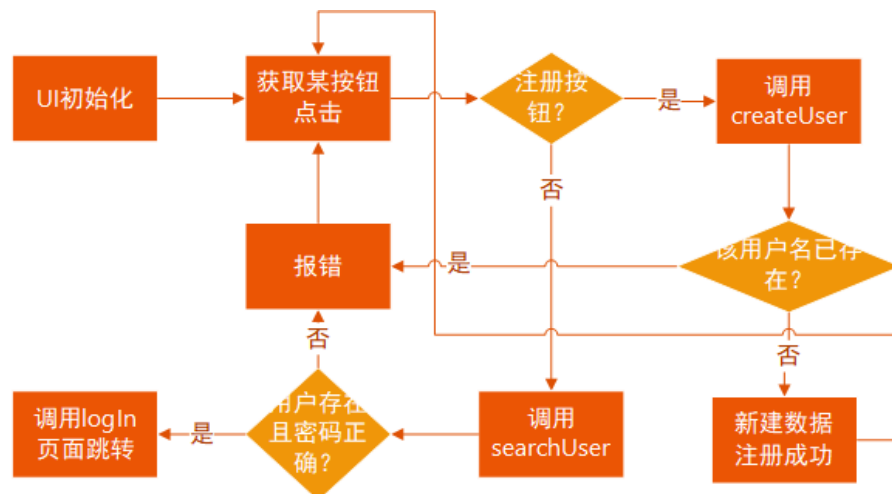


图 4.21 login 流程图

4.5.12 user 模块具体实现

user 是乘客应用子系统的操作页面，是整个系统中最复杂的模块和页面，总共包含 9 项功能，以下按功能划分，依次对功能下的方法实现进行说明。

1. 修改密码

包含一个方法：`modifyPasswd`。

在 `modifyPasswd` 内，首先用 `inputdialog` 获取用户输入的“原密码”和“新密码”；若原密码不正确，则报错，否则用新密码和用户 `id` 构造对 `user` 基表的 `update` 请求。

2. 代购乘客

包含两个方法：`createPassenger` 和 `showPassenger`，分别用于创建和展示本用户所登记的乘客记录。

在 `createPassenger` 中，首先用 `inputdialog` 获取用户输入的乘客基本信息，包含身份证号、姓名、出生年份和性别。之后检查输入信息中的身份证号和性别是否合乎规范，若不合规则报错；合规则向 `person` 基表查询该身份证号对应的各项基本信息，若当前输入信息与查询结果不符，则报错“与系统现有乘客信息冲突”；若查询结果不冲突或查询结果为空，则在为空时新插入该 `person` 数据项，在插入完成或非空时，向 `agent` 基表插入该乘客与当前用户的联系数据项，完成乘客登记。

`showPassenger` 负责在 `user` 页面的第五章 `tab` 页的 `table` 上展示出当前用户所登记的全部乘客信息，具体实现与之前表的创建、设置和填入相似，故不再赘述。

3. 金额充值

包含两个方法：`addMoney`，该方法在点击充值页的“确认”按钮后执行，如图 4.22 所示。



图 4.22 充值页展示图

在 `addmoney` 中，首先检查当前用户是否已绑卡，若未绑卡则引导用户输入 16 位银行卡号完成银行卡绑定；之后获取用户输入的充值金额，判断输入是否

为正整数，正确则更新 `user` 基表中的金额值。

`showMoney` 单纯负责将银行卡号和充值余额展示到页面上。

4. 行程查询

行程查询包含四个方法：`showCity`、`showAirpor`、`queryRoute t`、`showFly`。

其中，`showCity` 和 `showAirport` 如 4.5.6 对 `QComboBox` 的介绍，专用于下拉框的动态展示，在每一次上级下拉框的选项被修改时执行，向数据库动态请求数据并设置到自身下拉框上。

`queryRoute` 内，首先检查用户选定的起飞时间是否在明天以后，不符合则报错；之后结合用户选择合成 `sql` 查询语句，并调用 `showFly` 方法。

`showFly` 负责用 `table` 展示行程查询到的符合要求的实飞，其中每条实飞会显示三条，对应三种舱位，结果会按照具体价格升序排列展示。

5. 机票预定

机票预定含一个方法：`create_order`。

`create_order` 是绑定于实飞查询结果表的单击右键信号的槽函数，与 `administor` 中类似，其临时菜单仅包含“下单”一项。用户点击下单后，首先检查余额是否大于零，不满足则报错；再检查登记乘客数是否大于零，不满足则报错；之后使用 `selectdialog` 对象，引导用户在所有登记乘客中多选，获取购票乘客名单和请求购票张数；之后按照 4.5.5 所述的方式完成并行订票；倘若订票成功，会提示用户在 15 分钟内支付，并自动跳转到该用户所有订单的显示页；失败则会报错“余票不足”。

6. 订单支付

订单支付有四个相关方法：`showOrder`、`orderTimer`、`handleOrder` 和 `generateToken`。

`showOrder` 负责订单页的查询、展示和设置处理方式，与之前类似表展示方法相比，`showOrder` 多了一个用 `orderTimer` 实现的动态倒计时展示效果。在每次订单页被选中后，将开始一个一秒长的定时器，`orderTimer` 即为该定时器定时结束后的处理函数。

`orderTimer` 首先检查所有待支付状态下的订单，将其当前时间倒计时一栏中的文字获取，转换为时间，并再减去一秒。如果减去一秒后倒计时变 0，则通过修改 `porder` 基表修改该订单状态为“超时”，否则将该倒计时再转换成文字显示，最后再设置一个一秒的定时器，循环往复，达到动态倒计时的效果。如图 4. 23 所示为订单展示页的效果图。

`handleOrder` 是绑定于订单页单击右键信号上的槽函数，临时菜单包含“支付”“退订”和“详情”三个选项，此处暂时只介绍“支付”选项。首先检查状态是否为“待支付”，不满足则报错“无法支付”；其次检查月是否足够支付该订单，

不足则报错；最后则执行 `update` 的 `sql` 语句，更新内容包括用户的余额、订单的状态，以及调用 `generateToken` 方法为机票的值机生成一个随机密码。

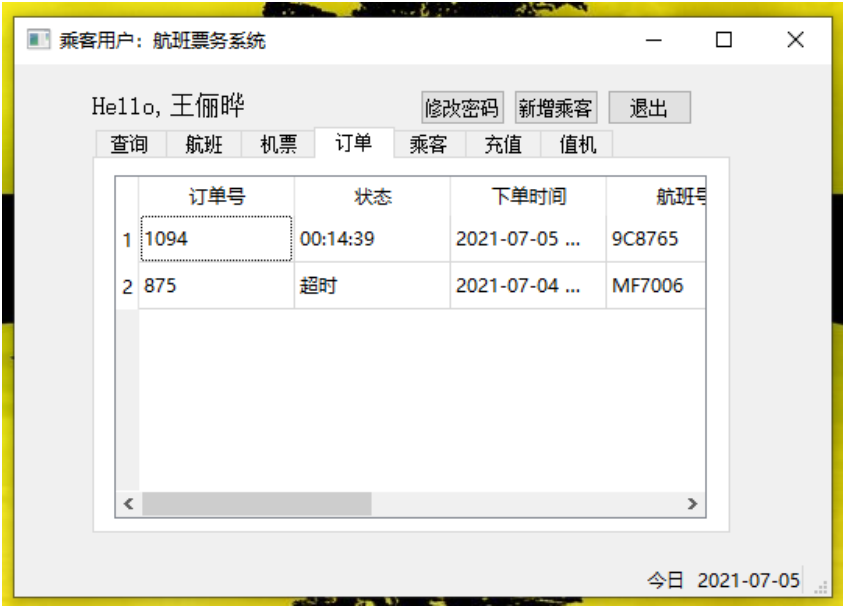


图 4.23 订单页成品图

`generateToken` 方法使用 `random.choice` 方法六次，生成一个包含大小写字母和数字的六位随机字符串。

7. 值机取票

值机取票与三个方法相关：`handleOrder`、`showTicket` 和 `checkIn`。

`handleOrder` 中的“详情”选项则是用于获取所选中行对应的订单编号，由此调用 `showTicket` 方法。

`showTicket` 负责查询并展示指定订单下所属的所有机票，在对 `sql` 查询结果转换成展示内容时，若值机密码为 `null` 则显示“未支付”，若座位号为 `null` 则显示“未值机”。如图 4.24 所示为机票展示页效果图。



图 4.24 机票展示页成品图

`checkIn` 负责读取用户在模拟值机时输入的信息，以这些信息查询有无符合的机票，如果没有则报错；如果有，则使用 `seat_ticket` 数据库函数为该机票安排座位，并将其座位号返回给客户端，展示给乘客值机成功后的座位号。如图 4.25 所示是值机页面。

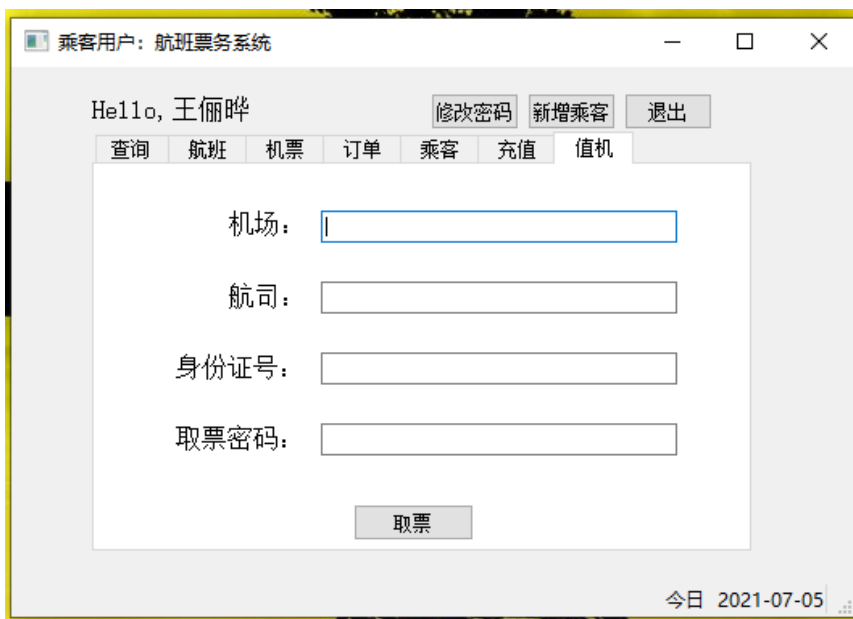


图 4.25 值机页面成品图

8. 订单取消

订单取消由方法 `handleOrder` 的“退订”选项实现。用户点击退订后，首先检查该订单是否已超时或已退订，若满足则报错“无法退订”；之后，查询属于该订单的机票中有无已经值机成功的，若有，则报错“无法退订”；之后，查询该订单所购实飞记录，若该实飞状态为已起飞，则也会报错“无法退订”。

最后，分四种情况处理退订时需要返回的金额：第一种，如果该订单状态为待支付，则返回金额为 0；第二种，如果已支付，且当前模拟时间距离起飞超过 24 个小时，则返回该订单的全部金额；第三种，如果已支付，且当前模拟时间距离起飞小于 24 小时大于 4 小时，则查询该实飞所属航班的所属航司的 24 小时以内退订手续费比例 `debit2`，返回金额为订单金额*（1-`debit2`）；第四种，如果已支付，且当前模拟时间距离起飞小于 4 小时，则查询该实飞所属航班的所属航司的 4 小时以内退订手续费比例 `debit1`，返回金额为订单金额*（1-`debit1`）。

9. 航班查询

我们获知亲朋好友乘坐的航班号后，输入航班号，`searchFlight` 方法即可查询并显示该航班的起飞地点、降落地点、预计起飞降落时间和所属航司等基本信息，方便接机等。

4.5.13 company 模块具体实现

`company` 是航司应用子系统的界面模块，提供了航班管理、实飞管理和机票查看等功能。

1. 航班管理

与航班管理有关的方法有三个：`showFlight`、`handleFlight` 和 `addFlight`。

`showFlight` 为常规的表展示函数。

`handleFlight` 为常规的绑定在航班展示表单单击右键信号上的槽函数，其临时菜单包含“修改”和“实飞”两个选项。若选择修改，可以延长选定航班的有效天数；若选择实飞，则会调用后续的 `showFly` 展示该航班的所有实飞表。

`addFlight` 与 `administor` 中常见的添加功能相似，均通过 `inputdialog` 获取用户输入的新建项信息，从而新建一条航班。如图 4. 26 所示为航班管理界面的效果图。



图 4.26 航班管理成品图

2. 实飞管理

与实飞管理相关的有两个方法：`showFly` 和 `handleFly`。

`showFly` 是常规的表展示函数。

`handleFly` 为常规的绑定在实飞展示表单击右键信号上的槽函数，其临时菜单包含“修改”和“机票”两个选项。若选择修改，则可以调整该航班各舱位的价格；若选择机票，则调用后续的 `showTicket` 方法展示该实飞的所有机票表。

如图 4.27 所示，为实飞管理的效果图。

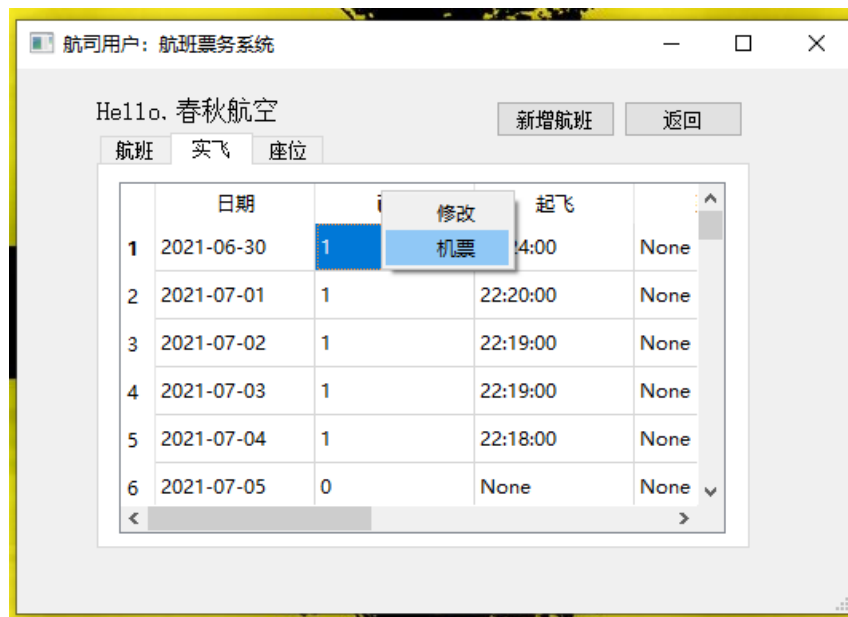


图 4.27 实飞管理成品图

3. 机票查看

方法 `showTicket` 具体实现了机票查看的功能，与常规的表展示函数别无二致，故不再赘述，如图 4.28 所示为机票查看的效果图。



图 4.28 机票查看成品图

4.6 系统测试

4.6.1 部分测试数据准备

本系统包括省份、省市、机场、航司和机型五项静态数据，其静态不意味着不能对其增删改，而是体现在它们不会随着票务处理而变化。为了便于测试，我收集到了 airport.json 和 com_airc_flg.xls 两个数据文件。

Airport.json 如图 4.29 所示，包含机场三字码、城市名、机场名和省份名，读取后。可作为导入省份、城市和机场的数据。

```
{
  "code": "AKU", "areaName": "阿克苏市", "airportName": "温宿机场", "provinceName": "新疆",
  "code": "AQG", "areaName": "安庆市", "airportName": "大龙山机场", "provinceName": "安徽",
  "code": "AYN", "areaName": "安阳市", "airportName": "安阳机场", "provinceName": "河南",
  "code": "BAV", "areaName": "包头市", "airportName": "二里半机场", "provinceName": "内蒙古",
  "code": "BHY", "areaName": "北海市", "airportName": "福城机场", "provinceName": "广西",
  "code": "BPX", "areaName": "昌都市", "airportName": "昌都马草机场", "provinceName": "西藏",
  "code": "BSD", "areaName": "保山市", "airportName": "保山机场", "provinceName": "云南",
  "code": "CAN", "areaName": "广州市", "airportName": "白云国际机场", "provinceName": "广东"
}
```

图 4.29 airport.json 部分图

com_airc_flg.xls 如图 4.30 所示，包含航班名、航空公司、机型、预计起飞时间、预计抵达时间、起飞机场、到达机场和准点率。本数据文件与 airport.json 按两类机场名作内连接，可作为导入航司、机型和航班的数据。

15067	Y87581	扬子江航空	波音737(中)	12:00	14:30	金湾机场	新郑机场	89%
15068	DZ6304	东海航空	波音737(中)	22:00	0:30	金湾机场	新郑机场	94%
15069	DZ6304	东海航空	波音737(中)	22:00	0:30	金湾机场	新郑机场	94%
15070	EU2234	成都航空	空客319(中)	13:40	15:45	金湾机场	遵义机场	95%
15071	EU2234	成都航空	空客319(中)	13:45	15:55	金湾机场	遵义机场	95%
15072	9C8827	春秋航空	空客320(中)	10:20	11:55	芷江机场	长水机场	95%
15073	9C8827	春秋航空	空客320(中)	10:30	12:05	芷江机场	长水机场	95%
15074	9C8828	春秋航空	空客320(中)	15:35	17:40	芷江机场	虹桥机场	100%
15075	GS6575	天津航空	ERJ-190(中)	15:45	17:30	芷江机场	咸阳机场	86%

图 4.30 com_airc_flg.xls 部分图

上述数据文件无法对数据库 airdb 中的数据项全覆盖，所以剩余数据项先采用默认初始值，之后由系统管理员或航司票务管理员操纵修改。

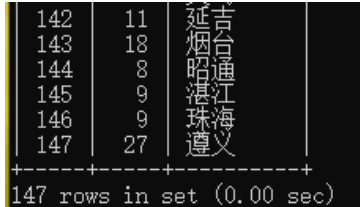
如图 4.31、图 4.32、图 4.33、图 4.34 和图 4.35 所示为 province 基表部分数据、city 基表部分数据、airport 基表部分数据、company 基表部分数据、aircraft 基表部分数据和 flight 基表部分数据。其中，company 基表第三列是初始登陆密码，第四、五列是初始手续费率；aircraft 基表的第三、四、五列依次是头等舱、商务舱、经济舱的座位数量；flight 基表的第九列是不允许 1 存在的准点率（为了满足数据库中 float(2,2)的完整性约束，需要将所有为 1 的准点率设置为 0.99），第十列和第十一列表示该航班有效期的范围，默认初始值为 2021-6-30 至 2021-8-31，第十二、十三、十四列是该航班三舱机票均价，是以 3000、1200、420 为基准加减 30%范围内的随机浮动得到。



26	江西	
27	贵州	
28	北京	
29	上海	
30	河北	
31	天津	

31 rows in set (0.00 sec)

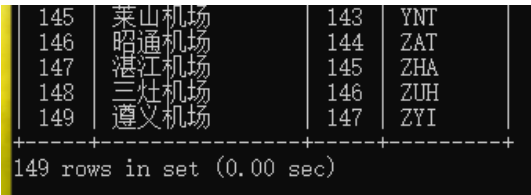
图 4.31 province 基表部分数据



142	11	延吉
143	18	烟台
144	8	昭通
145	9	湛江
146	9	珠海
147	27	遵义

147 rows in set (0.00 sec)

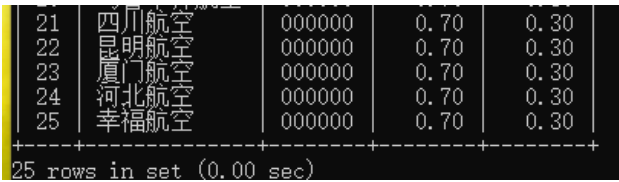
图 4.32 city 基表部分数据



145	莱山机场	143	YNT
146	昭通机场	144	ZAT
147	湛江机场	145	ZHA
148	三灶机场	146	ZUH
149	遵义机场	147	ZUI

149 rows in set (0.00 sec)

图 4.33 airport 基表部分数据



21	四川航空	000000	0.70	0.30
22	昆明航空	000000	0.70	0.30
23	厦门航空	000000	0.70	0.30
24	河北航空	000000	0.70	0.30
25	幸福航空	000000	0.70	0.30

25 rows in set (0.00 sec)

图 4.34 company 基表部分数据

5	ERJ-190(中)	3	6	12
6	JET	3	6	12
7	空客321(中)	3	6	12
8	ERJ(小)	3	6	12
9	新舟60(小)	3	6	12

9 rows in set (0.00 sec)

图 4.35 aircraft 基表部分数据

214	CZ9050	2	2	140	103	10:45:00	11:55:00	0.97	2021-06-30	2021-08-31	3727	1490	521
215	CZ6587	2	1	143	104	10:20:00	13:00:00	0.92	2021-06-30	2021-08-31	3409	1363	477
216	9C8731	9	2	54	21	16:30:00	19:00:00	0.98	2021-06-30	2021-08-31	3480	1392	487
217	MU5276	11	2	54	103	07:30:00	10:15:00	0.98	2021-06-30	2021-08-31	2343	937	328
218	JR1639	25	9	30	48	12:05:00	13:10:00	0.89	2021-06-30	2021-08-31	2288	915	320
219	11111	9	1	1	2	20:00:00	21:00:00	0.89	2021-07-10	2021-08-14	3000	2900	2800

219 rows in set (0.00 sec)

图 4.36 flight 基表部分数据

此外，根据触发器设置，航班插入引起实飞数据产生，如图 4.36 所示。

```
mysql> select count(*) from fly;
```

count(*)
13776

1 row in set (0.00 sec)

```
mysql> select * from fly limit 10000, 5;
```

id	fid	fdate	flew	off_real	land_real	pricef	pricec	pricey
10001	159	2021-08-15	0	NULL	NULL	2699	1079	377
10002	159	2021-08-16	0	NULL	NULL	2699	1079	377
10003	159	2021-08-17	0	NULL	NULL	2699	1079	377
10004	159	2021-08-18	0	NULL	NULL	2699	1079	377
10005	159	2021-08-19	0	NULL	NULL	2699	1079	377

5 rows in set (0.00 sec)

图 4.37 fly 基表部分数据

4.6.2 Welcome 界面测试

我们的测试从 welcome 欢迎页开始，主要测试模拟时间调整、页面弹出、触发器效果。

1. 调整系统模拟日期

如图 4.38 所示，当前系统模拟日期为 2021-7-5，尝试调至 2021-7-2 并选择游客入口，系统报错如图 4.39 所示。



图 4.38 调整模拟日期测试 1

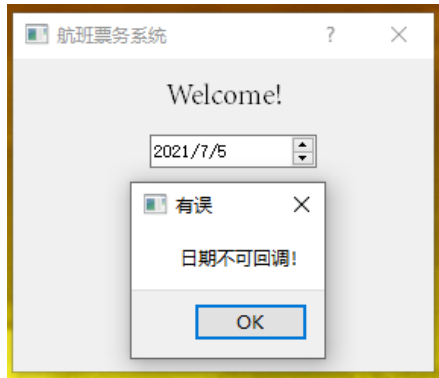


图 4.39 调整模拟日期测试 2

2. 页面弹出

再将模拟日期调至 2021-7-6，逐个点击各入口按钮，三个页面均能弹出。由于首次调至 2021-7-6 时会有触发器工作生成和更新大量数据，系统会卡顿 3 秒。

3. 日期插入触发的触发器效果

此时再用命令行工具查看日期为 2021-7-5 的实飞和对应的订单，发现触发器使其均已起飞，且均已售出一些匿名票，分别如图 4.40 和图 4.41 所示。

```
mysql> select id, flew from fly where fdate="2021-7-5";
```

id	flew
6	1
69	1
132	1
195	1
258	1

图 4.40 日期插入触发的触发器效果测试 1

```
mysql> select id, ffid, num from porder where ffid in (select id from fly where fdate="2021-7-5");
```

id	ffid	num
873	9582	1
875	9582	1
1095	6	4
1096	69	6

图 4.41 日期插入触发的触发器效果测试 2

3.6.1 administor 界面测试

administor 界面需要满足增删改基本功能，各表的操作处理函数类似，以下以省份表为例测试。

1. 增

选中省份表，点击新增按钮，弹出 inputdialog 窗口，输入新省份“川渝”，如图 4.42 所示，确认后再查看省份表，已能够查看新增省份，如图 4.43 所示。

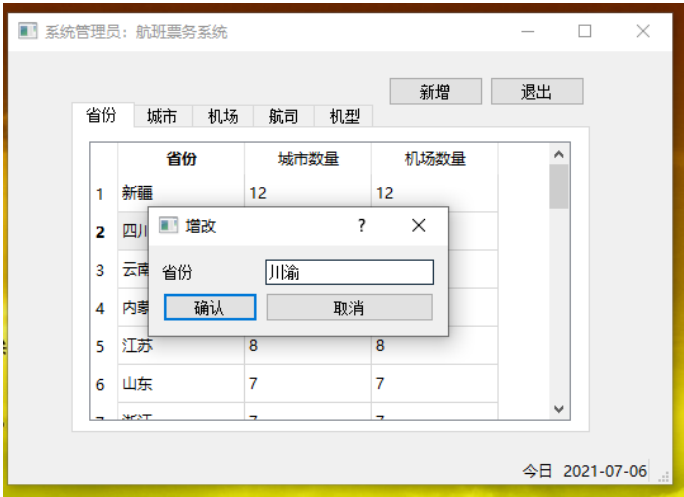


图 4.42 增加省份测试 1

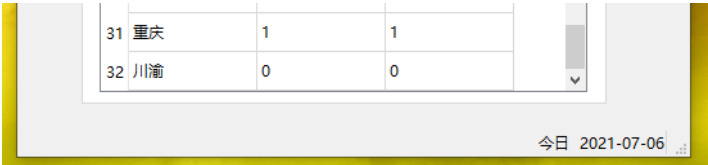


图 4.43 增加省份测试 2

2. 改

选中省份表中的“四川”所在行，选择修改为“四川行省”，确认后无报错，查看城市表中原属于“四川”的省份，如所示，先均属于“四川行省”，如图 4.44 所示。

31	西昌	四川行省	1
----	----	------	---

图 4.44 修改省份测试

3. 删

尝试删除城市数量和机场数量均不为 0 的“四川行省”，系统报错如图 4.45 所示，可见系统不允许级联删除。删除城市数量和机场数量均为 0 的“川渝”，“川渝”直接立刻从省份表消失，删除成功。

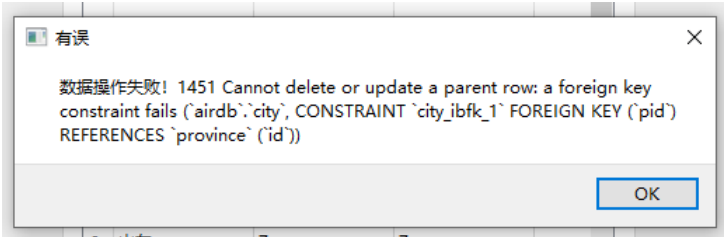


图 4.45 删除省份测试

3.6.1 login 界面测试

login 界面主要测试注册与登录是否逻辑正确，其中乘客和航司的处理逻辑相似，以下以航司的注册与登录为例。

注册成功如图 4.46 所示。

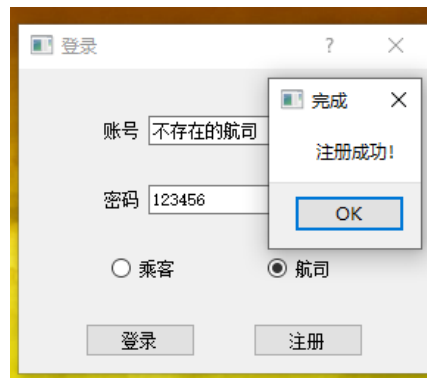


图 4.46 注册成功测试

因为航司已存在而注册失败如图 4.47 所示。

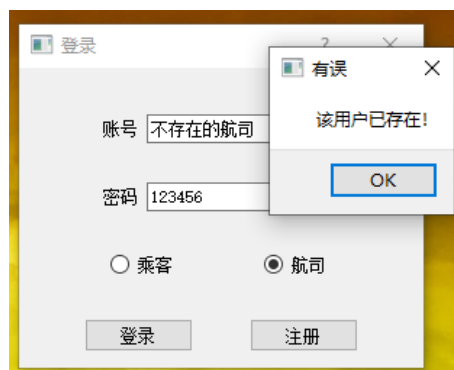


图 4.47 注册失败测试

因为航司不存在而登陆失败如图 4.48 所示。

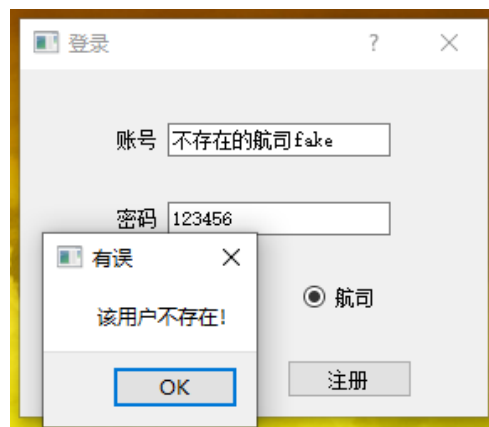


图 4.48 登陆失败测试 1

因为密码不正确而登陆失败如图 4.49 所示。

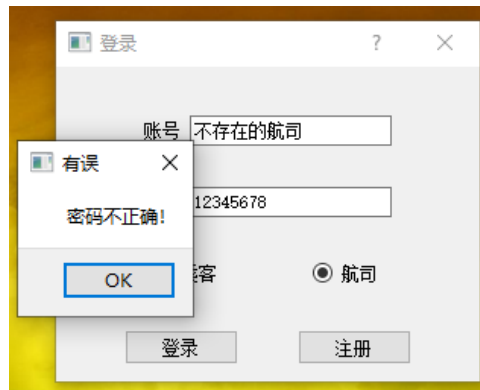


图 4.49 登陆失败测试 2

用刚注册的航司用户登陆成功如图 4.50 所示。



图 4.50 登陆成功测试 1

用原本存在的“春秋航空”航司登陆成功如图 4.51 所示。



图 4.51 登陆成功测试 2

3.6.2 user 界面测试

user 界面的测试关键在于查询、订票、支付、值机这一核心操作，以下创建一个全新的用户“小鱼”，测试上述完整流程。

首先，注册并登录“小鱼”乘客账户，如图 4.52 所示。

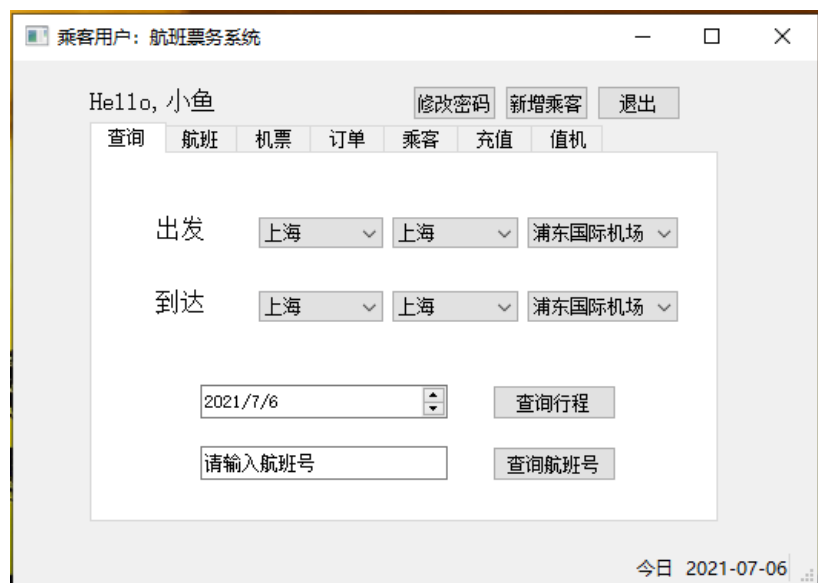


图 4.52 登录 user 界面

查询从如图 4.53 所示的实飞,先选择日期为系统模拟日期的“昨天”2021-7-5,系统报错如图 4.54 所示。再将选择日期改为 2021-7-8, 查询成功并自动跳转到航班展示页, 如图 4.55 所示。



图 4.53 查询指定行程测试

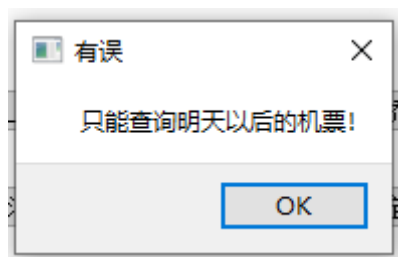


图 4.54 查询报错测试



图 4.55 行程查询成功测试

之后，选中航班“9C8765”的经济舱，选择下单，报错如图 4.56 所示。

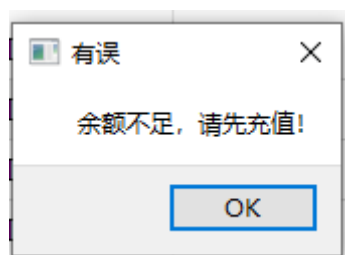


图 4.56 订票失败测试 1

于是选中“充值”页，输入充值 5000 元，确认后系统报错如图 4.57 所示。

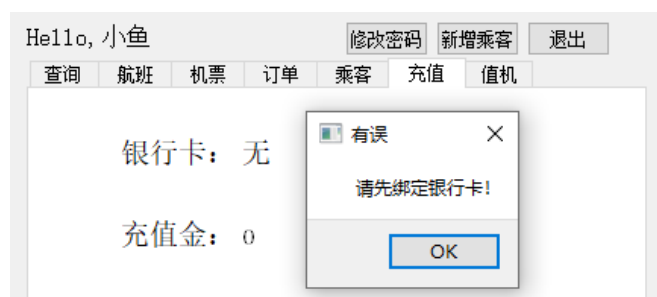


图 4.57 充值失败测试

于是顺应系统引导，输入十六位银行卡号绑定，如图 4.58 所示。

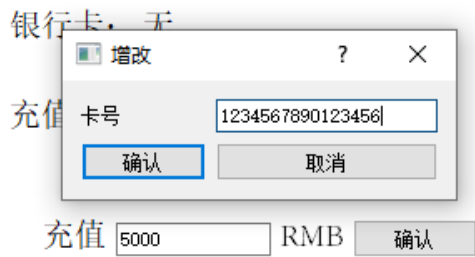


图 4.58 绑定银行卡测试

绑定后通知充值成功，如图 4.59 所示。

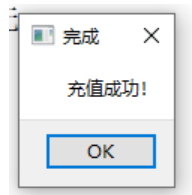


图 4.59 充值成功测试

再回到航班页，选中心仪航班实飞舱位并下单，系统报错如图 4.60 所示。



图 4.60 订票失败测试 2

于是点击页面右上方“新增乘客”按钮，新增“鱼霸霸”和“鱼淘气”两位乘客的登记，其中“鱼霸霸”的登记过程如图 4.61 所示。

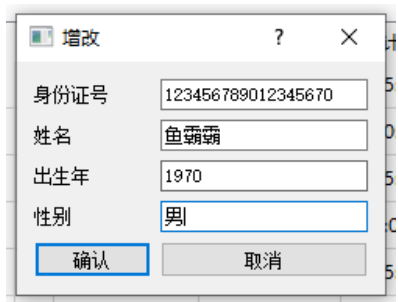


图 4.61 乘客登记测试

登记完毕后，选中乘客页，如图 4.62 所示。

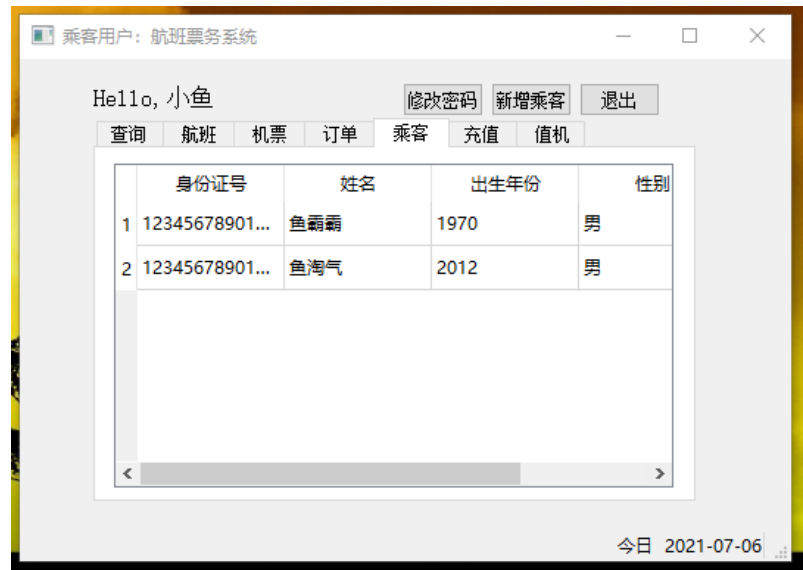


图 4.62 乘客展示页测试

第三次来到查询所得的航班页，选中心仪航班实飞舱位并下单，系统弹出多选对话框，选中两位乘客，为他们一人买一张机票，如图 4.63 所示。

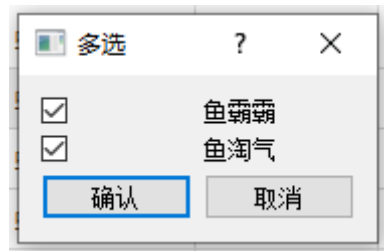


图 4.63 乘客多选测试

点击确认后，系统提示如图 4.64 所示。

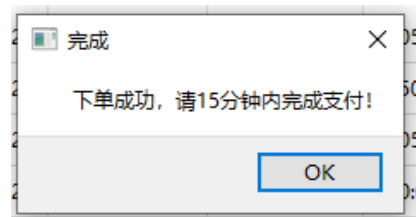


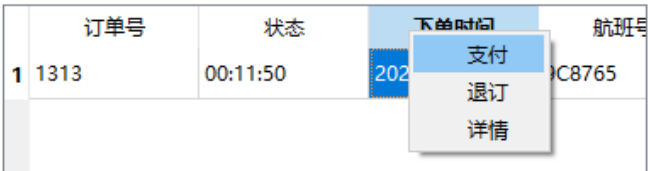
图 4.64 订票成功测试

选中订单页，查看到小鱼用户所创建的所有订单，如图 4.65 所示。



图 4.65 订单创建展示测试

如图 4.66 所示，右键单击该待支付订单项，选择“支付”，由于状态正确且余额充足，故系统直接将该订单的状态修改为“已支付”，如图 4.67 所示。



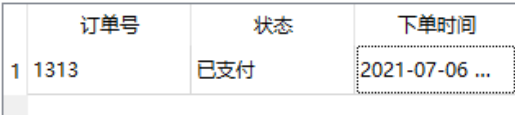
	订单号	状态	下单时间	航班号
1	1313	00:11:50	2021-07-06 ...	C8765

支付

退订

详情

图 4.66 订单支付测试 1



	订单号	状态	下单时间
1	1313	已支付	2021-07-06 ...

图 4.67 订单支付测试 2

支付成功后，选中查看该订单“详情”，页面自动跳转到“机票”页，如图 4.68 所示，机票的取票密码已经生成。



乘客用户: 航班票务系统

Hello, 小鱼

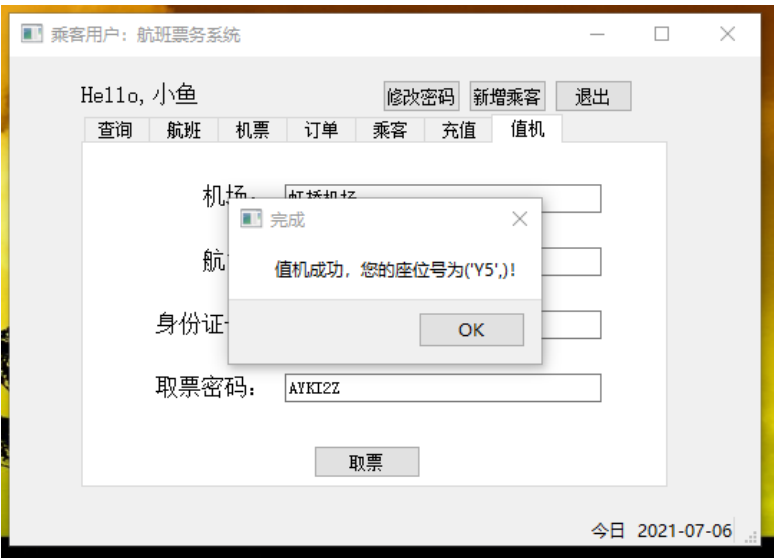
修改密码 新增乘客 退出

查询 航班 机票 订单 乘客 充值 值机

	编号	乘客姓名	取票密码	座位号
1	7073	鱼霸霸	AYKI2Z	未值机
2	7074	鱼淘气	AYKI2Z	未值机

图 4.68 取票密码生成测试

选中“值机”页，输入所需信息为鱼淘气值机，点击取票，系统提示值机成功，并通知生成的座位，如图 4.69 所示。返回“机票”页，发现鱼淘气乘客的座位号已生成，如图 4.70 所示。



乘客用户: 航班票务系统

Hello, 小鱼

修改密码 新增乘客 退出

查询 航班 机票 订单 乘客 充值 值机

机场: 广州白云

航班: 1313

身份证: 123456789012345678

取票密码: AYKI2Z

取票

完成

值机成功, 您的座位号为('Y5')!

OK

今日 2021-07-06

图 4.69 值机成功测试 1



图 4.70 值机成功测试 2

最后来到订单页，尝试对该订单取消，系统报错如图 4.71 所示。

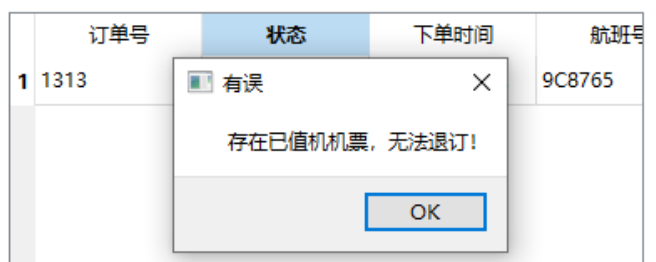


图 4.71 退订失败测试

3.6.3 company 界面测试

航空应用子系统的测试关键是航班管理、实飞管理和机票查询，以下以本身拥有航班的春秋航空为例测试。

如图 4.72 所示，选中春秋航空的“9C8765”航班，选择“修改”，可延长该航班的有效期，即增加实飞，如所示图 4.73。



图 4.72 航班管理测试 1

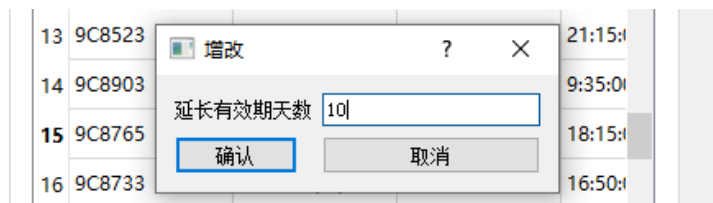


图 4.73 航班管理测试 2

选择“实飞”，系统自动跳转至“实飞”页，前几趟早于系统模拟日期的实飞已起飞，如图 4.74 所示；最后几趟的实飞日期也不止于最初默认初始的 2021-8-31，说明修改有效期成功，如图 4.75 所示。

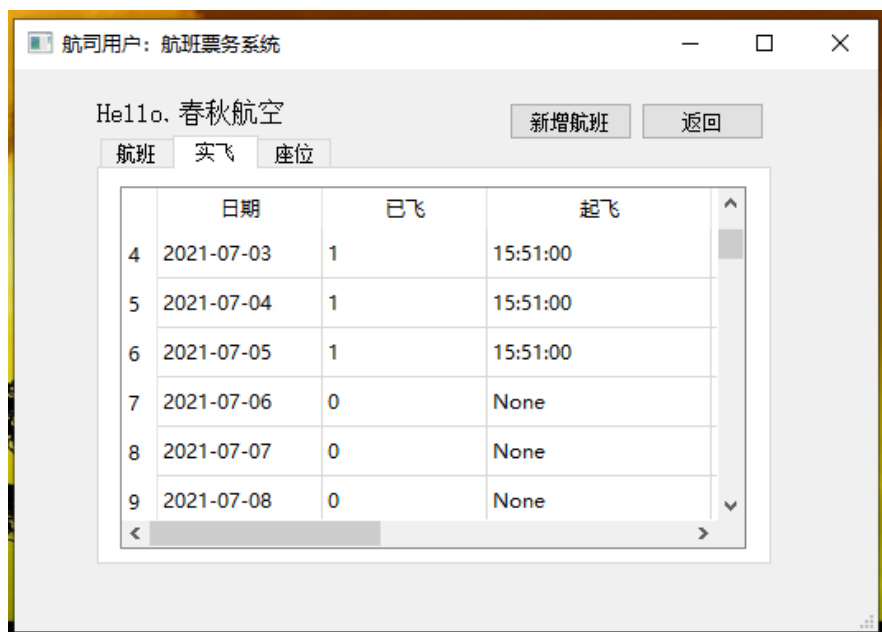


图 4.74 航班管理测试 3

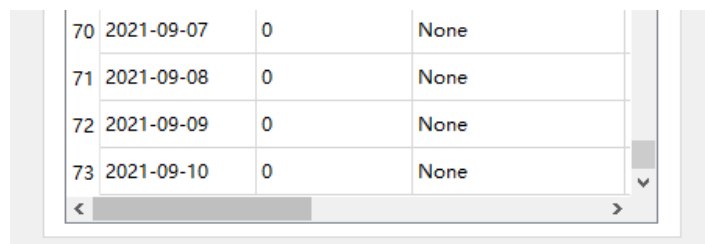


图 4.75 航班管理测试 4

选中 2021-7-7 的实飞，点击修改，将该趟实飞的价格下调 80 元，如图 4.76 所示。

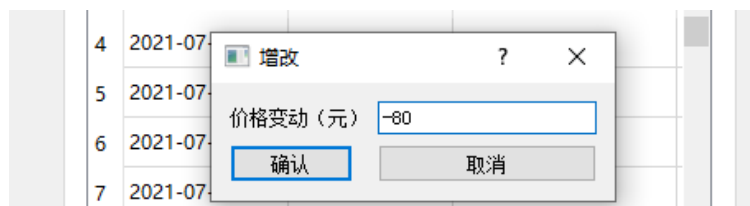


图 4.76 实飞管理测试 1

再次查看该实飞价格，发现三舱价格都已下降 80 元，如图 4.77 所示。

	头等舱价	商务舱价	经济舱价
4	2385	954	333
5	2385	954	333
6	2385	954	333
7	2385	954	333
8	2305	874	253
9	2385	954	333

图 4.77 实飞管理测试 2

选中 2021-7-7 的实飞，点击查看机票，系统自动跳转到“座位”页，如图 4.78 所示，可查看到数张实名票，均已支付、部分已值机。

航司用户: 航班票务系统

Hello. 春秋航空

新增航班 返回

航班 实飞 座位

	乘客身份证号	舱位	状态	订单号	座位	取
1	11111111...	3	2	1094	None	0tKF8n
2	11111111...	3	2	1094	None	0tKF8n
3	11111111...	3	2	1094	None	0tKF8n
4	12345678...	3	2	1313	None	AYKI2Z
5	12345678...	3	2	1313	Y5	AYKI2Z

图 4.78 机票管理测试

3.6.4 角色授权的验证与解决

在为航司应用子系统实现用户授权与视图机制时，我首先尝试例用触发器为每个新增的航司创建用户，但失败告终，原因是触发器不允许隐式提交，如图 4.79 所示。

```
mysql> create trigger add_cad after insert on company for each row begin
-> declare u_name char(30);
-> set u_name=new.name;
-> create user u_name;
-> grant company_ad to u_name;
-> end#
ERROR 1422 (HY000): Explicit or implicit commit is not allowed in stored function or trigger.
```

图 4.79 触发器报错

在用 python 代码为每个新增航司创建用户并授权为 company_ad 角色后，我始终遇到 1044 连接报错，如图 4.80 所示。

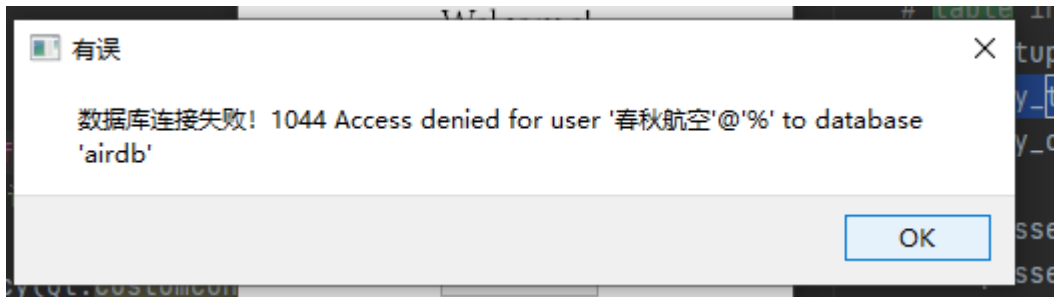


图 4.80 1044 连接报错

在此我先尝试将航司的数据库用户的 grant 和 super 权限设为 YES，并重新启动 mysql 服务器，如图 4.81 所示，但仍然为 1044 报错。

```
mysql> update mysql.user set Grant_priv='Y',Super_priv='Y' ;
Query OK, 31 rows affected (0.02 sec)
Rows matched: 33  Changed: 31  Warnings: 0

mysql> select host, user, grant_priv, super_priv from mysql.user
```

host	user	grant_priv	super_priv
%	company_ad	Y	Y
%	tt8	Y	Y
%	wly	Y	Y
%	上海航空	Y	Y
%	东方航空	Y	Y
%	东海航空	Y	Y
%	中国国航	Y	Y
%	乌鲁木齐航空	Y	Y
%	九元航空	Y	Y
%	北部湾航空	Y	Y
%	华夏航空	Y	Y
%	南方航空	Y	Y
%	厦门航空	Y	Y
%	吉祥航空	Y	Y
%	四川航空	Y	Y
%	多彩航空	Y	Y
%	天津航空	Y	Y
%	山东航空	Y	Y
%	幸福航空	Y	Y
%	我的航空	Y	Y
%	昆明航空	Y	Y
%	春秋航空	Y	Y
%	河北航空	Y	Y
%	海南航空	Y	Y
%	深圳航空	Y	Y
%	瑞丽航空	Y	Y
%	祥鹏航空	Y	Y
%	西藏航空	Y	Y
%	长龙航空	Y	Y
localhost	mysql.infoschema	Y	Y
localhost	mysql.session	Y	Y
localhost	mysql.sys	Y	Y
localhost	root	Y	Y

```
33 rows in set (0.00 sec)
```

图 4.81 授权尝试

之后通过 mysql workbench 以航司用户登录数据库，定位问题在于航司用户无法连接具体的 airdb 数据库，仅能在为各个用户专门授予对三个视图的操作权限时才能连接上 airdb，尝试记录如图 4.82 所示。

✖	4	10:08:16	use airdb
✔	5	10:08:53	use airdb
✖	6	10:10:05	use airdb

图 4.82 授权问题定位

为什么在将拥有 airdb 三个视图的所有权限的角色 company_ad 授予航司用户后，航司仍然没有连接 airdb 呢？经过其他多种尝试后，我发现问题在于 mysql 对角色管理有不同的设定。

依旧用已授权为 company_ad 角色的航司用户登录 mysql，查看 current_role，发现结果为 none，而不是预期的 company_ad，如图 4.83 所示。

```
3 • select current_role();
```

Result Grid	Filter Rows:	Export
current_role()		
NONE		

图 4.83 current_role 查询结果 1

用 root 账户登录 mysql，并为每个航司用户执行 sql 语句“set default role all to 航司用户名”后，再用航司用户登录 mysql 并查询 current_role，如图 4.84 所示。再执行航司应用子系统，发现 1044 的连接问题已解决。

```
1 • select current_role();
```

Result Grid	Filter Rows:	Export
current_role()		
▶	company_ad`@`%`	

图 4.84 current_role 查询结果 2

综上分析，在 mysql 中，一个用户可以被授予多个角色，但并不会随时都承担这些被授予的角色，需要通过“set default role”，才能够使用户默认执行该角色。

4.7 系统设计与实现总结

1. 确立基本设计思想

在确定选题为航空售票系统后，在开始具体设计系统前，我首先需要明确，自己期待的航空售票系统应该拥有哪些基本设计思想。

首先，机票的购票者和乘坐者分离。该系统售票时，面向的不是单独的、只能选择一张票的乘客本人，而是抽象的、能够一次性购多张票的购票人。这样的设计思想会提高系统使用的友好度。

其次，订单的建立过程和支付过程分离。订单的建立意味着用户暂时抢到心仪的机票了，但他还能够有一定的时间裕度反悔，但反悔时限又不能太长，所以订单首先需要建立，之后需要在十五分钟内完成支付，否则订单超时失效。这样的设计思想同样是为了提供系统使用的友好度。

最后，系统务必要有可前调的模拟日期。机票业务是典型的与日期、时间密不可分业务，所以即使是模拟系统，也不能割舍长达数日的时间变迁给机票系统带来的影响。但此处我们的系统需要在数分钟内完成测试，所以该系统必须要有可前调的模拟日期。

2. 勾勒系统框架

围绕着“买机票”这一核心业务，我开始构思整个系统会有哪些使用者，最终敲定游客、系统管理员、航司票务管理员和乘客用户四类角色，这四类角色构建了整个系统框架。

3. 细化核心业务流程

在敲定具体数据项前，必须先思考“购票”这一核心业务是如何执行的，执行期间会涉及到哪些数据项。

4. 确定系统 ER 图

根据上述步骤分析得到的关系与联系，确定系统的 ER 图，进而完成数据库建表操作。

5. 选择具体数据库技术

结合系统应用需求，我开始思考分别用哪些具体的数据库技术去优化系统实现，包括触发器、存储过程、排他锁、事务、视图和权限管理等等。

6. 设计 GUI 原型

为了使头脑中的系统成品具象化，在正式编程前，我首先用 QT 图形化工具绘制了各主要界面的 GUI 原型。

7. 编程与调试

按照页面出现的顺序，分模块依次编程实现，并阶段性测试、调试。

5 课程总结

1. 工作总结

在实现任务三时，最主要的工作在 4.7 中已大致总结，此外还有测试数据搜集和整理导入、报告书写等其他工作。

2. 技术收获

首先是对学会了如何灵活运用多项具体的数据库技术，能够较熟练的操作 mysql、SQL Server 等主流 DBMS。

其次是对 python、QT 等实现工具的熟练，尤其增进了面向对象编程的熟练度。

最后是遇到问题解决问题的能力得到提升。我在编程调试过程遇到过 mysql、python、QT 的各种问题，为了能让系统达到最初的设计预期，我放弃了绕道回避这一捷径，坚持学习和探索，最终预期的效果都基本实现。

3. 思想收获

本科阶段我们做了很多实验，在做任务三时，我时常思考数据库实践相比之前的基础实验的不同和提升在什么地方。

我认为，数据库实践的不同在于，它相比起数据结构课设，在后端的逻辑、算法方面明显更为薄弱（大部分情况），但这不意味着数据库实践在开倒车。实际上，数据库实践的提升，就是在于将我们的编程思维从深度，引导向了另一个同样重要的维度——广度。

在之前的编程中，我们常常集中精力攻克一个点，达到了编程的深度。但在实际应用中，代码业务逻辑常常需要处理多方的数据和操作请求，即为编程的广度。遗憾的是，在大学里，深度的考验很容易给出，但很难给出贴合实际的对广度的考验，久而久之，我自己也忘记了这一维度的存在。

数据库实践，通过给出对广度编程的问题的解决方案，提醒了我广度编程的问题的存在。所以，通过解决问题，让我意识到编程在很多时候也要考虑广度，是我在本次实践中的重要思想收获之一。

此外，积极思考和学习新技术的运用，遇到问题耐心解决等等，也是在编程锻炼中一些宝贵的思想收获。

4. 有待完善

该系统有最本质但也难以彻底改进的问题，就是业务覆盖面不广。比如中转飞机、中途休息飞机、航行时间跨天的飞机等因素，在本系统中都被忽略了。问题并非无法考虑和解决，但由于本系统完成时间有限，所以只能实现基本考虑和部分现实考虑。但我相信，每一个新考虑因素的解决，都会令本系统在数据库技术的运用上更加丰富。

5. 致谢

首先感谢左琼老师的悉心指导，在 ER 图绘制阶段，指出了我在“实飞-订单-机票”这三个实体的联系设计和属性设计的含糊之处；其次感谢物联网 1801 王英嘉同学，他在去年完成的数据库实践项目很优秀，我在选择实现工具时，参考了他的 GUI 设计工具，最终同样选择了 pyqt5，事实证明该开发工具效率确实很高。

附录

1. 建表

```
create database covid19;
```

```
use covid19;
```

```
-- 人员表
```

```
create table person (  
  id int constraint pk_person primary key,  
  fullname char(20) not null,  
  telephone char(11) not null  
);
```

```
-- 地点表
```

```
create table location(  
  id int constraint pk_location primary key,  
  location_name char(20) not null  
);
```

```
-- 行程表
```

```
create table itinerary(  
  id int constraint pk_itinerary primary key,  
  p_id int constraint fk_itinerary_pid foreign key references person(id),  
  loc_id int constraint fk_itinerary_lid foreign key references location(id),  
  s_time datetime,  
  e_time datetime  
);
```

```
-- 诊断表
```

```
create table diagnose_record(  
  id int constraint pk_diagnose_record primary key,
```

```

p_id int constraint fk_diagnose_pid foreign key references person(id),
diagnose_date datetime,
result int
);

```

-- 密切接触者表

```

create table close_contact(
id int constraint pk_close_contact primary key,
p_id int constraint fk_contact_pid foreign key references person(id),
contact_date datetime,
loc_id int constraint fk_contact_lid foreign key references location(id),
case_p_id int constraint fk_contact_caseid foreign key references person(id)
);

```

-- 隔离地点表

```

create table isolation_location(
id int constraint pk_isolation_loc primary key,
location_name char(20),
capacity int
);

```

-- 隔离表

```

create table isolation_record(
id int constraint pk_isolation primary key,
p_id int constraint fk_isolation_pid foreign key references person(id),
s_date datetime,
e_date datetime,
isol_loc_id int constraint fk_isolation_lid foreign key references
isolation_location(id),
state int
);

```

2. 查询

--1.0

```

select location_name, count(*) as visitors

```

```

from itinerary join location on location.id=itinerary.loc_id
group by loc_id,location_name
having count(*) > 30
order by count(*) desc, location_name asc;

```

--1.1

```

select location_name, visitors
from location, (select loc_id, count(loc_id) visitors from itinerary group by loc_id
having count(loc_id)>30) loc_count
where location.id=loc_count.loc_id
order by visitors desc, location_name asc;

```

--2

```

select location_name, count(*) as number
from isolation_record join isolation_location on
isolation_record.isol_loc_id=isolation_location.id
where state=1
group by isol_loc_id,location_name
order by number desc, location_name asc;

```

--3

```

select person.id as 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, itinerary I1, itinerary I2, location L1, location L2
where I1.e_time=I2.s_time and I1.loc_id=L1.id and I2.loc_id=L2.id and
person.id>30 and I1.p_id=I2.p_id and I1.p_id=person.id
order by person.id, reclosing_time;

```

--4

```

select fullname, telephone, location_name, convert(char(19), s_time, 20) as s_time,
convert(char(19), e_time, 20) as e_time
from (itinerary join location on itinerary.loc_id=location.id)right outer join person
on (person.id=p_id)

```

```
where fullname=N'充珉瑶' or fullname=N'贾涵山'
order by person.id desc, s_time;
```

```
--5
select id, location_name from location
where location_name like N'%店%'
order by id;
```

```
--6
select fullname, telephone
from person, itinerary
where person.id=itinerary.p_id and itinerary.loc_id=(select id from location where
location_name=N'活动中心') and
(((s_time between '2021-02-02 20:05:40.000' and '2021-02-02 21:25:40.000') or
(e_time between '2021-02-02 20:05:40.000' and '2021-02-02 21:25:40.000')) or
(s_time<'2021-02-02 20:05:40.000' and e_time > '2021-02-02 21:25:40.000'))
order by fullname;
```

```
--7 wei ce
select location_name
from isolation_location
where exists (
select * from isolation_record
where isolation_record.isol_loc_id=isolation_location.id and state=1
)
order by id;
```

```
--8
select top 30 fullname, telephone
from person
where exists (select * from itinerary where p_id=person.id)
order by person.id;
```

--9

```
select count(*) as number
from person
where not exists (select * from itinerary,location where
itinerary.loc_id=location.id and location.location_name=N'Today 便利店' and
p_id=person.id );
```

--10

```
select fullname
from person
where not exists (
select loc_id
from itinerary A
where not exists (
select *
from itinerary B
where B.loc_id=A.loc_id and B.p_id=person.id
)
)
order by fullname;
```

--11

```
GO;
create view isolation_location_status(id,location_name,capacity,occupied)
as
select isolation_location.id,location_name,capacity, sum(case when state = 1 then
1 else 0 end)as occupied
from isolation_location left outer join isolation_record on
(isolation_location.id=isolation_record.isol_loc_id)
group by isolation_location.id,location_name,capacity;
GO;
--select * from isolation_location_status;
```

--12

```
select location_name, capacity-occupied as available_rooms
from isolation_location_status
order by id;
```

--13-----

```
select fullname, telephone
from person PA
where exists (
  select *
  from itinerary A, itinerary B, person PB
  where A.p_id=PA.id and B.p_id=PB.id and PB.fullname=N'靳宛儿' and
  A.loc_id=B.loc_id and
    (((A.s_time between B.s_time and B.e_time) or (A.e_time between B.s_time
and B.e_time)) or
    (A.s_time < B.s_time and A.e_time>B.e_time))
) and fullname<>N'靳宛儿'
order by fullname;
```

```
select * from itinerary where p_id=14;
```

```
select *
```

```
from itinerary A, itinerary B, person PB
where A.p_id=14 and B.p_id=PB.id and PB.fullname=N'靳宛儿' and
A.loc_id=B.loc_id ;
```

--14

```
select location_name, count(*) as close_contact_number
from close_contact join location on close_contact.loc_id=location.id
group by location.id, location_name
```

```
order by close_contact_number desc, location_name;
```

```
--15
```

```
select top 1 person.id as case_p_id, fullname, count(*) as infected_number
from person, close_contact
where person.id=close_contact.case_p_id
group by person.id, fullname
order by infected_number desc;
```

```
--16
```

```
select top 3 fullname, count(*) as record_number
from person, itinerary
where person.id=itinerary.p_id
and (((s_time between '2021-02-02 10:00:00.000' and '2021-02-02 14:00:00.000')
or (e_time between '2021-02-02 10:00:00.000' and '2021-02-02 14:00:00.000'))
or (s_time< '2021-02-02 10:00:00.000' and e_time >'2021-02-02 14:00:00.000'))
group by person.id, fullname
order by count(*) desc;
```

```
--17
```

```
select location_name, iso_loc_rank.capacity as capacity
from isolation_location F, (
select distinct A.capacity, sum(case when A.capacity < B.capacity then 1 else 0
end) as ranked
from (select distinct capacity from isolation_location)A, (select distinct capacity
from isolation_location)B
group by A.capacity
)iso_loc_rank
where F.capacity=iso_loc_rank.capacity and iso_loc_rank.ranked=1;
```

3. 触发器

```
--triger
```

```

go
create trigger tri_confirmed
on diagnose_record
after insert,update
as
begin
    update isolation_record
    set state=3
    from isolation_record, inserted
    where isolation_record.p_id=inserted.p_id and inserted.result=1
end;

```

4. 函数

```

--function
go
create function Count_Records(@pid int) returns int
begin
    return (select count(*) from itinerary where p_id=@pid );
end

```