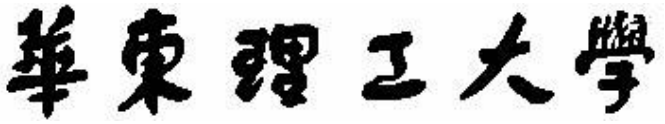


《数据库原理》

实验报告本

2022 ~ 2023 学年第一学期

专业	计算机科学与技术	计算机科学与技术
班级	计 203	计 203
姓名	刘子言	林芷珊
学号	20002462	20002451
指导教师	王占全	

计算机科学与工程系

2022 年 10 月

实验名称：“交通违章信息管理系统”数据库应用系统设计和实现	**实验地点**：信息大楼-216

所使用的工具软件及环境：Python+django+MySQL

姓名	数据库设计（50%）	系统答辩演示（25%）	实验报告（25%)	总分
刘子言				
林芷珊				

教师签名：
日期：

一、实验目的

1．通过实践，掌握数据库应用系统设计和实现方法；

2．学会在一个实际的 RDBMS 软件平台上创建数据库系统(建议使用 openGauss 软件)；

3．掌握通过 ODBC、ADO、JDBC 访问数据库的方法。

二、实验内容

1 系统背景和需求分析

1.1 系统背景

随着社会经济快速发展和城市化进程加快，交通需求持续高涨，车辆大量增加，流量迅猛增长，城市交通需求与供给的矛盾日益突出，已成为社会关注的热点、难点问题，公安交通管理工作面临着前所未有的压力。

面对21世纪经济、社会发展，及其对交通管理工作提出的新要求，靠简单的、拼体力的传统管理方式已经不能适应现代化管理的需要,必须在科学管理和科技手段的应用上找出路，走科技强警之路，向科技要管理、要效益。而现代科学技术的迅猛发展，为交通管理领域广泛应用科技手段提供了强大的技术支持，综合性交通信息管理系统既是必要的又是必然的,其建设能够为智能交通管理的未来发展奠定坚实的基础。

交通违章信息管理系统，是综合性交通信息管理系统的核心组成部分。我们通过查阅文献发现，在档案管理系统基本完善后交通管理基础信息已然具备，交通违章信息管理系统的建设就成为可能，该系统是交通管理业务最迫切需要的系统。公安部于2000年3月1日，在全国范围内颁发实施《机动车驾驶员交通违章记分办法》、《交通违章处理程序规定》(公安部45号令、46号令)，近期又制定了《交通违章数据库标准》，开始建设“违章交换平台”，可见在全国范围内交通违章信息管理系统的建设与完善已经成为重中之重。

1.2 需求分析

1.2.1 需求概述和系统边界

随着国内近几年联网数字信息共享和在线业务办理的普及，让在线发布、经办、处理交通违章处罚成为可能。本系统面向全体驾驶员以及公安部门相关工作人员，对驾驶员、车辆、警员等基本信息以及针对交通违章罚单的开具、经办、发布、处理等进行统一管理，能够最大限度地为公安部门工作人员节省时间与人力，提高工作和管理效率，同时能够为广大驾驶人员接收、查看、处理罚单事项带来便捷，助力警民数字化信息化业务模式的构建。

本系统包含针对交警和驾驶员不同视角的业务流程管理和违章信息处理，以及基本的数据检索：针对警务人员，主要是有权限限制的对违章信息的增删改查；针对普通驾驶人员，主要是违章处罚信息查看和罚款缴纳、扣分处理等。本系统可以促进业务办理效率、规范性、服务水平的提高，能够提供准确清晰的管理依据，从而促进管理水平的提高，最终达到促进交通安全意识形成、减少违章降低事故、提高道路通行能力的理想效果。

1.2.2 功能需求分析

通过查阅相关文献，考虑到公安部门及广大驾驶人员对交通违章信息管理系统的需求，本系统主要应提供如下功能（注：工作人员特指警务工作人员，不包括普通驾驶人员及其他）：

（1）驾驶员基本信息管理。提供驾驶员基本信息的录入、维护和查询功能，包括：

- 工作人员录入驾驶员基本信息；
- 工作人员修改、增加及删除驾驶员基本信息；
- 驾驶员登录系统时系统需要查询、对比基本信息中的驾驶执照号和密码；
- 所有已登录用户可以根据驾驶执照号查询驾驶员的基本信息。

（2）车辆基本信息管理。提供车辆基本信息的录入、维护和查询功能，包括：
• 工作人员录入车辆基本信息；
• 工作人员修改、增加及删除车辆基本信息；
• 所有已登录用户可以根据车牌照号查询车辆的基本信息。
（3）警察基本信息管理。提供在一定权限限制的基础之上，警察基本信息的录入、维护和查询功能，包括：
• 工作人员录入警察基本信息；
• 工作人员修改、增加及删除警察基本信息；
• 警察登录系统时系统需要查询、对比基本信息中的警号和密码；
• 警察作为已登录用户，可以根据警号查询警察的基本信息。
（4）交通违章通知书基本信息管理。提供交通违章通知书基本信息的录入、维护和查询功能，包括：
• 经办警察录入交通违章通知书基本信息；
• 经办警察修改、增加及删除交通违章通知书基本信息；
• 所有已登录用户可以根据通知书编号查询与自身相关的交通违章通知书的基本信息。
（5）处罚基本信息管理。提供处罚基本信息的录入、维护和查询功能，包括：
• 经办警察录入某个交通违章通知书的处罚基本信息；
• 经办警察修改、增加及删除某个交通违章通知书的处罚基本信息。
（6）处理处罚管理。提供针对不同处罚方式的处理、申请审核功能，包括：
• 所有已登录用户可以根据通知书编号查询与自身相关的交通违章通知书的处罚基本信息。
• 警告处理：驾驶员点击阅读通知书处罚的警告提示，阅读完毕以后，勾选“已阅读并接收”，该条处罚的处罚状态转变为“已处理”；
• 罚款处理：驾驶员按照规定金额缴纳罚款，缴纳成功以后，该条处罚的处罚状态转变为“已处理”；
• 扣分处理：驾驶员利用自身驾照扣除规定分数以后，该条处罚的处罚状态转变为“已处理”；
• 暂扣驾驶执照处理：驾驶员携带自身驾照前往派出所办理手续以后，该条处罚的处罚状态转变为“已处理”；

1.2.3 数据需求分析

（1）交通违章通知书：包括通知书编号、违章日期、地点、违章记载等信息，由通知书编号唯一标识。
（2）驾驶员：包括驾驶执照号、姓名、身份证号、地址、电话、登录密码等信息，由驾驶执照号唯一标识。其中，电话是多值属性，可能有多个，如移动号码、联通号码等。
（3）车辆：包括车牌照号、型号、生产厂家、生产日期等信息，由车牌照号唯一标识。
（4）警察：包括警号、姓名、电话、登录密码等信息，由警号唯一标识。其中，电话是多值属性，可能有多个，如移动号码、联通号码等。
（5）处罚：包括处罚编号、处罚方式、具体数值等信息，由处罚编号唯一标识。其中，处罚方式的取值，可以为暂扣驾驶执照、警告、扣分、罚款4种；具体数值的取值根据选择的处罚方式来定，比如，暂扣驾驶执照的具体数值在3-6个月之间，警告的具体数值为警告次数，扣分的具体数值在1-12分之间，罚款的具体数值在50-2000人民币之间。
（6）交通违章通知书与处罚之间的联系：需要记录针对每条处罚记录，驾驶员的处理状态。

1.2.4　业务规则及完整性约束分析

（1）一张交通违章通知书可以包含多条处罚记录，且一种处罚方式也可以对应多张违章通知书；

（2）一名驾驶员可以接收多张交通违章通知书，但一张交通违章通知书只能对应一个驾驶员；

（3）一辆车辆允许涉及多张交通违章通知书，但一张交通违章通知书只能对应一辆车；

（4）一名警察允许经办多张交通违章通知书，但一张交通违章通知书只能由一名警察经办；

（5）一次处罚中警告方式对应的具体数值为警告的次数；

（6）一次处罚中扣分方式对应的具体数值范围为2~12分；

（7）一次处罚中罚款方式对应的具体数值范围为50~2000人民币；

（8）一次处罚中暂扣驾驶执照方式对应的具体数值范围为3~6月；

（9）各种编号的编码规则：

①通知书编号的编码规则：由8位数字组成，其中前3位数字为违章类型的编号，后5位数字为序号；

②驾驶执照号的编码规则：由12位数字组成，其中前6位数字代表地域，后6位数字为序号；

③车牌照号的编码规则：第一位是汉字，是省份、直辖市以及自治区的简称，第二位是大写英文字母，是代表了地级市地区自治州的简称，部分城市采用的是顺序排列，第三位到第七位，是按字母以及数字规律排序。

④警号的编码规则：由6位数字组成，其中前2位数字代表部门或地区代码，后3位是个人编号；

⑤处罚的编号规则：由8位数字组成，一般按照时间顺序依次排序。

2　数据库概念结构设计（E-R模型）

2.1　基本实体集及属性

（1）**驾驶员（Driver）**实体集：具有驾驶执照号、姓名、身份证号、地址、电话、登录密码等属性，其数据字典如下图所示：

属性名	含义	类别	域及约束	实例
licenseId	驾驶执照号	主码	char(12)，不允许为空	320705005367
driverName	姓名		varchar(20)，不允许为空	李小勇
driverIdentify	身份证号		char(18)，不允许为空	342505200009250024
driverAddr	地址		varchar(40)	上海市徐汇区中山街道
driverPhone	电话	多值	varchar(13)	18607919999， 021-45687755
driverPwd	密码		char(20)，不允许为空	Qjs09257766

（2）**车辆（Car）**实体集：具有车牌照号、型号、生产厂家、生产日期等属性，其数据字典如下图所示：

属性名	含义	类别	域及约束	实例

carId	车牌照号	主码	char(12)，不允许为空	沪AL2925
carType	型号		varchar(30)，不允许为空	上汽大通MAXUS D90 Pro
carManu	生产厂家		varchar(30)	上汽大通制造
carMTime	生产日期		datetime	2022/01/01

（3）**警察（Police）**实体集：具有警号、姓名、电话、登录密码等属性，其数据字典如下：

属性名	含义	类别	域及约束	实例
policeId	警号	主码	char(12)，不允许为空	326568
policeName	姓名		varchar(20)，不允许为空	严正
policePhone	电话	多值	varchar(13)	13866967729，021-87009999
policePwd	密码		char(20)，不允许为空	Qjs09257766

（4）**交通违章通知书（Notification）**实体集：具有通知书编号、违章日期、违章地点、违章记载等属性，其数据字典如下：

属性名	含义	类别	域及约束	实例
notificationId	通知书编号	主码	char(8)，不允许为空	001 11000
punishTime	违章日期		datetime，不允许为空	2022/05/01
punishAddr	违章地点		varchar(40)，不允许为空	上海市徐汇区中山街道
punishDetail	违章记载		varchar(40)	公交站台违规停车
licenseId	驾驶执照号	外码	char(12)，不允许为空	320705005367
carId	车牌照号	外码	char(12)，不允许为空	沪AL2925
policeId	警号	外码	char(12)，不允许为空	326568

（5）**处罚（Punishment）**实体集：具有处罚编号、处罚方式、具体数值等属性，其数据字典如下：

属性名	含义	类别	域及约束	实例
punishId	处罚编号	主码	char(8)，不允许为空	0000 0001
punishWay	处罚方式		varchar(10)，取值为“警告”、“罚款”、“扣分”、“暂扣驾驶执照”4个选项	罚款
fine	具体数值		bigint，取值由处罚方式决定，警告对应次数，罚款对应金额（50-2000人民币），扣分对应分数（1-12分），暂扣对应月份（3-6月）	500

2.2 联系集及属性

（1）**接收（Receive）**联系集：实体集**驾驶员（Driver）**与**交通违章通知书（Notification）**之间的一对多联系集，表明一个驾驶员可接收多张罚单，而一张罚单只能由一个驾驶员接收。该联系集没有联系属性。

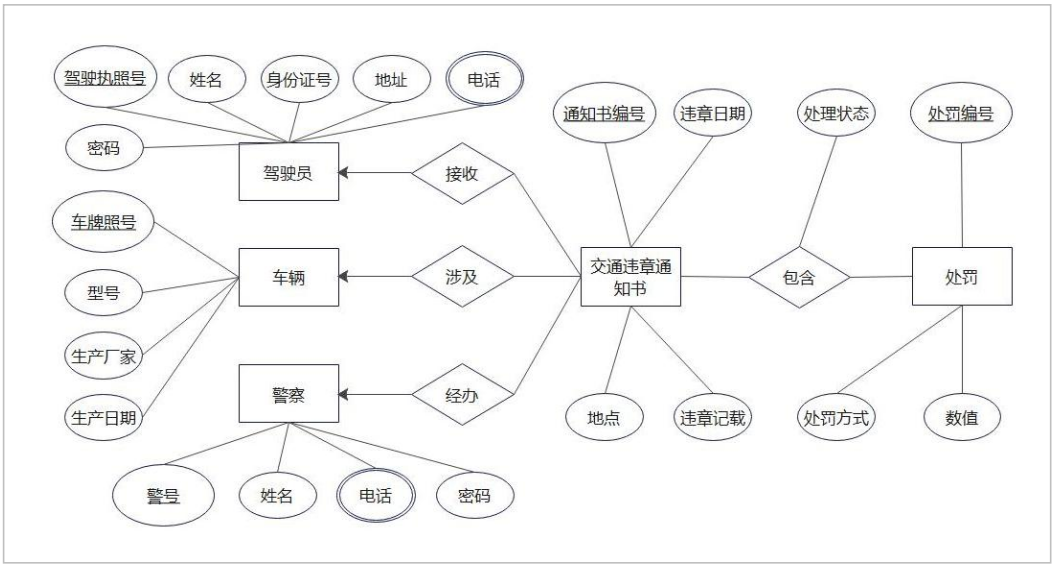
（2）**涉及（Involve）**联系集：实体集**车辆（Car）**与**交通违章通知书（Notification）**之间的一对多联系集，表明一辆车可涉及多张罚单，而一张罚单只能涉及一辆车。该联系集没有联系属性。

（3）**经办（Handle）**联系集：实体集**警察（Police）**与**交通违章通知书（Notification）**之间的一对多联系集，表明一名警察可经办多张罚单，而一张罚单只能由一个警察经办。该联系集没有联系属性。

（4）**包含（Include）**联系集：实体集**交通违章通知书（Notification）**与**处罚（Punishment）**之间的多对多联系集，表明一张交通违章通知书可以包含多种处罚方式，一个处罚方式也可以对应多张违章通知书。联系属性为处罚状态（punishState）。该联系集的数据字典如下：

属性名	含义	类别	域及约束	实例
notificationId	通知书编号	主码/外码	char(8)，不允许为空	001 11000
punishId	处罚编号	主码/外码	char(8)，不允许为空	001 11000
punishState	处理状态	联系	varchar(6)，取值为“已处理”或“未处理”，不允许为空	未处理

2.3 系统完整的E-R图



3　逻辑结构设计（关系模式集+视图）

3.1　关系模式集

（1）**驾驶员（Driver）**实体集转化的关系模式如下：

Driver (**licenseId**，driverName，driverIdentify，driverAddr，driverPhone，driverPwd)

属性名	数据类型	属性描述
licenseId	char(12)	驾驶执照号
driverName	varchar(20)	驾驶员姓名
driverIdentify	char(18)	驾驶员身份证号
driverAddr	varchar(40)	驾驶员地址
driverPhone	varchar(13)	驾驶员电话
driverPwd	char(20)	驾驶员登录密码

（2）**车辆（Car）**实体集转化的关系模式如下：

Car (**carId**，carType，carManu，carMTime)

属性名	数据类型	属性描述
carId	char(12)	车牌照号
carType	varchar(30)	车辆型号
carManu	varchar(30)	生产厂家
carMTime	datetime	生产日期

（3）**警察（Police）**实体集转化的关系模式如下：

Police (**policeId**，policeName，policePhone，policePwd)

属性名	数据类型	属性描述
policeId	char(12)	警号
policeName	varchar(20)	警员姓名
policePhone	varchar(13)	警员电话
policePwd	char(20)	警员登录密码

（4）**交通违章通知书（Notification）**实体集转化的关系模式如下：

Notification (**notificationId**，punishTime，punishAddr，punishDetail，***licenseId***，***carId***，***policeId***)

属性名	数据类型	属性描述
notificationId	char(8)	通知书编号
punishTime	datetime	违章日期
punishAddr	varchar(40)	违章地点
punishDetail	varchar(40)	违章记载
licenseId	char(12)	驾驶执照号
carId	char(12)	车牌照号
policeId	char(12)	警号

（5）**处罚（Punishment）**实体集转化的关系模式如下：

Punishment (**punishId**，punishWay，fine)

属性名	数据类型	属性描述

punishId	char(8)	处罚编号
punishWay	varchar(10)	处罚方式
fine	bigint	具体数值

（6）**包含（Include）**联系集转化的关系模式如下：

Include (***notificationId***，***punishId***，punishState)

属性名	数据类型	属性描述
notificationId	char(8)	通知书编号
punishId	char(8)	处罚编号
punishState	varchar(6)	处理状态

3.2 视图

（1）建立视图，分别显示指定的处罚方式的违章处罚信息

警告：

```
create view panish_warning
as
    select carId, licenseId, punishTime, punishAddr,
           punishWay, fine, punishState
    from   Notification a, Punishment b, Include c
    where  a.notificationId = c.notificationId
           and b.punishId = c.punishId
           and punishWay = '警告'
```

罚款：

```
create view panish_fine
as
    select carId, licenseId, punishTime, punishAddr,
           punishWay, fine, punishState
    from   Notification a, Punishment b, Include c
    where  a.notificationId = c.notificationId
           and b.punishId = c.punishId
           and punishWay = '罚款'
```

扣分：

```
create view panish_score
as
    select carId, licenseId, punishTime, punishAddr,
           punishWay, fine, punishState
    from   Notification a, Punishment b, Include c
    where  a.notificationId = c.notificationId
           and b.punishId = c.punishId
           and punishWay = '扣分'
```

暂扣驾驶执照：

```
create view panish_hold
as
    select carId, licenseId, punishTime, punishAddr,
           punishWay, fine, punishState
    from   Notification a, Punishment b, Include c
    where  a.notificationId = c.notificationId
           and b.punishId = c.punishId
           and punishWay = '暂扣驾驶执照'
```

（2）建立视图，显示驾驶员与对应违章车辆的详细信息

```
create view Driver_Car
as
    select a.licenseId,driverName,driverIdentify,driverPhone,b.carId
    from Driver a,Car b,Notification c
    where a.licenseId = c.licenseId and b.carId = c.carId
```

（3）建立视图，显示警察的可见信息

```
create view Police_Detail
as
    select policeId,policeName,policePhone
    from Police
```

4 物理设计（存储结构及索引）

4.1 存储结构

（1）建立存储过程，输入车牌照号，显示未处理的违章罚单的相关信息

```
if exists(select name from sysobjects where name='driver_punish_todo' and type='p')
    drop procedure driver_punish_todo
go
create procedure driver_punish_todo(@carid char(12))
as
    select carId, licenseId, punishTime, punishAddr,
           punishWay, fine, punishState
    from   Notification a, Punishment b, Include c
    where  a.notificationId = c.notificationId
           and b.punishId = c.punishId
           and carId = @carid
           and punishState = '未处理'
```

（2）建立存储过程，输入警号，显示该警察经办的未处理的违章罚单的相关信息

```
if exists(select name from sysobjects where name='police_punish_todo' and type='p')
    drop procedure police_punish_todo
go
```

```
create procedure police_punish_todo(@policeid char(12))
as
    select policeId, carId, licenseId, punishTime, punishAddr,
           punishWay, fine, punishState
    from   Notification a, Punishment b, Include c
    where  a.notificationId = c.notificationId
           and b.punishId = c.punishId
           and policeId = @policeid
       and punishState = '未处理'
```

（3）建立触发器，约束Punishment表中的punishWay只能取“警告”、“罚款”、“扣分”、“暂扣驾驶执照”

```
create trigger insert_punishway
on Punishment for insert,update
as
    if exists(select * from inserted where punishWay not in ('警告','罚款','扣分','暂扣驾驶执照'))
    ROLLBACK
```

（4）建立触发器，约束Punishment表中的fine在不同的punishWay的情况下的取值范围

```
create trigger insert_fine
on Punishment for insert,update
as
begin
    if exists(select * from inserted where punishWay in ('警告') and fine not in (0,1))
        ROLLBACK
    if exists(select * from inserted where punishWay in ('罚款') and fine not in (50,100,200,300,500,1000,2000))
        ROLLBACK
    if exists(select * from inserted where punishWay in ('扣分') and fine not in (1,3,6,9,12))
        ROLLBACK
    if exists(select * from inserted where punishWay in ('暂扣驾驶执照') and fine not between 3 and 6)
        ROLLBACK
end
```

4.2 索引

（1）为Notification表，建立按时间降序排序的非聚簇索引

```
create unique nonclustered
index noti_time
on Notification(punishTime desc)
```

5 完整性和安全性设计

5.1 完整性设计

（1）实体完整性

基本表的主码值唯一且不为空值，具体如下：

- **driver**

```
DROP TABLE IF EXISTS `driver`;
/*!40101 SET @saved_cs_client     = @@character_set_client */;
/*!50503 SET character_set_client = utf8mb4 */;
CREATE TABLE `driver` (
  `licenseId` varchar(12) NOT NULL,
  `driverName` varchar(20) NOT NULL,
  `driverIdentify` varchar(18) NOT NULL,
  `driverAddr` varchar(40) DEFAULT NULL,
  `driverPhone` varchar(13) DEFAULT NULL,
  `driverPwd` varchar(20) NOT NULL,
  PRIMARY KEY (`licenseId`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
```

- **car**

```
DROP TABLE IF EXISTS `car`;
/*!40101 SET @saved_cs_client     = @@character_set_client */;
/*!50503 SET character_set_client = utf8mb4 */;
CREATE TABLE `car` (
  `carId` varchar(12) NOT NULL,
  `carType` varchar(30) DEFAULT NULL,
  `carManu` varchar(30) DEFAULT NULL,
  `carMTime` datetime(6) DEFAULT NULL,
  PRIMARY KEY (`carId`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
```

- **police**

```
DROP TABLE IF EXISTS `police`;
/*!40101 SET @saved_cs_client     = @@character_set_client */;
/*!50503 SET character_set_client = utf8mb4 */;
CREATE TABLE `police` (
  `policeId` varchar(12) NOT NULL,
  `policeName` varchar(20) NOT NULL,
  `policePhone` varchar(13) DEFAULT NULL,
  `policePwd` varchar(20) NOT NULL,
  PRIMARY KEY (`policeId`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
```

（2）参照完整性

参照完整性为若干表中的相应元组建立多对一的参照关系。在SQL中，参照完整性定义是使用create table语句中的foreign key和references短语来实现，或通过使用alter table语句中的add foreign key短语来实现。具体如下：

- **notification**

```
DROP TABLE IF EXISTS `notification`;
/*!40101 SET @saved_cs_client     = @@character_set_client */;
/*!50503 SET character_set_client = utf8mb4 */;
CREATE TABLE `notification` (
  `notificationId` varchar(8) NOT NULL,
  `punishTime` datetime(6) NOT NULL,
  `punishAddr` varchar(40) NOT NULL,
  `punishDetail` varchar(40) DEFAULT NULL,
  `carId` varchar(12) NOT NULL,
  `licenseId` varchar(12) NOT NULL,
  `policeId` varchar(12) NOT NULL,
  PRIMARY KEY (`notificationId`),
  KEY `notification_carId_6fe9b0c2_fk_car_carId` (`carId`),
  KEY `notification_licenseId_e59e6e9b_fk_driver_licenseId` (`licenseId`),
  KEY `notification_policeId_3132c492_fk_police_policeId` (`policeId`),
  CONSTRAINT `notification_carId_6fe9b0c2_fk_car_carId`
           FOREIGN KEY (`carId`) REFERENCES `car` (`carId`),
  CONSTRAINT `notification_licenseId_e59e6e9b_fk_driver_licenseId`
           FOREIGN KEY (`licenseId`) REFERENCES `driver` (`licenseId`),
  CONSTRAINT `notification_policeId_3132c492_fk_police_policeId`
           FOREIGN KEY (`policeId`) REFERENCES `police` (`policeId`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
```

- **include**

```
DROP TABLE IF EXISTS `include`;
/*!40101 SET @saved_cs_client     = @@character_set_client */;
/*!50503 SET character_set_client = utf8mb4 */;
CREATE TABLE `include` (
  `notificationId` varchar(8) NOT NULL,
  `punishState` varchar(6) DEFAULT NULL,
  `punishId` varchar(8) NOT NULL,
  PRIMARY KEY (`notificationId`),
  UNIQUE KEY `include_notificationId_punishId_e494e073_uniq` (`notificationId`,`punishId`),
  KEY `include_punishId_15cb9a3b_fk_punishment_punishId` (`punishId`),
  CONSTRAINT `include_notificationId_ffd95f00_fk_notification_notificationId`
           FOREIGN KEY (`notificationId`) REFERENCES `notification` (`notificationId`),
  CONSTRAINT `include_punishId_15cb9a3b_fk_punishment_punishId`
           FOREIGN KEY (`punishId`) REFERENCES `punishment` (`punishId`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
```

- **punishment**

```
DROP TABLE IF EXISTS `punishment`;
/*!40101 SET @saved_cs_client     = @@character_set_client */;
/*!50503 SET character_set_client = utf8mb4 */;
CREATE TABLE `punishment` (
  `punishId` varchar(8) NOT NULL,
  `punishWay` varchar(20) NOT NULL,
  `fine` bigint NOT NULL,
  PRIMARY KEY (`punishId`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
```

（3）用户自定义完整性

用户自定义完整性是定义具体应用中数据必须满足的语义要求，由RDBMS提供定义、检查和处理等功能，而不必由应用程序承担。在定义关系模式时，用户自定义完整性通常仅使用列约束和元组约束两种。具体如下：

• punishment表中关于处罚方式punishway的列约束

```
`punishWay` varchar(20) NOT NULL,
    constraint punishway_CK check (punishWay in ('警告', '罚款', '扣分', '暂扣驾驶执照')),
```

• punishment表中关于警告次数/罚金/所扣分数/暂扣月数的fine的元组约束

```
`fine` bigint NOT NULL,
    constraint fine_CK check (
        (punishWay='警告' and (fine=0 or fine=1))
        or (punishWay='罚款' and fine>=50 and fine<=2000)
        or (punishWay='扣分' and fine>=2 and fine<=12)
        or (punishWay='暂扣驾驶执照' and fine>=3 and fine<=6)
        ),
```

5.2 安全性设计

本系统分为不同角色登录，分别授予不同的权限：

（1）关于驾驶员登录的视角

对于punishment表和notification表，只能进行查询，并且查询范围为与本人相关的记录；

对于driver表和car表，可以进行查询和修改，并且查询修改范围为本人相关的记录；

对于include表，可以进行查询和修改，并且查询范围为本人相关的记录，同时仅能修改include表中的punishState的值。

（2）关于警员登录的视角

对于punishment表和notification表，可以进行查询和更新，并且查询范围为所有记录；

对于driver表和car表，可以进行查询，并且查询范围为所有记录，但是无法查看驾驶员的隐私信息（如密码、身份证号等）；

对于include表，可以进行查询和更新，并且查询范围为所有记录；

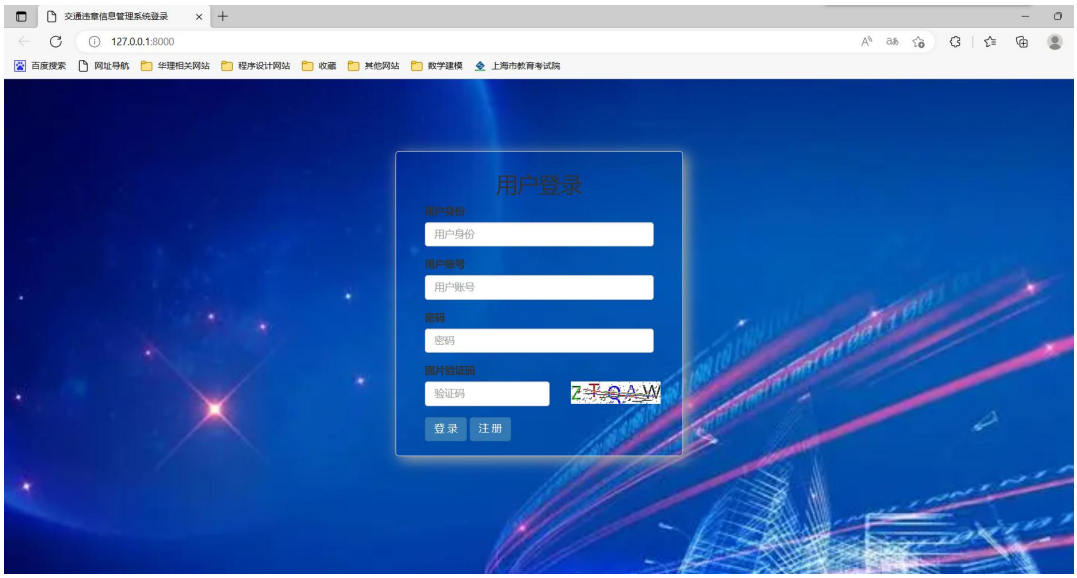
对于police表，可以进行查询和修改，并且查询范围为与本人相关的记录。

6 系统实现（界面+程序核心代码）

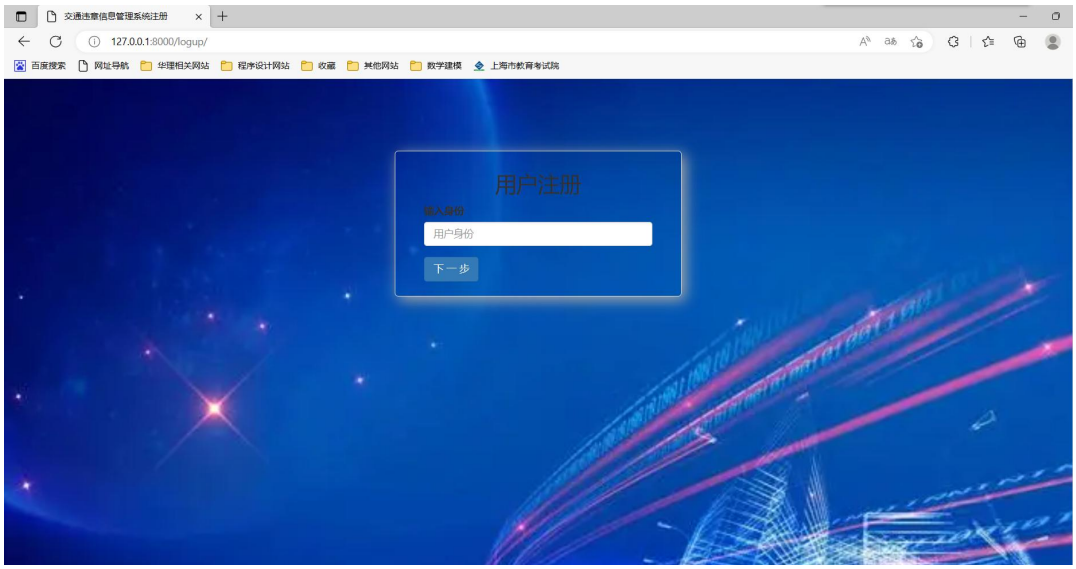
6.1 系统界面展示

（1）登录与注册

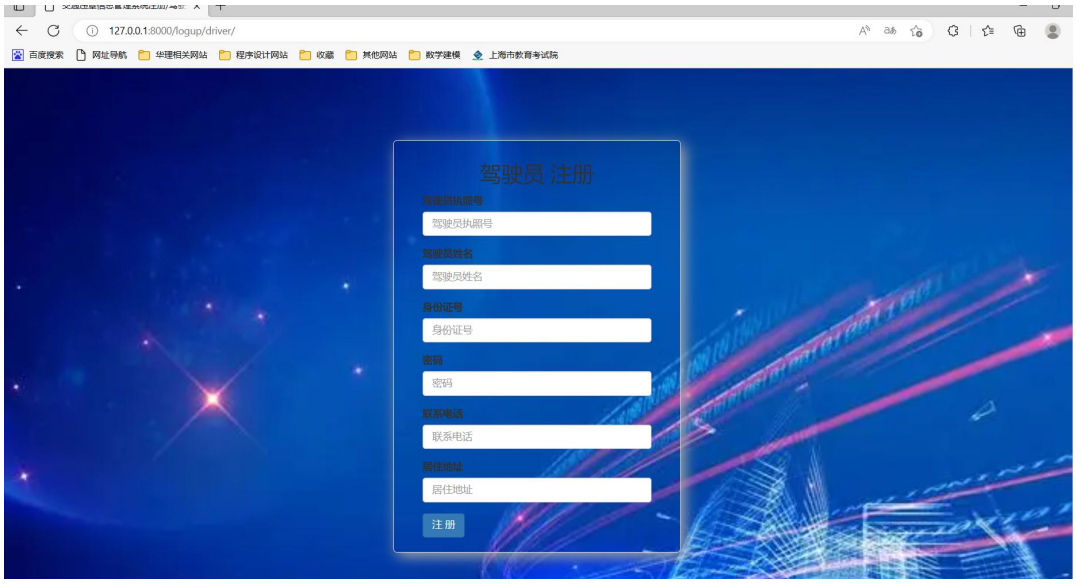
• 登录界面：


交通违章信息管理系统登录
127.0.0.1:8000
用户登录
用户身份
用户身份
用户账号
用户账号
密码
密码
图片验证码
验证码
登录
注册

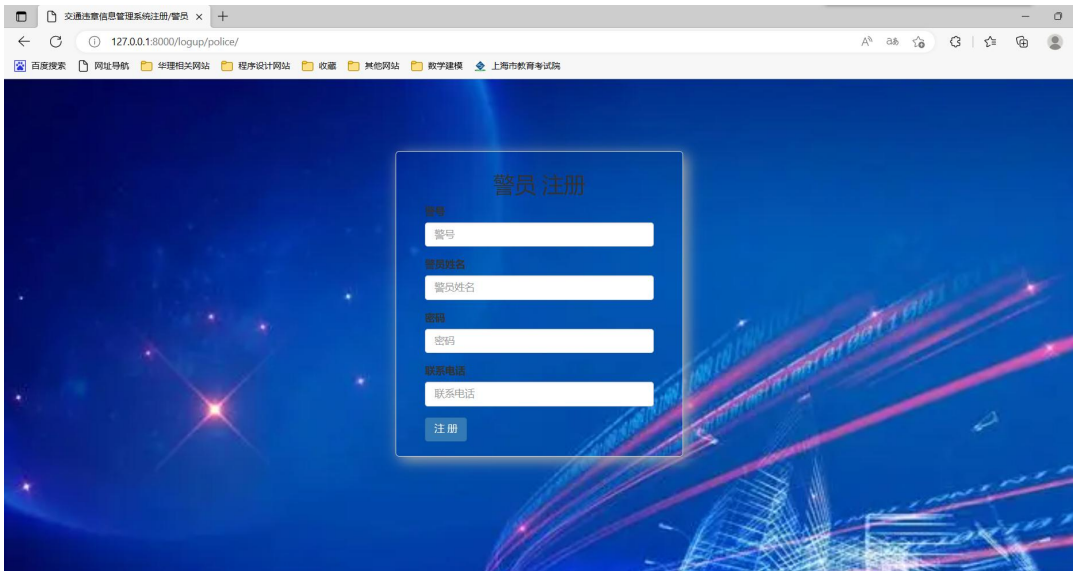
- 注册界面


交通违章信息管理系统注册
127.0.0.1:8000/logup/
用户注册
用户身份
用户身份
下一步

- 驾驶员注册

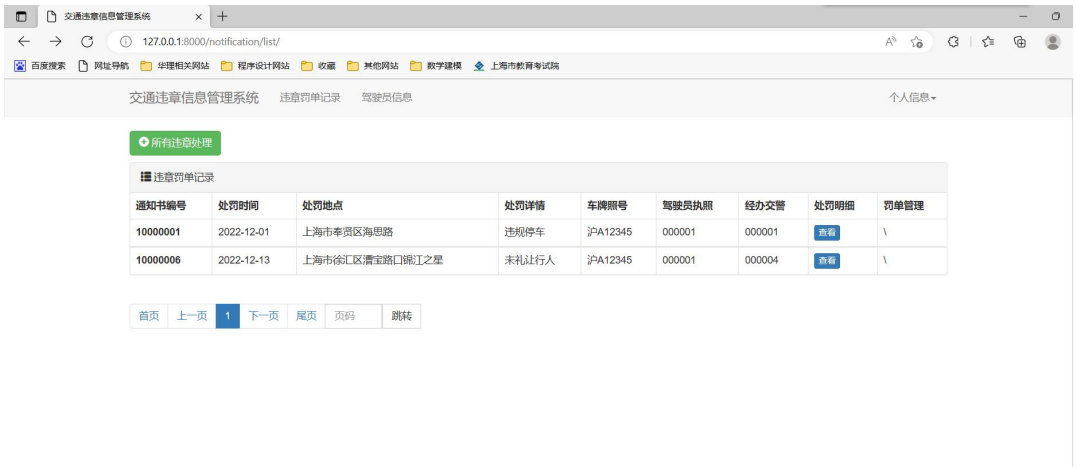

127.0.0.1:8000/logup/driver/
驾驶员 注册
驾驶员执照号
驾驶员执照号
驾驶员姓名
驾驶员姓名
身份证号
身份证号
密码
密码
联系电话
联系电话
居住地址
居住地址
注册

• 警员注册



（2）驾驶员视角

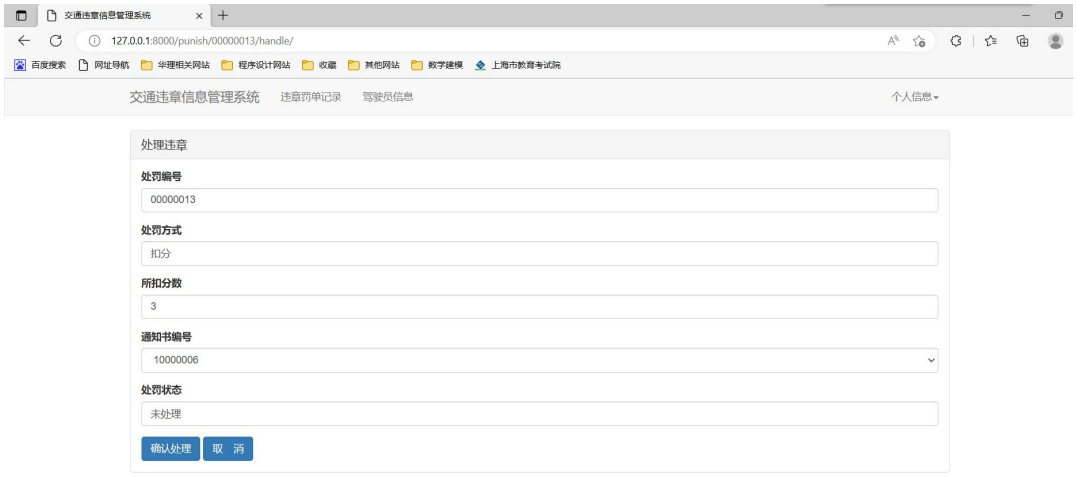
• 违章罚单记录界面（只能看到与个人相关的违章记录）



• 查看所有的违章处罚信息

• 查看单个的违章处罚详情

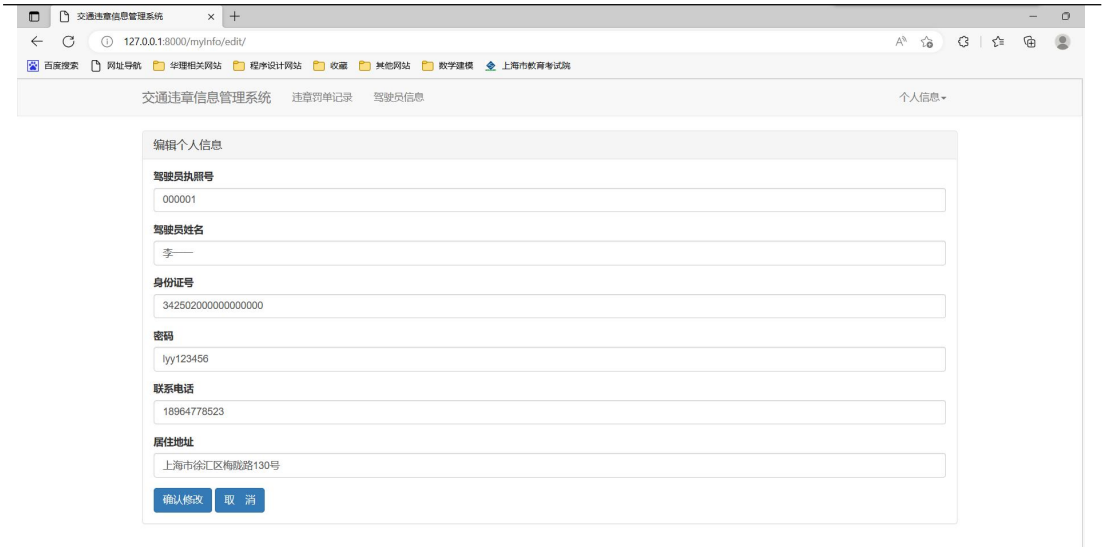
- 逐个点击处理违章



- 所有违章处理完成

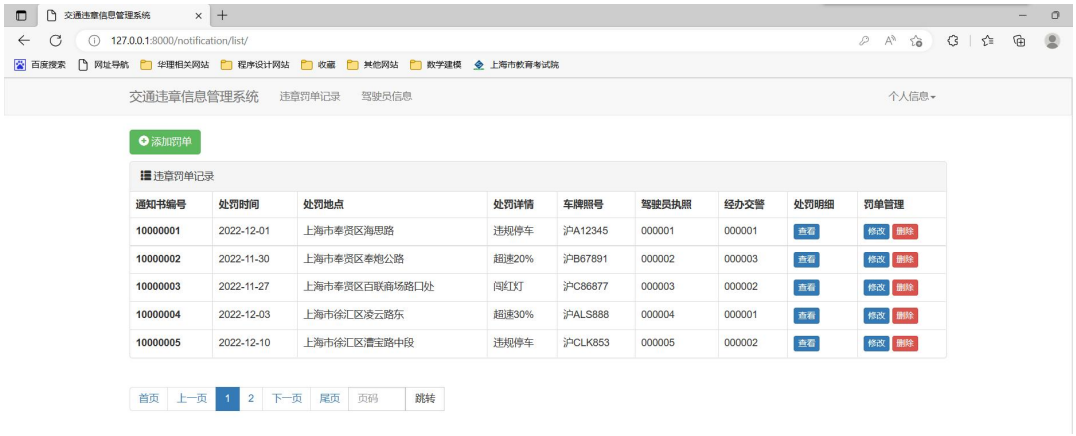
- 查看个人信息

- 修改个人信息



（3）警员视角

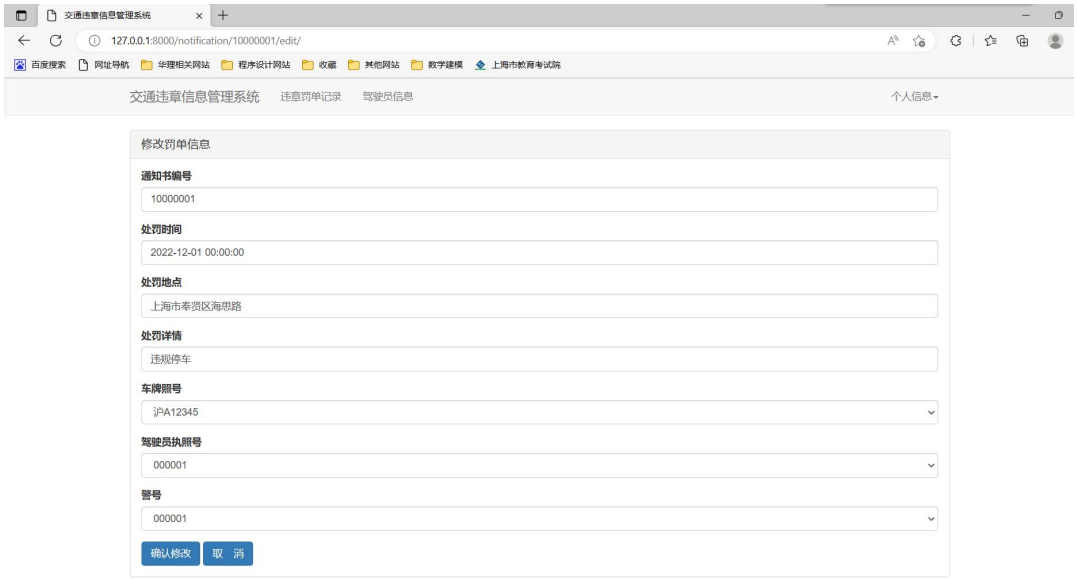
• 违章罚单记录界面（可以看到所有的违章记录）



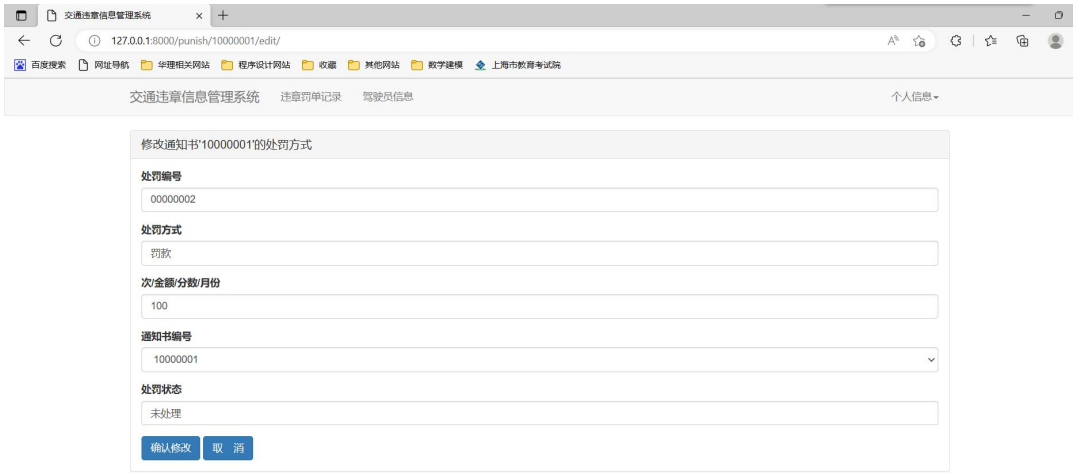
满 5 条记录则分页：

• 查看单个违章处罚详情

- 修改单个罚单详情



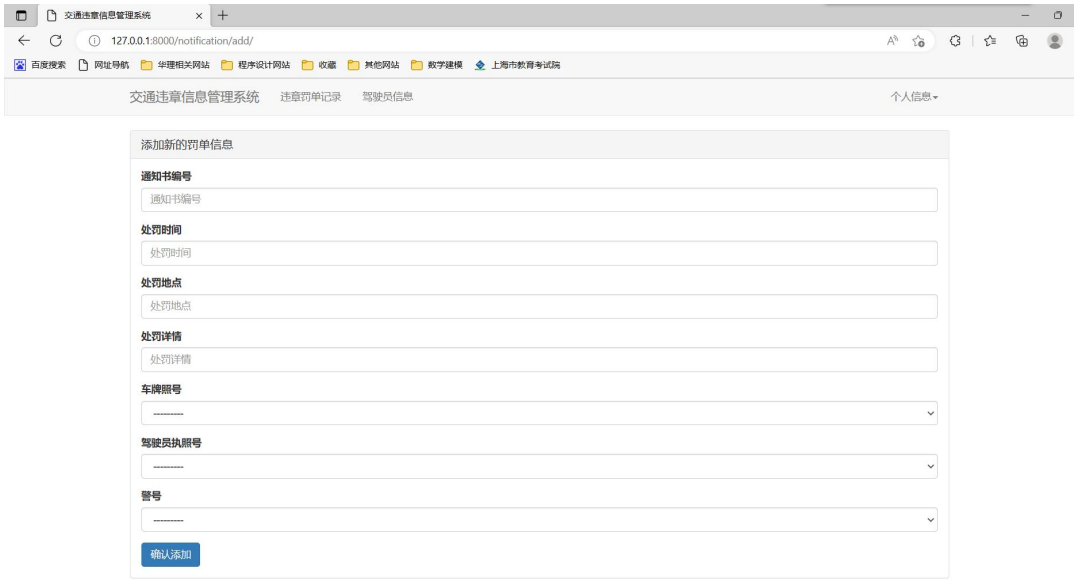
- 修改单个处罚方式



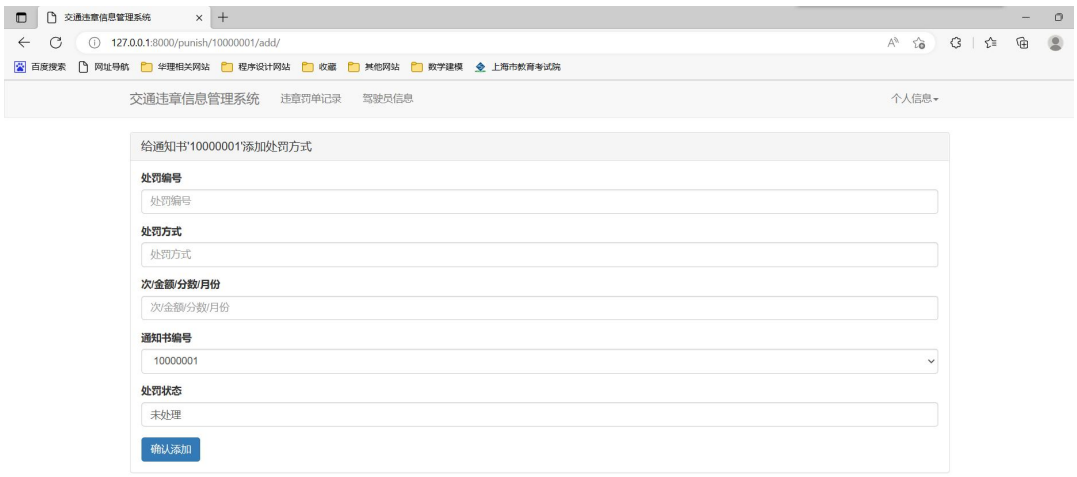
- 删除单个罚单信息

- 删除单个处罚信息

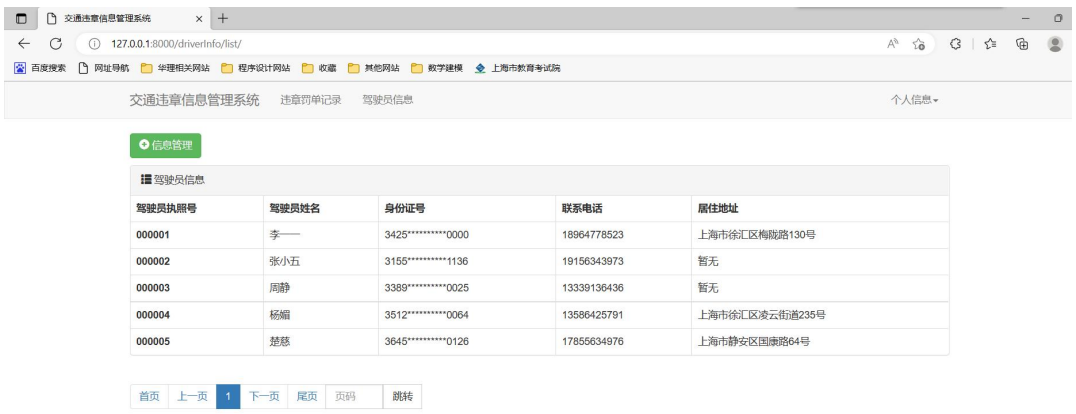
- 添加新的罚单



- 添加新的处罚

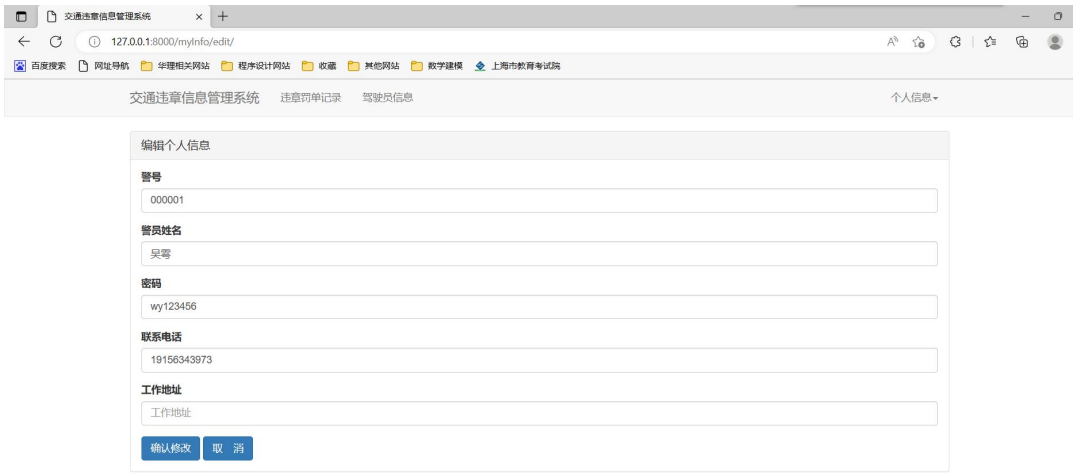


- 查看所有驾驶员信息



- 查看个人信息

• 修改个人信息



• 退出登录

6.2 程序核心代码

（1）路由设计

```
urlpatterns = [
    # 用户登录 & 注册 & 退出
    path('', user.login),
    path('login/', user.login),
    path('logup/', user.logup),
    path('logup/police/', user.logup_police),
    path('logup/driver/', user.logup_driver),
    path('logout/', user.logout),
    path('image/code/', user.image_code, name='image_code'),

    # 通知书管理
    path('notification/list/', notification.notification_list),
```

```
    path('notification/add/', notification.notification_add),
    path('notification/<int:nid>/edit/', notification.notification_edit),
    path('notification/<int:nid>/delete/', notification.notification_delete),

    # 违章处罚处理
    path('punish/list/', punish.punish_list),
    path('punish/<int:nid>/detail/', punish.punish_detail),
    path('punish/<nid>/handle/', punish.punish_handle),
    path('punish/<int:nid>/add/', punish.punish_add),
    path('punish/<int:nid>/edit/', punish.punish_edit),
    path('punish/<int:nid>/delete/', punish.punish_delete),

    # 驾驶员信息
    path('driverInfo/list/', driverInfo.driverInfo_list),

    # 个人信息
    path('myInfo/list/', myInfo.myInfo_list),
    path('myInfo/edit/', myInfo.myInfo_edit),

]
```

（2）数据库连接

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': 'traffic_db',
        'USER': 'root',
        'PASSWORD': '123456',
        'HOST': '127.0.0.1',
        'PORT': 3306,
    }
}
```

（3）重点函数设计及权限设置

• user：

```
def login(request):
    """用户  登录"""
    if request.method == "GET":
        form = LoginForm()
        return render(request, 'login.html', {'form': form})
    form = LoginForm(data=request.POST)
    if form.is_valid():
        # 请求成功，获取到的用户名、密码、图片验证码，格式如下
        # {'userid': 'wupeiqi', 'password': '5e5c3bad7eb35cba3638e145c830c35f',
```

```
                "code":xxx}
            # 校验验证码
            user_input_code = form.cleaned_data.pop('code')
            code = request.session.get('image_code', "")    # 从 session 中获取验证码进行校
验
            if code.upper() != user_input_code.upper():
                form.add_error("code", "验证码错误")
                return render(request, 'login.html', {'form': form})

            # 校验用户 id、密码
            user_input_pwd = form.cleaned_data.pop('password')
            global user_input_id
            user_input_id = form.cleaned_data.pop('userid')
            global user_input_class
            user_input_class = form.cleaned_data.pop('userclass')
            # driver_object = models.Driver.objects.filter(**form.cleaned_data).first()
            if user_input_class == '驾驶员':
                driver_object = models.Driver.objects.filter(licenseid=user_input_id,
                                    driverpwd=user_input_pwd).first()
                police_object = 0
            elif user_input_class == '警员':
                police_object = models.Police.objects.filter(policeid=user_input_id,
                                    policepwd=user_input_pwd).first()
                driver_object = 0
            else:
                form.add_error("userclass", "身份无效，请选择”驾驶员“或”警员“！ ")
                return render(request, 'login.html', {'form': form})
            if not (driver_object or police_object):
                form.add_error("password", "用户 id 或密码错误")
                return render(request, 'login.html', {'form': form})

            # 用户名和密码正确，网站生成随机字符串;，写到用户浏览器的 cookie 中，再
写入 session 中，保存 1 天；
            request.session["info"] = {'id': user_input_id, 'pwd': user_input_pwd}
            request.session.set_expiry(60 * 60 * 24 * 1)
            return redirect("/notification/list/")
        return render(request, 'login.html', {'form': form})

    def logout(request):
        """ 注销 用户 """
        request.session.clear()
        return redirect('/login/')

    def logup(request):
```

```
    """ 注册 用户 """
    if request.method == "GET":
        form = LogupForm()
        return render(request, 'logup.html', {"form": form})

    # 用户 POST 提交数据，数据校验
    form = LogupForm(data=request.POST)
    if form.is_valid():
        # 如果数据合法，则保存到数据库
        user_logup_class = form.cleaned_data.pop('userclass')
        if user_logup_class == '警员':
            return redirect('/logup/police/')
        elif user_logup_class == '驾驶员':
            return redirect('/logup/driver/')
        else:
            form.add_error("userclass", "身份无效，请选择”驾驶员“或”警员“！ ")
            return render(request, 'logup.html', {'form': form})


def logup_police(request):
    """ 注册 警员 """
    if request.method == "GET":
        form = PoliceModelForm()
        return render(request, 'logup_police.html', {"form": form})

    # 用户 POST 提交数据，数据校验
    form = PoliceModelForm(data=request.POST)
    if form.is_valid():
        # 如果数据合法，则保存到数据库
        form.save()
        return redirect('/login/', {'msg': '注册成功，请登录！ '})
    # 校验失败则在页面上显示错误信息
    return render(request, 'logup_police.html', {"form": form})


def logup_driver(request):
    """ 注册 驾驶员 """
    if request.method == "GET":
        form = DriverModelForm()
        return render(request, 'logup_driver.html', {"form": form})

    # 用户 POST 提交数据，数据校验
    form = DriverModelForm(data=request.POST)
    if form.is_valid():
```

```
            # 如果数据合法，则保存到数据库
            form.save()
            return redirect('/login/', {'msg': '注册成功，请登录！'})
        # 校验失败则在页面上显示错误信息
        return render(request, 'logup_driver.html', {"form": form})
```

• notification：

```
def notification_list(request):
    """交通违章通知书 展示"""
    from .user import user_input_class, user_input_id
    if user_input_class == '警员':
        queryset = models.Notification.objects.all()
    elif user_input_class == '驾驶员':
        queryset = models.Notification.objects.filter(licenseid=user_input_id)
    else:
        return HttpResponse('未提交用户信息，请重新登录')
    page_object = Pagination(request, queryset, page_size=5)
    context = {
        "queryset": page_object.page_queryset,  # 违章罚单信息
        "page_string": page_object.html(),
        "user_input_class": user_input_class,  # 用户身份
    }
    return render(request, 'notification_list.html', context)

def notification_add(request):
    """ 添加 新的违章罚单 """
    if request.method == "GET":
        form = NotificationModelForm()
        return render(request, 'notification_add.html', {"form": form})
    # 用户 POST 提交数据，数据校验
    form = NotificationModelForm(data=request.POST)
    if form.is_valid():
        # 如果数据合法，则保存到数据库
        form.save()
        return redirect('/notification/list/')
    # 校验失败则在页面上显示错误信息
    return render(request, 'notification_add.html', {"form": form})

def notification_edit(request, nid):
    """ 编辑 违章罚单 """
    row_object = models.Notification.objects.filter(notificationid=nid).first()
    if request.method == "GET":
        # 根据 notificationid 去数据库获取要编辑的那一行数据（对象）
        form = NotificationModelForm(instance=row_object)
```

```
            return render(request, 'notification_edit.html', {'form': form})

        form = NotificationModelForm(data=request.POST, instance=row_object)
        if form.is_valid():
            # 默认保存的是用户输入的所有数据，如果想要再用户输入以外增加一点值: form.instance.字段名 = 值
            form.save()
            return redirect('/notification/list/')
        return render(request, 'notification_edit.html', {"form": form})

def notification_delete(request, nid):
    """ 删除 违章罚单 """
    models.Notification.objects.filter(notificationid=nid).delete()
    return redirect('/notification/list/')
```

• punish：

```
def punish_detail(request, nid):
    """具体某个罚单的 处罚方式列表"""
    from .user import user_input_class
    punishids = []
    include_all = models.Include.objects.filter(notificationid=nid)
    for obj in include_all:
        punishids.append(obj.punishid)
    punish_all = models.Punishment.objects.filter(punishid__in=punishids)

    return render(request, 'punish_detail.html', {"punish_all": punish_all, "include_all": include_all, "nid": nid, "user_input_class": user_input_class})

def punish_list(request):
    """查看 与该驾驶员/警员相关的 所有处罚列表 """
    from .user import user_input_class, user_input_id
    includes = []
    punishids = []
    notificationids = models.Notification.objects.filter(licenseid=user_input_id)
    for obj in notificationids:
        includes.append(obj.notificationid)
    include_all = models.Include.objects.filter(notificationid__in=includes)
    for obj in include_all:
        punishids.append(obj.punishid)
    punish_all = models.Punishment.objects.filter(punishid__in=punishids)

    return render(request, 'punish_list.html', {"punish_all": punish_all, "include_all": include_all, "user_input_class": user_input_class})
```

```
def punish_handle(request, nid):
    """驾驶员处理违章处罚 """
    row_object1 = models.Punishment.objects.filter(punishid=nid).first()
    row_object2 = models.Include.objects.filter(punishid=nid).first()

    if request.method == "GET":
        # 根据 punishid 去数据库获取要编辑的那一行数据（对象）
        form1 = PunishmentModelForm(instance=row_object1)
        form2 = IncludeModelForm(instance=row_object2)
        name = ''
        for field in form1:
            if field.label == '处罚方式':
                name = str(field)
            if field.label == '次/金额/分数/月份':
                if name == '<input type="text" name="punishway" value="警告" maxlength="20" class="form-control" placeholder="处罚方式" required id="id_punishway">':
                    field.label = '警告次数'
                elif name == '<input type="text" name="punishway" value="罚款" maxlength="20" class="form-control" placeholder="处罚方式" required id="id_punishway">':
                    field.label = '金额/元'
                elif name == '<input type="text" name="punishway" value="扣分" maxlength="20" class="form-control" placeholder="处罚方式" required id="id_punishway">':
                    field.label = '所扣分数'
                else:
                    field.label = '暂扣月份'

        return render(request, 'punish_handle.html', {'form1': form1, 'form2': form2, "nid": nid})

    form1 = PunishmentModelForm(data=request.POST, instance=row_object1)
    form2 = IncludeModelForm(data=request.POST, instance=row_object2)
    if form1.is_valid() and form2.is_valid():
        # 默认保存的是用户输入的所有数据，如果想要再用户输入以外增加一点值: form.instance.字段名 = 值
        form1.save()
        form2.save()
        return redirect('/punish/list/')
    return render(request, 'punish_handle.html', {"form1": form1, "form2": form2, "nid": nid})
```

```
def punish_add(request, nid):
    """ 对某个违章通知书  添加 新的处罚方式 """
    row_object = models.Include.objects.filter(notificationid=nid).first()
    if request.method == "GET":
        form1 = PunishmentModelForm()
        form2 = IncludeModelForm(instance=row_object)
        return render(request, 'punish_add.html', {"form1": form1, "form2": form2, "nid": nid})

    # 用户 POST 提交数据，数据校验
    form1 = PunishmentModelForm(data=request.POST)
    form2 = IncludeModelForm(data=request.POST, instance=row_object)
    if form1.is_valid():
        # 默认保存的是用户输入的所有数据，如果想要再用户输入以外增加一点值: form.instance.字段名 = 值
        form1.save()
        # form2.save()
        return redirect('/notification/list/')
    # 校验失败则在页面上显示错误信息
    return render(request, 'punish_detail.html', {"form1": form1, "form2": form2, "nid": nid})


def punish_edit(request, nid):
    """ 对某个违章通知书  编辑 处罚方式 """
    row_object = models.Include.objects.filter(notificationid=nid)
    row_object1 = models.Include.objects.filter(notificationid=nid).first()
    row_object2 = ''
    for obj in row_object:
        row_object2 = obj.punishid

    if request.method == "GET":
        # 根据 notificationid 去数据库获取要编辑的那一行数据（对象）
        form1 = PunishmentModelForm(instance=row_object2)
        form2 = IncludeModelForm(instance=row_object1)
        return render(request, 'punish_edit.html', {"form1": form1, "form2": form2, "nid": nid})

    form1 = PunishmentModelForm(data=request.POST, instance=row_object2)
    form2 = IncludeModelForm(data=request.POST, instance=row_object1)
    if form1.is_valid():
        # 默认保存的是用户输入的所有数据，如果想要再用户输入以外增加一点值: form.instance.字段名 = 值
        form1.save()
        # form2.save()
```

```
            return redirect('/notification/list/')
        # 校验失败则在页面上显示错误信息
        return render(request, 'punish_edit.html', {"form1": form1, "form2": form2, "nid": nid})

def punish_delete(request, nid):
    """ 对某个违章通知书  删除  该处罚方式 """
    models.Include.objects.filter(notificationid=nid).delete()
    return redirect('/notification/list/')
```

- myInfo:

```
def myInfo_list(request):
    """个人信息 展示"""
    from .user import user_input_class, user_input_id
    if user_input_class == '警员':
        queryset = models.Police.objects.filter(policeid=user_input_id)
        # 加密输出密码 ********
        for obj in queryset:
            pwd = obj.policepwd
            num = (len(pwd)) * '*'   # 用*代替
            obj.policepwd = num
    elif user_input_class == '驾驶员':
        queryset = models.Driver.objects.filter(licenseid=user_input_id)
        # 加密输出密码 ********
        for obj in queryset:
            pwd = obj.driverpwd
            num = (len(pwd)) * '*'   # 用*代替
            obj.driverpwd = num
    else:
        return HttpResponse('未提交用户信息，请重新登录')
    page_object = Pagination(request, queryset, page_size=5)
    context = {
        "queryset": page_object.page_queryset,   # 个人信息
        "page_string": page_object.html(),
        "user_input_class": user_input_class,   # 用户身份
    }
    return render(request, 'myInfo.html', context)

def myInfo_edit(request):
    """ 编辑 个人信息 """
    from .user import user_input_class, user_input_id
    if user_input_class == '警员':
        row_object = models.Police.objects.filter(policeid=user_input_id).first()
        if request.method == "GET":
            # 根据 user_input_id 去数据库获取要编辑的那一行数据（对象）
```

```
                form = PoliceModelForm(instance=row_object)
                return render(request, 'myInfo_edit.html', {'form': form})
            form = PoliceModelForm(data=request.POST, instance=row_object)
        elif user_input_class == '驾驶员':
            row_object = models.Driver.objects.filter(licenseid=user_input_id).first()
            if request.method == "GET":
                # 根据 user_input_id 去数据库获取要编辑的那一行数据（对象）
                form = DriverModelForm(instance=row_object)
                return render(request, 'myInfo_edit.html', {'form': form})
            form = DriverModelForm(data=request.POST, instance=row_object)
        else:
            return HttpResponse('未提交用户信息，请重新登录')
        if form.is_valid():
            # 默认保存的是用户输入的所有数据，如果想要再用户输入以外增加一点值:
form.instance.字段名 = 值
            form.save()
            return redirect('/myInfo/list/')
        return render(request, 'myInfo_edit.html', {"form": form})
```

• driverInfo：

```
def driverInfo_list(request):
    """驾驶员信息列表 展示"""
    from .user import user_input_class, user_input_id
    if user_input_class == '警员':
        queryset = models.Driver.objects.all()
    elif user_input_class == '驾驶员':
        queryset = models.Driver.objects.filter(licenseid=user_input_id)
    else:
        return HttpResponse('未提交用户信息，请重新登录')
    # 加密输出身份证号码 3425**********0024
    for obj in queryset:
        number = obj.driveridentify
        num1 = number[0:4]  # 切出前四位
        num2 = (len(number) - 8) * '*'  # 中间用*代替
        num3 = number[-4:]  # 切出后四位
        obj.driveridentify = num1 + num2 + num3

    page_object = Pagination(request, queryset, page_size=5)
    context = {
        "queryset": page_object.page_queryset,
        "page_string": page_object.html(),
        "user_input_class": user_input_class,
    }
    return render(request, 'driverInfo_list.html', context)
```

7 结论

在本次实验中，我们运用Django框架和mysql数据库构建了一个交通违章信息管理系统。该系统共创建了6个数据表，分别为表Driver、Car、Police、Notification、Punishment、Include.在Django框架中对这六个表进行调用，分别以驾驶员与警员的视角设计了交通违章信息的查询与更新的网页界面，实现了对数据库的增删改查等功能。

登录/注册界面：

用户通过输入用户身份、用户账号、密码、图片验证码等信息可进行登录，其中用户身份部分，分别有“警员”与“驾驶员”两种身份可选择，输入对应的账号与密码后，点击登录即可进入相应身份的网页界面；注册时用户可根据身份选择进入不同的注册通道，填写相关信息。

警员的网页端：

违章罚单记录界面中，警员可以查询到所有的违章罚单记录，并可查看处罚明细、对罚单、处罚详情进行增加、修改、删除等操作；驾驶员信息界面中，警员可查看到所有驾驶员的信息；个人信息中，警员可对自己的个人信息进行修改的操作。

驾驶员的网页端：

违章罚单记录界面中，驾驶员可以查询到本人相关的违章信息，并可对处罚明细进行查看，同时也可以进行违章处理；个人信息界面，驾驶员可以查看到本人的相关信息，并可对其进行修改。

本系统面向全体驾驶员以及公安部门相关工作人员，全方面涵盖了警员、驾驶员、违章车辆信息的查询与交通违章罚单的开具、经办、发布、处理等功能，助力警民数字化信息化业务模式的构建，为各方均带来了极大的便利。

三、出现的问题和解决方案

问题 1：如何实现驾驶员和警员两种视角

解决方案：首先在数据库中单独建立两张表（driver、police）来分别存放驾驶员和警员的基本信息；在登录设计时，在后端运用函数（本质是存储过程）进行身份判断，例如：如果登陆者为驾驶员，则后续界面的呈现均限制在个人相关信息之下（比如只能查询个人相关罚单及处罚处理情况），并且控制用户只能对特定的信息进行增删改查；如果为警员登录，则权限会比驾驶员更高。

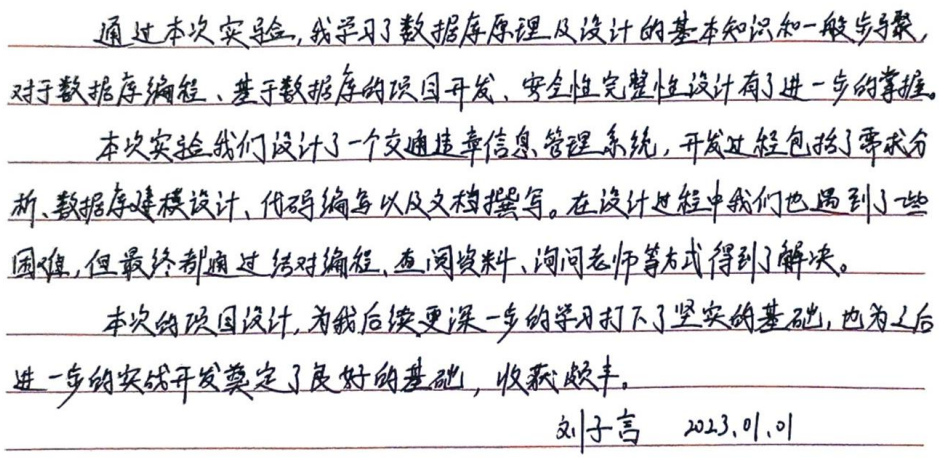
问题 2：如何实现罚单与处罚方式的对应关系

解决方案：首先建立交通违章通知书与处罚方式之间多对多的关系，即在数据库中建立三张表（notification、punishment、include）来分别储存通知书、处罚方式以及包含关系（处理状态）等相关信息；在系统具体设计时，点击查看某一交通违章通知书的详细信息，则可以看到多条处罚方式记录，并可逐个实现增删改查。

问题 3：如何实现前后端及数据库之间数据的传输、修改和存储

解决方案：我们在开发系统时采用了 Django 框架+mysql 数据库：配置好数据库相关信息后，在数据调用时利用了 Django 框架中的 Form/ModelForm 组件，自定义类，辅助数据库、前端、后端之间进行数据的快捷传输；同时运用了 Django 内部自带的 ORM 框架，简化数据库编程语言，实现对数据的增删改查，使对表中数据的操作更加便捷。

四、收获和体会(手写)



通过本次实验，我学习了数据库原理及设计的相关知识与技术，对于运用数据库相关知识进行需求分析设计、数据库概念设计、数据库逻辑设计、数据库编程，最终实现及一个数据库系统的构建有了进一步的掌握。

本次实验中，我们设计并构建了一个交通违章信息管理系统，在开发过程中，我们也遇到了一些问题，如如何处理罚单与处罚方式间的对应关系等，但最终通过询问老师、查阅资料等一系列方法解决了问题。

本次实验中，我对数据库设计有了进一步的了解，受益匪浅。

林芷珊 2023.01.01