目录

1 系统概述与需求分析

1.1 功能需求

1.1.1 总体需求分析

（1）图书馆中的图书具有书号、书名、作者、馆藏册数、在馆册数、价格、出版社及摘要等必要信息。其中每种书具有唯一的书号，即一个书号对应一种书而不是一本书。书名可以重复，但如果只是两本书重名，则这两本书具有不同的书号。馆藏册数和在馆册数可以用于让读者判断是否可以借，另外，馆藏册数必须大于等于在馆册数。

（2）图书馆中的读者具有卡号、姓名、性别、单位、类型、级别和在借册数这些必要信息。卡号用于唯一地标识读者身份，每个读者对应一个卡号，同样每个卡号也不能对应多个读者。类型这个属性可以用于区分教师、学员，区分他们的意义在于不同类型的读者可同时借阅的书籍数大不相同。

（3）从读者的角度，可以查阅图书馆相应的图书资料（也可以按要求查找，如输入一些关键字）以及每本书的在馆册数；同时可以提交借书和还书的申请；能判断自己是否因超期而欠款，若超期则自动显示出所需的罚款金额。

（4）图书馆中的工作人员可以分为图书管理员和系统管理员。图书管理员是图书馆中最普通的工作人员，图书管理员可以对读者的借书和还书操作进行批准或者驳回，同时能够查询读者信息，并且为图书馆中增加、删除、修改书籍。系统管理员主要管理各类人员信息，包括读者信息和管理员信息这两项，同时系统管理员能够查看读者登录记录。

（5）图书馆中应当有书架，书架具有书架号和类型两个属性，不同的书架号对应的图书类型也应该不同；另外还需要在图书中增加存放位置这个属性。

1.1.2 登录功能

用户输入正确的用户名及密码，并且选择对应的身份，方可进入系统，正确登录之后才可进行进一步的管理，对于输入错误用户名或密码的用户将弹出提示。

1.1.3 读者管理功能

（1）读者自助管理：在读者界面有“用户信息修改”页，支持姓名、性别、密码修改（对于性别修改有字段提醒），实时显示读者 ID、单位、可借册数等信息；还有“逾期罚款”页显示欠款金额并支持在线支付，支付后自动清零，对于没有欠款的，点击付款后也有对应的提示。

（2）管理员集中管理：在图书管理员界面有“查询读者信息”页支持按 ID、姓名、单位多条件查询，表格展示读者全量信息；系统管理员界面有“读者信息录入”页可创建新读者账户，根据读者类型（本科生/研究生/教师）自动设置可借册数（10/20/30 本），并支持批量修改与删除操作，在修改与插入时对于部分选项有下拉菜单防止信息违法，在删除操作时也会有对应的提示操作。

1.1.4 图书管理功能

（1）图书信息维护：在图书管理员界面有“管理书籍信息”页，支持书号、书名、作者等 11 项属性的增删改查，删除图书前自动检查外键依赖，避免删除在借图书；系统管理员无需直接管理图书，通过基础数据维护间接支持图书分类（如书架号与书籍类型关联）。

（2）自助查询功能：读者界面有“查询图书信息”页，支持按书名、书号、作者等条件组合查询，对

于书名做了模糊搜索处理，表格展示馆藏册数、在馆册数等流通信息，便于读者定位可借图书。

1.1.5 借阅管理功能

（1）借阅流程自动化：读者通过“借书”页查询图书后提交申请；图书管理员在“借阅/归还审批”页批准申请，系统自动删除记录：借书时更新借阅记录、减少读者可借册数与图书在馆册数。

（2）还书自助服务：读者通过“还书”页可以查询已借图书并提交还书申请，管理员经过审批后，同步更新借阅状态，还书时计算逾期天数并累加欠款，无需人工干预库存与读者借阅记录。

1.1.6 系统维护功能

系统管理员通过“图书管理员信息录入”页创建与管理管理员账户，区分“图书管理员”与“系统管理员”权限；通过“登录记录查看”页支持按读者 ID 查询历史登录日志，便于审计系统访问记录，保障数据安全。

1.2 安全性与完整性要求

系统中，对各种信息进行管理和修改时应规范输入。

（1）安全性要求：系统中用户登录时应要求用户输入密码，用户不可注册，只有系统管理员才可增添系统用户，以保证系统的安全性。不同级别的用户有相应的权限范围，例如读者只能查看与修改自己的个人信息，图书管理员可以查看所有读者信息而不能修改，系统管理员可以查看与修改所有读者的个人信息。

（2）数据完整性约束：各类信息需要具备的完整性，例如读者的姓名、ID，书的书名、ID，借阅记录的读者 ID、书 ID、借阅时间等不能为空。各个数据间确保外键依赖关系，如图书管理员删除图书时，需要先检查外键依赖；系统管理员删除读者时，读者信息录入验证是否存在借阅记录，避免脏数据产生。

1.3 数据流图

1.3.1 总体功能图

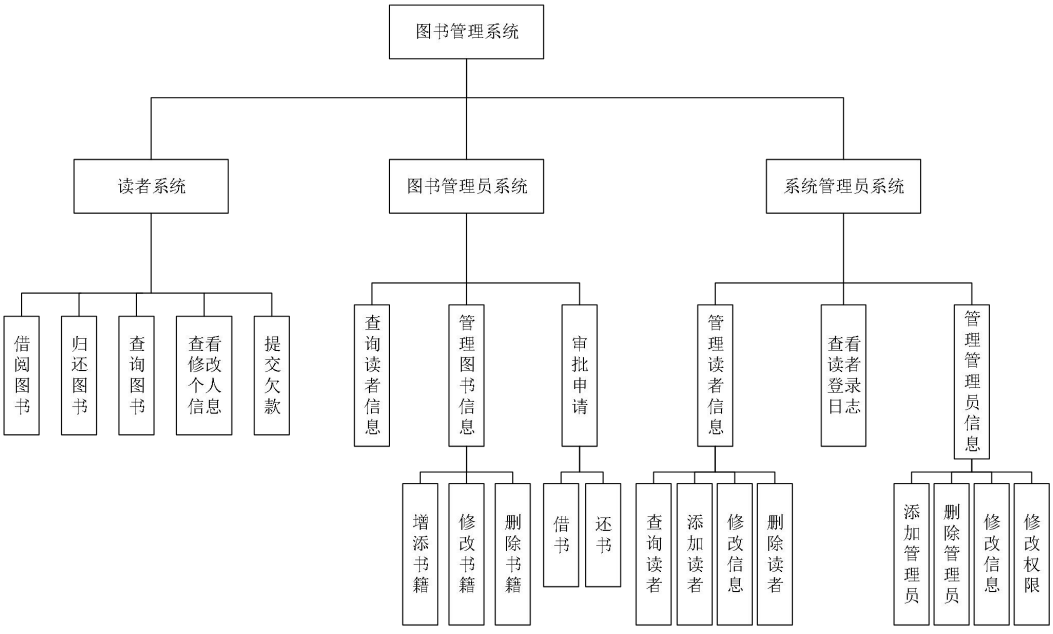


图 1.1 系统总功能图

1.3.2 总数据流图

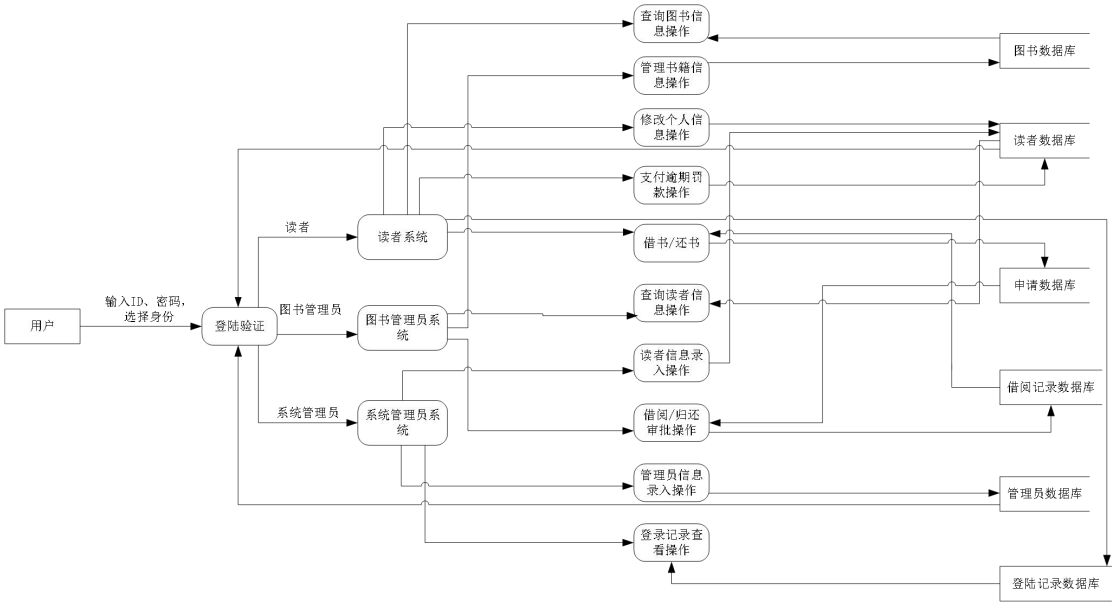


图 1.2 系统数据流图

1.4 数据字典

1.4.1 数据项

（1） 书架信息

表 1.1 书架信息中的数据项

数据项	数据类型	长度	取值范围	含义说明
书架号	varchar	10	Not Null	每个书架的唯一标识
书籍类型	varchar	45	Not Null	书架内书籍类型

（2） 书籍信息

表 1.2 书籍信息中的数据项

数据项	数据类型	长度	取值范围	含义说明
书号	varchar	15	Not Null	每本书的唯一标识
书名	varchar	30	Not Null	书的名字（可重名）
作者	varchar	20		书的作者
类型	varchar	15		书的类型
价格	float			书的价格（浮点数）
出版社	varchar	45		书的出版社
摘要	varchar	50		内容摘要
馆藏册数	int			馆内书的总量
在馆册数	int			馆内剩余数量
书架号	varchar	10	Not Null	书所在书架
被借次数	int			书的外借次数

（3） 读者类型信息

表 1.3 读者类型信息中的数据项

数据项	数据类型	长度	取值范围	含义说明
读者类型	varchar	10	本科生、研究生、教师	读者身份
借书时间	int			一本书最多可借时间
最多在借册数	int			最多可以同时借的数量

（4） 读者信息

表 1.4 读者信息中的数据项

数据项	数据类型	长度	取值范围	含义说明
ID	varchar	15	Not Null	读者唯一标识
姓名	varchar	20	Not Null	读者姓名（可重名）
性别	enum		男、女	读者性别
单位	varchar	45	所有学院名	读者所在学院
读者类型	varchar	45	本科生、研究生、教师	读者身份
可借册数	int			读者当前剩余可借
在借册数	int			读者已借册数
密码	varchar	20	Not Null	账号密码
欠款	float		Not Null	读者逾期欠款

（5） 借还申请信息

表 1.5 借还申请信息中的数据项

数据项	数据类型	长度	取值范围	含义说明
书号	varchar	15	Not Null	申请的书号
ID	varchar	15	Not Null	读者 ID
时间	date		Not Null	申请时间
类型	enum		借、还	借书或还书

（6） 借阅记录信息

表 1.6 借阅记录信息中的数据项

数据项	数据类型	长度	取值范围	含义说明
书号	varchar	15	Not Null	借阅的书号
ID	varchar	15	Not Null	读者 ID
时间	date		Not Null	借书时间

（7） 登录记录信息

表 1.7 登录记录信息中的数据项

数据项	数据类型	长度	取值范围	含义说明
ID	varchar	15	Not Null	登录的读者 ID

续表 1.7 登录记录信息中的数据项

时间	date	Not Null	登录时间
次数	int	Not Null	当天的登录次数

（8）管理员信息

表 1.8 管理员信息中的数据项

数据项	数据类型	长度	取值范围	含义说明
ID	varchar	15	Not Null	唯一辨识 ID
姓名	varchar	20	Not Null	管理员姓名
类型	enum		系统管理员、图书管理员	管理员类型
密码	varchar	20	Not Null	账户密码

1.4.2 数据结构

表 1.9 数据结构表

数据结构名	组成	含义说明
书架信息	书架号、书籍类型	描述书架的基本信息
书籍信息	书号、书名、作者、类型、价格、出版社、 摘要、馆藏册数、在馆册数、书架号、被借次数	描述每本书的基本信息， 以书号作为唯一识别标志
读者类型信息	读者类型、借书时间、最多在借册数	读者身份信息，共分为本科生、研究生、 教师，对每个身份有不同的借阅权限
读者信息	ID、姓名、性别、单位、读者类型、 可借册数、在借册数、密码、欠款	读者的基本信息， 以 ID 号作为唯一识别码
借还申请信息	书号、ID、时间、类型	申请记录的表，包含 基本信息与申请类型
借阅记录	书号、ID、时间	借阅记录的表，保存借阅的书、 读者与借阅时间
登录记录	ID、时间、次数	描述登录记录的表
管理员信息	ID、姓名、类型、密码	管理员基本信息，包含权限类别， 以 ID 号作为唯一识别码

2 开发运行环境

开发环境：PyCharm 2020.1

数据库：MySQL

数据库管理工具：Navicat for MySQL

开发语言：Python 3.9

图形化界面：PyQT5

运行环境：Windows 11 系统

3 概念结构设计（E - R 图）

3.1 E - R 图概述

本数据库的实体主要有 “书架”、“书籍”、“管理员”、“读者”、“读者类型”、“登录记录”、“借还申请”。

实体关系：

书架与书籍：存在“属于”关系，一个书架可放多本书籍，关系为一对多。

管理员与书籍：存在“管理”关系，一名管理员可管理多本书籍，关系为一对多。

管理员与登录记录：存在“查看”关系，一名管理员可以查看多条登录记录，关系为一对多。

管理员与读者：存在“管理”关系，一名管理员管理多个读者，关系为一对多。

读者与书籍：存在“借阅”关系，一名读者借多本书，一本书被多名读者借，关系为多对多。

读者与读者类型：存在“身份”关系，一种读者类型对应多名读者，关系为一对多。

读者与借还申请：存在“提交”关系，一名读者可以提交多次申请，关系为一对多。

管理员与借还申请：存在“审批”关系，一名管理员可以审批多个申请，关系为一对多。

3.2 分 E - R 图

图 3.1 书架 E - R 图

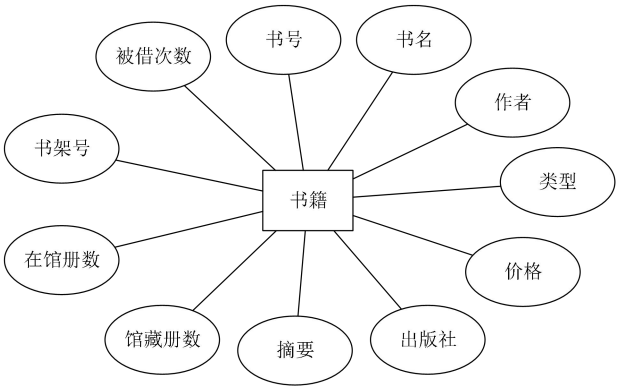


图 3.2 书籍 E - R 图

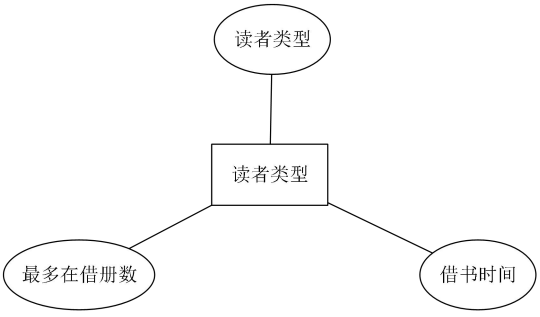


图 3.3 读者类型 E - R 图

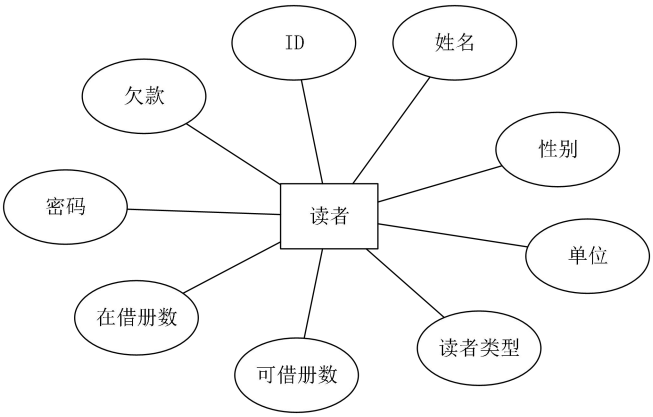


图 3.4 读者 E - R 图

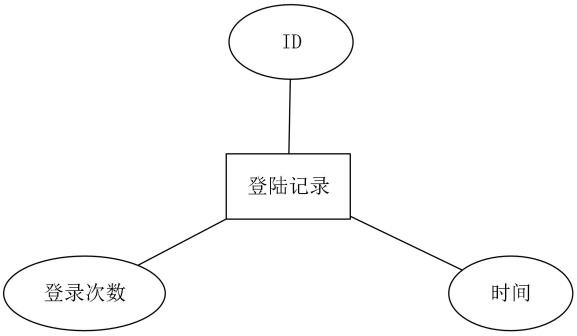


图 3.5 登录记录 E - R 图

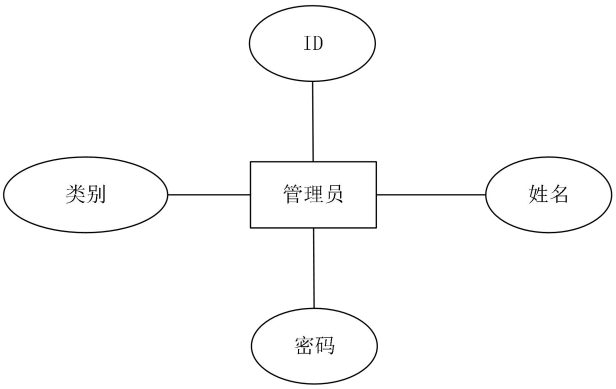


图 3.6 管理员 E - R 图

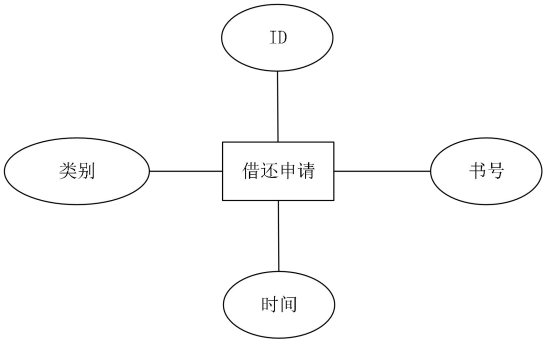


图 3.7 借还申请 E - R 图

3.3 总 E - R 图

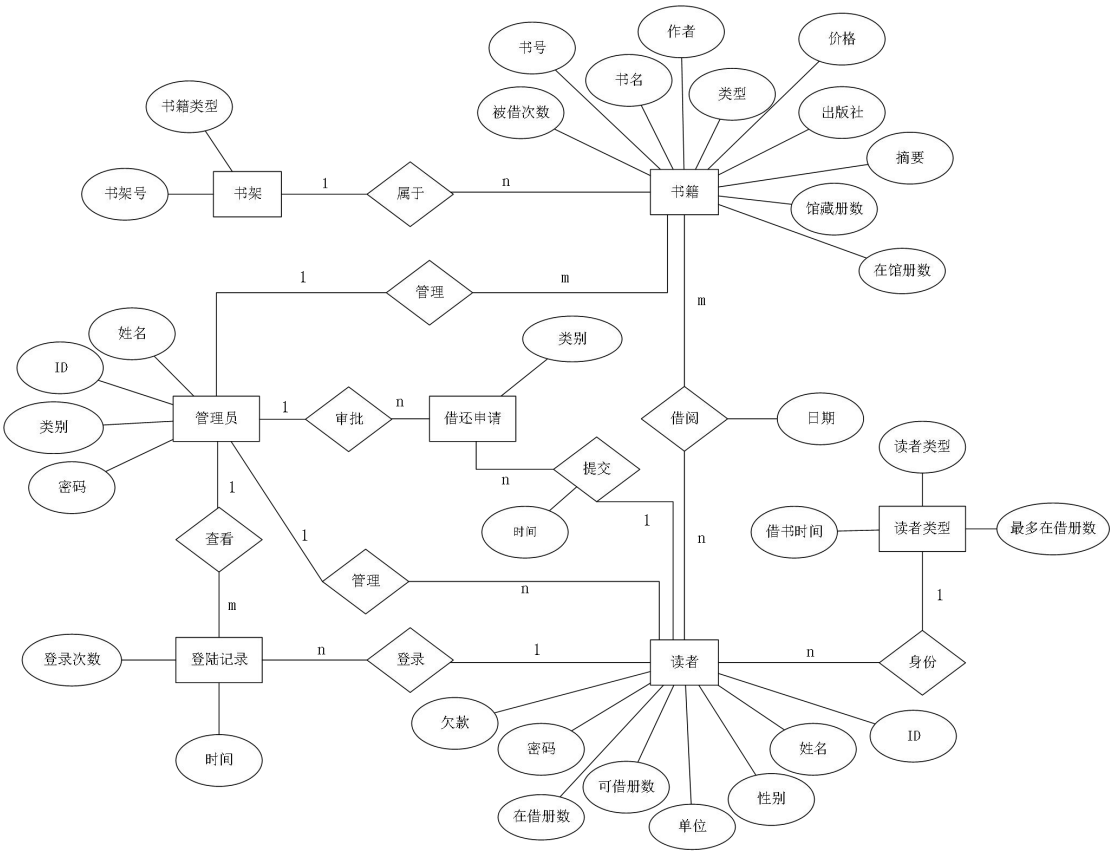


图 3.8 总 E - R 图

4 逻辑结构设计

根据 E - R 图设计对应的关系模式，将所有的实体转化成一个关系模式，将所有多对多的关系转化为关系模式，由此得到如下 8 个关系模式：

4.1 书架关系模式

书架的 E-R 图中，实体为书架，其中书架号用于确保唯一性，因此作为书架的主码。

关系模式：bookshelfs (书架号, 书籍类型)

由于只有一个候选码（书架号），关系模式中不存在非主属性部分依赖的情况；由于除了候选码以外的所有非主属性只有书籍类型，不存在传递依赖，且候选码是唯一的决定因素，该关系模式满足 BCNF。

由此创建对应的 bookshelfs 表：

图 4.1 bookshelfs 表的创建

对应的 SQL 语句：

```
-- 书架信息
CREATE TABLE bookshelfs  (
  书架号 varchar(10) NOT NULL,
  书籍类型 varchar(45) NOT NULL,
  PRIMARY KEY (书架号) USING BTREE
);

INSERT INTO bookshelfs VALUES
('C01', '计算机技术'),('C02', '自动化'),('C03', '文学'),('A01', '人工智能'),
('B01', '航空航天'), ('C04', '材料科学'),('A02', '自动化控制'),('B02', '经济管理'),
('C05', '外语学习'), ('A03', '文学艺术'),('B03', '马克思主义理论'),('B04', '物理科学'),
('A04', '人文社科');
```

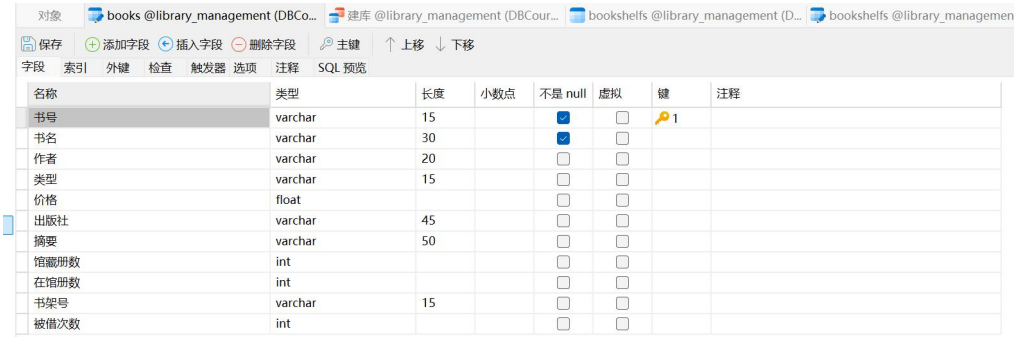
4.2 书籍关系模式

书籍的 E-R 图中，实体为书籍，其中书号用于确保唯一性，因此作为书籍的主码。

关系模式：books (书号, 书名, 作者, 类型, 价格, 出版社, 摘要, 馆藏册数, 在馆册数, 书架号, 被借次数)

同理可知，不存在非主属性部分依赖，不存在传递依赖，且候选码书号是唯一的决定因素，满足 BCNF。

由此创建对应的 book 表：



名称	类型	长度	小数点	不是 null	虚拟	键	注释
书号	varchar	15		☑	☐	🔑 1	
书名	varchar	30		☑	☐		
作者	varchar	20		☐	☐		
类型	varchar	15		☐	☐		
价格	float			☐	☐		
出版社	varchar	45		☐	☐		
摘要	varchar	50		☐	☐		
馆藏册数	int			☐	☐		
在馆册数	int			☐	☐		
书架号	varchar	15		☐	☐		
被借次数	int			☐	☐		

图 4.2 book 表的创建

对应的 SQL 语句：

```
-- 图书信息
CREATE TABLE books  (
  书号 varchar(15) NOT NULL,
  书名 varchar(30) NOT NULL,
  作者 varchar(20) DEFAULT NULL,
  类型 varchar(15) DEFAULT NULL,
  价格 float DEFAULT NULL,
  出版社 varchar(45) DEFAULT NULL,
  摘要 varchar(50) DEFAULT NULL,
  馆藏册数 int(0) DEFAULT NULL,
  在馆册数 int(0) DEFAULT NULL,
  书架号 varchar(15) DEFAULT NULL,
  被借次数 int(0) DEFAULT NULL,
  PRIMARY KEY (书号) USING BTREE,
  INDEX 书架号_idx(书架号) USING BTREE,
  CONSTRAINT 书架号 FOREIGN KEY (书架号) REFERENCES bookshelfs (书架号)
   ON DELETE RESTRICT ON UPDATE RESTRICT
) ;
```

4.3 读者类型关系模式

读者类型的 E-R 图中，读者类型用于确保唯一性，因此作为主码。

关系模式：readertype (读者类型, 借书时间, 最多在借册数)

同理可知，不存在非主属性部分依赖，不存在传递依赖，且候选码读者类型是唯一的决定因素，满足 BCNF。

由此创建对应的 readertype 表：

图 4.3 readertype 表的创建

对应的 SQL 语句：

```
-- 读者类型表
CREATE TABLE readertype  (
  读者类型 varchar(10)  NOT NULL,
  借书时间 int(0) DEFAULT NULL,
  最多在借册数 int(0) DEFAULT NULL,
  PRIMARY KEY (读者类型) USING BTREE
);
INSERT INTO readertype VALUES ('教师', 3, 30);
INSERT INTO readertype VALUES ('本科生', 1, 10);
INSERT INTO readertype VALUES ('研究生', 2, 20);
```

4.4 读者关系模式

读者的 E-R 图中，实体为读者，其中 ID 用于确保唯一性，因此作为读者的主码。

关系模式：readers (ID, 姓名, 性别, 单位, 读者类型, 可借册数, 在借册数, password, 欠款)

同理可知，不存在非主属性部分依赖，不存在传递依赖，且候选码 ID 是唯一的决定因素，满足 BCNF。

由此创建对应的 readers 表：

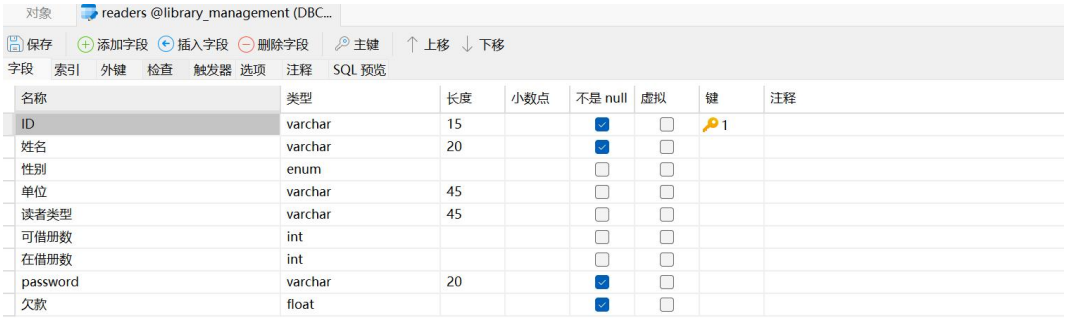


图 4.4 readers 表的创建

对应的 SQL 语句：

```
-- 读者表
CREATE TABLE readers  (
  ID varchar(15) NOT NULL,
```

```
  姓名 varchar(20) NOT NULL,
  性别 enum('男','女') DEFAULT NULL,
  单位 varchar(45) DEFAULT NULL,
  读者类型 varchar(45) DEFAULT NULL,
  可借册数 int(0) DEFAULT NULL,
  在借册数 int(0) DEFAULT NULL,
  password varchar(20) NOT NULL DEFAULT '123456',
  欠款 float NOT NULL DEFAULT 0,
  PRIMARY KEY (ID) USING BTREE,
  CONSTRAINT 读者类型 FOREIGN KEY (读者类型) REFERENCES readertype (读者类型)
  ON DELETE RESTRICT ON UPDATE RESTRICT
);
```

4.5 登录记录关系模式

登录记录的 E-R 图中，实体为登录记录，时间与次数作为联和主码。

关系模式：loginrecord (ID, time, number)

同理可知，不存在非主属性部分依赖，不存在传递依赖，且候选码(time, number)是唯一的决定因素，满足 BCNF。

由此创建对应的 loginrecord 表：

图 4.5 loginrecord 表的创建

对应的 SQL 语句：

```
-- 登录记录
CREATE TABLE loginrecord  (
  ID varchar(15) NOT NULL,
  time date NOT NULL,
  number int(0) NOT NULL,
  PRIMARY KEY (time, number) USING BTREE,
  INDEX readerid_idx(ID) USING BTREE,
  CONSTRAINT readerid2 FOREIGN KEY (ID) REFERENCES readers (ID)
  ON DELETE RESTRICT ON UPDATE RESTRICT
);
```

4.6 管理员关系模式

管理员的 E-R 图中，实体为管理员，ID 与类型作为联和主码。

关系模式：workers (ID, 姓名, type, password)

同理可知，不存在非主属性部分依赖，不存在传递依赖，且候选码(ID, type)是唯一的决定因素，满足 BCNF。

由此创建对应的 workers 表：

图 4.6 workers 表的创建

对应的 SQL 语句：

```
-- 工作人员
CREATE TABLE workers  (
  ID varchar(15) NOT NULL,
  姓名 varchar(20) NOT NULL,
  type enum('图书管理员','系统管理员') NOT NULL,
  password varchar(20) NOT NULL DEFAULT '123456',
  PRIMARY KEY (ID, type) USING BTREE
);
INSERT INTO workers VALUES ('01', '落兮呀', '系统管理员', '050330zzp');
INSERT INTO workers VALUES ('10001', '张三', '图书管理员', '123456');
```

4.7 借还申请关系模式

借还申请的 E-R 图中，实体为借还申请，主码为全码。

关系模式：item (bookid, ID, time, type)

由于主码为全码，必定满足 BCNF。

由此创建对应的 item 表：

图 4.7 item 表的创建

对应的 SQL 语句：

```
-- 借书/还书申请
CREATE TABLE item  (
  bookid varchar(15) NOT NULL,
  ID varchar(15) NOT NULL,
  time date NOT NULL,
  type enum('borrow','return') NOT NULL,
  PRIMARY KEY (bookid, ID, time, type) USING BTREE,
  INDEX readerid_idx(ID) USING BTREE,
  CONSTRAINT bookid FOREIGN KEY (bookid) REFERENCES books (书号)
  ON DELETE RESTRICT ON UPDATE RESTRICT,
  CONSTRAINT readerid FOREIGN KEY (ID) REFERENCES readers (ID)
  ON DELETE RESTRICT ON UPDATE RESTRICT
);
```

4.8 借阅记录关系模式

在读者与书籍之间存在“借阅”关系，并且是多对多，需要单独转化成一个关系模式“借阅记录”，包含了书号与读者 ID，以及时间，三者联合作为主码。

关系模式：borrow (ID, 书号, 借书时间)

由于主码为全码，必定满足 BCNF。

由此创建对应的 borrow 表：

图 4.8 borrow 表的创建

对应的 SQL 语句：

```
-- 借书记录
CREATE TABLE borrow  (
  ID varchar(15) NOT NULL,
  书号 varchar(15) NOT NULL,
  借书时间 date NOT NULL,
  PRIMARY KEY (ID, 书号, 借书时间) USING BTREE,
  INDEX 书号_idx(书号) USING BTREE,
  CONSTRAINT ID FOREIGN KEY (ID) REFERENCES readers (ID)
  ON DELETE RESTRICT ON UPDATE RESTRICT,
  CONSTRAINT 书号 FOREIGN KEY (书号) REFERENCES books (书号)
  ON DELETE RESTRICT ON UPDATE RESTRICT
);
```

5 物理结构设计

5.1 索引设计

5.1.1 主键索引

主键索引通过唯一标识每一行记录，在大多数情况下自动采用 B+ 树结构，提高数据的插入、查询效率，并确保数据唯一性。（对应 SQL 语句在逻辑结构设计对应表部分已给出）

表 5.1 主键索引设计

表名	主键字段	设计理由	索引方法
books	书号	每本图书的唯一编号，便于定位图书信息	B+ 树
bookshelfs	书架号	唯一标识每个书架类型，供图书归类	B+ 树
readers	ID	每个读者具有唯一编号， 支撑借阅、登录等模块	B+ 树
readertype	读者类型	用于唯一标识“教师、本科生”等类型	B+ 树
borrow	ID, 书号, 借书时间	联合主键，防止重复借阅同一本书	B+ 树
item	bookid, ID, time, type	联合主键，确保一条申请记录唯一	B+ 树

续表 5.1 主键索引设计

loginrecord	time, number	保证每日登录编号唯一	B+ 树
workers	ID, type	同一 ID 可有不同管理员类型 联合主键保障唯一性	B+ 树

5.1.2 辅助索引

为了优化查询性能，在部分外键字段或常用于条件查询的字段上设置了普通索引。（对应 SQL 语句在逻辑结构设计对应表部分已给出）

表 5.2 辅助索引设计

表名	字段	索引名	作用说明	索引方法
books	书架号	书架号_idx	加快按书架分类查询图书	B+ 树
borrow	书号	书号_idx	加速按书号查找借阅记录	B+ 树
item	ID	readerid_idx	便于查询某个读者的借/还申请	B+ 树
loginrecord	ID	readerid_idx	快速定位读者登录历史	B+ 树

5.2 完整性约束设计

5.2.1 外键约束

外键连接两个表，确保子表中的字段必须与父表中存在的值一致，从而维护数据间的逻辑联系。（对应 SQL 语句在逻辑结构设计对应表部分已给出）

表 5.3 外键约束设计

表名	外键字段	关联父表	限制
books	书架号	bookshelfs	图书必须归属于已存在的书架类型
readers	读者类型	readertype	保证读者类型合法，便于设置借书上限
borrow	ID, 书号	readers, books	借书记录必须对应存在的读者与图书
item	ID, bookid	readers, books	借还申请只能由合法读者发起， 对应图书必须存在
loginrecord	ID	readers	登录记录必须对应存在的读者信息

所有外键统一采用约束策略：ON DELETE RESTRICT ON UPDATE RESTRICT

ON DELETE RESTRICT 为了防止删除父表记录后导致子表引用失效（如不能删除仍有借书记录的图书或读者），ON UPDATE RESTRICT 为了防止更新主键值导致子表数据引用错误，保证关系的一致性。

5.2.2 属性上的约束

系统中大量采用属性上的约束增强数据合法性，用来确保用户输入或程序操作不会破坏数据规范。（对应 SQL 语句在逻辑结构设计对应表部分已给出）

表 5.4 属性上的约束设计

类型	示例字段	说明
非空约束（NOT NULL）	书号, ID, 姓名, 密码	保证关键字段不可省略

续表 5.4 属性上的约束设计

默认值	readers.password, readers.欠款	简化插入操作，提升初始化效率
枚举类型	性别 enum('男','女') item.type enum('borrow','return')	限制字段值只能在预设集合中

5.3 触发器设计

触发器是数据库在指定操作前后自动执行的语句块，用于实现复杂或自动化的数据维护逻辑。本系统共设计了四个触发器，涵盖用户插入、读者删除、借书审批、还书审批等关键场景。

5.3.1 读者插入触发器

当插入新读者记录时，自动根据其所属读者类型在 readertype 表中查找并设置其“可借册数”，自动设置可借上限，实现借阅权限的自动初始化。触发时间为 BEFORE INSERT ON readers

对应 SQL 语句如下：

```
-- 读者初始化触发器
CREATE TRIGGER set_borrow_limit_on_insert
BEFORE INSERT ON readers
FOR EACH ROW
BEGIN
  SET NEW.可借册数 = (
    SELECT 最多在借册数 FROM readertype WHERE 读者类型 = NEW.读者类型
  );
END;
```

5.3.2 读者删除触发器

若直接删除 readers 表中的用户，而不删除其在 loginrecord 表中的关联数据，会违反外键约束。因此此触发器在删除读者之前，会先清理其所有登录记录。触发时间为 BEFORE DELETE ON readers

对应 SQL 语句如下：

```
-- 删除读者时的触发器，删除对应登录记录
CREATE TRIGGER delete_reader_login_record
BEFORE DELETE ON readers
FOR EACH ROW
BEGIN
  DELETE FROM loginrecord WHERE ID = OLD.ID;
END;
```

5.3.3 借书审批触发器

用户提交借书申请记录存在于 item 表，当管理员审批通过后删除该记录时，系统自动执行以下动作：在 borrow 表中插入正式借书记录；将 readers 表中读者的“可借册数”减 1，“在借册数”加 1；将 books 表中图书的“在馆册数”减 1，“被借次数”加 1。触发时间为 AFTER DELETE ON item（type 为 borrow）

对应 SQL 语句如下：

```
-- 批准借书时的自动操作
CREATE TRIGGER after_item_delete_borrow
AFTER DELETE ON item
FOR EACH ROW
BEGIN
  IF OLD.type = 'borrow' THEN
    -- 插入 borrow 表
```

```
    INSERT INTO borrow(ID, 书号, 借书时间)
    VALUES (OLD.ID, OLD.bookid, OLD.time);
    -- 更新 readers 表
    UPDATE readers SET 可借册数 = 可借册数 - 1, 在借册数 = 在借册数 + 1
    WHERE ID = OLD.ID;
    -- 更新 books 表
    UPDATE books SET 在馆册数 = 在馆册数 - 1, 被借次数 = 被借次数 + 1
    WHERE 书号 = OLD.bookid;
  END IF;
END;
```

5.3.4 还书审批触发器

当管理员批准还书申请后删除 item 表记录时，触发器自动删除 borrow 表中对应的借阅记录；更新 readers 的借阅状态（可借册数+1，在借册数-1）；更新 books 的库存状态（在馆册数+1）。触发时间为 AFTER DELETE ON item（type 为 return）

对应 SQL 语句如下：

```
-- 批准还书时的自动操作
CREATE TRIGGER after_item_delete_return
AFTER DELETE ON item
FOR EACH ROW
BEGIN
  IF OLD.type = 'return' THEN
    -- 删除 borrow 表中的借阅记录
    DELETE FROM borrow
    WHERE ID = OLD.ID AND 书号 = OLD.bookid;
    -- 更新 readers 表
    UPDATE readers SET 可借册数 = 可借册数 + 1, 在借册数 = 在借册数 - 1
    WHERE ID = OLD.ID;
    -- 更新 books 表
    UPDATE books SET 在馆册数 = 在馆册数 + 1
    WHERE 书号 = OLD.bookid;
  END IF;
END;
```

6 系统具体介绍及实现

6.1 数据库连接模块

建立与 MySQL 数据库 library_management 的连接，并返回用于执行 SQL 语句的游标对象和管理数据库连接的连接对象，提供统一的数据库连接接口，简化其他模块的数据库操作，为图书馆管理系统的其他模块提供数据库操作的基础支持。

通过 pymysql.connect 方法，根据指定的主机地址（localhost）、用户名（root）、密码（050330zzp）和数据库名（library_management）建立数据库连接，返回 Connection 对象 conn。调用 conn.cursor() 方法创建游标对象 cursor，用于执行 SQL 语句和处理查询结果。

Python 代码如下：

```
# 数据库链接模块
import pymysql
# 数据库链接
def connect():
    conn = pymysql.connect(host='localhost',
```

```
                        user='root',
                        password='050330zzp',
                        database='library_management')
    cursor = conn.cursor()
    return cursor, conn
```

6.2 登录界面

6.2.1 登录功能概述

登录界面为图书管理系统的入口，支持三种不同身份（读者、图书管理员、系统管理员）的用户登录。用户需输入 ID 和密码，并选择身份类型，点击“登录”按钮进行验证。若输入信息正确，将根据用户身份跳转到相应的管理界面；若信息错误，会弹出警告提示。同时，用户可点击“退出”按钮关闭程序。

图 6.1 登录界面

6.2.2 登录验证逻辑

在 login 方法中，根据用户选择的身份类型执行不同的数据库查询语句进行验证。当未输入任何信息时，会有对应的提示；在输入错误的用户名或密码时，也会提示。

图 6.2 登录界面输入提示

读者登录：查询 readers 表中的 ID 与密码，若验证通过，记录登录时间，根据登录时间在 loginrecord 表中搜索记录，得到对应的登录次数，将记录插入 loginrecord 表，然后跳转到读者界面。

Python 代码如下：

```
        # 读者登录
        if self.idbox.currentText() == '读者':
            sql = 'SELECT * FROM readers ' \
                  'WHERE ID = "%s" AND password="%s"' % (ID, PW)
```

```
            res = cursor.execute(sql)
            if res:
                logintime = time.strftime("%Y-%m-%d", time.localtime())
                sql = 'SELECT * FROM loginrecord ' \
                      'WHERE time="%s"' % logintime
                res = cursor.execute(sql)
                logined = cursor.fetchall()
                if res:
                    last = logined[-1]
                    number = last[-1]
                    num = number + 1
                    sql = 'INSERT INTO loginrecord(ID,time,number) ' \
                          'VALUES(%s,"%s",%d)' % (ID, logintime, num)
                    cursor.execute(sql)
                    conn.commit()
                else:
                    sql = 'INSERT INTO loginrecord(ID,time,number) ' \
                          'VALUES(%s,"%s",%d)' % (ID, logintime, 1)
                    cursor.execute(sql)
                    conn.commit()
                self.read = Readerui()
                self.read.show()
                self.close()
            else:
                QMessageBox.warning(self, "警告", "用户 ID 或密码错误，请重新输入!", QMessageBox.Yes)
```

管理员登录：查询 workers 表中的 ID、密码与对应类型，若验证通过，跳转到对应的管理员界面。

Python 代码如下：

```
        # 图书管理员登录
        elif self.idbox.currentText() == '图书管理员':
            type = '图书管理员'
            sql = 'SELECT * FROM workers ' \
                  'WHERE ID = "%s" AND password="%s" AND type="%s" ' % (ID, PW, type)
            res = cursor.execute(sql)
            # 进行判断
            if res:
                self.bookadmin = bookadminui()
                self.bookadmin.show()
                self.close()
                pass
            else:
                QMessageBox.warning(self, "警告", "用户 ID 或密码错误，请重新输入!", QMessageBox.Yes)
        # 系统管理员登录
        elif self.idbox.currentText() == '系统管理员':
            type = '系统管理员'
            sql = 'SELECT * FROM workers ' \
                  'WHERE ID = "%s" AND password="%s" AND type="%s"' % (ID, PW, type)
            res = cursor.execute(sql)
            # 进行判断
            if res:
                self.sysadmin = sysadminui()
                self.sysadmin.show()
                self.close()
            else:
                QMessageBox.warning(self, "警告", "用户 ID 或密码错误，请重新输入!", QMessageBox.Yes)
```

6.3 读者系统

6.3.1 读者系统功能概述

读者系统内含 5 个页面，用于管理读者的相关操作，包括借书、还书、查询图书信息、修改用户信息以及处理逾期罚款等功能。系统通过连接数据库，实现了与数据库的数据交互，以存储和管理读者和图书的相关信息。具体提供了以下功能：

借书：读者可以通过书号或书名查询可借图书，并提交借书申请，其中对书名进行了模糊搜索处理。

还书：读者可以输入书号查询已借图书，并提交还书申请。

查询图书信息：读者可以根据书名、书号、作者、类别、出版社等条件查询图书信息，同样对书名进行了模糊搜索处理。

用户信息修改：读者可以修改自己的姓名、性别和密码。

逾期罚款支付：读者可以查看需支付的罚款金额，并通过不同方式支付。

6.3.2 借书功能

借书界面主要用于读者查询可借图书，并提交借书申请。读者可以通过输入书号或书名进行查询，对于书名做了模糊搜索处理。系统会显示符合条件的图书信息，包括书号、书名、作者、出版社和在馆册数。在默认情况下，界面会自动显示全部书籍。读者选择想要借阅的图书后，点击“提交”按钮即可提交借书申请。同一次登录，同一本书只能借一本，否则会弹出提示。

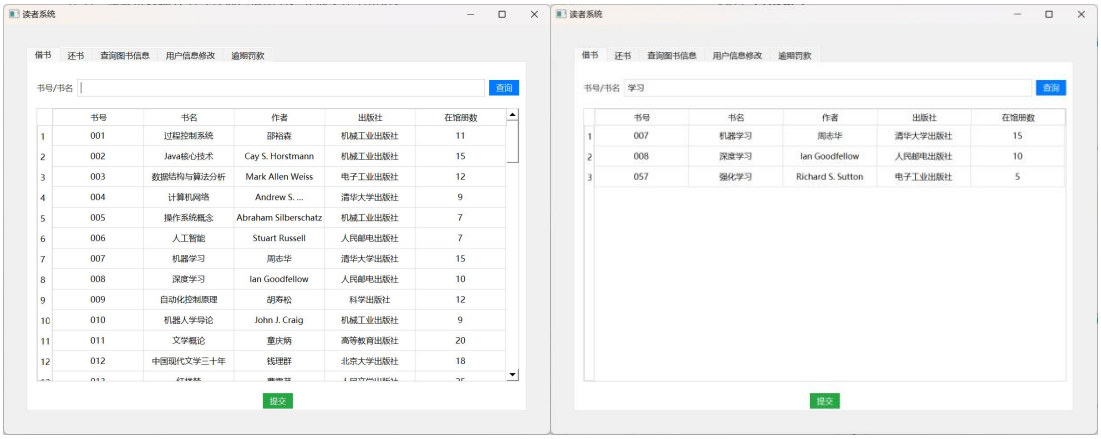


图 6.3 借书界面及搜索功能演示

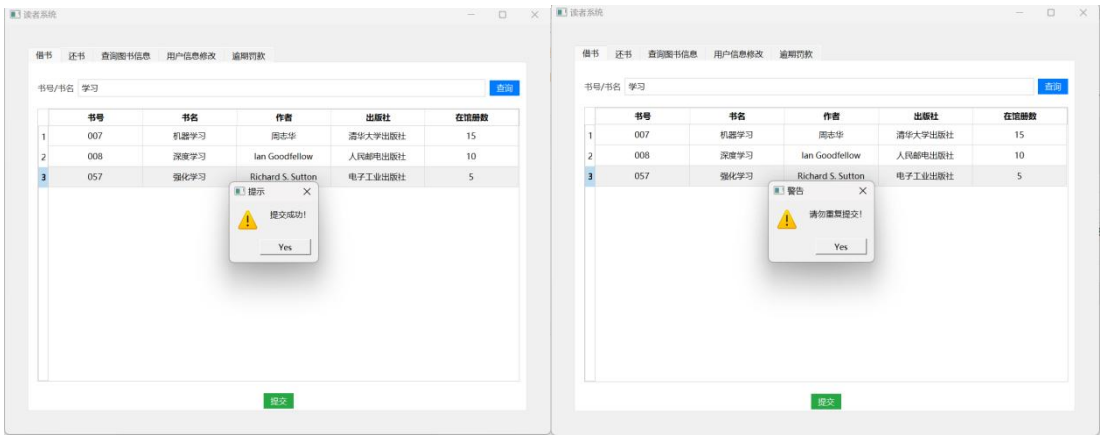


图 6.4 借书提交及重复提交提示演示

借书界面的查询函数获取用户在输入框中输入的书号或书名，如果输入不为空，则生成包含输入信息的 SQL 查询语句；如果输入为空，则查询所有图书信息。执行 SQL 查询并获取结果，根据结果的数量设置表格的行数，然后将每本图书的书号、书名、作者、出版社和在馆册数显示在表格中。

Python 代码如下：

```
# 借书信息查询
def borrowidcheck(self):
    bookin = self.borrowbookid.text()
    if bookin:
        sql = 'SELECT * FROM books WHERE 书号="%s" OR 书名 LIKE "%%%s%%"' % (bookin, bookin)
    else:
        sql = 'SELECT * FROM books'
    cursor.execute(sql)
    bookinfo = cursor.fetchall()
    n = len(bookinfo)
    self.borrowtable.setRowCount(n)
    for i in range(n):
        book = bookinfo[i]
        bookid = QTableWidgetItem(book[0])
        bookid.setTextAlignment(QtCore.Qt.AlignCenter)
        bookname = QTableWidgetItem(book[1])
        bookname.setTextAlignment(QtCore.Qt.AlignCenter)
        bookauthor = QTableWidgetItem(book[2])
        bookauthor.setTextAlignment(QtCore.Qt.AlignCenter)
        bookpress = QTableWidgetItem(book[5])
        bookpress.setTextAlignment(QtCore.Qt.AlignCenter)
        booknumber = QTableWidgetItem(str(book[8]))
        booknumber.setTextAlignment(QtCore.Qt.AlignCenter)
        self.borrowtable.setItem(i, 0, bookid)
        self.borrowtable.setItem(i, 1, bookname)
        self.borrowtable.setItem(i, 2, bookauthor)
        self.borrowtable.setItem(i, 3, bookpress)
        self.borrowtable.setItem(i, 4, booknumber)
```

提交时需要先获取当前登录的读者 ID，实现方法为获取当前日期，生成查询当天登录记录的 SQL 语句，按照编号排序。执行 SQL 查询并获取所有结果，取最后一条记录作为当前读者的登录记录，提取其中的 ID 并返回。Python 代码如下：

```
# 获取当前读者 ID
def getreaderid(self):
    nowtime = time.strftime("%Y-%m-%d", time.localtime())
    sql = 'SELECT * FROM loginrecord ' \
          'WHERE time = "%s" ' \
          'ORDER BY number' % nowtime
    cursor.execute(sql)
    todaylogin = cursor.fetchall()
    readerlogin = todaylogin[-1]
    ID = readerlogin[0]
    return ID
```

获取到用户的读者 ID 后，检查用户是否选择了要借阅的图书，如果没有选择，则弹出警告消息。如果选择了图书，检查该书的在馆册数是否为 0，如果为 0，则弹出警告消息。接着检查用户是否已经提交过该图书的借书申请，如果已经提交过，则弹出警告消息。如果以上条件都不满足，则插入一条借书记录到

item 表中，并提交事务，最后弹出提交成功的提示消息。Python 代码如下：

```
# 提交借书申请
    def submit(self):
        s = self.borrowtable.currentRow()
        if s == -1:
            QMessageBox.warning(self.mainwindow, "警告", "请点击想借阅的书！", QMessageBox.Yes)
        else:
            remain = int(self.borrowtable.item(s, 4).text())
            if remain == 0:
                QMessageBox.warning(self.mainwindow, "警告", "这本书已经借光啦！", QMessageBox.Yes)
            else:
                t = self.borrowtable.item(s, 0).text()
                sql = 'SELECT * FROM item ' \
                      'where ID="%s" and bookid="%s" and type="borrow"' % (self.loginID, t)
                res = cursor.execute(sql)
                if res:
                    QMessageBox.warning(self.mainwindow, "警告", "请勿重复提交！", QMessageBox.Yes)
                else:
                    nowtime = time.strftime("%Y-%m-%d", time.localtime())
                    sql = 'INSERT INTO item(bookid,ID,time,type) ' \
                          'VALUES ("%s","%s","%s","borrow")' % (t, self.loginID, nowtime)
                    cursor.execute(sql)
                    conn.commit()
                    QMessageBox.warning(self.mainwindow, "提示", "提交成功！", QMessageBox.Yes)
```

6.3.3 还书功能

还书界面用于读者查询已借图书，并提交还书申请。界面默认显示全部已有借阅记录，读者可以输入书号进行查询，也可以点击“显示全部”按钮查看所有已借图书，查询结果会显示书籍基本信息和借书时间，选择要归还的图书后，点击“还书”按钮提交还书申请。若未选择，会弹出提示。由于提交还书申请后还需管理员审批，所以系统会检测是否为重复提交。

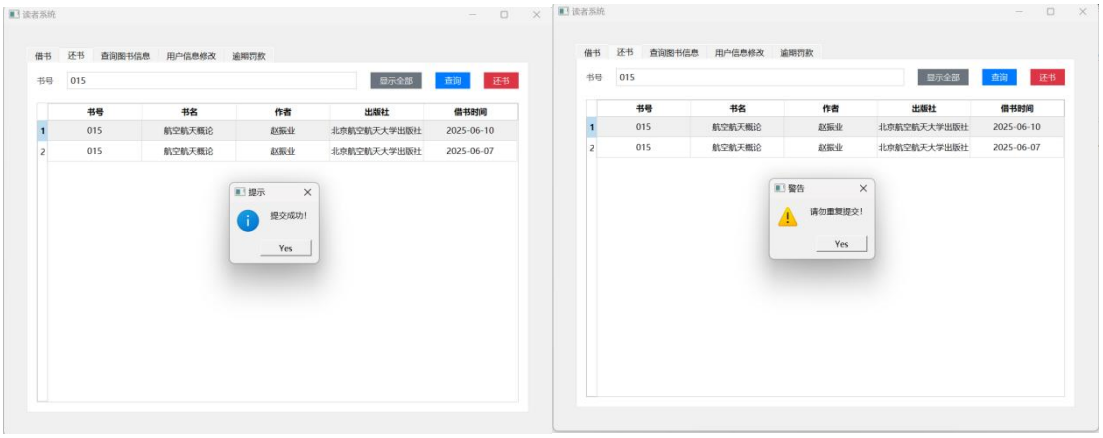


图 6.5 还书提交及重复提交提示演示

还书界面查询函数获取用户在还书输入框中输入的书号，进行查询；如果输入为空，则查询当前读者的所有已借图书信息。然后将每本已借图书的基本信息和借书时间显示在表格中。Python 代码如下：

```
# 还书查询
    def returnquery(self):
        bookid = self.returnbookid.text().strip()
        if bookid:
```

```
            sql = 'SELECT br.书号, b.书名, b.作者, b.出版社, br.借书时间 ' \
                  'FROM borrow br, books b ' \
                  'WHERE br.书号 = b.书号 AND br.ID = "%s" AND br.书号 = "%s" ' \
                  'ORDER BY br.借书时间 DESC' % (self.loginID, bookid)
        else:
            sql = 'SELECT br.书号, b.书名, b.作者, b.出版社, br.借书时间 ' \
                  'FROM borrow br, books b ' \
                  'WHERE br.书号 = b.书号 AND br.ID = "%s" ' \
                  'ORDER BY br.借书时间 DESC' % self.loginID
        cursor.execute(sql)
        results = cursor.fetchall()
        self.return_table.setRowCount(len(results))
        for i, row in enumerate(results):
            for j, val in enumerate(row):
                item = QTableWidgetItem(str(val))
                item.setTextAlignment(QtCore.Qt.AlignCenter)
                self.return_table.setItem(i, j, item)
```

提交还书申请时，检查用户是否输入了书号，没有输入则弹出警告消息。接着检查用户是否已经提交过该图书的还书申请，如果已经提交过，则弹出警告消息。随后插入一条还书记录到 item 表中，并提交事务，最后弹出提交成功的提示消息，并重新查询已借图书信息。Python 代码如下：

```
    # 提交还书申请
    def ReturnBook(self):
        bookid = self.returnbookid.text().strip()
        if not bookid:
            QMessageBox.warning(self.mainwindow, "警告", "请输入书号或从列表中选择！", QMessageBox.Yes)
            return
        nowtime = time.strftime("%Y-%m-%d", time.localtime())
        sql = 'SELECT * FROM item ' \
              'WHERE ID="%s" AND time="%s" AND type="return" AND bookid="%s"' % (
              self.loginID, nowtime, bookid)
        res = cursor.execute(sql)
        if res:
            QMessageBox.warning(self.mainwindow, "警告", "请勿重复提交！", QMessageBox.Yes)
        else:
            sql = 'INSERT INTO item(bookid, ID, time, type) ' \
                  'VALUES ("%s", "%s", "%s", "return")' % (bookid, self.loginID, nowtime)
            cursor.execute(sql)
            conn.commit()
            QMessageBox.information(self.mainwindow, "提示", "提交成功！", QMessageBox.Yes)
            self.returnquery()
```

6.3.4 查询书籍功能

此界面允许读者根据多种条件查询图书信息，包括书名、书号、作者、类别和出版社。读者可以点击“查询所有”按钮查看所有图书信息，也可以输入具体条件后点击“查询”按钮进行精确查询。同样对书名搜索进行了模糊搜索处理。若不满足搜索条件会弹出对应提示框。

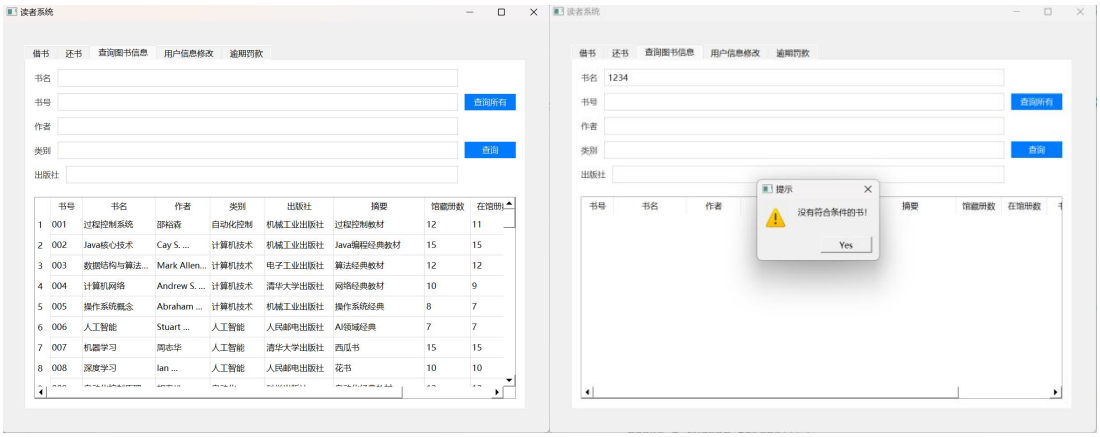


图 6.6 书籍查询演示

条件查询函数获取用户在各个输入框中输入的书名、书号、作者、类别和出版社信息，根据输入情况生成相应的 SQL 查询语句。执行查询并获取结果，如果有结果，则将结果显示在表格中；如果没有结果，则弹出提示消息。Python 代码如下：

```
# 条件查询书籍
def selectbook(self):
    bookname = self.bookid.text()
    bookid = self.bookname.text()
    bookauthor = self.author.text()
    booktype = self.type.text()
    bookpress = self.press.text()
    an = 0
    if bookname:
        bookname = ' 书名 LIKE "%%%s%%"' % (bookname)
        an = 1
    if bookid:
        if an == 1:
            bookid = ' and 书号="%s"' % (bookid)
        else:
            bookid = ' 书号="%s"' % (bookid)
            an = 1
    if bookauthor:
        if an == 1:
            bookauthor = 'and 作者="%s"' % (bookauthor)
        else:
            bookauthor = '作者="%s"' % (bookauthor)
            an = 1
    if booktype:
        if an == 1:
            booktype = ' and 类型="%s"' % (booktype)
        else:
            booktype = ' 类型="%s"' % (booktype)
            an = 1
    if bookpress:
        if an == 1:
            bookpress = ' and 出版社="%s"' % (bookpress)
        else:
            bookpress = ' 出版社="%s"' % (bookpress)
    sql1 = 'SELECT * FROM books where'
    sql = sql1 + bookname + bookid + bookauthor + booktype + bookpress
    res = cursor.execute(sql)
```

```
        books = cursor.fetchall()
        if res:
            booknumber = len(books)
            self.tableWidget.setRowCount(booknumber)
            for i in range(booknumber):
                book = books[i]
                self.tableWidget.setItem(i, 0, QTableWidgetItem(book[0]))
                self.tableWidget.setItem(i, 1, QTableWidgetItem(book[1]))
                self.tableWidget.setItem(i, 2, QTableWidgetItem(book[2]))
                self.tableWidget.setItem(i, 3, QTableWidgetItem(book[3]))
                self.tableWidget.setItem(i, 4, QTableWidgetItem(book[5]))
                self.tableWidget.setItem(i, 5, QTableWidgetItem(book[6]))
                self.tableWidget.setItem(i, 6, QTableWidgetItem(str(book[7])))
                self.tableWidget.setItem(i, 7, QTableWidgetItem(str(book[8])))
                self.tableWidget.setItem(i, 8, QTableWidgetItem(book[9]))
                self.tableWidget.setItem(i, 9, QTableWidgetItem(str(book[10])))
        else:
            QMessageBox.warning(self.mainwindow, "提示", "没有符合条件的书！", QMessageBox.Yes)
```

6.3.5 用户信息修改功能

该界面用于读者修改自己的个人信息，包括姓名、性别和密码。读者输入新的信息后，点击“修改”按钮即可更新个人信息。对于性别修改，若输入不是男/女，会有对应提示。

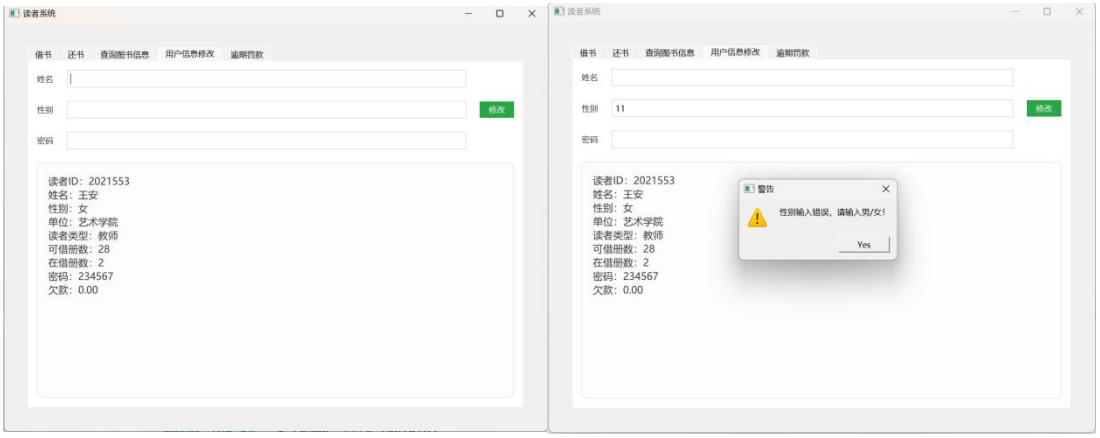


图 6.7 用户信息修改演示

修改信息函数获取用户在姓名、性别和密码输入框中输入的信息。如果姓名输入不为空，则更新读者的姓名信息；如果性别输入为“男”或“女”，则更新读者的性别信息；如果性别输入不为空且不是“男”或“女”，则弹出警告消息；如果密码输入不为空，则更新读者的密码信息。最后调用函数更新界面上显示的读者信息。Python 代码如下：

```
    # 读者修改自己的信息
    def alterinfo(self):
        name = self.lineEdit_8.text()
        sex = self.lineEdit_9.text()
        password = self.lineEdit_11.text()
        if name:
            sql = 'UPDATE readers SET 姓名="%s" ' \
                  'WHERE ID="%s"' % (name, self.loginID)
            cursor.execute(sql)
            conn.commit()
        if sex == '男' or sex == '女':
```

```
            sql = 'UPDATE readers SET 性别="%s" ' \
                  'WHERE ID="%s"' % (sex, self.loginID)
            cursor.execute(sql)
            conn.commit()
        elif sex != '':
            QMessageBox.warning(self.mainwindow, "警告", "性别输入错误，请输入男/女！", QMessageBox.Yes)
        if password:
            sql = 'UPDATE readers SET password="%s" ' \
                  'WHERE ID="%s"' % (password, self.loginID)
            cursor.execute(sql)
            conn.commit()
        self.getreaderinfo()
```

6.3.6 逾期罚款功能

此界面用于显示读者的逾期罚款金额，并提供微信和支付宝支付方式。读者可以查看需支付的罚款金额，点击“微信支付”或“支付宝支付”按钮进行支付。若罚款已经为 0 则会弹出提示窗。

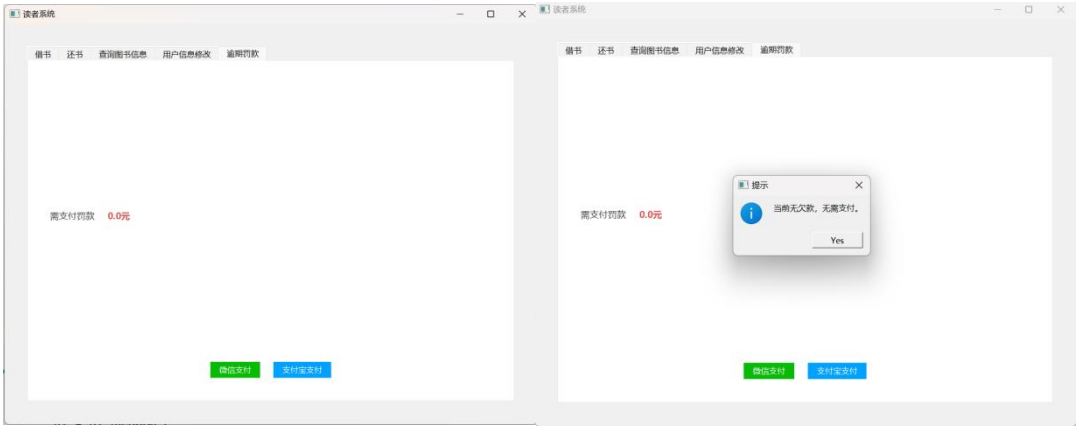


图 6.8 逾期罚款界面演示

支付罚款函数查询当前读者的欠款金额，如果欠款金额为 0，则弹出提示消息表示无需支付。如果欠款金额不为 0，则将欠款金额清零，更新读者信息，并弹出支付成功的提示消息。Python 代码如下：

```
    # 支付欠款（链接支付接口）
    def paymoney(self):
        # 获取当前欠款金额
        sql = 'SELECT 欠款 FROM readers ' \
              'WHERE ID="%s"' % self.loginID
        cursor.execute(sql)
        result = cursor.fetchone()
        if result and float(result[0]) == 0.0:
            QMessageBox.information(self.mainwindow, "提示", "当前无欠款，无需支付。", QMessageBox.Yes)
            return
        # 否则清零并提示成功
        money = 0.00
        sql = 'UPDATE readers SET 欠款="%f" ' \
              'WHERE ID="%s"' % (money, self.loginID)
        cursor.execute(sql)
        conn.commit()
        self.getreaderinfo()
        QMessageBox.information(self.mainwindow, "提示", "支付成功，欠款已清零！", QMessageBox.Yes)
```

6.4 图书管理员系统

6.4.1 图书管理员系统概述

图书管理员系统内含 3 个界面，主要包含读者信息查询、书籍信息管理以及借阅/归还审批等功能。系统通过连接数据库，实现对读者、书籍和借阅申请信息的增删改查操作，实现了基本的图书管理操作。

6.4.2 查询读者信息功能

此页面的主要功能是查询读者信息，用户既能够查看所有读者的信息，也可以依据读者 ID、姓名和单位进行筛选查询。由于权限设置，图书管理员只能查询读者信息而不能做任何修改。若输入的信息插叙不到读者，会弹出对应提示框。

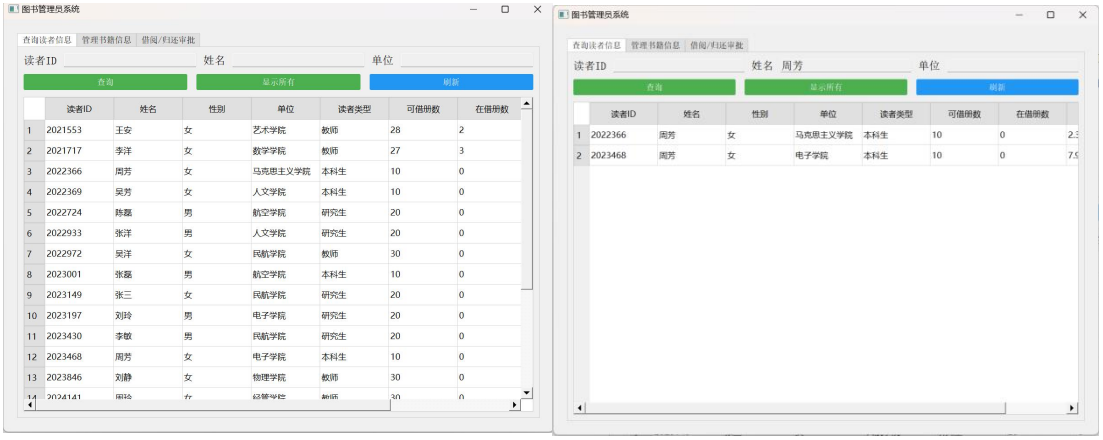


图 6.9 查询读者界面演示

查询全部读者函数从数据库的 readers 表中查询所有读者信息，并将其显示在表格中。条件查询读者函数根据用户在输入框中输入的读者 ID、姓名和单位信息，从数据库的 readers 表中筛选出符合条件的读者信息。Python 代码如下：

```
# 显示所有的读者
def show_all_readers(self):
    sql = "SELECT * FROM readers"
    res = cursor.execute(sql)
    readerinfo = cursor.fetchall()
    self.readerinfors.setRowCount(res)
    for i in range(res):
        reader = readerinfo[i]
        readerid = QTableWidgetItem(reader[0])
        readername = QTableWidgetItem(reader[1])
        readersex = QTableWidgetItem(reader[2])
        readerunit = QTableWidgetItem(reader[3])
        readertype = QTableWidgetItem(reader[4])
        books_can_borrow = QTableWidgetItem(str(reader[5]))
        books_borrowed = QTableWidgetItem(str(reader[6]))
        charge_to_pay = QTableWidgetItem(str(reader[8]))
        self.readerinfors.setItem(i, 0, readerid)
        self.readerinfors.setItem(i, 1, readername)
        self.readerinfors.setItem(i, 2, readersex)
        self.readerinfors.setItem(i, 3, readerunit)
        self.readerinfors.setItem(i, 4, readertype)
        self.readerinfors.setItem(i, 5, books_can_borrow)
```

```
            self.readerinfors.setItem(i, 6, books_borrowed)
            self.readerinfors.setItem(i, 7, charge_to_pay)
    # 显示读者信息
    def display_readerinfo(self):
        row = self.readerinfors.currentRow()
        self.readerid.setText(self.readerinfors.item(row, 0).text())
        self.readername.setText(self.readerinfors.item(row, 1).text())
        self.unit.setText(self.readerinfors.item(row, 3).text())
    # 选择读者
    def readerselect(self):
        readerid = self.readerid.text()
        readername = self.readername.text()
        readerunit = self.unit.text()
        abc = 0
        if readerid:
            readerid = 'ID="%s"' % (readerid)
            abc = 1
        if readername:
            if abc == 1:
                readername = 'and 姓名="%s"' % (readername)
            else:
                readername = '姓名="%s"' % (readername)
                abc = 1
        if readerunit:
            if abc == 1:
                readerunit = 'and 单位="%s"' % (readerunit)
            else:
                readerunit = '单位="%s"' % (readerunit)
        sql0 = 'SELECT * FROM readers WHERE '
        sql1 = readerid + readername + readerunit
        sql = sql0 + sql1
        if sql1:
            res = cursor.execute(sql)
            if res:
                readerinfo = cursor.fetchall()  # 返回一堆元组组成的元组
                self.readerinfors.setRowCount(res)
                for i in range(res):
                    reader = readerinfo[i]
                    readerid = QTableWidgetItem(reader[0])
                    readername = QTableWidgetItem(reader[1])
                    readersex = QTableWidgetItem(reader[2])
                    readerunit = QTableWidgetItem(reader[3])
                    readertype = QTableWidgetItem(reader[4])
                    books_can_borrow = QTableWidgetItem(str(reader[5]))
                    books_borrowed = QTableWidgetItem(str(reader[6]))
                    charge_to_pay = QTableWidgetItem(str(reader[8]))
                    self.readerinfors.setItem(i, 0, readerid)
                    self.readerinfors.setItem(i, 1, readername)
                    self.readerinfors.setItem(i, 2, readersex)
                    self.readerinfors.setItem(i, 3, readerunit)
                    self.readerinfors.setItem(i, 4, readertype)
                    self.readerinfors.setItem(i, 5, books_can_borrow)
                    self.readerinfors.setItem(i, 6, books_borrowed)
                    self.readerinfors.setItem(i, 7, charge_to_pay)
            else:
                QMessageBox.warning(self.mainwindow, "警告", "没有符合条件的读者！", QMessageBox.Yes)
        else:
            QMessageBox.warning(self.mainwindow, "警告", "至少输入一项读者信息！", QMessageBox.Yes)
```

6.4.3 管理书籍信息功能

该页面主要用于管理书籍信息，涵盖添加、删除和修改书籍信息，以及查看所有书籍信息的功能。添加按钮将输入框中的书籍信息添加到数据库中。删除按钮删除表格中选中的书籍信息，删除前会检查该书籍是否存在关联记录。修改按钮修改表格中选中的书籍信息，并更新到数据库中。刷新按钮重新加载并显示所有书籍的信息。

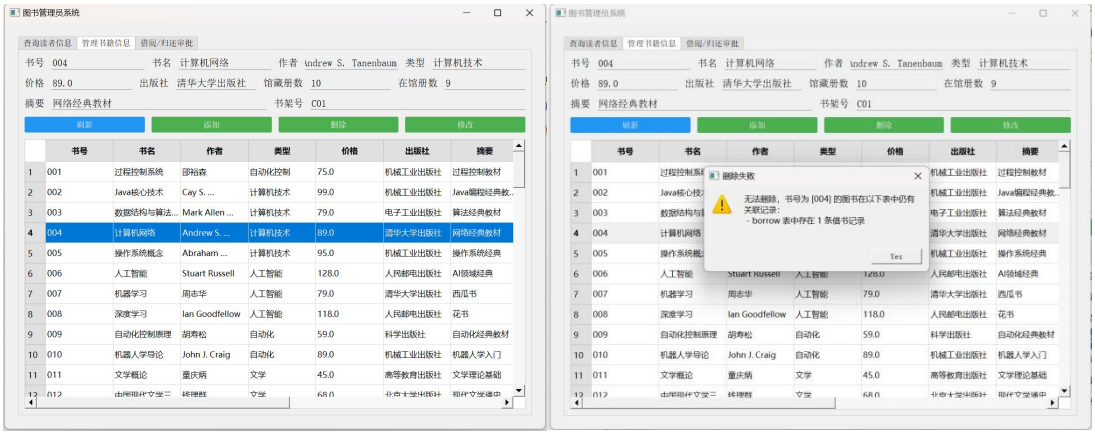


图 6.10 管理书籍信息界面功能演示

添加书籍函数获取用户输入的书籍信息，检查用户输入的信息是否完整且合法。若信息完整且合法，则执行 SQL 插入语句将书籍信息插入到数据库中，并提交事务，刷新表格显示所有书籍信息；若信息不完整或不合法，则弹出警告框提示用户。Python 代码如下：

```
    # 添加图书
    def add_book(self):
        bookid = self.bookid.text()
        bookname = self.bookname.text()
        author = self.author.text()
        booktype = self.type.text()
        prize1 = self.prize.text()
        press = self.press.text()
        abstract_2 = self.abstract_2.text()
        bookshelf = self.bookshelf.text()
        try:
            totalnum = int(self.totalnum.text())
            num = int(self.num.text())
            if prize1 != '':
                prize = float(prize1)
                if bookid and bookname and author and booktype and prize
                    and press and totalnum and num and abstract_2 and bookshelf:
                    sql = 'INSERT INTO books(书号,书名,作者,类型,价格,出版社,摘要,馆藏册数,在馆册数,书架
号,被借次数) ' \
                          'VALUES ("%s","%s","%s","%s","%.2f","%s","%s","%d","%d","%s","%d")' % (
                    bookid, bookname, author, booktype, prize, press, abstract_2, num, totalnum, boo
kshelf, 0)
                    cursor.execute(sql)
                    conn.commit()
                    self.show_all_books()
                else:
                    QMessageBox.warning(self.mainwindow, "警告", "请补全书目信息！", QMessageBox.Yes)
            else:
                QMessageBox.warning(self.mainwindow, "警告", "请补全书目信息！", QMessageBox.Yes)
```

```
        except (pymysql.err.IntegrityError, ValueError):
            QMessageBox.warning(self.mainwindow, "警告", "书目信息错误！", QMessageBox.Yes)
```

删除书籍函数检查用户是否选中了要删除的书籍，获取选中书籍的书号。检查该书籍在 item 表和 borrow 表中是否存在关联记录。若存在关联记录，则弹出警告框提示用户无法删除。若不存在关联记录，则执行删除语句将该书籍信息从数据库中删除，并提交事务。刷新表格显示所有书籍信息。若删除过程中出现错误，则回滚事务并弹出警告框提示用户。Python 代码如下：

```
    # 删除图书
    def drop_book(self):
        row = self.bookinfos.currentRow()
        if row < 0:
            QMessageBox.warning(self.mainwindow, "提示", "请先选择要删除的图书", QMessageBox.Yes)
            return
        bookid = self.bookinfos.item(row, 0).text()
        # 检查外键依赖项
        cursor.execute("SELECT COUNT(*) FROM item "
                       "WHERE bookid=%s", (bookid,))
        item_count = cursor.fetchone()[0]
        cursor.execute("SELECT COUNT(*) FROM borrow "
                       "WHERE 书号=%s", (bookid,))
        borrow_count = cursor.fetchone()[0]
        if item_count > 0 or borrow_count > 0:
            msg = "无法删除，书号为 [%s] 的图书在以下表中仍有关联记录：\n" % bookid
            if item_count > 0:
                msg += " - item 表中存在 %d 条借阅/还书申请记录\n" % item_count
            if borrow_count > 0:
                msg += " - borrow 表中存在 %d 条借书记录\n" % borrow_count
            QMessageBox.warning(self.mainwindow, "删除失败", msg, QMessageBox.Yes)
            return
        try:
            cursor.execute("DELETE FROM books "
                           "WHERE 书号=%s", (bookid,))
            conn.commit()
            self.show_all_books()
            QMessageBox.information(self.mainwindow, "成功", "图书 [%s] 删除成功!
" % bookid, QMessageBox.Yes)
        except Exception as e:
            conn.rollback()
            QMessageBox.warning(self.mainwindow, "错误", "数据库错误：" + str(e), QMessageBox.Yes)
```

修改图书信息函数获取用户修改后的书籍信息，检查用户输入的信息是否完整且合法。若信息完整且合法，则执行更新语句将书籍信息更新到数据库中，并提交事务。刷新表格显示所有书籍信息。若信息不完整或不合法，则弹出警告框提示用户。Python 代码如下：

```
    # 修改图书
    def update_book(self):
        bookid = self.bookid.text()
        bookname = self.bookname.text()
        author = self.author.text()
        booktype = self.type.text()
        prize1 = self.prize.text()
        press = self.press.text()
        abstract_2 = self.abstract_2.text()
        bookshelf = self.bookshelf.text()
```

```
        try:
            totalnum = int(self.totalnum.text())
            num = int(self.num.text())
            if prize1 != '':
                prize = float(prize1)
                if bookid and bookname and author and booktype and prize and press and totalnum and
num and abstract_2 and bookshelf:
                    sql = 'UPDATE books ' \
                          'SET 书号="%s",书名="%s",作者="%s",类型="%s",价格="%.2f",出版社="%s",' \
                          '摘要="%s",馆藏册数="%d",在馆册数="%d",书架号="%s",被借次数="%d" ' \
                          'WHERE 书号="%s"' % (
                    bookid, bookname, author, booktype, prize, press, abstract_2, num, totalnum, boo
kshelf, 0, bookid)
                    cursor.execute(sql)
                    conn.commit()
                    self.show_all_books()
                else:
                    QMessageBox.warning(self.mainwindow, "警告", "请补全书目信息！", QMessageBox.Yes)
            else:
                QMessageBox.warning(self.mainwindow, "警告", "请补全书目信息！", QMessageBox.Yes)
        except (pymysql.err.IntegrityError, ValueError):
            QMessageBox.warning(self.mainwindow, "警告", "书目信息错误！", QMessageBox.Yes)
```

6.4.4 借阅/归还审批功能

此页面主要用于处理借阅/归还申请，用户可以查看所有待审批的申请信息，并进行批准或驳回操作。在批准过后，会触发数据库中的触发器，自发地修改对应信息。

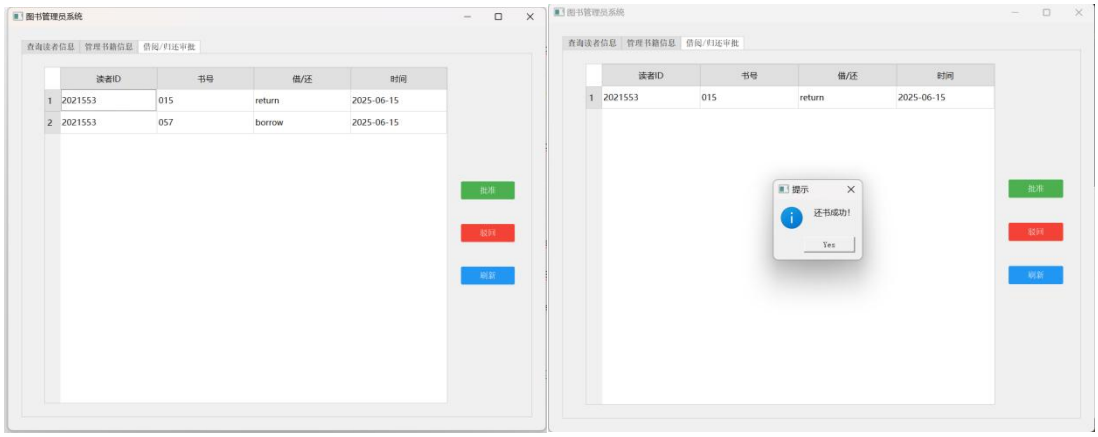


图 6.11 借阅/归还审批界面

显示申请记录函数从数据库的 item 表中查询所有待审批的借阅/归还申请信息，并将其显示在表格中。Python 代码如下：

```
    # 显示所有待审批信息
    def show_all_items(self):
        sql = 'SELECT * FROM item'
        res = cursor.execute(sql)
        items = cursor.fetchall()
        self.iteminfo.setRowCount(res)
        for i in range(res):
            item = items[i]
            self.iteminfo.setItem(i, 1, QTableWidgetItem(item[0]))
            self.iteminfo.setItem(i, 0, QTableWidgetItem(item[1]))
```

```
            self.iteminfo.setItem(i, 3, QTableWidgetItem(str(item[2])))
            self.iteminfo.setItem(i, 2, QTableWidgetItem(item[3]))
```

批准函数批准用户在 item 表格中选中的借阅/归还申请。检查用户是否选中了要批准的申请，获取选中申请的读者 ID、书号、申请类型和申请时间。若申请类型为还书，则检查是否逾期，若逾期则计算欠款并更新读者的欠款信息。执行 SQL 删除语句将该申请记录从数据库中删除，并提交事务。在删除时，会自动触发 5.3 中设计的借书与还书的触发器，自动地修改对应的关联数据与记录。Python 代码如下：

```
    # 批准借阅请求
    def pizhun(self):
        s = self.iteminfo.currentRow()
        if s < 0:
            QMessageBox.warning(self.mainwindow, "提示", "请先选择一项申请", QMessageBox.Yes)
            return
        readerid = self.iteminfo.item(s, 0).text()
        bookid = self.iteminfo.item(s, 1).text()
        selecttype = self.iteminfo.item(s, 2).text()
        itemtime_str = self.iteminfo.item(s, 3).text()
        if selecttype == 'return':
            # 检查是否逾期，若逾期则计算欠款
            sql = 'SELECT 借书时间 FROM borrow ' \
                  'WHERE ID=%s AND 书号=%s'
            cursor.execute(sql, (readerid, bookid))
            result = cursor.fetchone()
            if result:
                borrow_date = time.strptime(str(result[0]), "%Y-%m-%d")
                return_date = time.strptime(str(itemtime_str), "%Y-%m-%d")
                days = (time.mktime(return_date) - time.mktime(borrow_date)) // (60 * 60 * 24)
                cursor.execute('SELECT 读者类型 FROM readers '
                               'WHERE ID=%s', (readerid,))
                rtype = cursor.fetchone()[0]
                limits = {'研究生': 60, '本科生': 30, '教师': 90}
                allowed = limits.get(rtype, 30)
                overdue = int(days - allowed)
                if overdue > 0:
                    cursor.execute('UPDATE readers '
                                   'SET 欠款 = 欠款 + %s '
                                   'WHERE ID=%s', (overdue, readerid))
                    conn.commit()
                    QMessageBox.warning(self.mainwindow, "注意", f"超期 {overdue} 天，已计入欠款
", QMessageBox.Yes)
                else:
                    QMessageBox.information(self.mainwindow, "提示", "还书成功！", QMessageBox.Yes)
        # 统一删除申请记录（触发器自动处理）
        cursor.execute("DELETE FROM item "
                       "WHERE bookid=%s AND ID=%s AND time=%s AND type=%s",
                       (bookid, readerid, itemtime_str, selecttype))
        conn.commit()
        self.show_all_items()
```

驳回函数则直接删除对应申请即可，无需其他操作，Python 代码如下：

```
    # 驳回借阅请求
    def bohui(self):
        s = self.iteminfo.currentRow()
        bookid = self.iteminfo.item(s, 1).text()
```

```
        sql = 'DELETE FROM item ' \
              'WHERE bookid="%s"' % (bookid)
        cursor.execute(sql)
        conn.commit()
```

6.5 系统管理员系统

6.5.1 系统管理员系统概述

主要用于管理图书馆系统中的读者信息、图书管理员信息以及查看登录记录。该界面包含三个界面，分别实现不同的管理功能：

读者信息录入：可以添加、修改、删除读者信息，还能根据读者 ID 进行查询。

图书管理员信息录入：可以添加、修改、删除图书管理员信息。

登录记录查看：可以查看所有读者的登录记录，也能根据读者 ID 进行筛选查询。

6.5.2 读者信息录入功能

该页面主要用于对读者信息进行管理，包括添加、修改、删除读者信息，以及根据读者 ID 进行查询。同时，由于触发器，能够自动根据读者类型设置可借册数。在删除读者信息时会检测是否有借阅记录或借还申请，如果有会弹出提示无法删除。对于输入都有合法性检查。

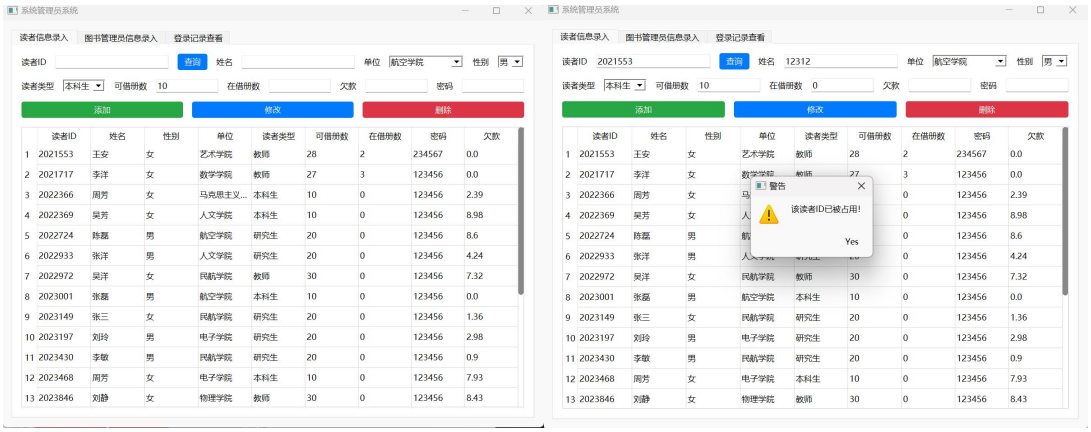


图 6.12 录入读者信息界面及添加提示演示

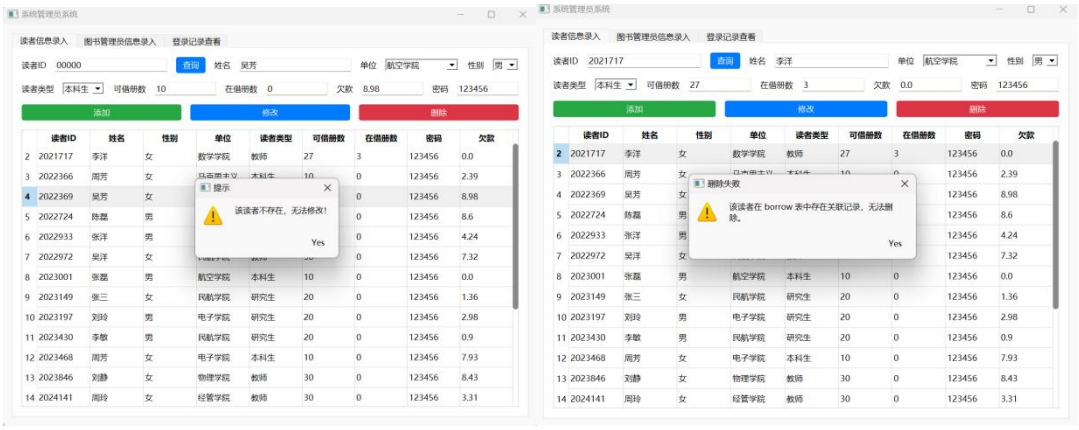


图 6.13 录入读者信息界面修改与删除提示演示

添加读者函数获取输入框和下拉框中的读者信息，执行查询语句，检查该读者 ID 是否已经存在。如果 ID 已存在，弹出警告框提示用户；否则，执行插入语句将读者信息插入到数据库中。提交数据库事务，弹出提示框告知用户添加成功。清空输入框，取消表格的选中状态。插入时会自动触发 5.3 中设计的触发器，无需手动设置可借册数。Python 代码如下：

```
# 添加读者
def addreader(self):
    id = self.readerid.text()
    name = self.readname.text()
    runit = self.unit.currentText()
    rsex = self.sex.currentText()
    rtype = self.readtype.currentText()
    num = int(self.number.text())
    rpassword = self.readerpassword.text()
    sql = 'SELECT * FROM readers WHERE ID="%s"' % id
    res = cursor.execute(sql)
    if res:
        QMessageBox.warning(self.mainwindow, "警告", "该读者 ID 已被占用！", QMessageBox.Yes)
    else:
        value = (id, name, rsex, runit, rtype, num, rpassword)
        sql = 'INSERT INTO readers' \
              '(ID, 姓名, 性别, 单位, 读者类型, 在借册数, password) ' \
              'VALUES ("%s","%s","%s","%s","%s",%d, "%s");' % value
        cursor.execute(sql)
        conn.commit()
        QMessageBox.warning(self.mainwindow, "提示", "添加成功！", QMessageBox.Yes)
        self.getreadersinfo()
        self.clearReaderInputs()
        self.readerinfos.clearSelection()
```

读者查询函数获取输入的读者 ID，检查是否为空。如果为空，弹出提示框要求用户输入 ID。执行查询语句，查找该读者信息。如果找到该读者，将其信息显示在输入框中，并在表格中选中该行；否则，弹出提示框告知用户未找到该读者信息，清空输入框，取消表格的选中状态。Python 代码如下：

```
# 读者 ID 查询
def searchReaderByID(self):
    id = self.readerid.text().strip()
    if not id:
        QMessageBox.warning(self.mainwindow, "提示", "请输入读者 ID 进行查询", QMessageBox.Yes)
        return
    sql = 'SELECT * FROM readers ' \
          'WHERE ID="%s"' % id
    cursor.execute(sql)
    reader = cursor.fetchone()
    if reader:
        self.readname.setText(reader[1])
        self.sex.setCurrentText(reader[2])
        self.unit.setCurrentText(reader[3])
        self.readtype.setCurrentText(reader[4])
        self.totalnumber.setText(str(reader[5]))
        self.number.setText(str(reader[6]))
        self.readerpassword.setText(reader[7])
        self.money.setText(str(reader[8]))
        # 定位表格选中行
        for i in range(self.readerinfos.rowCount()):
            if self.readerinfos.item(i, 0).text() == id:
```

```
                    self.readerinfos.selectRow(i)
                    break
        else:
            QMessageBox.information(self.mainwindow, "提示", "未找到该读者信息", QMessageBox.Yes)
            self.clearReaderInputs()
            self.readerinfos.clearSelection()
```

修改读者信息函数获取输入框和下拉框中的读者信息，检查该读者是否存在。如果读者不存在，弹出提示框告知用户无法修改；否则，执行更新语句将新的读者信息更新到数据库中。提交数据库事务，弹出提示框告知用户修改成功。Python 代码如下：

```
    # 修改读者信息
    def altereader(self):
        id = self.readerid.text()
        name = self.readname.text()
        runit = self.unit.currentText()
        rsex = self.sex.currentText()
        rtype = self.readtype.currentText()
        total = int(self.totalnumber.text())
        num = int(self.number.text())
        rmoney = float(self.money.text())
        rpassword = self.readerpassword.text()
        sql = 'SELECT * FROM readers ' \
              'WHERE ID="%s"' % id
        res = cursor.execute(sql)
        if res == 0:
            QMessageBox.warning(self.mainwindow, "提示", "该读者不存在，无法修改！", QMessageBox.Yes)
            return
        value = (name, rsex, runit, rtype, total, num, rmoney, rpassword, id)
        sql = 'UPDATE readers ' \
              'SET 姓名="%s",性别="%s",单位="%s",读者类型="%s",可借册数=%d,在借册数=%d,欠款
=%.2f,password="%s" ' \
              'WHERE ID="%s"' % value
        cursor.execute(sql)
        conn.commit()
        QMessageBox.warning(self.mainwindow, "提示", "修改成功！", QMessageBox.Yes)
        self.getreadersinfo()
        self.clearReaderInputs()
        self.readerinfos.clearSelection()
```

删除读者函数弹出确认对话框，询问用户是否确定删除。如果用户确认删除，获取当前输入的读者 ID。检查该读者是否在 borrow 和 item 表中存在关联记录。如果存在，弹出警告框告知用户无法删除。如果没有关联记录，执行删除语句，将该读者信息从数据库中删除。删除时会触发 5.3 中触发器，自动删除该 ID 对应的所有登录记录。Python 代码如下：

```
    # 删除读者信息
    def deletereader(self):
        res = QMessageBox.warning(self.mainwindow, "警告", "即将删除，确定？
", QMessageBox.Yes | QMessageBox.No, QMessageBox.No)
        if res == QMessageBox.Yes:
            id = self.readerid.text()
            # 外键依赖检查
            check_sqls = [
                ('borrow', 'SELECT * FROM borrow WHERE ID="%s" LIMIT 1' % id),
                ('item', 'SELECT * FROM item WHERE ID="%s" LIMIT 1' % id),
```

```
            ]
            for name, sql in check_sqls:
                cursor.execute(sql)
                if cursor.fetchone():
                    QMessageBox.warning(self.mainwindow, "删除失败", "该读者在 %s 表中存在关联记录，无法
删除。" % name, QMessageBox.Yes)
                    return
            try:
                sql = 'DELETE FROM readers ' \
                      'WHERE ID="%s"' % id
                cursor.execute(sql)
                conn.commit()
                QMessageBox.information(self.mainwindow, "提示", "删除成功！", QMessageBox.Yes)
                self.getreadersinfo()
                self.clearReaderInputs()
                self.readerinfos.clearSelection()
            except Exception as e:
                QMessageBox.critical(None, "错误", "删除失败：%s" % str(e), QMessageBox.Yes)
```

6.5.3 管理员信息录入功能

该页面用于对管理员信息进行管理，包括添加、修改、删除管理员信息，同时还能够修改权限。同样的也有合法性检查。

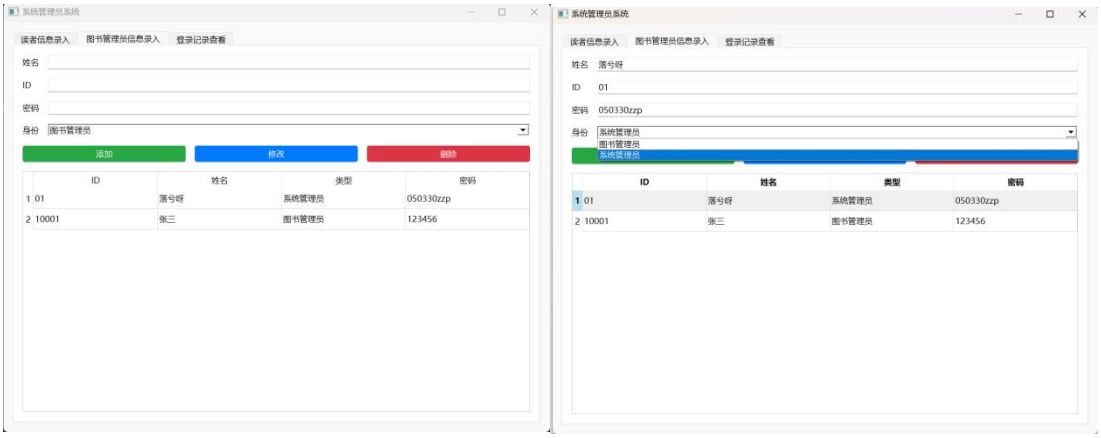


图 6.14 管理员信息录入界面

此处的添加、修改、删除操作与读者部分基本相同且更加简单，原理不再赘述，Python 代码如下：

```
    # 获取图书管理员的信息
    def getadmininfo(self):
        sql = 'SELECT * FROM workers ' \
              'order by ID'
        res = cursor.execute(sql)
        admins = cursor.fetchall()
        self.admininfos.setRowCount(res)
        for i in range(res):
            admin = admins[i]
            for j in range(4):
                self.admininfos.setItem(i, j, QTableWidgetItem(str(admin[j])))

    # 增加管理员
    def addadmin(self):
        id = self.adminID.text()
```

```
        name = self.adminname.text()
        password = self.adminpassword.text()
        rtype = self.admintype.currentText()
        sql = 'SELECT * FROM workers WHERE ID="%s"' % id
        res = cursor.execute(sql)
        if res:
            QMessageBox.warning(self.mainwindow, "警告", "该 ID 已被占用！", QMessageBox.Yes)
        else:
            value = (id, name, rtype, password)
            sql = 'INSERT INTO workers(ID, 姓名, type, password) ' \
                  'VALUES ("%s","%s","%s","%s");' % value
            print(sql)
            cursor.execute(sql)
            conn.commit()
            QMessageBox.warning(self.mainwindow, "提示", "添加成功！", QMessageBox.Yes)
            self.getadmininfo()
            self.clearReaderInputs()
            self.readerinfos.clearSelection()

    def displayadmininfo(self):
        s = self.admininfos.currentRow()
        self.adminname.setText(self.admininfos.item(s, 1).text())
        self.adminID.setText(self.admininfos.item(s, 0).text())
        self.adminpassword.setText(self.admininfos.item(s, 3).text())
        self.admintype.setCurrentText(self.admininfos.item(s, 2).text())

    # 修改管理员
    def alteradmin(self):
        id = self.adminID.text()
        name = self.adminname.text()
        password = self.adminpassword.text()
        rtype = self.admintype.currentText()
        value = (name, rtype, password, id)
        sql = 'UPDATE workers SET 姓名="%s",type="%s",password="%s" ' \
              'WHERE ID="%s"' % value
        cursor.execute(sql)
        conn.commit()
        self.getadmininfo()
        QMessageBox.warning(self.mainwindow, "提示", "修改成功！", QMessageBox.Yes)
        self.getadmininfo()
        self.clearAdminInputs()
        self.admininfos.clearSelection()
    # 删除管理员
    def deleteadmin(self):
        res = QMessageBox.warning(self.mainwindow, "警告", "即将删除，确定？
", QMessageBox.Yes | QMessageBox.No, QMessageBox.No)
        if res == QMessageBox.Yes:
            id = self.adminID.text()
            sql = 'DELETE FROM workers WHERE ID="%s"' % id
            cursor.execute(sql)
            conn.commit()
            QMessageBox.warning(self.mainwindow, "提示", "删除成功！", QMessageBox.Yes)
            self.getadmininfo()
            self.clearReaderInputs()
            self.readerinfos.clearSelection()
```

6.5.4 登录记录查看功能

该页面用于查看读者的登录记录，支持根据读者 ID 进行筛选查询。显示读者基本信息与登录信息。

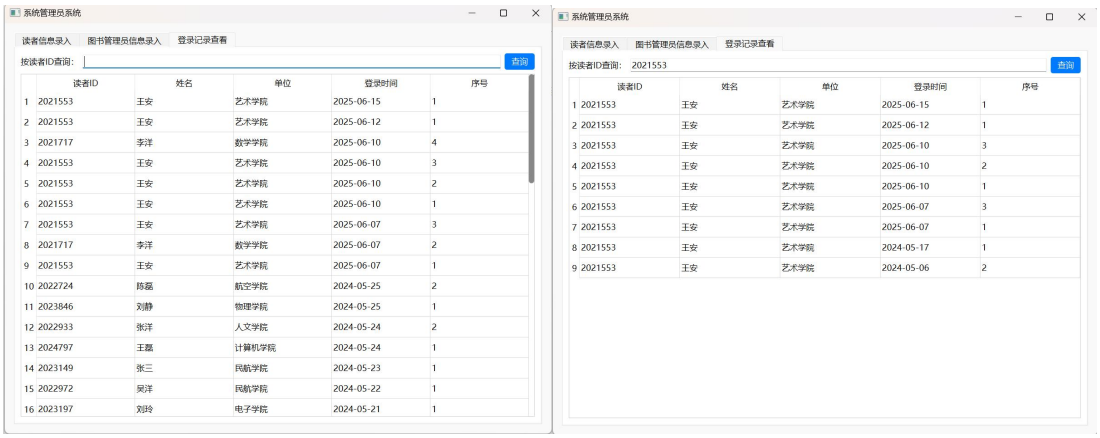


图 6.15 登录信息查看界面

登录记录查看函数从数据库中获取登录记录，并将其显示在登录记录表格中。如果输入了读者 ID，则只显示该读者的登录记录，若未输入，则显示全部记录。默认显示全部记录。Python 代码如下：

```
# 登录记录查看
def loadLoginRecords(self, filter_id=None):
    if filter_id:
        sql = 'SELECT L.ID, R.姓名, R.单位, L.time, L.number ' \
              'FROM loginrecord L, readers R ' \
              'WHERE L.ID = R.ID AND L.ID = "%s" ' \
              'ORDER BY L.time DESC, L.number DESC' % filter_id
    else:
        sql = 'SELECT L.ID, R.姓名, R.单位, L.time, L.number ' \
              'FROM loginrecord L, readers R ' \
              'WHERE L.ID = R.ID ' \
              'ORDER BY L.time DESC, L.number DESC'
    cursor.execute(sql)
    records = cursor.fetchall()
    self.loginrecordTable.setRowCount(len(records))
    for i, row in enumerate(records):
        for j in range(5):
            self.loginrecordTable.setItem(i, j, QTableWidgetItem(str(row[j])))

# 查询登录记录
def onSearchLoginRecord(self):
    id = self.searchID.text().strip()
    self.loadLoginRecords(filter_id=id if id else None)
```

7 系统运行与系统特色

7.1 系统运行

使用流程：用户打开系统后，首先进入登录界面，需输入用户名、密码，并选择用户类型（读者、图书管理员、系统管理员）。系统会根据用户选择的类型在相应的数据库表中验证用户名和密码。验证通过后，记录登录信息（针对读者），并打开对应类型的操作界面。界面采用多标签页设计，进入对应标签页后即可进行功能使用。系统通过执行 SQL 语句与 MySQL 数据库进行交互，实现数据的增删改查操作。

为检测系统是否按预期进行，进行了如下的测试：检测各个存储过程的功能，检测系统运行界面运行是否正常，检测数据库的增、删、查、改功能是否正常实施，检测对于用户的非法输入是否不会崩溃并且给出对应提示。

7.2 系统特色

（1）功能模块清晰，角色权限分明

系统按照角色划分为读者端、图书管理员端、系统管理员端，各司其职，互不干扰。每个角色的功能权限符合实际业务逻辑，如：读者只能申请借书/还书、查询信息、修改个人资料；图书管理员可处理借还申请、维护图书与读者信息；系统管理员专责账号与登录记录管理，保障系统安全。

（2）借还流程自动化，数据库逻辑完善

借书/还书采用申请+审批机制，流程规范；利用 MySQL 触发器实现审批后自动插入/删除借阅记录、更新库存和状态，减少前端逻辑压力；字段默认值、外键约束、枚举类型等机制确保数据合法性与一致性。

（3）多条件查询支持，信息检索灵活

图书信息查询支持多字段组合查询（书号、书名、作者、类别、出版社），读者查询、借阅记录查询等功能均支持精确匹配与筛选，提高信息定位效率，满足不同用户查询需求。

（4）自动化与便捷操作

在添加读者信息时，当管理员选择读者类型为本科生、研究生或教师时，系统会自动根据读者类型设置可借册数，减少了管理员的手动输入工作量，提高了操作效率。在进行添加、修改或删除操作后，系统会自动刷新相应的表格数据，确保界面显示的数据与数据库中的数据保持一致，方便管理员实时查看和管理数据。

8 总结

8.1 存在的不足

（1）百万条数据测试

由于时间问题以及技术问题，本系统并没有经过百万条数据的测试，一是暂时不知道如何获得百万条数据，存在技术问题（例如 Python 爬虫），二是对自己设计的系统没有自信能够支持百万条数据的测试，因为并未针对这个问题做专门的优化，这一点也是改进的一大重点方向。

（2）视图设计

视图一方面可以帮我们使用表的一部分而不是所有的表，另一方面也可以针对不同的用户制定不同的查询视图。在一开始的设计中，没有设计视图，想法是可以用 Python 语句来做到视图的作用，但经过实践过后发现这种方法没有使用视图方便。例如：对于不同页面里所看到的不同的书籍信息，完全可以使用视图来设计，这一点也是一个优化的方向。

（3）代码结构与可维护性问题

重复代码较多，在读者信息和书籍信息的查询、显示功能中，存在大量重复的表格数据填充代码。例如展示读者信息和书籍信息的函数中，表格行设置逻辑相似，可封装为通用函数。

同时数据库操作（如查询、插入、更新）直接在界面逻辑中实现，未分离数据访问层，导致代码耦合度高，维护困难。

（4）可视化界面设计一般

由于这是第一次学习并且使用前端来构建可视化界面，形成一个小的应用，对于可视化界面的创建与设计还很稚嫩，在这方面存在很多不足，这也是未来的一个学习与努力的方向。

8.2 心得体会

在本次图书管理系统的开发过程中，我深刻体会到了信息系统从设计到实现的完整流程，也进一步提升了我在数据库设计、图形界面开发和 Python 编程方面的实际动手能力。整个系统的开发虽然充满挑战，但收获也颇丰。

在课堂上学习的数据库范式、E - R 图建模、SQL 数据操作语言，在本项目中得到了实际应用。在本项目中，我一步步从头开始，体会到了数据库系统的设计过程，从需求分析到概念结构设计，从逻辑结构设计到物理结构部署，我不仅理解了数据库结构合理性的重要性，还掌握了实际业务中如何通过主键、外键、触发器等机制来保证数据一致性和完整性。尤其是触发器的使用，让我认识到数据库不只是存储工具，更能承载部分业务逻辑，减轻前端开发负担。原本抽象的知识，甚至都很难全部记住，经过实际上手开发过后，每一个步骤都变得清晰明了，每一步都能感受到其存在意义。

同时这也是我第一次接触前端开发设计，从一开始的迷茫无助，再到随着网上的各种资料与 AI 的帮助一步步入门，这是一个很好的个人技能提升的过程。虽然最后的成果 UI 设计并没有多么出色，但也是我一开始想象不到的程度。

开发过程中遇到了不少实际问题，如：数据库外键冲突导致的删除失败；多次借书申请导致的重复数据；图形界面控件信号与槽函数未匹配造成功能失效；非法数据的处理与提示等。通过查阅资料、调试代码与反复测试，我逐步学会了定位问题、分析原因、查找资料并解决问题的完整流程，也增强了我的独立思考与解决问题的能力。

在后期测试中，我逐渐意识到系统中存在一些潜在隐患，如密码明文存储、表格信息无法分页、界面操作无反馈等问题。虽然这些功能不在基本要求之内，但它们暴露了系统安全性与用户友好性的薄弱环节。这让我认识到，一个优秀的数据库系统不仅要“能用”，还要“好用”“安全”，这将成为我今后编程与设计中持续关注的重要方向。

本次图书管理系统项目是一次宝贵的实战经历，它不仅检验了我的所学知识，也提升了我的综合实践能力。在未来的学习与工作中，我将继续巩固基础，不断拓展技术广度，同时注重系统的可用性、健壮性与用户体验设计，努力成为一名既懂技术又注重质量的开发者。

9 参考文献

[1] 王珊, 杜小勇, 陈红. 数据库系统概论（第 6 版）[M]. 北京：高等教育出版社，2023.
[2] 陈文. Python 编程从入门到实践（第 2 版）[M]. 北京：人民邮电出版社，2021.
[3] PyMySQL 官方文档. https://pymysql.readthedocs.io/en/latest/
[4] Qt for Python 官方文档. https://doc.qt.io/qtforpython/

10 项目链接

Github 链接：https://github.com/jiuyealxy/Library_management_system_databaseCourseDesign