设计文档

目录

[一、 项目背景 1](#_Toc139195968)

[二、 需求分析 1](#_Toc139195969)

[三、 概念设计 2](#_Toc139195970)

[四、 逻辑设计 3](#_Toc139195971)

[五、 物理设计 4](#_Toc139195972)

[六、 开发文档 4](#_Toc139195973)

[七、 心得体会 10](#_Toc139195974)

# 项目背景

在交通无比发达的现在，乘坐各种交通工具变得极为简单，前往目的地变得更快、更安全。乘坐各种交通工具离不开的就是购票系统，本次课程设计结合所学到的基础数据库知识，利用python+Flask+SQL Alchemy完成一个简易的车票购买系统。

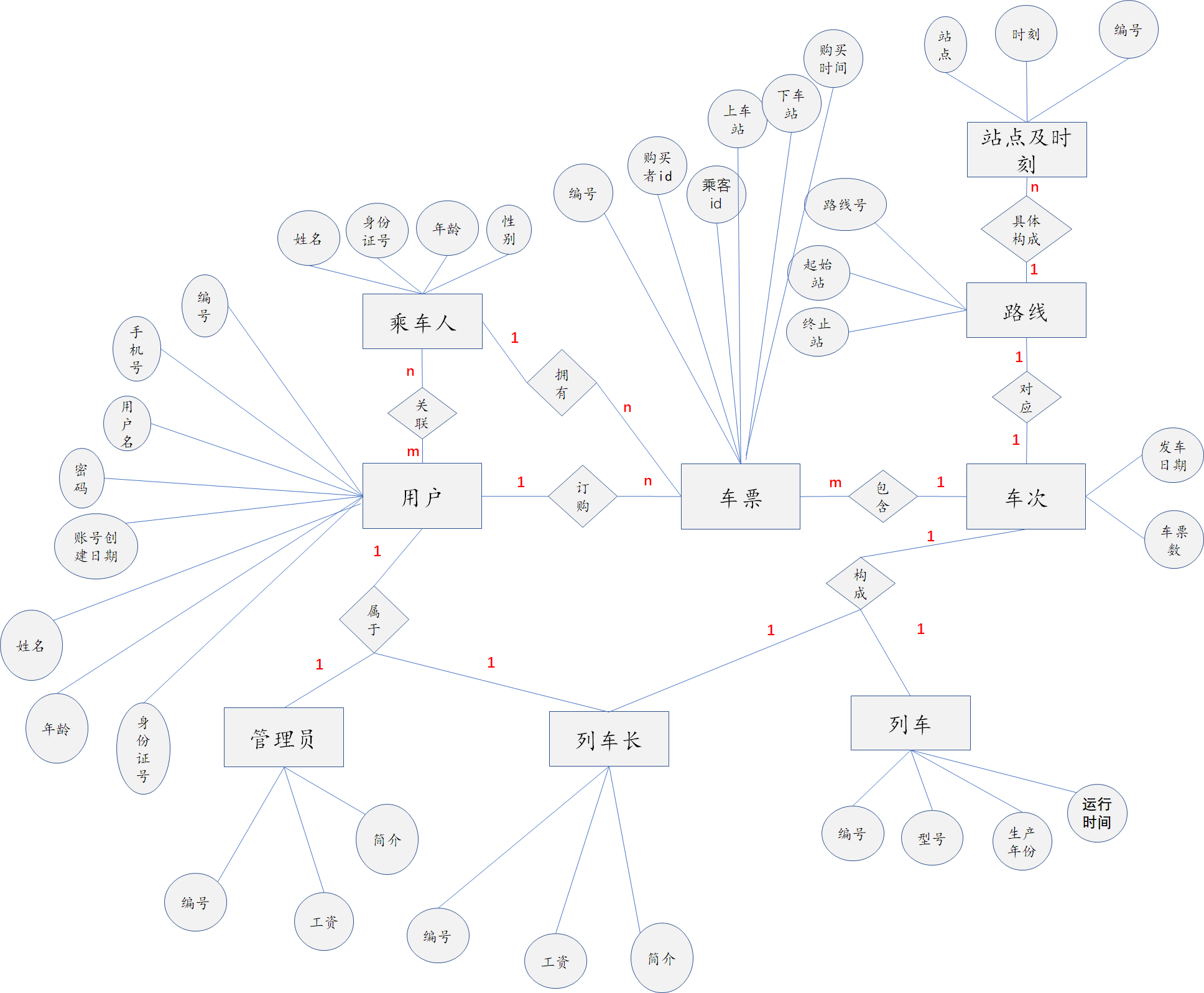
# 需求分析

本系统意在构建一个简易的车票购买系统。需要实现的功能包括以下部分：

* 根据用户手机号以及密码实现登录、注册，并且将用户分为：普通用户（普通乘客）、列车长、管理员（车站管理者，能够修改列车、票等信息）。
* 切换身份（管理员🡨->乘客，列车长🡨->乘客）
* 根据给出的信息：时间、始发站、终点站等进行车次（车票）信息查询。（时间、站点信息、剩余票数、价格、列车号等车票的基本信息）
* 根据给出的列车号查询列车信息（包含列车概况：型号，列车座位信息等等；列车长信息等等）
* 列车车长自己的排班情况。
* 用户（普通乘客）购买车票。
* 用户查询自己购买的车票。
* 管理员创建车票（车次、站点）信息。
* 用户（普通乘客）取消订单。
* Web端则是上述各种功能、按钮、页面跳转的实现。

# 概念设计

E-R图



# 逻辑设计

关系模型如下

用户（**用户编号**，手机号（唯一），用户名，密码，创建日期，姓名，年龄，性别，用户身份）

管理员（**管理员编号**，用户编号，工资，个人简介）//用户id做主键

乘车人（**身份证号**，姓名，年龄，性别）

列车长（**列车长编号**，用户编号，工资，简介）//可以直接拿用户id直接做主键

用户乘客表(**id**，用户编号，乘客身份证号) // 外键为用户id（购票者）和passenger idcard

车票（**车票编号**，用户编号，乘客身份证号，车次编号，使用状态，下单时间，上车站，下车站）

车次（**车次号**，发车日期，路线号，列车号，列车长编号，车票数，余票数）

路线（**路线号**，名称，起始站，终点站）

站点时刻（**站点号**，路线号，站点名，抵达时刻，停留时间）

列车（**列车编号**，型号，生产年份，数量，简介，运行时间）

# 物理设计

由于构建的数据库，仅仅是对于数据库知识的简单运用，所以物理设计部分没有过多考虑。

关系模式的存取方法为MySQL默认的B+树索引方法。

而对于每个Table，即使可能那个表有许多能够作为主键的，例如user表中的身份证、电话号…但是我们仍为每个表建立一个自增的id，原因在于对于这个小型的系统我们不必要为其考虑太多，使用自增id编程实现较为方便。当然劣势在于id自增后无法控制，不像身份证和电话一样。

数据的物理存储结构就完全依赖于MySQL的存储了，并没有特别考虑。

# 开发文档

1. 开发环境、语言

关系数据库：本次大作业使用的关系数据库为MySQL 5.7。

语言+库/包：Flask+flask-SQL-Alchemy+python（3.9）进行开发。（附有requirement.txt）

操作系统：Windows11

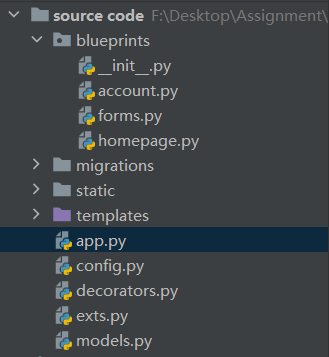
1. 开发思路

使用MySQL作为关系数据库，负责数据存储，而使用Flask进行后端构建，同时利用sqlalchemy的orm特性，对数据库进行操作。（实际编程实现中还是写了不少原生sql语句）。

大体上，利用继承自SQL alchemy的model来建表（当然也可以直接使用table），然后使用query语句已经原生sql语句进行查询、插入、删除、修改。利用视图函数对各种功能实现（类似使用视图）。使用html+css+js（实际上没怎么使用css+js，因为相应知识匮乏，只用html搭建了一个骨架）进行前端的交互。

由于Flask为我们短期速成的，所以编程实现上可能比较丑陋（例如建表时指定了relationship属性，但是在增删改查时，并没有怎么使用…以及一些复杂的查询都是拿原生SQL语句写的）。

1. 代码简要介绍
2. 代码结构



Source code

|---blueprints（蓝图，主要功能实现的部分）

|---

|---\_init\_.py

|---account.py（实现的功能和“account”有些不符，结构没划分好，包含登录、注册、查询购买的车票、关联乘客、展示captain、trip、tain等等以及相应对象的增添等等）

|---forms.py(表单验证的功能函数)

|---homepage.py（主页，主要就是实现了查询车票的功能）

|---migrants（版本控制产生的，提交代码时会删去，运行时会产生）

|---static （存储css、js、图片等的地方，本次只使用了一点点js）

|---templates（各种html文件储存的地方，各种html的命名应该比较合理一眼就能看出是什么功能）

|---app.py （主函数，函数入口）

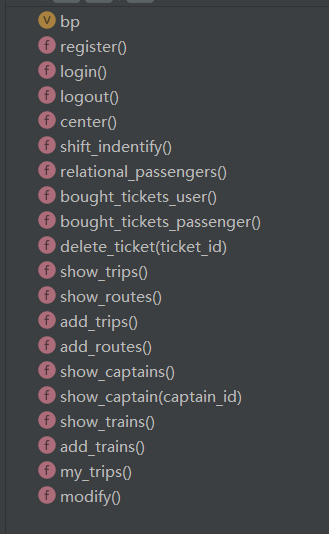
|---config.py（存储各种参数，运行时可能需要修改）

|---decorator.py（要求登录的功能函数）

|---exts.py（防止重复引用…）

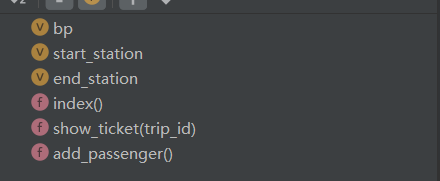
|---models.py（建表）

1. Buleprints--account内容简介



* Register为注册功能，get请求则返回渲染register.html页面，post也即提交表单时，进行输入插入user+passenger+（？captain/admin，依据注册的身份），相应检查格式（表单），但是身份证格式、电话格式（首位为1）只是简单判断了长度，没有做更详细的校验。值得一提的是，密码采用了werkzeug.security进行了hash加密。（所以密码是varchar（100））
* Login\logout，分别为登录、退出登录，登录就是简单查询一下电话号，使用werkzeug.security的check-password-hash比较hash值比对储存的密码。Logout为登出，并且清除cookie。
* Center为用户中心，渲染用户中心页面（基于身份）。而shift\_indentify则切换身份（passenger<->captain or passenger<->admin,前提是有相应权限，admin、captain都可以切换为passenger）。Modify为修改信息。（由于差不多是只使用了html，所以会看着有点不太合理2333）
* Relational\_passenger为用户中心功能，乘客（身份设置为乘客时）查询关联的乘客，也即userpassenger表。同时支持删除关联，但是不能删除和自己的关联。
* Bought\_ticket\_user/passenger,查看自己购买的票、作为乘客的票。都是查询ticket表只不过查询条件是user\_id,passenger\_idcard的区别罢了。当然如果只是显示ticket表中的内容当然显得单调了，这里的查询关联了一系列表，使用原生SQL语句进行的查询。Delete\_ticket即根据ticket\_id删除车票。（这里传参是使用了url传参，可能会显得不安全？）（值得一提的是，本文件使用的时间都为utc时间，所以可能和你的预期有差距？）
* Show\_trip\routes\captain\captains，则是字面意思根据查询的数据（大都关联了多个表，使得渲染的页面的信息没那么单调）（show\_trips只会显示当前日期之后的车次…，但是其它板块却疏忽了对于日期的考虑，诸如查询车票时…值得改进）
* Add\_trips\routes\trains，针对管理员身份，添加车次（trips）、路线（routes）、列车（trains），其中routes插入了route、station两个表。
* My\_trips为列车长查询自己的排版，也即根据trip的captain\_id查询（关联route…）。

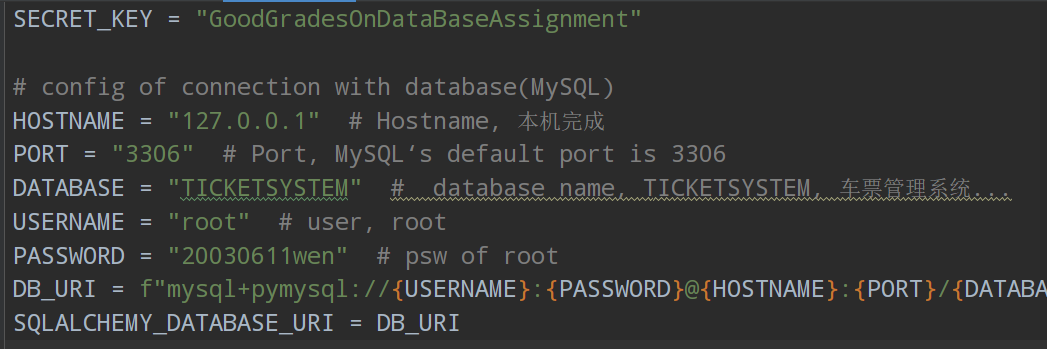
1. Blueprints—homepage简介



* Index为首页显示内容，get请求渲染页面，post请求（其实有点不妥，直接使用<a>跳转即可）跳往show\_ticket，也即买票详情页。
* Show\_ticket则是根据用户的点击，进行买票、增加关联乘车人…
* Add\_passenger插入userpassenger表（根据输入数据）

1. Template 中的html使用了flask依赖的jinjia2的模板继承功能。
2. 如何使用？

* 首先确保安装了MySQL 5.7，python3.8
* 然后根据requirement.txt进行各种包、库的安装（如果有什么疏漏的包的话，请按照报错安装…由于这个环境有一些多余的包，所以没有直接导出该环境的所有包…）
* 其次登录MySQL，根据配置



诸如用户名、密码相对应地登录MySQL，然后创建数据库：

CREATE DATABASE IF NOT EXISTS TICKETSYSTEM DEFAULT CHARSET utf8 COLLATE utf8\_general\_ci;

（SECRET\_KEY为密码hash加密的hash值，可以修改，port为端口号）

* 随后终端进入本文件夹，输入flask db init,flask db migrate, flask db upgrade,完成表的创建（我们开发使用的是pycharm-official+anoconda，运行则是激活虚拟环境，然后终端输入这三条指令即可，ps：或许不开发，使用db.session.create\_all()建表就好了…不必使用flask-migreate）
* 随后运行即可。（如果不导入数据，需要以管理员身份，先创建路线，然后创建车次…当然，创建车次需要有列车长，也就是说还需要先注册列车长）
* 可以使用附加的data.sql导入一些数据（包含route、trip、station，但是不包含用户、列车长、管理员，用户需要自己创建…）（注意！由于导入数据依赖于一些id，所以如果用户自己尝试过添加数据后，导入的data.sql就有问题了）

1. 值得改进的地方：包括前端美化，以及一些页面逻辑，以及对于时间的考虑诸如车票过期（其实是有state属性的…，但是由于ddl…）本次大作业删除部分设计较少，例如站点、路线、车次的删除，虽然实际生活中很少产生但是确实实实在在可能发生的，这也是有待改进的地方，毕竟本系统缺少级联删除的设定…以及一些页面增加一些按钮，比如用户可以查看所有车次等等。

# 心得体会

首先是，本次大作业给了我们一个实践数据库知识的机会，不仅仅是实验的黑压压的命令行，大作业的成果是可见的，令人高兴的。

其次，本次大作业使用了flask，缘由是在b站看到了一个讲flask的很基础的视频，于是学了一下，然后使用到了本次作业当中。算是锻炼了自学能力😏😏

然后，最终结果来看，由于前端知识的匮乏，导致页面不怎么好看（css部分求助于ChatGPT，但是程序完全是自己写的，包括html代码）…

最后，感谢小组成员（虽然只有两位，能加分么🐼🐼）的共同努力！🥰🥰