

# 图书管理系统-数据库原理实验报告

162110132, 谢骏鑫, 17849890562

计算机科学与技术

Email: xiejunxin0126@foxmail.com

**摘要:** 随着近几年科技水平的飞速发展，社会整体信息化程度的大幅提高，借助信息工具来提升人们的信息管理效率，降低管理成本已经成为常态。本次实验最终所完成的图书管理系统也是基于此出发，通过网站 Web 前端开发技术和数据库管理系统相结合，开发出一个易于使用的图书管理系统，方便图书管理人员管理图书，也利于普通用户查看图书，并提供了图书借阅功能，进一步提升了图书管理系统的易用性。

**关键词:** OpenGauss; Python; Django; HTML; JavaScript

**ABSTRACT:** With the rapid development of science and technology in recent years, the overall degree of information technology has been greatly improved, and it has become normal to use information tools to improve people's information management efficiency and reduce management costs. The library management system completed in this experiment is also based on this starting point, through the combination of Web front-end development technology and database management system, an easy-to-use library management system has been developed, which is convenient for library management personnel to manage books, but also conducive to ordinary users to view books, and provides book borrowing function, further improving the ease of use of the library management system.

**KEYWORDS:** OpenGauss; Python; Django; HTML; JavaScript

## 1 实验内容基本介绍

### 1.1 实验内容介绍

本次数据库实验一共完成了六个实验内容，其中实验一到实验五为基本实验内容，主要通过自己编写 SQL 语句体验 SQL 定义功能、数据插入、数据查询、数据修改、删除、视图的操作以及库函数、授权控制相关数据库提供的功能，在第六个实验综合实验中自己独立设计并完成了一个简单的小型图书管理系统，极大地提升了自己的代码编写能力，也增强了自己对数据库原理的认识和理解。

### 1.2 技术工具介绍

本次实验使用主要的编程语言为 Python，在进行前端网站开发时使用了 HTML 语言和 Javascript 语言，数据库系统采用要求的华为数据库 OpenGauss，在开发第六个实验要求的小型信息管理系统时使用的前端可视化框架为 Django，也是 Python 语言中用于开发网站的主流框架。对于编程语言和 OpenGauss 数据库不过多介绍，重点说明一下 Django 框架的开发原理。

Django 是一个高级的 Python 语言编写的 Web 应用程序开发框架，使用 Django 可以快速开发出一个网站应用，其采用的设计模式为模型-视图-模板（MVT）模式，其包含很多用于前端网站开发的强大特性和模块，是一个特别适合快速开发复杂的数据库驱动的网站开发框架。这里重点说明一下其 MVT 设计模式和底

层数据库直接的关系。

MVT 设计模式中的 M 表示 Model，即模型，在 Django 中模型用于定义数据库结构的对象，使用 Python 类来表示数据库中的表，每一个模型类都对应于数据库中的一张表，而类中的属性则对应数据表中的列。在模型中定义的数据字段（比如在代码中所见的 CharField、IntegerField 等）和行为方法（如数据验证方法）共同构成了数据的逻辑结构和业务规则。Django 通过提供的模型层来与数据库进行交互，通过使用 ORM（对象关系映射，Object-Relational Mapping）技术允许开发者可以以面向对象的方式来操作数据库，通过模型 Model 提供的接口直接获取和操作数据库中的数据，从而避免了编写大量繁杂的 SQL 语句工作。

MVT 中的 V 表示表示 View，即视图，视图是 Django 中负责处理 HTTP 请求并返回响应的组件，它是在部署前端网站中处理业务逻辑的主要部分，视图本质上就是一个个需要自己编写的函数，也是自己所设计的前端网站的核心逻辑代码部分，这些函数接收 Web 请求，根据请求内容执行响应的逻辑处理（比如从模型中获取数据），然后决定要展示给客户的响应内容。

MVT 中的 T 表示 Template，即模板，本质上就是 Django 中用来呈现视图处理后数据的 HTML 文件，HTML 文件定义了页面的布局和结构，并允许插入动态数据。模板系统使用特定的语法（如 Django 模板语言的变量{{ }}和标签{% %}）来指示数据如何插入到静态 HTML 中，其确保了表现层（UI）与业务逻辑和数据的解耦。

最后再简单介绍一下 JavaScript 语言，这是一个广泛用于网站开发的编程语言，其主要作用就是用于实现网页的动态功能。通过给一个 HTML 静态网页界面添加对应的使用 JavaScript 语言编写的脚本，可以处理用户事件，实现较好的用户交互效果。

## 2 基础实验内容报告

这一部分主要介绍实验一到实验五的完成情况和报告说明，由于第六个综合实验所要求实现的小型信息系统使用 Django 开发，基于其设计模式进行开发使得对数据库的所有相关操作全都转化为了对 Django 中模型类对象的接口调用，因而这里再说明一下实验一到实验五的完成内容，报告其中 SQL 语句的使用和设计情况。

### 2.1 实验一：SQL 定义功能、数据插入

#### 2.1.1 实验内容

1. 建立教学数据库的三个基本表：

S(Sno, Sname, Ssex, Sbirthdate, Sdept)——学生（学号，姓名，性别，出生日期，主修专业）；

SC(Sno, Cno, Grade, Semester, Teachingclass)——选课（学号，课程号，成绩）；

C(Cno, Cname, Cpno, Ccredit)——课程（课程号，课程名，先行课，学分）；

2. DROP TABLE、ALTER TABLE、CREATE INDEX、DROP INDEX 及 INSERT 语句输入数据；

#### 2.1.2 实验结果

以所写的代码进行展示，代码如下代码 2.1 所示。

---

```
import psycopg2
```

```
# 创建连接对象
```

```

conn = psycopg2.connect(database="db_tpcc", user="joe", password="Bigdata@123", host="121.36.
37.200", port=26000)
cur = conn.cursor() # 创建指针对象

# 创建表

cur.execute("CREATE TABLE S"
           "(Sno VARCHAR(8) PRIMARY KEY,"
            "Sname VARCHAR(20),"
            "Ssex VARCHAR(6),"
            "Sbirthdate Date,"
            "Sdept VARCHAR(40)"
           ") ;")

cur.execute("CREATE TABLE C"
           "(Cno VARCHAR(8) PRIMARY KEY,"
            "Cname VARCHAR(40) NOT NULL,"
            "Cpno VARCHAR(8),"
            "Ccredit SMALLINT,"
            "FOREIGN KEY(Cpno) REFERENCES C(Cno)"
           ") ;")

cur.execute("CREATE TABLE SC"
           "(Sno VARCHAR(8),"
            "Cno VARCHAR(5),"
            "Grade SMALLINT,"
            "Semester VARCHAR(10),"
            "Teachingclass VARCHAR(10),"
            "PRIMARY KEY(Sno,Cno),"
            "FOREIGN KEY(Sno) REFERENCES S(Sno),"
            "FOREIGN KEY(Cno) REFERENCES C(Cno)"
           ") ;")

# 建立索引

cur.execute("CREATE UNIQUE INDEX Idx_StuSname ON S(Sname);")
cur.execute("CREATE UNIQUE INDEX Idx_CouCname ON C(Cname);")
cur.execute("CREATE UNIQUE INDEX IdxSCCno ON SC(Sno ASC,Cno DESC);")

# 删除索引

cur.execute("DROP INDEX Idx_StuSname;")
```

```

cur.execute("DROP INDEX Idx_CouCname;")
cur.execute("DROP INDEX IdxSCCno;")

# 插入数据 为实验一到实验五服务

# 插入 s 表

cur.execute("INSERT INTO S VALUES('20180001', '李勇', '男', '2000-3-8', '信息安全');")
cur.execute("INSERT INTO S VALUES('20180002', '刘晨', '女', '1999-9-1', '计算机科学与技术');")
cur.execute("INSERT INTO S VALUES('20180003', '王敏', '女', '2001-8-1', '计算机科学与技术');")
cur.execute("INSERT INTO S VALUES('20180004', '张立', '男', '2000-1-8', '计算机科学与技术');")
cur.execute("INSERT INTO S VALUES('20180005', '陈新奇', '男', '2001-11-1', '信息管理与信息系统');")
cur.execute("INSERT INTO S VALUES('20180006', '赵明', '男', '2000-6-12', '数据科学与大数据技术');")
cur.execute("INSERT INTO S VALUES('20180007', '王佳佳', '女', '2001-12-7', '数据科学与大数据技术');")

# 插入 c 表

cur.execute("INSERT INTO C VALUES('81001', '程序设计基础与 C 语言', NULL, 4);")
cur.execute("INSERT INTO C VALUES('81002', '数据结构', '81001', 4);")
cur.execute("INSERT INTO C VALUES('81003', '数据库系统概论', '81002', 4);")
cur.execute("INSERT INTO C VALUES('81004', '信息系统概论', '81003', 4);")
cur.execute("INSERT INTO C VALUES('81005', '操作系统', '81001', 4);")
cur.execute("INSERT INTO C VALUES('81006', 'Python 语言', '81002', 3);")
cur.execute("INSERT INTO C VALUES('81007', '离散数学', NULL, 4);")
cur.execute("INSERT INTO C VALUES('81008', '大数据技术概论', '81003', 4);")

# 插入 sc 表

cur.execute("INSERT INTO SC VALUES('20180001', '81001', 85, '20192', '81001-01');");
cur.execute("INSERT INTO SC VALUES('20180001', '81002', 96, '20201', '81002-01');");
cur.execute("INSERT INTO SC VALUES('20180001', '81003', 87, '20202', '81003-01');");
cur.execute("INSERT INTO SC VALUES('20180002', '81001', 80, '20192', '81001-02');");
cur.execute("INSERT INTO SC VALUES('20180002', '81002', 98, '20201', '81002-01');");
cur.execute("INSERT INTO SC VALUES('20180002', '81003', 71, '20202', '81003-02');");
cur.execute("INSERT INTO SC VALUES('20180003', '81001', 81, '20192', '81001-01');");
cur.execute("INSERT INTO SC VALUES('20180003', '81002', 76, '20201', '81002-02');");
cur.execute("INSERT INTO SC VALUES('20180004', '81001', 56, '20192', '81001-02');");
cur.execute("INSERT INTO SC VALUES('20180004', '81002', 97, '20201', '81002-02');");
cur.execute("INSERT INTO SC VALUES('20180005', '81003', 68, '20202', '81003-01');")

conn.commit()

```

---

```
# 获取结果
# cur.execute('SELECT * FROM student')
# results = cur.fetchall()
# print(results)
```

---

代码 2.1 基础实验一代码

## 2.2 实验二：数据查询

### 2.2.1 实验内容

1. 查询选修 1 号课程的学生学号与姓名；
2. 查询选修课程名为数据结构的学生学号与姓名；
3. 查询不选 1 号课程的学生学号与姓名；
4. 查询学习全部课程学生姓名；
5. 查询所有学生除了选修 1 号课程外所有成绩均及格的学生的学号和平均成绩，其结果按平均成绩的降序排列；
6. 查询选修数据库原理成绩第 2 名的学生姓名；
7. 查询所有 3 个学分课程中有 3 门以上（含 3 门）课程获 80 分以上（含 80 分）的学生的姓名；
8. 查询选课门数唯一的学生的学号；
9. SELECT 语句中各种查询条件的实验；

### 2.2.2 实验结果

以所写的代码进行展示，代码如下代码 2.2 所示。

---

```
import psycopg2

conn = psycopg2.connect(database="db_tpcc", user="joe", password="Bigdata@123", host="121.36.
37.200", port=26000)
cur = conn.cursor()

# 获取结果模板
# cur.execute('SELECT * FROM student')
# results = cur.fetchall()
# print(results)

# 查询 1
# 查询选修 1 号课程的学生学号与姓名
cur.execute("SELECT S.Sno,S.Sname FROM S,SC WHERE S.Sno = SC.Sno AND SC.Cno = '81001';")
results = cur.fetchall()
# print(results)
```

```

# 查询 2

# 查询选修课程名为数据结构的学生学号与姓名

cur.execute("SELECT S.Sno,S.Sname FROM S,C,SC WHERE S.Sno = SC.Sno AND C.Cno = SC.Cno AND C.Cna
me = '数据结构';")
results = cur.fetchall()

# print(results)

# 查询 3

# 查询不选 1 号课程的学生学号与姓名

cur.execute("SELECT S.Sno,S.Sname FROM S,SC WHERE S.Sno = SC.Sno AND SC.Cno NOT LIKE '81001';
")
results = cur.fetchall()

# print(results)

# 查询 4

# 查询学习全部课程学生姓名

cur.execute('''
    SELECT Sname FROM S WHERE NOT EXISTS
        (SELECT * FROM C WHERE NOT EXISTS
            (SELECT * FROM SC WHERE Sno = S.Sno AND Cno = C.Cno));
        ''')
results = cur.fetchall()

# print(results)

# 查询 5

# 查询所有学生除了选修 1 号课程外所有成绩均及格的学生的学号和平均成绩 其结果按平均成绩的降序排列

cur.execute('''
    SELECT SCX.Sno,AVG(SCX.Grade) FROM SC SCX
    WHERE 59 < all(SELECT SCY.Grade FROM SC SCY
    WHERE SCY.Sno = SCX.Sno AND SCY.Cno != '81001')
    AND SCX.Cno != '1'
    GROUP BY SCX.Sno
    ORDER BY AVG(SCX.Grade) DESC;
        ''')
results = cur.fetchall()

# print(results)

```

```

# 查询 6

# 查询选修数据库原理成绩第 2 名的学生姓名

# 所建表中无数据库原理这一课程 用数据库系统概论课程代替

cur.execute('''

    SELECT Sname FROM S

    WHERE Sno IN (

        SELECT Sno FROM SC,C

        WHERE C.Cname = '数据库系统概论' AND SC.Cno = C.Cno

        ORDER BY Grade DESC

        LIMIT 1 OFFSET 1);

    ''')

results = cur.fetchall()

# print(results)

# 查询 7

# 查询所有 3 个学分课程中有 3 门以上(含 3 门)课程获 80 分以上(含 80 分)的学生的姓名

cur.execute('''

    SELECT Sname FROM S

    WHERE Sno IN (

        SELECT Sno FROM SC,C

        WHERE C.Cno = SC.Cno AND C.Ccredit = 3 AND SC.Grade >= 80

        GROUP BY Sno

        HAVING COUNT(Grade) >=3);

    ''')

results = cur.fetchall()

# print(results)

# 查询 8

# 查询选课门数唯一的学生的学号

cur.execute('''

    SELECT SCX.Sno FROM SC SCX

    GROUP BY SCX.Sno

    HAVING COUNT(*) NOT IN (

        SELECT COUNT(*) FROM SC SCY

        WHERE SCY.Sno != SCX.Sno

        GROUP BY SCY.Sno);

    ''')

results = cur.fetchall()

# print(results)

```

```

# 查询 9

# 自行 SELECT test

# e.g. 查询选修了 81002 号课程的学生的学号及其成绩, 查询结果按分数的降序排列

cur.execute('''
    SELECT Sno,Grade
    FROM SC
    WHERE Cno = '81002'
    ORDER BY Grade DESC;
    ''')

results = cur.fetchall()

# print(results)

# e.g. 查询平均成绩大于或等于 60 分的学生学号和平均成绩

cur.execute('''
    SELECT Sno,AVG(Grade)
    FROM SC
    GROUP BY Sno
    HAVING AVG(Grade) >= 60;
    ''')

results = cur.fetchall()

# print(results)

# e.g. 使用 IN 谓词查询选修了课程名为“数据库系统概论”的学生的学号和姓名

cur.execute('''
    SELECT Sno,Sname FROM S
    WHERE Sno IN (
        SELECT Sno FROM SC
        WHERE Cno IN (
            SELECT Cno FROM C
            WHERE Cname = '数据库系统概论'));
    ''')

results = cur.fetchall()

# print(results)

# e.g. 查询没有选修 81002 号课程的学生姓名

cur.execute('''
    SELECT Sname FROM S
    ''')

```

```

        WHERE NOT EXISTS (
            SELECT * FROM SC
            WHERE Sno = S.Sno AND Cno = '81002');

        '')
    results = cur.fetchall()
    # print(results)

# e.g. 查询至少选修了学生 20180002 选修的全部课程的学生的学号
cur.execute('''
    SELECT Sno FROM S
    WHERE NOT EXISTS (
        SELECT * FROM SC SCX
        WHERE SCX.Sno = '20180002' AND
        NOT EXISTS (
            SELECT * FROM SC SCY
            WHERE SCY.Sno = S.Sno AND SCY.Cno = SCX.Cno));
    ''')
results = cur.fetchall()
# print(results)

```

代码 2.2 基础实验二代码

### 2.3 实验三：数据修改、删除

#### 2.3.1 实验内容

1. 把 1 号课程的非空成绩提高 10%；
2. 在 SC 表中删除课程名为数据结构的成绩的元组；
3. 在 S 和 SC 表中删除学号为 202415122 的所有数据；

#### 2.3.2 实验结果

以所写的代码进行展示，代码如下代码 2.3 所示。

```

import psycopg2

conn = psycopg2.connect(database="db_tpcc", user="joe", password="Bigdata@123", host="121.36.3
7.200", port=26000)
cur = conn.cursor()

# 修改 1
# 把 1 号课程的非空成绩提高 10%
cur.execute("SELECT * FROM SC WHERE Cno = '81001';")

```

```
cmd1 = """
UPDATE SC
SET Grade = Grade * 1.1
WHERE Grade IS NOT NULL AND Cno = '81001';
"""

# cur.execute(cmd1)

cur.execute("SELECT * FROM SC WHERE Cno = '81001';")
results = cur.fetchall()
# print(results)

# 修改 2
# 在 SC 表中删除课程名为数据结构的成绩的元组
cmd2 = """
DELETE FROM SC
WHERE Cno IN (SELECT Cno FROM C WHERE Cname = '数据结构');
"""

# cur.execute(cmd2)

cur.execute("SELECT * FROM SC;")
results = cur.fetchall()
# print(results)

# 修改 3
# 在 S 和 SC 表中删除学号为 202415122 的所有数据
# 所建表无学号为 202415122 用 20180007 代替
cmd3 = """
DELETE FROM SC WHERE Sno = '20180007';
DELETE FROM S WHERE Sno = '20180007';
"""

cur.execute(cmd3)
cur.execute("SELECT * FROM S;")
results = cur.fetchall()
print(results)
```

---

代码 2.3 基础实验三代码

## 2.4 实验四：视图的操作

### 2.4.1 实验内容

1. 建立男学生的视图，属性包括学号、姓名、选修课程名和成绩；
2. 在男学生视图中查询平均成绩大于 80 分的学生学号与姓名；

### 2.4.2 实验结果

以所写的代码进行展示，代码如下代码 2.4 所示。

```
import psycopg2

conn = psycopg2.connect(database="db_tpcc", user="joe", password="Bigdata@123", host="121.36.3
7.200", port=26000)
cur = conn.cursor()

# 视图 1
# 建立男学生的视图 属性包括学号 姓名 选修课程名和成绩
cur.execute('''
CREATE VIEW Boys_view AS
SELECT S.Sno,S.Sname,C.Cname,SC.Grade
FROM SC,S,C
WHERE S.Ssex = '男' AND S.Sno = SC.Sno AND C.Cno = SC.Cno;
'''')

# 视图 2
# 在男学生视图中查询平均成绩大于 80 分的学生学号与姓名
cur.execute('''
SELECT Sno,Sname FROM Boys_view
GROUP BY Sno,Sname
HAVING AVG(Grade) > 80;
''')

results = cur.fetchall()
print(results)
```

代码 2.4 基础实验四代码

### 2.5 实验五：库函数，授权控制

#### 2.5.1 实验内容

1. 计算每个学生有成绩的课程门数、平均成绩；
2. 使用 GRANT 语句，把对基本表 S、SC、C 的使用权限授给其它用户；
3. 实验完成后，撤消建立的基本表和视图；

#### 2.5.2 实验结果

以所写的代码进行展示，代码如下代码 2.5 所示。

```
import psycopg2
```

```

conn = psycopg2.connect(database="db_tpcc", user="joe", password="Bigdata@123", host="121.36.3
7.200", port=26000)
cur = conn.cursor()

# 计算每个学生有成绩的课程门数 平均成绩
cur.execute('''
    SELECT Sno,COUNT(Cno),AVG(Grade)
    FROM SC
    WHERE Grade IS NOT NULL
    GROUP BY Sno;
    ''')
results = cur.fetchall()
print(results)

# 使用 GRANT 语句 把对基本表 s sc c 的使用权限授给其它用户
cur.execute('''
    GRANT ALL PRIVILEGES
    ON TABLE S
    TO PUBLIC;
    ''')

cur.execute('''
    GRANT ALL PRIVILEGES
    ON TABLE C
    TO PUBLIC;
    ''')

cur.execute('''
    GRANT ALL PRIVILEGES
    ON TABLE SC
    TO PUBLIC;
    ''')

# 实验完成后 撤消建立的基本表和视图
cur.execute('''
    DROP VIEW Boys_view;
    DROP TABLE S,C,SC;
    ''')

```

...)

#### 代码 2.5 基础实验五代码

### 2.6 基础实验总结

在完成了一系列涵盖 SQL 基础操作的基本实验之后，我获得了对关系型数据库系统深入且实践性的理解。对于数据库的定义功能，即表结构的创建，可以通过 CREATE TABLE 语句来定义表结构，包括列名、数据类型、约束条件等；对于数据插入功能，我掌握了如何使用 INSERT INTO 语句插入数据，同时也注意到了数据格式与数据完整性约束的匹配问题，这些都与理论课上老师所讲述的内容相对应起来；对于最关键也是最为困难的数据查询部分，这也是数据库的核心功能，我学会了通过 SELECT 语句结合不同的子句（比如 WHERE、ORDER BY、GROUP BY? HAVING、EXIST 子句等），以及运用不同的查询方式（连接查询、嵌套查询、基于派生表的查询等）去灵活检索到所需要的信息，同时也使用了聚合函数（SUM、AVG、COUNT 等）进行数据分析；对于数据修改和数据删除，我学会了使用 UPDATE 语句修改表中记录，以及使用 DELETE 语句删除表中记录；最后，我也通过创建、使用和删除视图（VIEW）认识到其在简化复杂查询、保护数据安全以及提供定制化数据视角方面的价值，同时对 GRANT 语句的使用也进一步认识到应该如何给不同用户或角色分配读、写、执行等权限来确保数据访问的合规性和安全性。

综上所述，通过完成这五个基本实验，我对于数据库系统理论的指示得到了进一步的认识和理解，通过亲身动手实践也进一步熟悉了数据库 SQL 语句的操作，获得了一定的实践经验，这些对于我后续进行小型信息管理系统的开发都大有裨益。

## 3 数据库设计

这一部分介绍实验六所设计的图书管理系统的数据库设计部分，数据库设计按照课程所讲述的六个基本步骤进行，即需求分析、概念结构设计、逻辑结构设计、物理结构设计、数据库实施和数据库运行和维护。

需求分析阶段是整个数据库设计的基础，也是最困难和最耗费时间的一步，这一步主要是未来准确了解与分析用户的应用需求。作为“地基”的需求分析是否做得充分与准确，决定了在其上构建数据库“大厦”的速度与质量。

概念结构设计是整个数据库设计的关键，它通过对用户需求进行综合、归纳与抽象，形成一个独立于具体数据库管理系统的概念模型。

逻辑结构设计是按某种转换规则将上一步的概念结构设计转换为某个数据库管理系统所支持的数据模型，并对其进行优化。

物理结构设计是为了逻辑数据模型选取一个最适合应用环境的物理结构，包括存储结构和存取方法。

在数据库实施阶段，设计人员运用数据库管理系统提供的数据库语言及高级语言，根据逻辑结构设计和物理结构设计的结果创建数据库，编写与调试应用程序，组织数据入库并进行试运行。

最后数据库应用系统经过试运行后即可投入正式运行，在数据库系统运行过程中必须不断地对其进行评估、调整与修改。

### 3.1 需求分析阶段

需求分析简单地说就是分析用户的要求，是数据库设计的起点，这一部分从本图书管理系统的四个模块的基本需求出发进行分析。

本图书管理系统一共自顶向下设计了四个基本模块，分别是普通用户模块、管理员模块、图书馆模块

和借书模块，这四个模块相辅相成，共同组成了最终所设计出的图书管理系统。

### 3.1.1 普通用户模块

无论是普通用户模块，还是管理员模块，他们在模型设计上都是将主键设计为指向 Django 中的 User 模型对象，Django 中的 User 模型是 Django 自带的权限认证系统的核心组件，对应到数据库关系中就是一张表。使用 User 可以完成在网站开发中所涉及的与网站交互的人有关的权限访问设计。

普通用户在模块功能上，普通用户具有一系列自定义的属性用于描述自己的个人信息，同时具有注册和登录已使用本图书管理系统的功能，同时具有本管理系统的查询权限，普通用户可以查询此系统中已上架的所有图书相关信息，并可以利用提供的检索元件去检索到想要了解的图书。

### 3.1.2 管理员用户模块

管理员用户模块也是以 Django 中自带的认证系统的核心模型 User 为主键，并增加一个可以管理员自定义的名字字段加以区分和标识。

本图书管理系统的管理员用户模块是操作本系统的所有权限的一个集大成模块，管理员负责绝对意义上的图书管理，可以添加图书、修改图书数量、更改图书信息、查看用户信息、查询所有图书和用户的信息、与借书模块进行交互等功能，同时管理员用户可以访问 Django 所开发网站自带的 Admin 后台管理系统。

前面说过，Django 是一个强数据库驱动型的网站开发框架，并才有 MVT 的设计模型，Admin 后台是 Django 一个强大的内置功能，它允许开发者通过 Web 界面轻松地管理站点内所有部署的数据库内容，支持以可视化的操作实现对所有设计的表中数据的增删改查。

### 3.1.3 图书馆模块

图书馆模块是图书管理系统的中心模块，这个模块设计了图书的所有信息和属性，比如图书的名称、图书的作者、图书的类型、图书的出版社、图书的作者、图书的价格等属性，同时还需要维护每本图书的剩余数量，用于借书模块的管理。

功能上图书馆模块支持查看所有已上架图书的信息，可以分页展示所有图书的简介，提供检索功能以检索包含指定书名、指定作者以及指定出版社的图书信息。

### 3.1.4 借书模块

借书模块是在图书馆已有的基本信息管理功能的基础上额外提供的一个子功能模块，该模块通过维护图书馆书的库存数量来决定一本书可不可以进行外借，同时有借书就需要设计对应的还书功能，因而需要设计一张借书信息表，维护每一条的借书记录，记录借给哪位用户并标记此次借书是否已经归还。

功能上借书模块的权限是全权由管理员负责，管理员通过输入要外接的图书 ID 和需要借书的用户 ID 实现借书的功能，普通用户无法直接操作借书功能，如此便实现了权限的集中处理。

## 3.2 概念结构设计阶段

概念结构设计就是将需求分析得到的用户需求抽象为信息结构（即概念模型）的过程，它是整个数据库设计的关键，本图书管理系统使用 E-R 图进行概念结构的设计。

从需求分析得到的结果出发，本管理系统的概念结构设计如下两幅 E-R 图所示，图 3.2.1 为表示用户和图书管理功能之间的联系的 E-R 图，图 3.2.2 为表示用户注册与登录的 E-R 图。

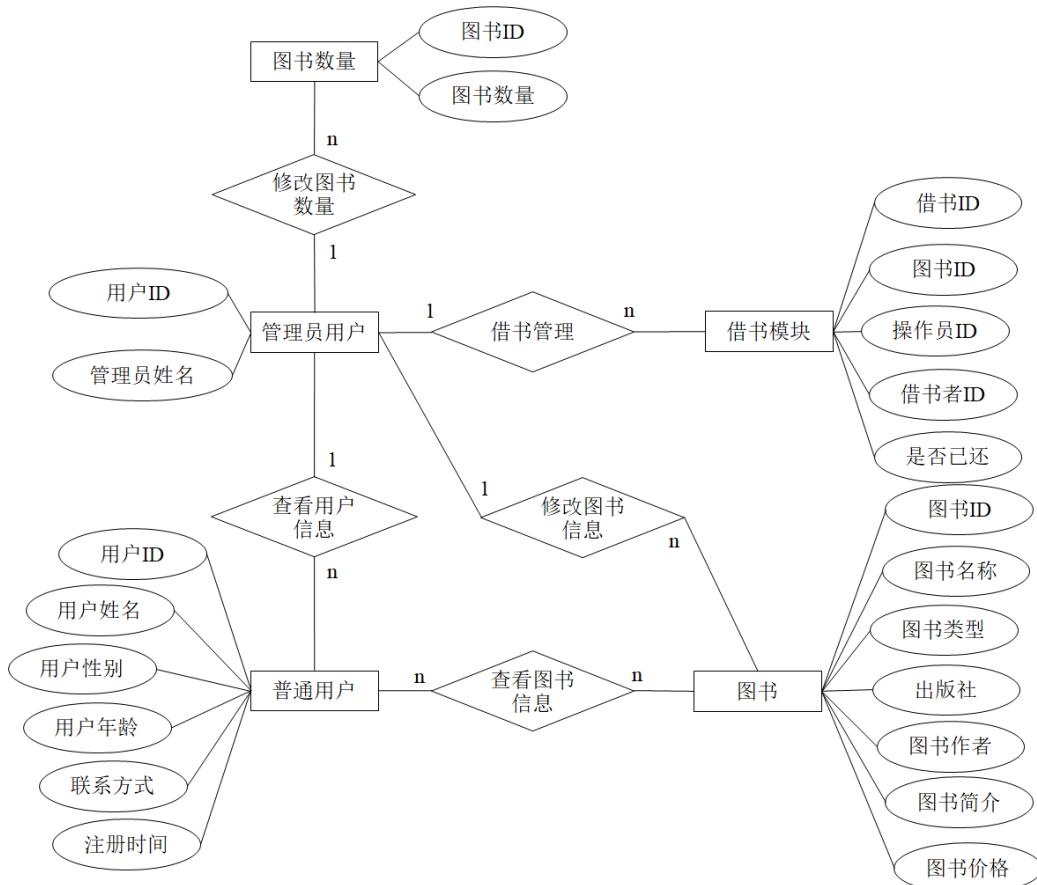


图 3.2.1 用户与图书管理的 E-R 图

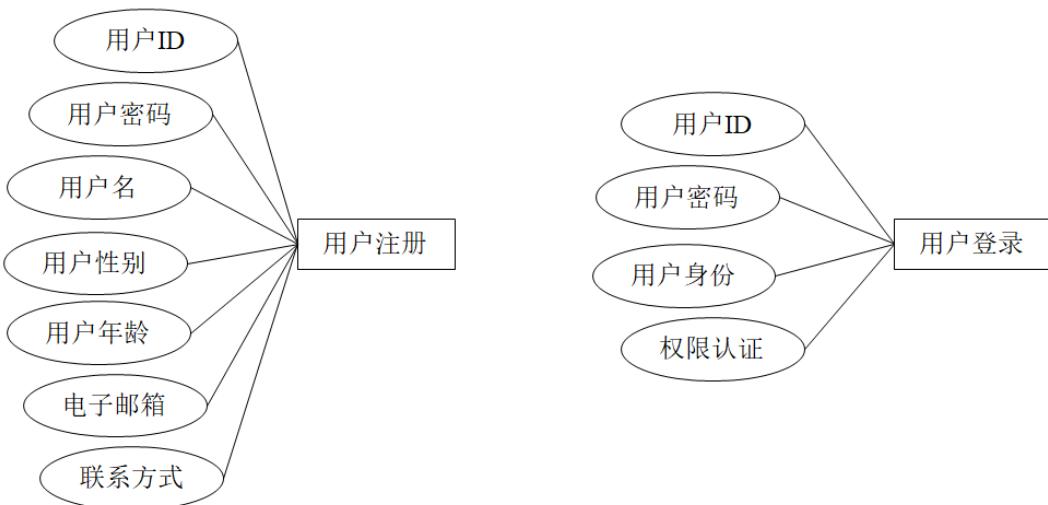


图 3.2.2 用户注册与登录的 E-R 图

### 3.3 逻辑结构设计阶段

上一步设计得到的概念结构是独立于任何一种数据模型的信息结构，而逻辑结构设计的任务就是把概念结构设计阶段得到的基本 E-R 图转换为与选用数据库管理系统产品支持的数据模型相符合的逻辑结构。

通过上述设计得到的 E-R 图可以得到如下的本图书管理系统的数据模型，即一张张在数据库中自己设

计实现的表结构。

### 3.3.1 Django 自带的认证系统的 User 表

如前所述，无论是普通用户还是管理员用户，它们的用户 ID 都是指向 Django 中自带的 User 模型的外键，从面向对象的角度去看，普通用户类和管理员用户类都是继承自 User 基类的子类。这样设计可以方便使用 Django 中的用户认证功能，很好地实现用户功能上权限的区分。以下为数据库中最终设计的 User 模型的属性信息，数据库中对表结构的描述如图 3.3.1 所示，具体表中各属性含义解释如表 3.3.1 所示。

```
db_bms=> \d auth_user
          Table "joe.auth_user"
   Column |      Type       | Modifiers
---+-----+-----+
 id | integer | not null default nextval('auth_user_id_seq'::regclass)
 password | character varying(128) | not null
 last_login | timestamp with time zone | not null
 is_superuser | boolean | not null
 username | character varying(150) | not null
 first_name | character varying(150) |
 last_name | character varying(150) |
 email | character varying(254) | not null
 is_staff | boolean | not null
 is_active | boolean | not null
 date_joined | timestamp with time zone | not null
Indexes:
 "auth_user_pkey" PRIMARY KEY, btree (id) TABLESPACE pg_default
 "auth_user_username_key" UNIQUE CONSTRAINT, btree (username) TABLESPACE pg_default
 "auth_user_username_6b21ab7c_like" btree (username varchar_pattern_ops) TABLESPACE pg_default
Referenced by:
 TABLE "app_staffinfo" CONSTRAINT "app_staffinfo_staffid_id_1291ca3_fk_auth_user_id" FOREIGN KEY ("staffid_id") REFERENCES auth_user(id) DEFERRABLE INITIALLY DEFERRED
 TABLE "app userinfo" CONSTRAINT "app userinfo_userid_id_5e972e5e_fk_auth_user_id" FOREIGN KEY ("userid_id") REFERENCES auth_user(id) DEFERRABLE INITIALLY DEFERRED
 TABLE "auth_user_groups" CONSTRAINT "auth_user_groups_user_id_a612ed6b_fk_auth_user_id" FOREIGN KEY ("user_id") REFERENCES auth_user(id) DEFERRABLE INITIALLY DEFERRED
 TABLE "auth_user_user_permissions" CONSTRAINT "auth_user_user_permissions_user_id_a95ead1b_fk_auth_user_id" FOREIGN KEY ("user_id") REFERENCES auth_user(id) DEFERRABLE INITIALLY DEFERRED
 TABLE "django_admin_log" CONSTRAINT "django_admin_log_user_id_c564e6a6_fk_auth_user_id" FOREIGN KEY ("user_id") REFERENCES auth_user(id) DEFERRABLE INITIALLY DEFERRED
```

图 3.3.1 Django 中 auth\_user 表结构

表 3.3.1 Django 中 auth\_user 表各属性含义

属性名	类型	是否允许空值	含义
id	integer	not null	User 表的主键 ID，在 OpenGauss 中设置为 default nextval ('auth_user_id_seq'::regclass)，从而支持自增，并建立 B+树索引方便查询
password	character varying(128)	not null	用户账户的密码，用于进行 Django 中的身份和权限认证，是必要的
last_login	timestamp with time zone	not null	上一次在 Django 身份认证中登录 login 的时间，由 Django 框架在允许时自动填充
is_superuser	boolean	not null	是否为超级用户的字段，为真时指定该用户所有权限，可以执行所有操作而无需进行权限认证
username	character varying(150)	not null	用户名，是唯一 UNIQUE 字段，来标识一个记录，是必要的，并建立了 B+树索引
first_name	character varying(150)		用户姓名中的名，由用户自定义，非必要
last_name	character varying(150)		用户姓名中的姓，由用户自定义，非必要
email	character varying(254)	not null	电子邮件地址，由用户自定义，非必要，创建用户时指定
is_staff	boolean	not null	是否为为管理员的标识字段，为真时指定该用户可以访问管理站点
is_active	boolean	not null	是否活跃的字段，为真时指定该用户应被是为活跃账户，是通过 Django 用户身份验证的必要条件之一
date_joined	timestamp with time zone	not null	用户创建账号或首次加入系统的日期和时间，当一个新的用户账户被创建时，Django 会自动将当前的日期和时间存储在这个字段中

### 3.3.2 普通用户个人信息 UserInfo 表

根据概念结构设计阶段得到的 E-R 图分析可以设计出普通用户个人信息 UserInfo 表，数据库中对表结

构的描述如图 3.3.2 所示，具体表中各属性含义解释如表 3.3.2 所示。

```
db_bms=> \d app_userinfo
          Table "joe.app_userinfo"
  Column |      Type       | Modifiers
-----+-----+-----+
 id    | bigint        | not null default nextval('app_userinfo_id_seq'::regclass)
 username | character varying(30) | not null
 usersex | boolean       | not null
 userage | integer        | not null
 userphone | character varying(12) | 
 userRegisterTime | timestamp(0) without time zone | not null
 userId_id | integer        | not null
Indexes:
 "app_userinfo_pkey" PRIMARY KEY, btree (id) TABLESPACE pg_default
 Foreign-key constraints:
 "app_userinfo_userid_id_5e972e5e_fk_auth_user_id" FOREIGN KEY ("userId_id") REFERENCES auth_user(id) DEFERRABLE INITIALLY DEFERRED
 Referenced by:
 TABLE "app_borrowbook" CONSTRAINT "app_borrowbook_userid_id_82705f35_fk_app_userinfo_id" FOREIGN KEY ("userId_id") REFERENCES app_userinfo(id) DEFERRABLE INITIALLY DEFERRED
```

图 3.3.2 普通用户个人信息 UserInfo 表结构

表 3.3.2 普通用户个人信息 UserInfo 表中各属性含义

属性名	类型	是否允许空值	含义
id	bigint	not null	Userinfo 表的主键 ID，设置为 default nextval('app_userinfo_id_seq'::regclass)，进而允许自增，并在其上建立了 B+ 树索引方便查询
userName	character varying(30)	not null	普通用户的姓名，由用户注册时自定义创建
userSex	boolean	not null	普通用户的性别，采用布尔类型，为真时表示男性，否则表示女性
userAge	integer	not null	普通用户的年龄，由用户注册时自定义创建
userPhone	character varying(12)		普通用户的手机号码，允许为空值，由用户注册时自定义创建
userRegisterTime	timestamp(0) without time zone	not null	普通用户第一次注册账户的时间，当用户首次注册时由系统自动填充
userId_id	integer	not null	指向 User 表主键 id 的外键字段，用来指向对应的记录在 User 中对应的记录，并在其上建立了 B+ 树索引

### 3.3.3 管理员用户个人信息 StaffInfo 表

根据概念结构设计阶段得到的 E-R 图分析可以设计出管理员用户个人信息 StaffInfo 表，数据库中对表结构的描述如图 3.3.3 所示，具体表中各属性含义解释如表 3.3.3 所示。

```
db_bms=> \d app_staffinfo
          Table "joe.app_staffinfo"
  Column |      Type       | Modifiers
-----+-----+-----+
 id    | bigint        | not null default nextval('app_staffinfo_id_seq'::regclass)
 staffName | character varying(40) | not null
 staffId_id | integer        | not null
Indexes:
 "app_staffinfo_pkey" PRIMARY KEY, btree (id) TABLESPACE pg_default
 Foreign-key constraints:
 "app_staffinfo_staffId_id_1291cfa3_fk_auth_user_id" FOREIGN KEY ("staffId_id") REFERENCES auth_user(id) DEFERRABLE INITIALLY DEFERRED
 Referenced by:
 TABLE "app_addbook" CONSTRAINT "app_addbook_staffId_id_ef020c07_fk_app_staffinfo_id" FOREIGN KEY ("staffId_id") REFERENCES app_staffinfo(id) DEFERRABLE INITIALLY DEFERRED
 TABLE "app_borrowbook" CONSTRAINT "app_borrowbook_staffId_id_a1a87efd_fk_app_staffinfo_id" FOREIGN KEY ("staffId_id") REFERENCES app_staffinfo(id) DEFERRABLE INITIALLY DEFERRED
```

图 3.3.3 管理员用户个人信息 StaffInfo 表结构

表 3.3.3 管理员用户个人信息 StaffInfo 表中各属性含义

属性名	类型	是否允许空值	含义
id	bigint	not null	StaffInfo 表的主键 ID，设置为 default nextval('app_staffinfo_id_seq'::regclass)，进而允许自增，并在其上建立了 B+ 树索引方便查询
staffName	character varying(40)	not null	表示管理员身份信息的管理员名字字段，在创建管理员用户时指定
staffId_id	integer	not null	指向 User 表中主键 id 的外键字段，用来指向对应记录在 U

---

ser 表中对应的记录，并在其上建立了 B+树索引

---

### 3.3.4 图书类型信息 BookType 表

根据概念结构设计阶段得到的 E-R 图分析可以设计出图书类型信息 BookType 表，数据库中对表结构的描述如图 3.3.4 所示，具体表中各属性含义解释如表 3.3.4 所示。

```
db_bms-> \d app_booktype
Table "joe.app_booktype"
 Column | Type | Modifiers
-----+-----+-----+
 typeId | integer | not null
 typeName | character varying(20) | not null
Indexes:
 "app_booktype_pkey" PRIMARY KEY, btree ("TypeId") TABLESPACE pg_default
Referenced by:
 TABLE "app_book" CONSTRAINT "app_book_booktype_id_41acd2c6_fk_app_booktype_TypeId" FOREIGN KEY ("bookType_id") REFERENCES app_booktype("TypeId") DEFERRABLE INITIALLY DEFERRED
```

图 3.3.4 图书类型信息 BookType 表结构

表 3.3.4 图书类型信息 BookType 表中各属性含义

属性名	类型	是否允许空值	含义
TypeId	integer	not null	BookType 表的主键 ID，每一个 ID 对应一个具体的图书类型，所有图书的类型均由图书管理员预先设置好并作为后续图书信息的可选项，不设置自增，并建立了 B+树索引方便查询
TypeName	character varying(20)	not null	类型名字字段，由管理员指定，表示具体的图书类型

### 3.3.5 图书基本信息 Book 表

根据概念结构设计阶段得到的 E-R 图分析可以设计出图书基本信息 Book 表，数据库中对表结构的描述如图 3.3.5 所示，具体表中各属性含义解释如表 3.3.5 所示。

```
db_bms-> \d app_book
Table "joe.app_book"
 Column | Type | Modifiers
-----+-----+-----+
 bookId | character varying(40) | not null
 bookName | character varying(40) | not null
 bookPublisher | character varying(40) | not null
 bookAuthor | character varying(40) | not null
 bookIntroduction | text | not null
 bookPrice | integer | not null
 bookType_id | integer | not null
Indexes:
 "app_book_pkey" PRIMARY KEY, btree ("bookId") TABLESPACE pg_default
 "app_book_bookId_58ad1191_like" btree ("bookId" varchar_pattern_ops) TABLESPACE pg_default
 "app_book_booktype_id_41acd2c6" btree ("bookType_id") TABLESPACE pg_default
Foreign-key constraints:
 "app_book_booktype_id_41acd2c6_fk_app_booktype_TypeId" FOREIGN KEY ("bookType_id") REFERENCES app_booktype("TypeId") DEFERRABLE INITIALLY DEFERRED
Referenced by:
 TABLE "app_addbook" CONSTRAINT "app_addbook_bookId_id_049631c3_fk_app_book_bookId" FOREIGN KEY ("bookId") REFERENCES app_book("bookId") DEFERRABLE INITIALLY DEFERRED
 TABLE "app_booknum" CONSTRAINT "app_booknum_BookId_id_f78d1434_fk_app_book_bookId" FOREIGN KEY ("bookId") REFERENCES app_book("bookId") DEFERRABLE INITIALLY DEFERRED
 TABLE "app_borrowbook" CONSTRAINT "app_borrowbook_bookId_id_96c5d913_fk_app_book_bookId" FOREIGN KEY ("bookId") REFERENCES app_book("bookId") DEFERRABLE INITIALLY DEFERRED
```

图 3.3.5 图书基本信息 Book 表结构

表 3.3.5 图书基本信息 Book 表中各属性含义

属性名	类型	是否允许空值	含义
bookId	character varying(40)	not null	Book 表的主键 ID，字符串类型唯一表示一本图书，值由管理员添加图书时指定，不设置自增，并建立 B+树索引方便查询
bookName	character varying(40)	not null	图书的名称字段，由管理员添加图书时指定
bookPublisher	character varying(40)	not null	图书的出版社字段，由管理员添加图书时指定
bookAuthor	character varying(40)	not null	图书的作者字段，由管理员添加图书时指定
bookIntroduction	text	not null	图书的简介字段，由管理员添加图书时指定
bookPrice	integer	not null	图书的价格字段，由管理员添加图书时指定
bookType_id	integer	not null	指向 BookType 图书类型表主键 TypId 的外键字段，用整数 ID 值来实现到具体图书类型的映射，并建立了 B+树索引

### 3.3.6 图书数量 BookNum 表

根据概念结构设计阶段得到的 E-R 图分析可以设计出图书数量 BookNum 表，数据库中对表结构的描述如图 3.3.6 所示，具体表中各属性含义解释如表 3.3.6 所示。

```
db_bms=> \d app_booknum
Table "joe.app_booknum"
 Column | Type | Modifiers
-----+-----+-----+
 id    | bigint | not null default nextval('app_booknum_id_seq'::regclass)
 bookNum | integer | not null
 bookId_id | character varying(40) | not null
Indexes:
 "app_booknum_pkey" PRIMARY KEY, btree (id) TABLESPACE pg_default
 "app_booknum_bookId_id_f78d1434" btree ("bookId_id") TABLESPACE pg_default
 "app_booknum_bookId_id_f78d1434_like" btree ("bookId_id" varchar_pattern_ops) TABLESPACE pg_default
Foreign-key constraints:
 "app_booknum_bookId_id_f78d1434_fk_app_book_bookId" FOREIGN KEY ("bookId_id") REFERENCES app_book("bookId") DEFERRABLE INITIALLY DEFERRED
```

图 3.3.6 图书数量 BookNum 表结构

表 3.3.6 图书数量 BookNum 表中各属性含义

属性名	类型	是否允许空值	含义
id	bigint	not null	BookNum 表的主键 ID，设置为 default nextval('app_booknum_id_seq'::regclass)从而允许自增，并建立了 B+树索引方便查询
bookNum	integer	not null	表示图书数量的整型字段，在添加图书或添加图书数量时由管理员指定
bookId_id	character varying(40)	not null	指向 Book 表中主键 bookId 的外键字段，来表示每一本书的具体数量关系，并建立 B+树索引方便查询

### 3.3.7 添加图书数量 AddBook 表

根据概念结构设计阶段得到的 E-R 图分析可以设计出添加图书数量 AddBook 表，数据库中对表结构的描述如图 3.3.7 所示，具体表中各属性含义解释如表 3.3.7 所示。

```
db_bms=> \d app_addbook
Table "joe.app_addbook"
 Column | Type | Modifiers
-----+-----+-----+
 id    | bigint | not null default nextval('app_addbook_id_seq'::regclass)
 addNum | integer | not null
 bookId_id | character varying(40) | not null
 staffId_id | bigint | not null
Indexes:
 "app_addbook_pkey" PRIMARY KEY, btree (id) TABLESPACE pg_default
 "app_addbook_bookId_id_049631c3" btree ("bookId_id") TABLESPACE pg_default
 "app_addbook_bookId_id_049631c3_like" btree ("bookId_id" varchar_pattern_ops) TABLESPACE pg_default
 "app_addbook_staffId_id_ef02c0c7" btree ("staffId_id") TABLESPACE pg_default
Foreign-key constraints:
 "app_addbook_bookId_id_049631c3_fk_app_book_bookId" FOREIGN KEY ("bookId_id") REFERENCES app_book("bookId") DEFERRABLE INITIALLY DEFERRED
 "app_addbook_staffId_id_ef02c0c7_fk_app_staffinfo_id" FOREIGN KEY ("staffId_id") REFERENCES app_staffinfo("staffId_id") DEFERRABLE INITIALLY DEFERRED
```

图 3.3.7 添加图书数量 AddBook 表结构

表 3.3.7 添加图书数量 AddBook 表中各属性含义

属性名	类型	是否允许空值	含义
id	bigint	not null	AddBook 表的主键 ID，设置为 default nextval('app_addbook_id_seq'::regclass)以实现自增，并建立 B+树索引方便查询
addNum	integer	not null	表示每次添加图书数量时的具体数量，由管理员添加图书数量时指定
bookId_id	character varying(40)	not null	指向 Book 表中主键 bookId 的外键字段，来表示给哪一本书来添加数量，并建立 B+树索引方便查询
staffId_id	bigint	not null	指向 StaffInfo 表中主键 id 的

外键字段，用来表示是哪一个管理员执行的添加记录，并建立 B+树索引方便查询

### 3.3.8 借书记录 BorrowBook 表

根据概念结构设计阶段得到的 E-R 图分析可以设计出借书记录 BorrowBook 表，数据库中对表结构的描述如图 3.3.8 所示，具体表中各属性含义解释如表 3.3.8 所示。

```
db_bms=> \d app_borrowbook
          Table "joe.app.borrowbook"
  Column |          Type          | Modifiers
-----+-----+-----+
borrowId | integer          | not null
hasReturned | boolean          | not null
bookId_id | character varying(40) | not null
staffId_id | bigint           | not null
userId_id | bigint           | not null
Indexes:
"app_borrowbook_pkey" PRIMARY KEY, btree ("borrowId") TABLESPACE pg_default
"app_borrowbook_bookId_id_06c5d913" btree ("bookId_id") TABLESPACE pg_default
"app_borrowbook_bookId_id_06c5d913_like" btree ("bookId_id" varchar_pattern_ops) TABLESPACE pg_default
"app_borrowbook_staffId_id_a1a87efd" btree ("staffId_id") TABLESPACE pg_default
"app_borrowbook_userId_id_82705f35" btree ("userId_id") TABLESPACE pg_default
Foreign-key constraints:
"app_borrowbook_bookId_id_06c5d913_fk_app_book_bookId" FOREIGN KEY ("bookId_id") REFERENCES app_book("bookId") DEFERRABLE INITIALLY DEFERRED
"app_borrowbook_staffId_id_a1a87efd_fk_app_staffinfo_id" FOREIGN KEY ("staffId_id") REFERENCES app_staffinfo(id) DEFERRABLE INITIALLY DEFERRED
"app_borrowbook_userId_id_82705f35_fk_app_userinfo_id" FOREIGN KEY ("userId_id") REFERENCES app_userinfo(id) DEFERRABLE INITIALLY DEFERRED
```

图 3.3.8 借书记录 BorrowBook 表结构

表 3.3.8 借书记录 BorrowBook 表中各属性含义

属性名	类型	是否允许空值	含义
borrowId	integer	not null	BorrowBook 表的主键 ID，来表示表中每一条借书记录，并建立 B+树索引方便查询
hasReturned	boolean	not null	是否已还的标记变量，采用布尔类型，若为真表示此条借书记录借出的书已经还回
bookId_id	character varying(40)	not null	指向 Book 表中主键 bookId 的外键字段，用来表示此条借书记录借出的书是哪一本，并建立 B+树索引方便查找
staffId_id	bigint	not null	指向 StaffInfo 表主键 id 的外键字段，用来表示此次借书记录是由哪个管理员操作的，并建立 B+树索引方便查找
userId_id	bigint	not null	指向 UserInfo 表主键 id 的外键字段，用来表示此次借书是借给哪一位用户的，并建立 B+树索引方便查找

## 3.4 物理结构设计阶段

图书管理系统设计采用的数据库管理系统为华为的 OpenGauss 数据库，在多个表的主键和外键字段上都建立 B+树索引以提高查找效率。在确定数据库的存储结构方面，包括确定关系、索引、聚簇、日志、备份等的存储安排和存储结构，确定系统配置等均采取数据库默认的配置。

## 3.5 数据库实施阶段

在数据库实施阶段，主要使用 Python 语言连接远程 OpenGauss 数据库，采用 Python 库 Psycopg2，这是一个 Python 适配器，它允许 Python 程序与 PostgreSQL 数据库进行交互，执行 SQL 命令，处理结果集并实现数据库中的连接操作。在使用 Django 进行网站前端页面的开发过程中，Django 支持使用相应的数据库连接远程数据库，从而实现模型和数据库之间表的迁移。

Django 中连接 OpenGauss 远程数据库服务器的配置如下代码 3.5.1 所示。

```

DATABSES = {
    'default': {
        # 'ENGINE': 'django.db.backends.sqlite3',
        'ENGINE': 'django.db.backends.postgresql_psycopg2',
        'NAME': 'db_bms',
        'USER': 'joe',
        'PASSWORD': 'Bigdata@123',
        'HOST': '121.36.37.200',
        'PORT': 26000
    }
}

```

代码 3.5.1 Django 中链接远程 OpenGauss 数据库配置

### 3.6 数据库运行和维护阶段

最后一个阶段，是数据库的运行和维护阶段，在实际开发过程中数据库运行效果良好，即使在后续压力测试添加了一百万条数据后查询效率依旧保持在一个较高的水平，无需再进行人工对数据库的修改。

## 4 Django 界面设计

### 4.1 界面介绍

#### 4.1.1 图书管理系统登录主页面

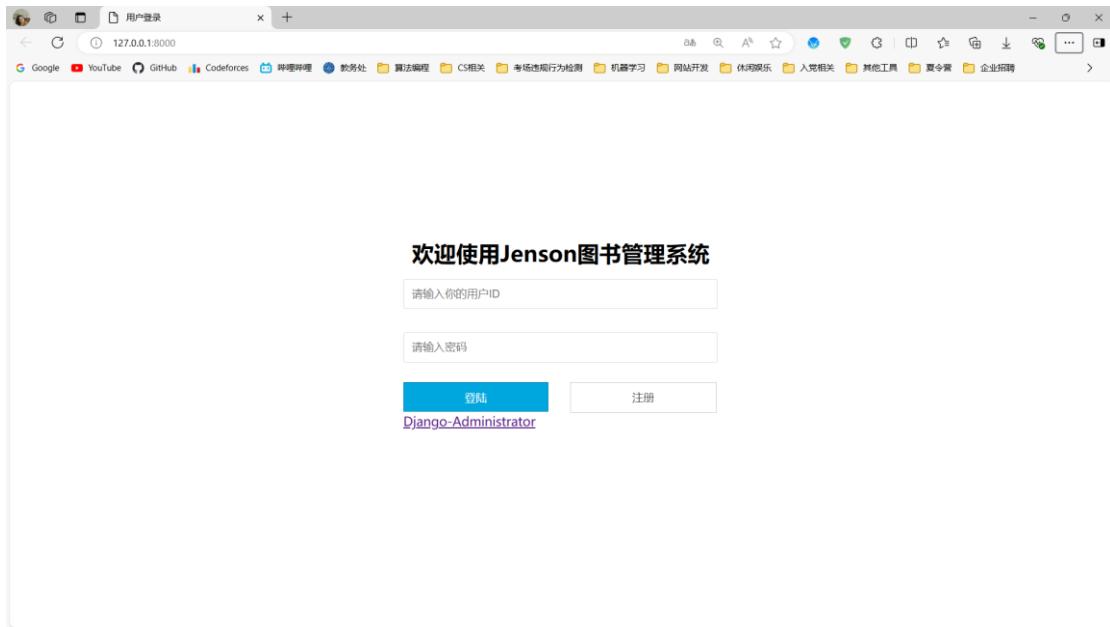


图 4.1.1 图书管理系统登录的主页面

如上图 4.1.1 所示，此页面是本图书管理系统启动 Web 后的主页面，用户通过输入自己在注册账户时所输入的用户 ID 和密码进行登录，点击登录按钮之后后台会进行身份验证，若数据库中存在该 ID 的用户

并且密码输入正确即可通过验证进入普通用户中心页面。若事先没有注册过对应的用户 ID 或者密码输入错误，系统将无法通过验证并弹出登录失败的提示，请求重新操作，如下图 4.1.2 所示。

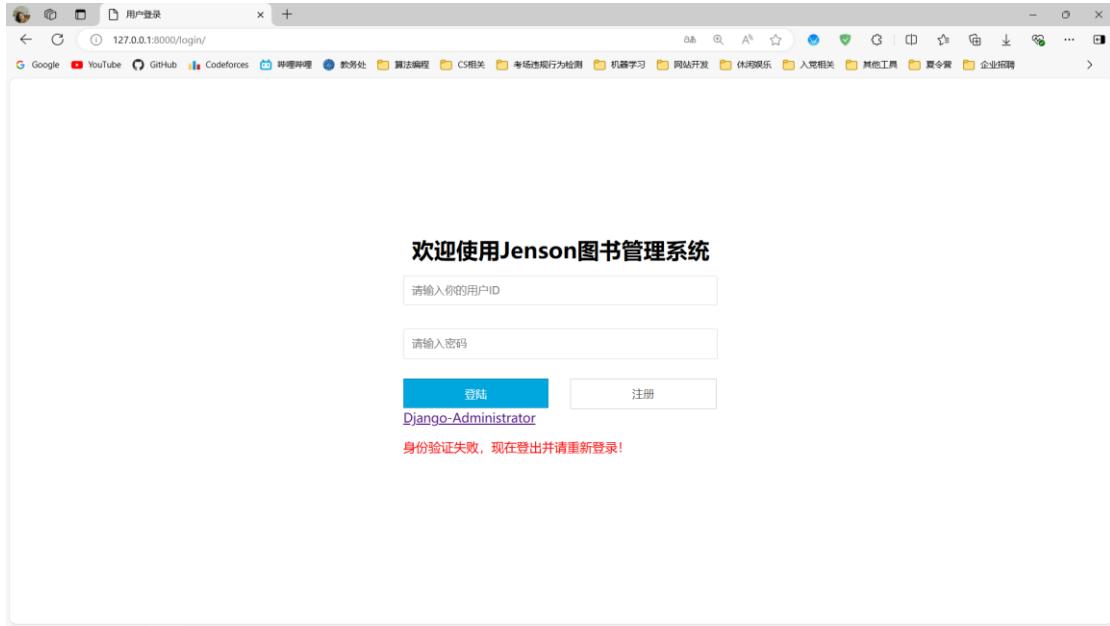


图 4.1.2 用户登录时用户名或密码错误的提示信息

同样，管理员登录时输入管理员用户的 ID 和对应的密码，通过系统校对数据和验证之后即可登录系统，然后进入管理员中心的页面进行管理员具有的一系列操作。在“登录”按钮下面的“Django-Administrator”超链接是用来进入 Django 所开发的 Web 自带的 Admin 站点的，需要创建对应的超级用户（在本系统中与管理员用户是一致的）并输入账号和密码，进入 Admin 站点后可以对系统中所有的数据模型进行修改。

#### 4.1.2 注册页面

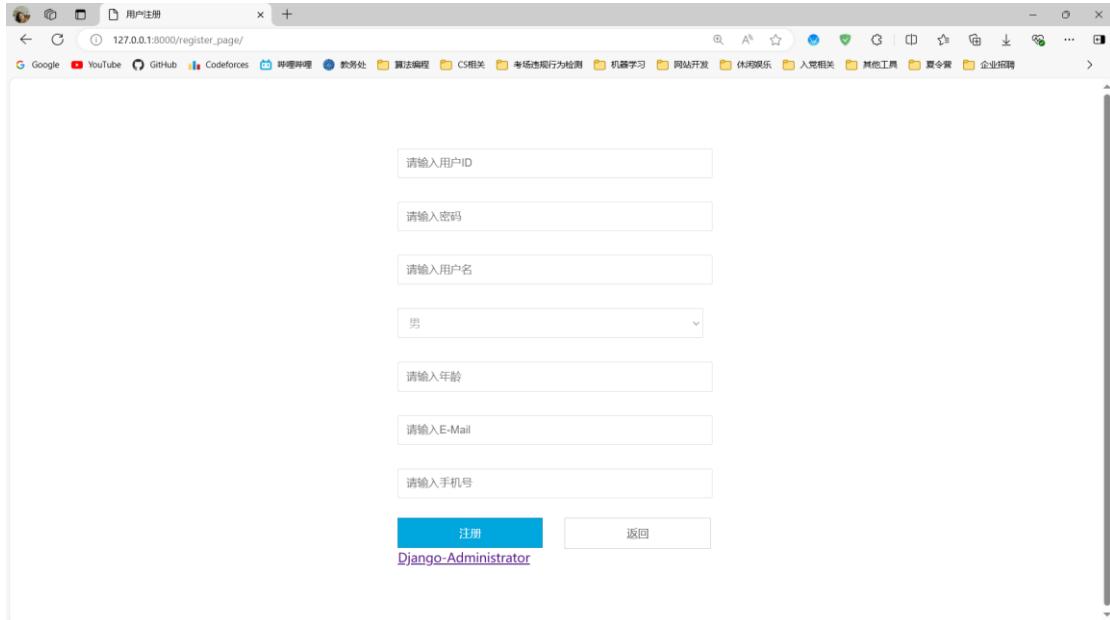


图 4.1.3 普通用户进行注册的主页面

如上图 4.1.3 所示，这是本图书管理系统用于普通用户注册的页面，普通用户根据输入文本框的提示信息输入自定义的数据，当输入的信息格式合法时会提示成功创建用户并自动跳转到登录页面，若输入的信息和数据库中现有的信息发生冲突或者格式出现问题会提示注册失败并请求重新操作，如下图 4.1.4、图 4.1.

5、图 4.1.6、图 4.1.7 所示。

The screenshot shows a user registration form with several input fields and error messages. The fields are:

- 请输入用户ID: A text input field containing "003".
- 请输入密码: A text input field containing "\*\*\*\*\*".
- 请输入用户名: A text input field containing "buger".
- 男: A dropdown menu set to "男".
- 请输入年龄: A text input field containing "21".
- 请输入E-Mail: A text input field containing "123@123.com".
- 请输入手机号: A text input field containing "12312313212".

Below the form are two buttons: a blue "注册" (Register) button and a white "返回" (Return) button. A red message at the bottom states "注册失败, 请重新操作!" (Registration failed, please try again!).

图 4.1.4 注册时输入信息不合规的提示信息

The screenshot shows the same user registration form with valid input. The fields are:

- 请输入用户ID: A text input field containing "003".
- 请输入密码: A text input field containing "\*\*\*\*\*".
- 请输入用户名: A text input field containing "buger".
- 男: A dropdown menu set to "女".
- 请输入年龄: A text input field containing "21".
- 请输入E-Mail: A text input field containing "123@123.com".
- 请输入手机号: A text input field containing "12312313212".

Below the form are two buttons: a blue "注册" (Register) button and a white "返回" (Return) button. A purple message at the bottom indicates "Django-Administrator" has been registered.

图 4.1.5 普通用户输入个人信息进行注册的主页面

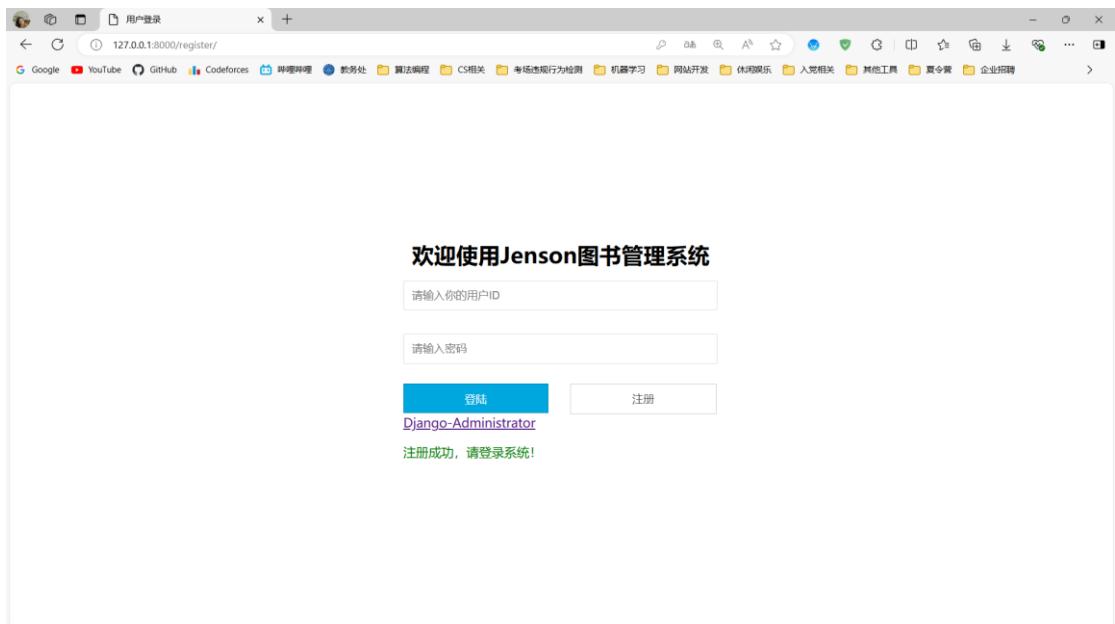


图 4.1.6 注册成功后的跳转页面

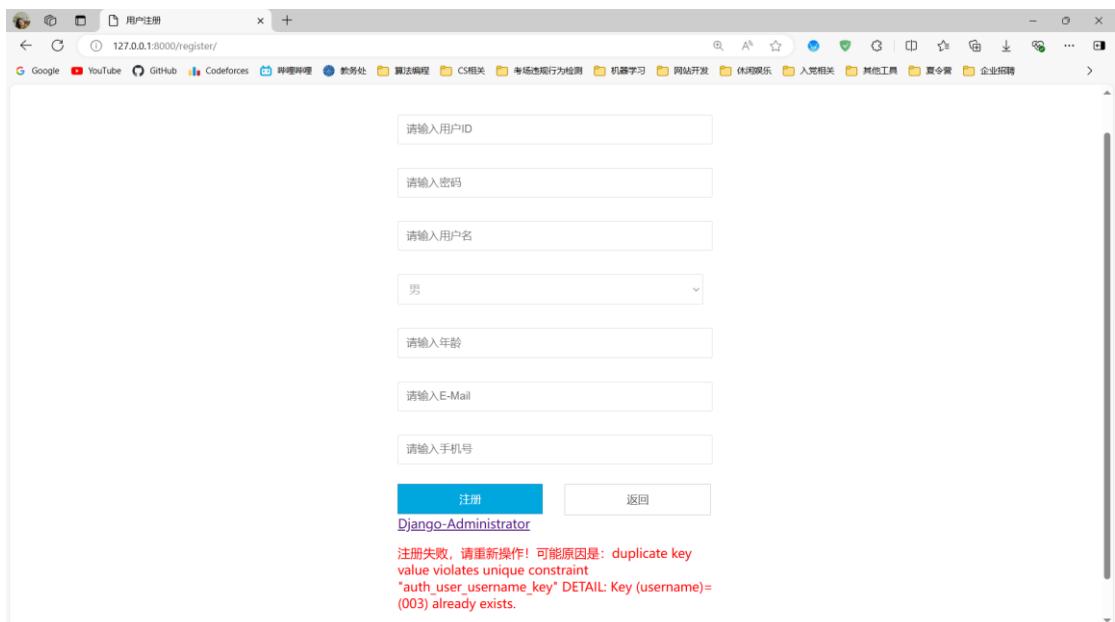


图 4.1.7 注册失败的提示信息

注册功能仅用于普通用户的注册，管理员用户的身份无法进行注册得到，其账户和密码在创建系统时已经指定。

#### 4.1.3 普通用户中心页面

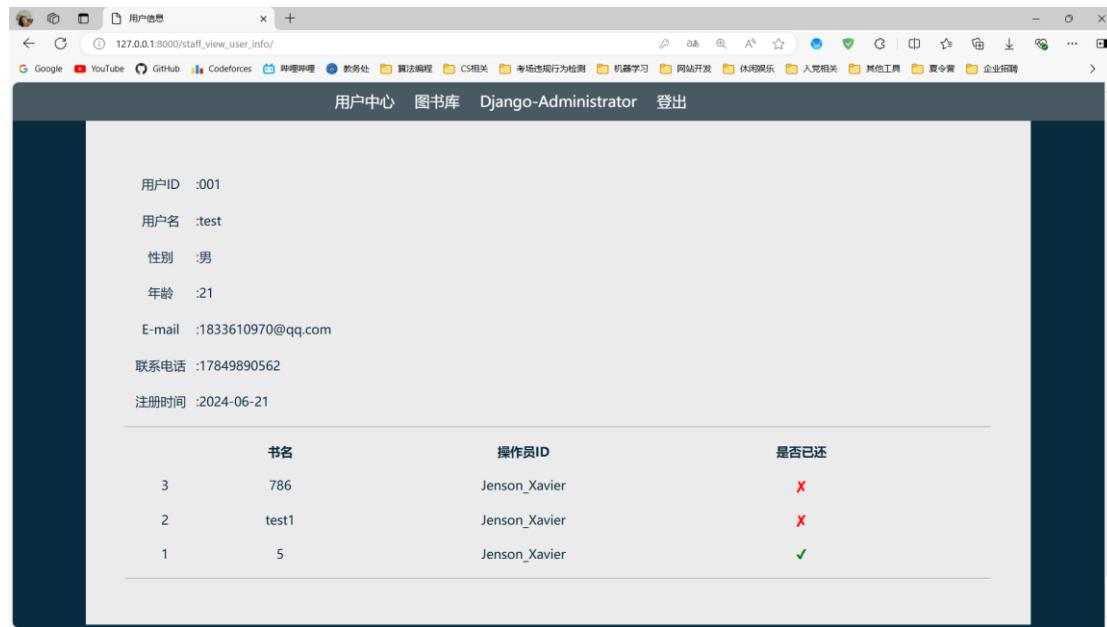


图 4.1.8 普通用户 001 登陆后显示信息的主页面

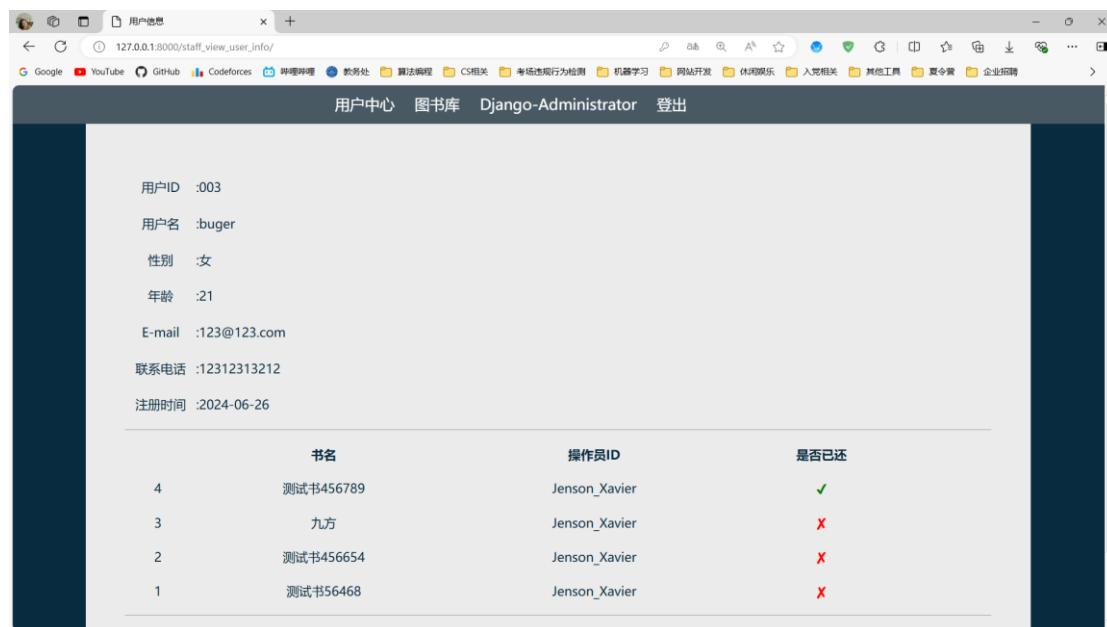


图 4.1.9 普通用户 003 登陆后显示信息的主页面

如上图 4.1.8、图 4.1.9 所示，普通用户注册并且输入正确的用户 ID 和登录密码通过验证后即可进入用户中心页面，在这个页面，用户可以看到自己的个人信息，在下面会展示自己的借书情况和还书情况，以及对应的操作管理员的 ID。在顶部操作栏中提供四个可跳转的链接，点击“用户中心”可以重新回到此页面；点击“图书库”可以进入本图书管理系统的图书库页面；点击“Django-Administrator”可以进入本 Web 开发的 Django 内置的管理员 Admin 站点，只有以管理员身份登录时此链接才会正确跳转，普通用户登录时需要输入对应的超级管理员的账户和密码；点击“退出”可以退出当前已经登录的用户并重新回到登录页面，重新输入账号和密码进行登录。

#### 4.1.4 管理员中心页面

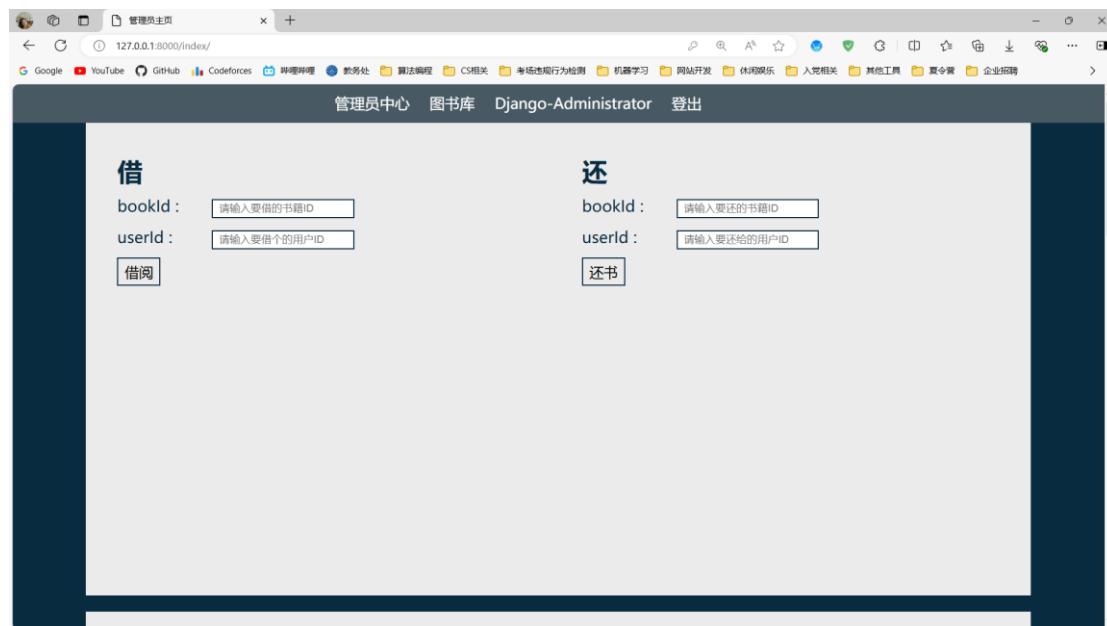


图 4.1.10 管理员借还书的主页面

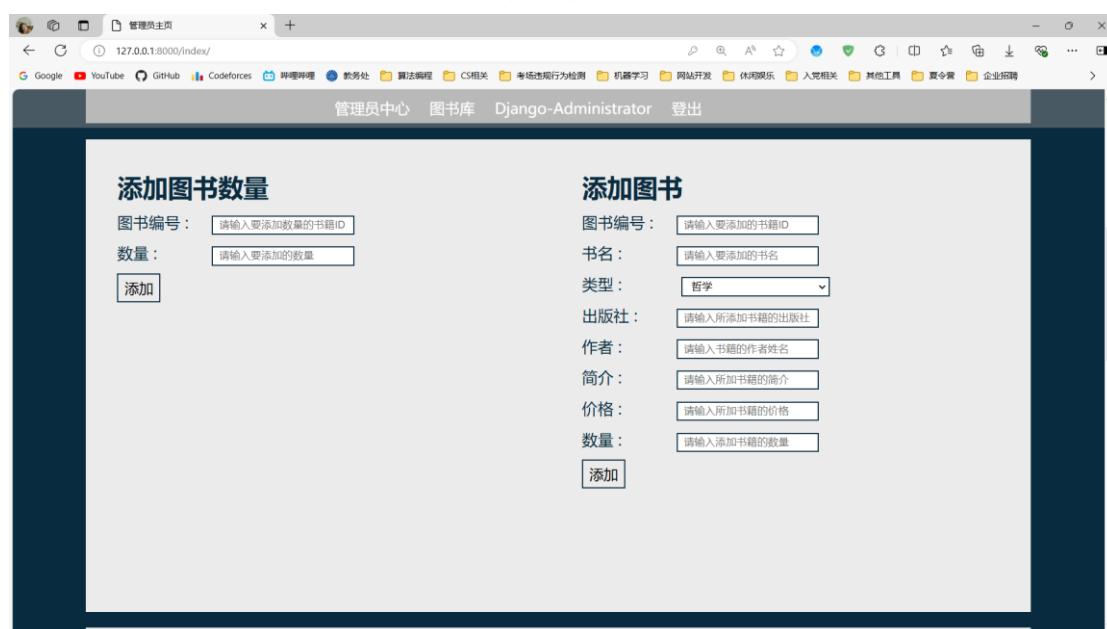


图 4.1.11 管理员添加图书数量和添加图书的主页面



图 4.1.12 管理员更改图书信息的主界面

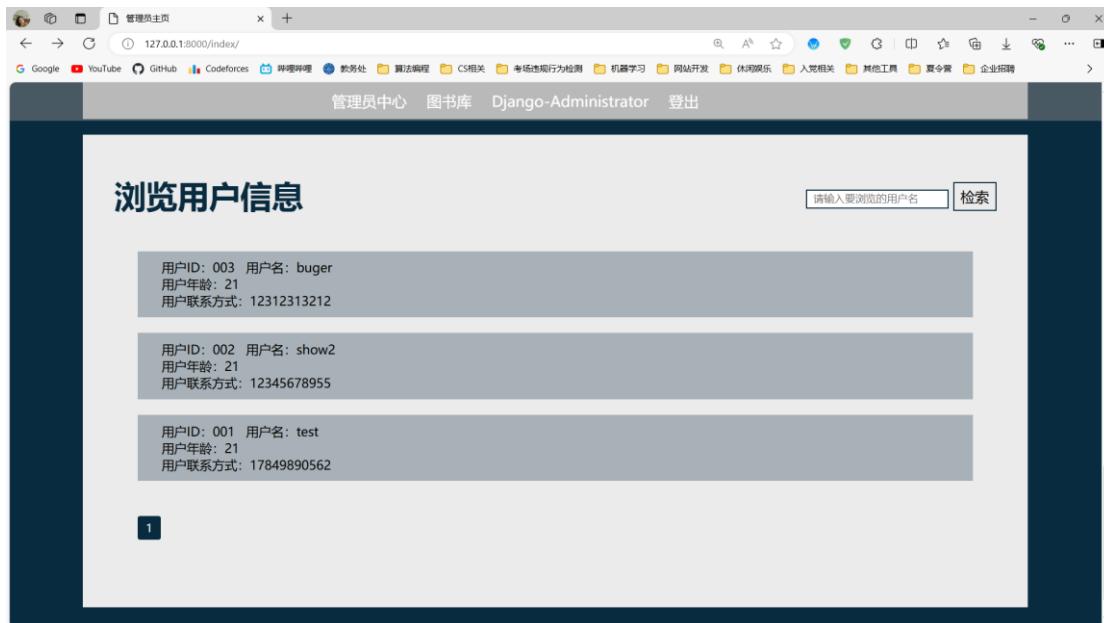


图 4.1.13 管理员浏览用户信息的主界面

如上图 4.1.10、图 4.1.11、图 4.1.12、图 4.1.13 所示，当输入正确内置的管理员的账户和密码之后登录成功即可进入管理员页面，管理员可以进行将指定 ID 的书籍借给指定 ID 的用户，并将指定 ID 的用户已经借的书进行归还；可以添加新的图书信息，并增加指定 ID 图书的数量；可以通过指定图书的 ID 检索得到该图书的具体信息并修改其中的每个字段的信息；也可以浏览当前系统已经注册的所有普通用户信息，可以通过用户的用户名进行检索。当点击页面顶部边栏中的“管理员中心”时会重新跳转到这个页面，其他点击效果如前普通用户中心页面所述，不过当以管理员身份成功登录时点击“Django-Administrator”链接之后可以成功跳转到 Django 自带的 Admin 后台站点进行数据库中信息的管理和维护。

当输入正确的图书 ID 和用户 ID，并且对应的图书还存在库存时可以将图书成功借出，当指定 ID 的用户的确存在对应图书的借书记录时，借书和还书的操作会提示成功，否则会提示出错，这也包括格式错误的情况，相关效果如下图 4.1.14、图 4.1.15、图 4.1.16 所示。

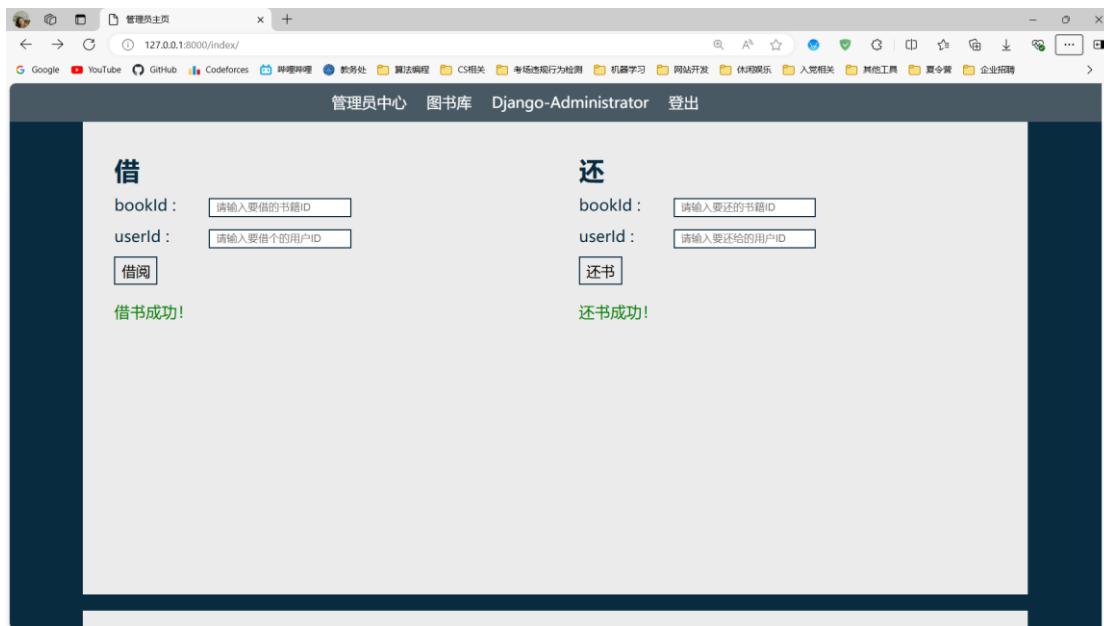


图 4.1.14 管理员借还书成功的提示信息

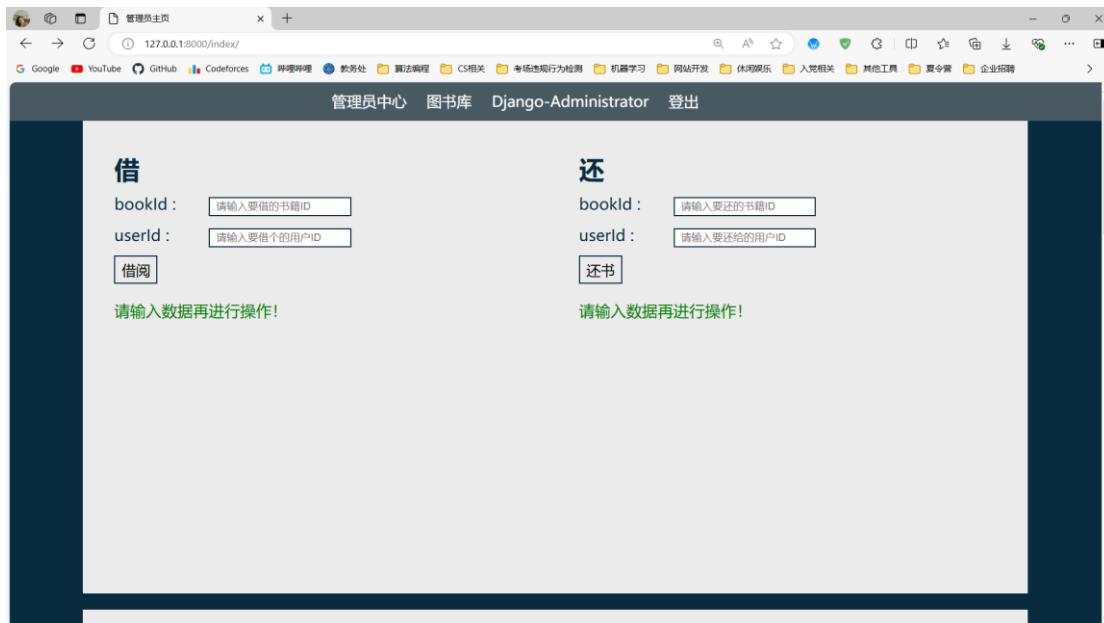


图 4.1.15 管理员未输入数据时的提示信息

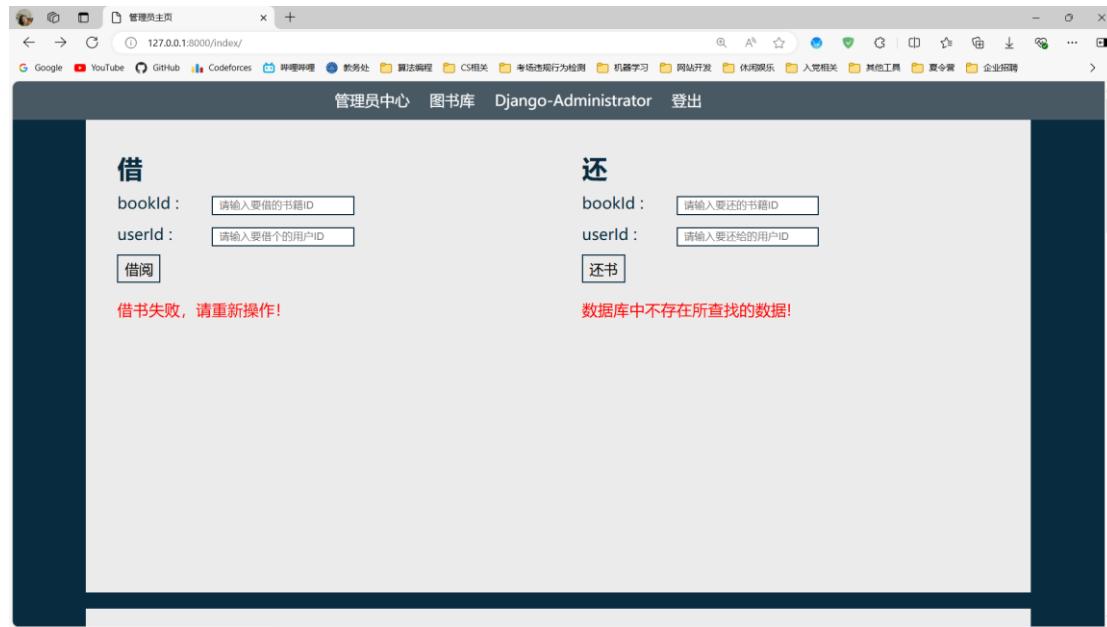


图 4.1.16 管理员借还书失败的提示信息

添加图书数量和添加图书的效果如上所述，当指定的书籍 ID 存在于图书数据库时可以成功添加数据，当要添加图书的信息格式输入无误时，并且输入的对应图书编号不与现有图书数据库冲突时会提示成功添加数据，否则均会提示出错并请求重新操作，如下图图 4.1.17、图 4.1.18、图 4.1.19 所示。



图 4.1.17 管理员添加图书数量成功的提示信息



图 4.1.18 管理员添加图书失败的提示信息



图 4.1.19 管理员成功添加书籍并添加图书数量失败的提示信息

当管理员需要更改图书信息时，输入正确的图书 ID 进行检索，对检索得到的指定 ID 的图书可以修改相关信息，如果检索输入的格式不正确或者不存在对应的图书 ID 则会提示报错并请求重新操作，修改图书信息和添加图书的操作流程基本一致，当输入正确格式的信息后提交更新会提示更新成功，否则提示失败并请求重新操作，如下图 4.1.20、图 4.1.21、图 4.1.22、图 4.1.23 所示。

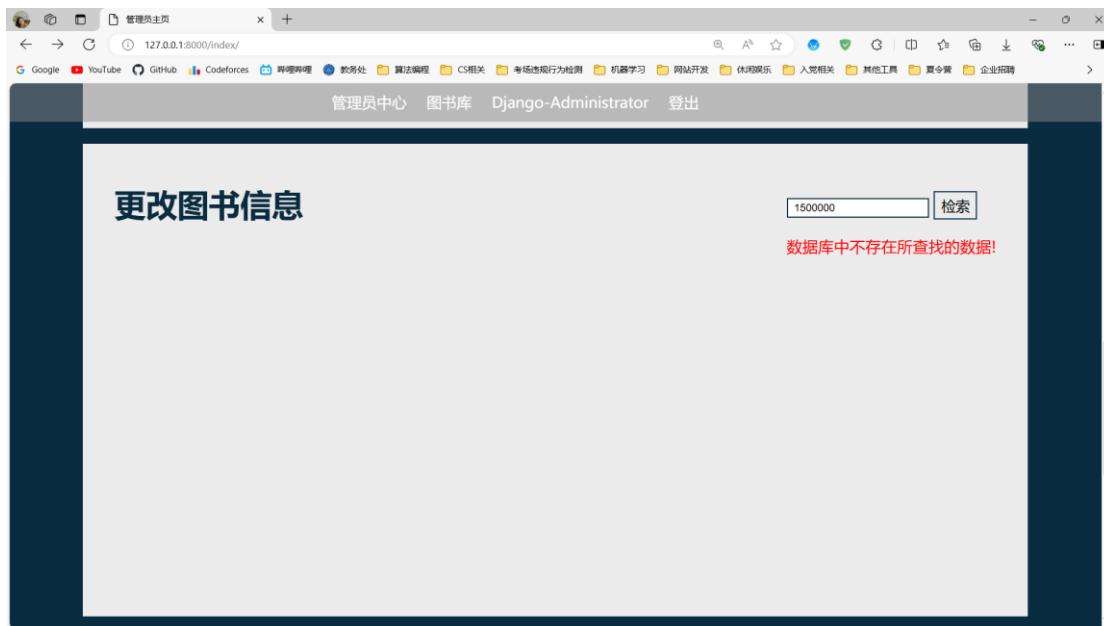


图 4.1.20 管理员更改图书信息前检索图书失败的提示信息

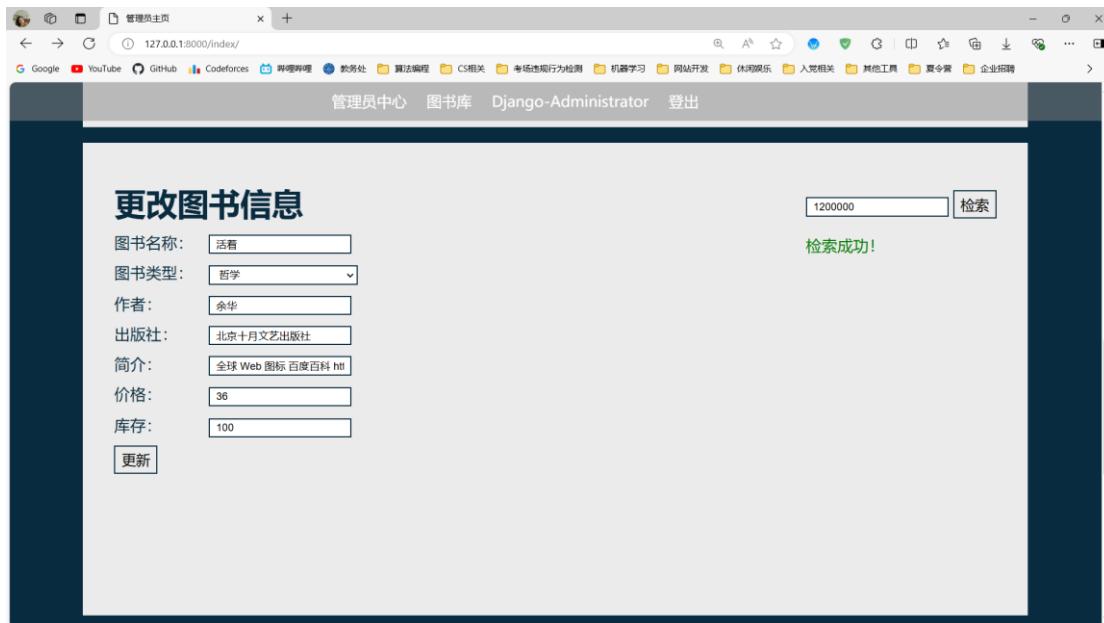


图 4.1.21 管理员尝试进行更改图书信息前检索成功的提示信息

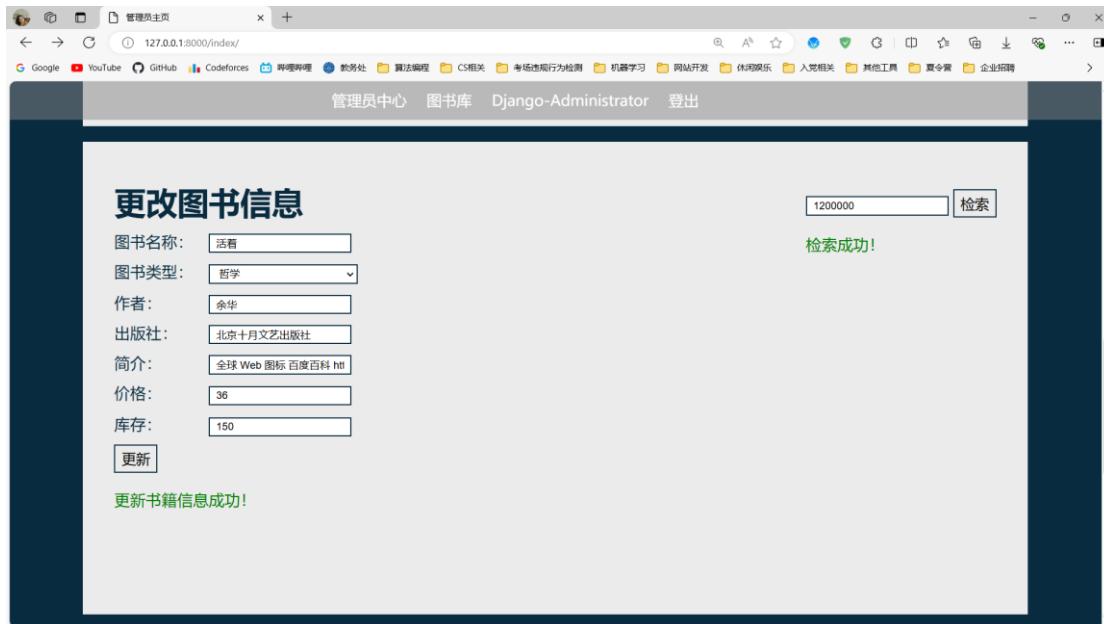


图 4.1.22 管理员尝试进行更改图书信息成功的提示信息

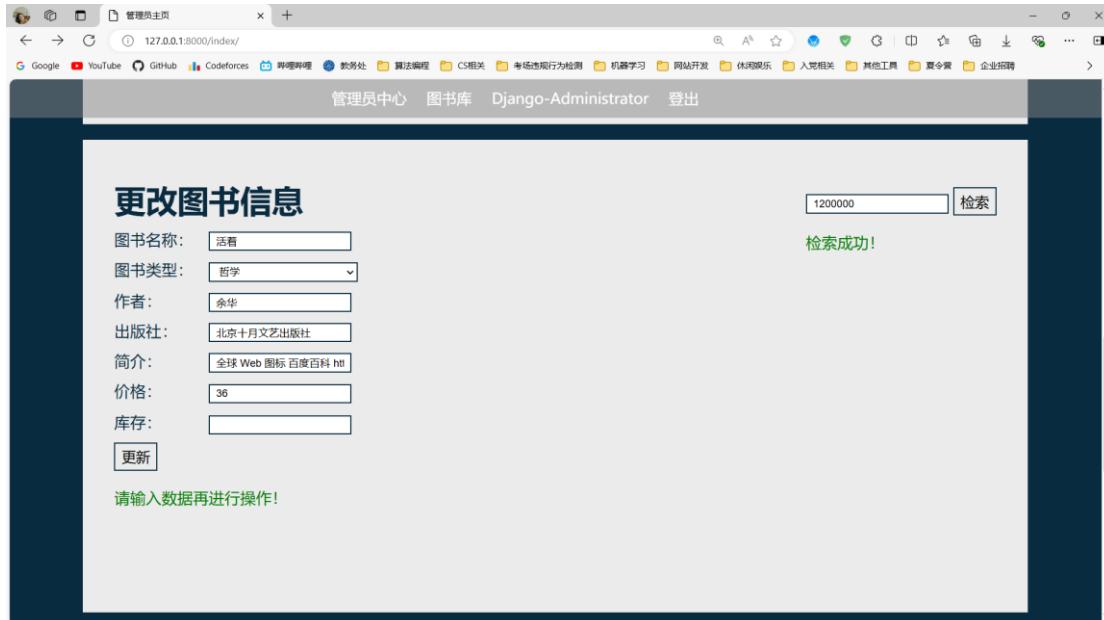


图 4.1.23 管理员尝试进行更改图书信息的提示信息

在最后一个浏览用户信息界面，以分页的方式将系统中所有已经注册过的普通用户信息的简介呈现出来，点击某个具体的用户可以跳转到展示其详细信息的页面，在右上角可以输入指定用户的用户名进行检索，支持模糊搜索，即若某个用户其名字中含有检索时输入的内容都会呈现出来，若不存在用户名符合要求的则不会显示任何用户信息的条目，若不输入任何内容默认展示所有用户，如下图 4.1.24、图 4.1.25、图 4.1.26、图 4.1.27 所示。

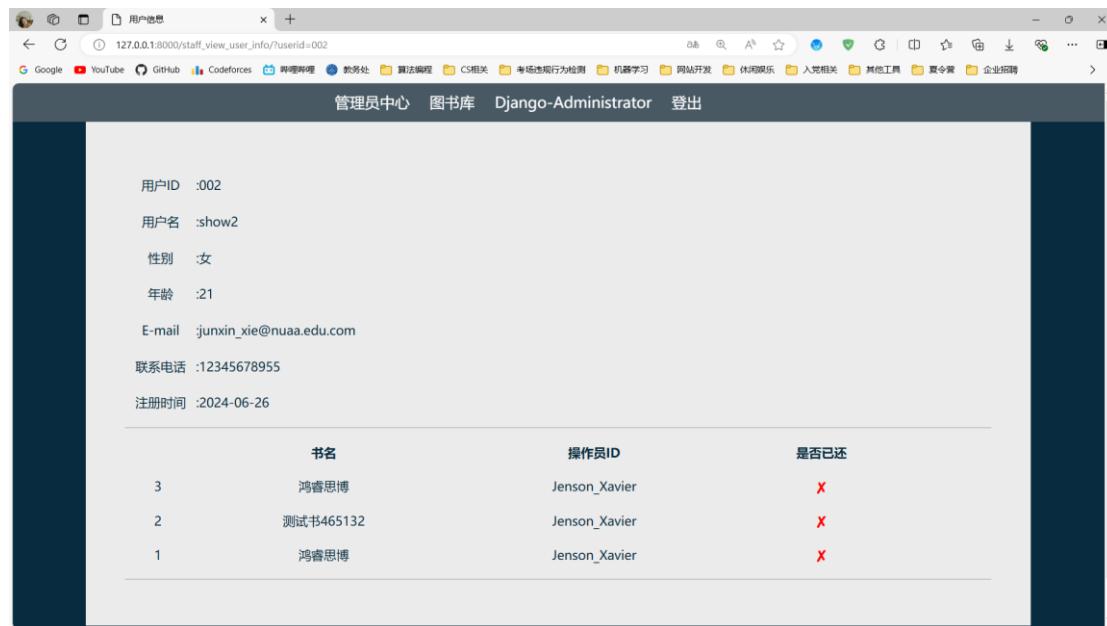


图 4.1.24 管理员浏览具体用户信息

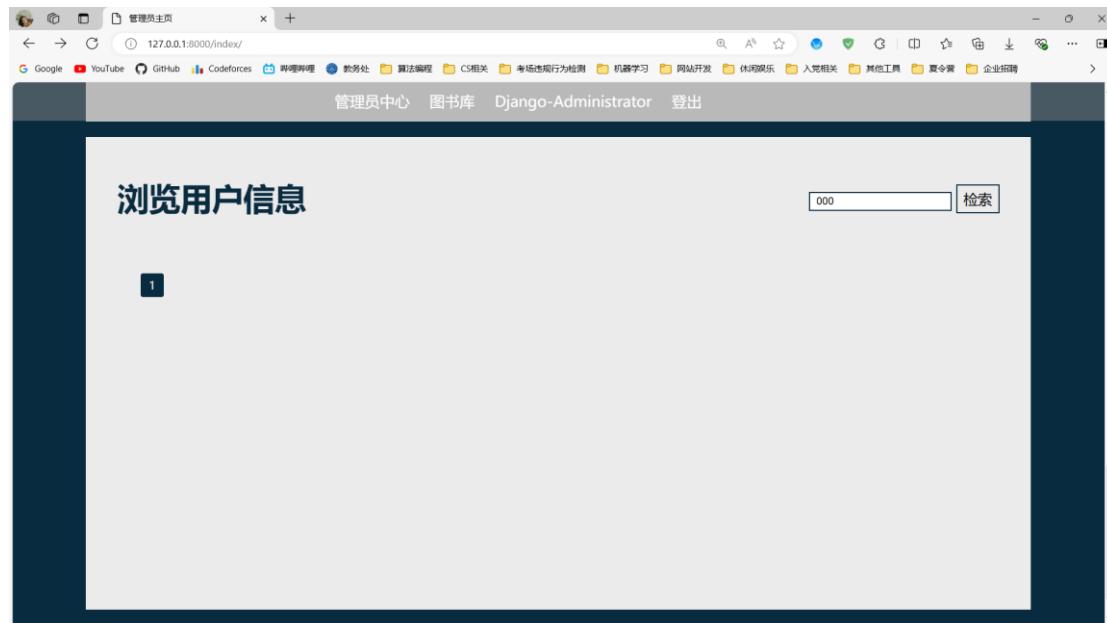


图 4.1.25 检索的用户不存在

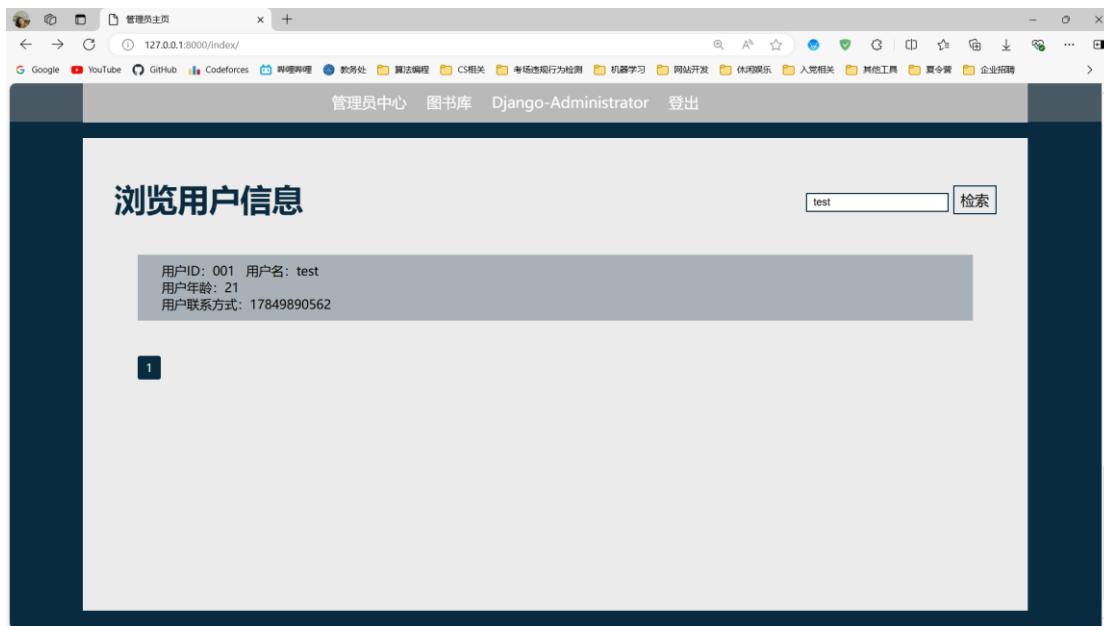


图 4.1.26 管理员检索特定用户名的用户

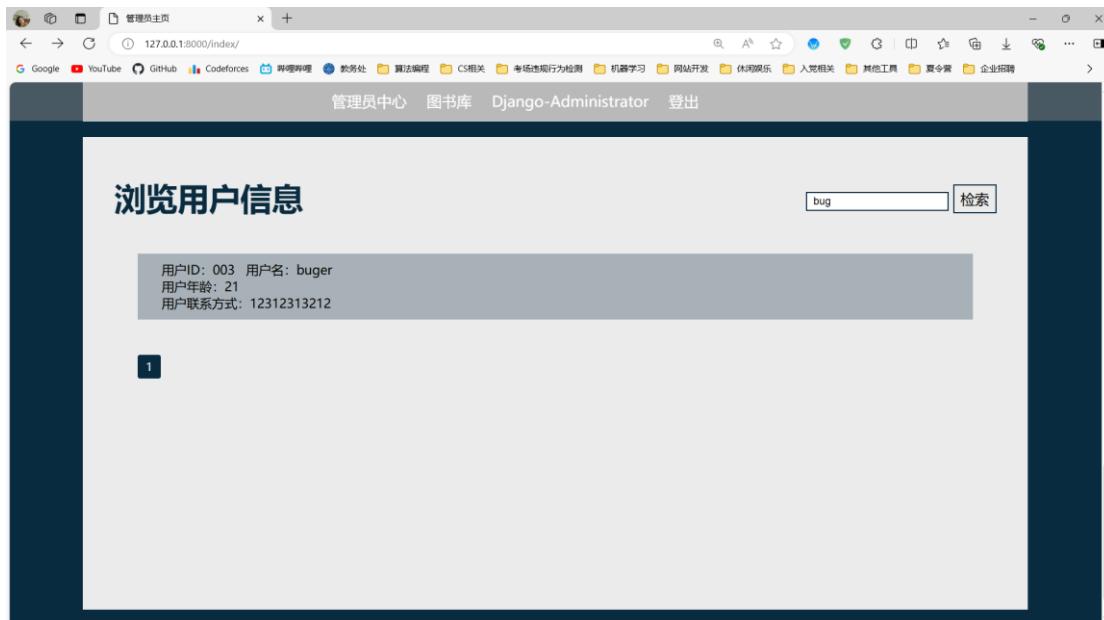


图 4.1.27 管理员检索含有指定用户名的用户

## 4.1.5 图书库页面



图 4.1.28 图书库主页面第一页页面的上半部分

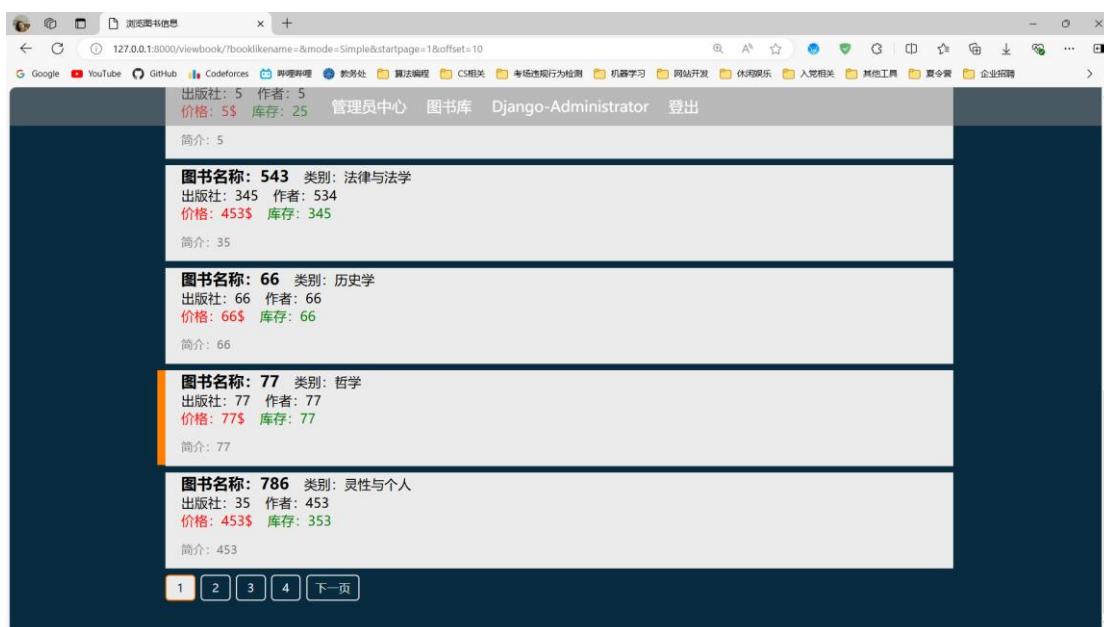


图 4.1.29 图书库主页面第一页页面的下半部分

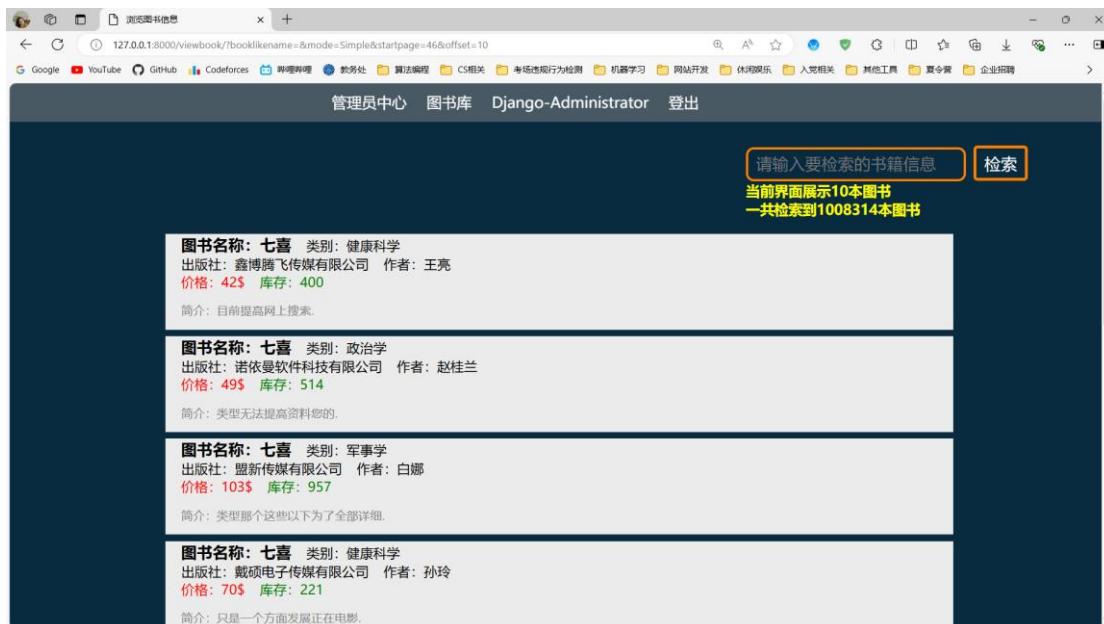


图 4.1.30 图书库分页跳转后页面的上半部分



图 4.1.31 图书库分页跳转后页面的下半部分

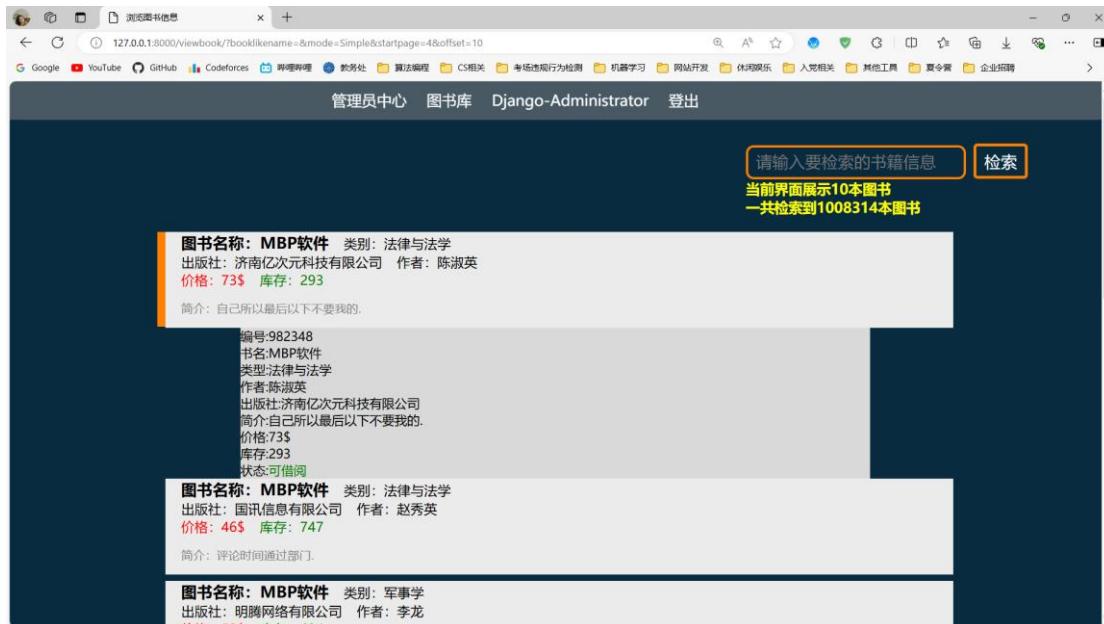


图 4.1.32 图书库主页面查看某图书的详细信息

如上图 4.1.28、图 4.1.29、图 4.1.30、图 4.1.31、图 4.1.32 所示，图书库页面是本图书管理系统的一个核心页面，这个页面负责展示图书库中已上架的所有图书，因为图书库中图书总数可能会非常之多，所以在展示界面中作了分页处理，每页最多展示 10 本图书，每个图书条目将部分信息标注在上面，点击某个条目可以展开去查看该图书的详细信息。

同时在右上角检索栏可以对想要找的指定图书进行检索，支持多个属性的模糊搜索，可以检索含有指定书名、指定出版社和指定作者的图书，当检索成功后输入栏下方会提示检索的结果，有多少符合条件的图书，并继续以分页的形式显示在下方，当输入的内容有误或者检索不到符合条件的图书时就展示为空，若输入为空则默认展示所有图书，如下图 4.1.33、图 4.1.34 所示。

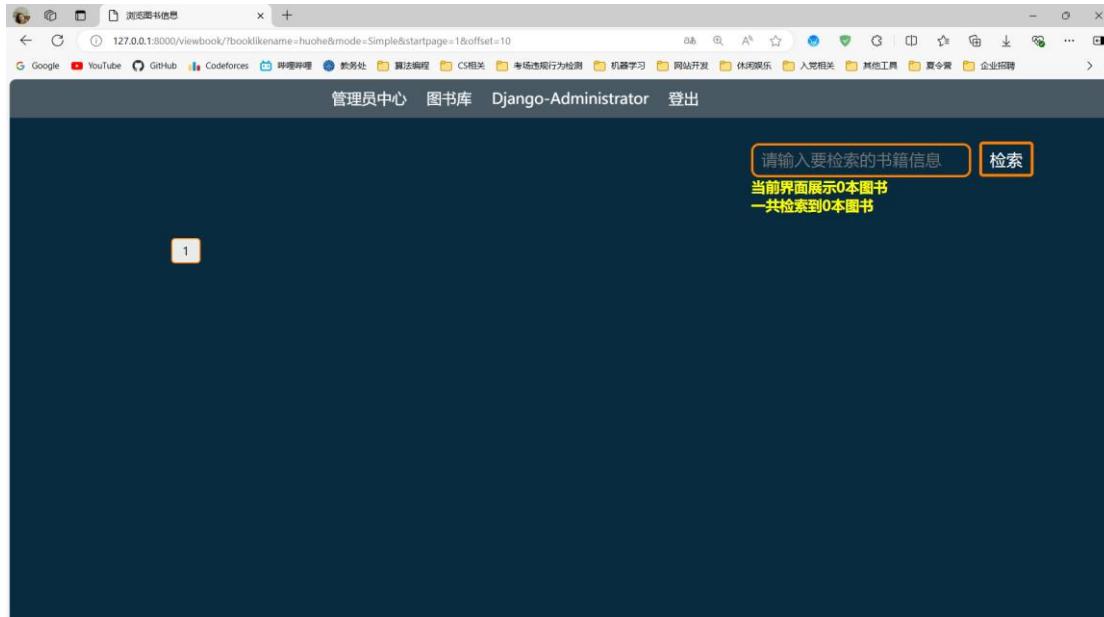


图 4.1.33 检索的图书库中不存在

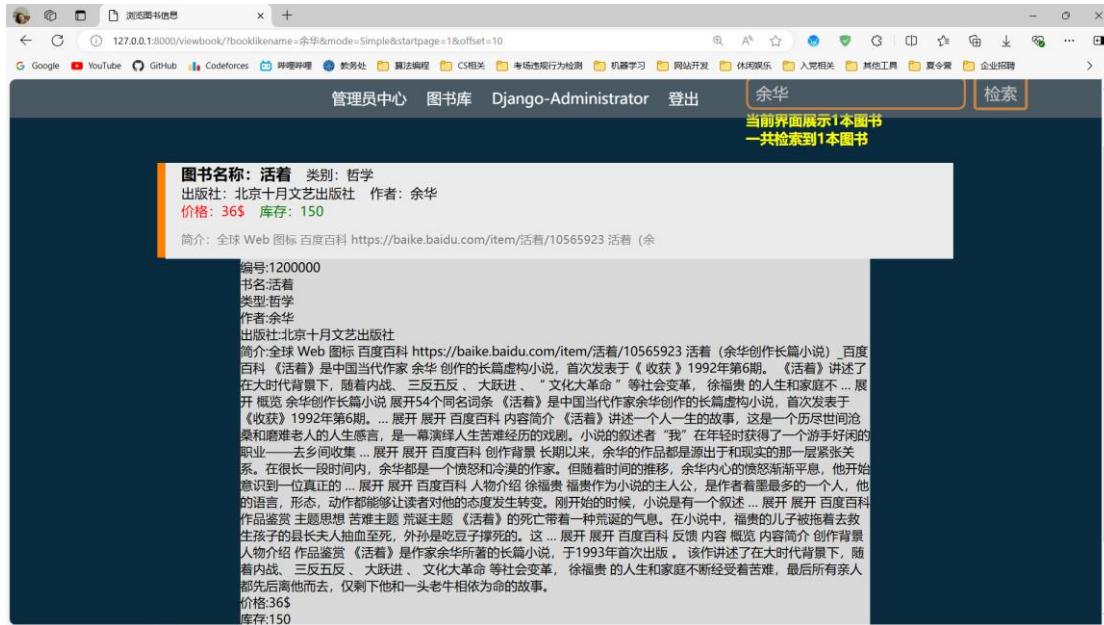


图 4.1.34 检索图书并查看图书的详细信息

### 4.1.6 登出页面

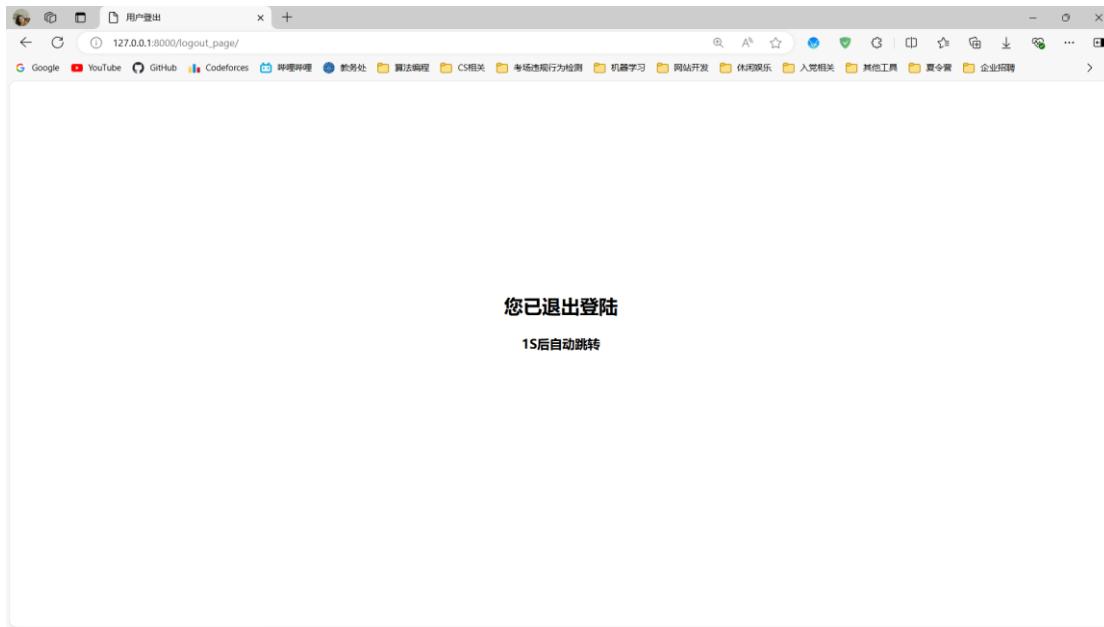


图 4.1.35 登出跳转界面

如上图 4.1.35 所示，当普通用户或者管理员用户点击登出时系统会退出系统，弹出对应的登出提示并在 1s 的倒计时后自动刷新页面并跳转到主页登录页面。

#### 4.1.7 Django-Admin 管理站点

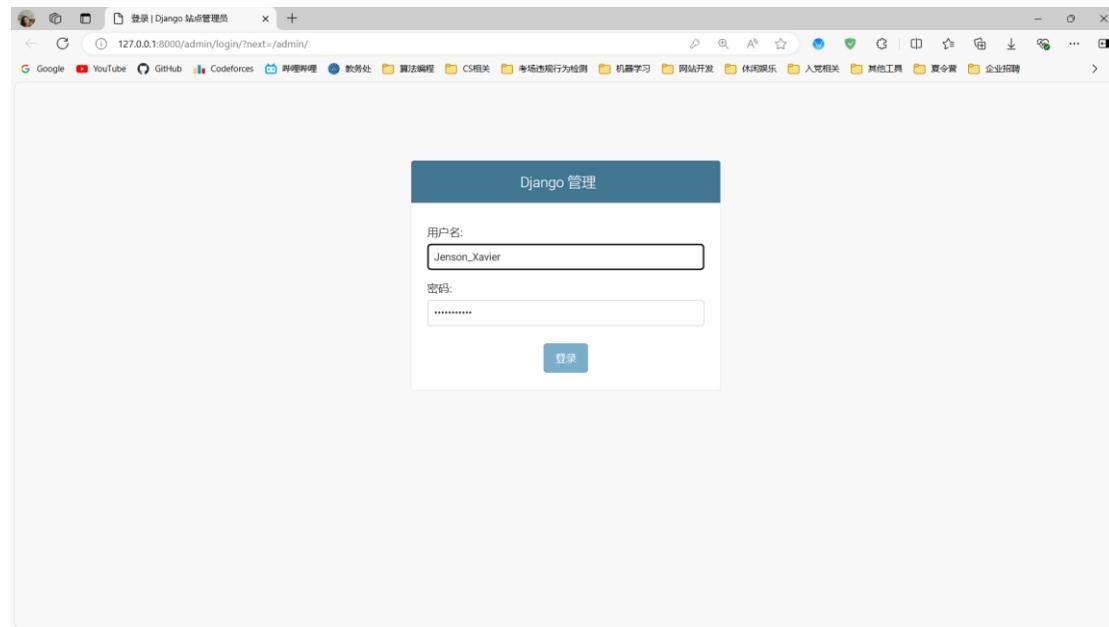


图 4.1.36 Django-Admin 管理站点登录界面

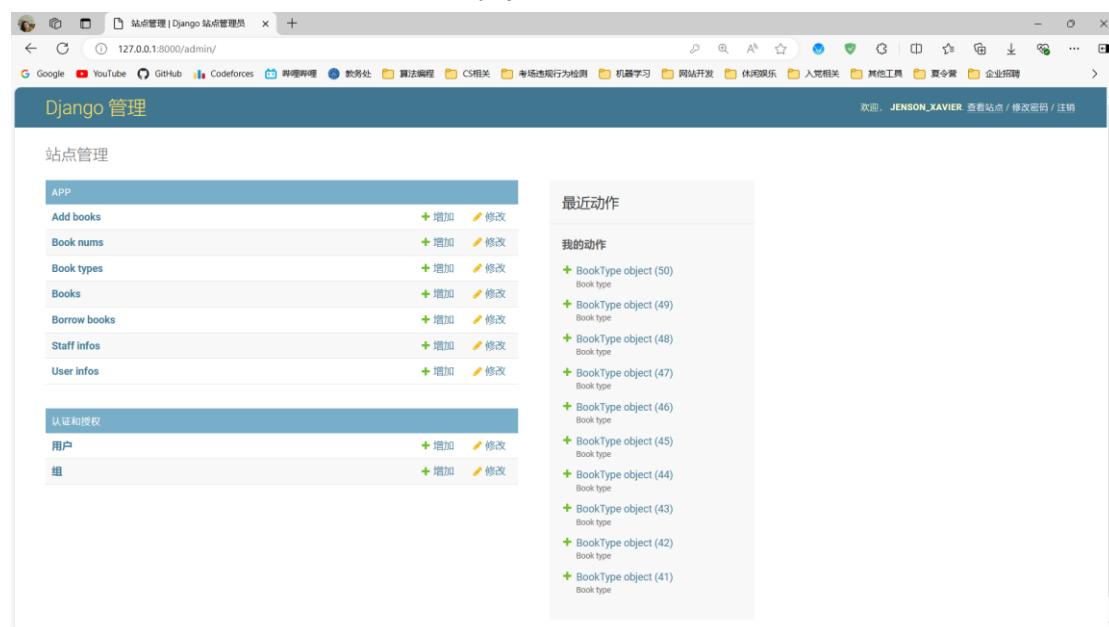


图 4.1.37 Django-Admin 管理站点管理界面

如上图 4.1.36、图 4.1.37 所示，最后简单介绍一下使用 Django 开发数据库驱动类型的项目时，其自带的 Django 管理站点界面，这需要使用创建项目时建立的超级管理员账户的用户名和密码才能够进入，本系统中与管理员用户一致，进入后可以看到自己构建项目所建立的所有模型，并可以自己对模型进行任何操作，可以对任意一个模型添加新数据、删除旧数据、修改已有数据，或者增加某个用户、修改指定用户的信息，或者删除已注册的用户信息，从而实现对某个指定用户的注销功能，也可以进一步给某个用户赋予指定的权限，使其可以进行某些普通用户无法进行的操作，等等。

此站点支持非常复杂且全面的对数据模型的种种操作，由 Django 框架本身自带支持，这里不再赘述。

## 4.2 视图逻辑

上面介绍了本图书管理系统的所有界面及其跳转逻辑，在 Django 中，其后端的逻辑是由其中的视图（V）

iew) 提供的, 视图的主要任务就是接收 HTTP 请求执行相应的业务逻辑之后返回一个 HTTP 响应给客户端, 并且每个视图都需要通过 Django 的 URL 配置来关联一个特定的 URL 模式, 在网站前端实现页面访问和跳转时就是对不同 URL 的访问, 而访问一个 URL 时, Django 就会调用对应的视图函数来处理请求。整个系统还综合运用 HTML 语言以及 CSS 样式表、JavaScript 脚本语言共同实现如上所述的效果, 如果相关技术均面面俱到报告会显得过于冗长且会脱离主题, 且 HTML、JavaScript 等相关内容均与网站显示逻辑与交互反馈有关, 核心逻辑均体现在视图部分, 因此这里重点说明一下在开发网站时所设计的视图逻辑相关内容, 其他相关内容不再赘述。

整个图书管理系统前端的 URL 配置和视图的对应关系如下代码 4.2.1 所示.

```
"""
URL configuration for BMS project.

The 'urlpatterns' list routes URLs to views. For more information please see:
    https://docs.djangoproject.com/en/4.2/topics/http/urls/
Examples:
Function views
    1. Add an import: from my_app import views
    2. Add a URL to urlpatterns: path('', views.home, name='home')
Class-based views
    1. Add an import: from other_app.views import Home
    2. Add a URL to urlpatterns: path('', Home.as_view(), name='home')
Including another URLconf
    1. Import the include() function: from django.urls import include, path
    2. Add a URL to urlpatterns: path('blog/', include('blog.urls'))
"""

from django.contrib import admin
from django.urls import path
from django.conf.urls import *

import app.views

urlpatterns = [
    path('', app.views.login_view, name='main'),
    path('admin/', admin.site.urls),
    path('register_page/', app.views.register_view, name="register_page"),
    path('login_page/', app.views.login_view, name="login_page"),
    path('logout_page/', app.views.logout_view, name='logout_page'),
]
```

```

path('register/', app.views.register, name='register'),
path('login/', app.views.login, name='login'),

path('staff_add_book/', app.views.staff_add_book, name='staff_add_book'),
path('staff_alter_book/', app.views.staff_alter_book, name='staff_alter_book'),
path('viewbook/', app.views.view_bookinfo, name='view_bookinfo'),
path('index/', app.views.index_staff, name='index_staff'),

path('staff_borrow_book/', app.views.staff_borrow_book, name='staff_borrow_book'),
path('staff_return_book/', app.views.staff_return_book, name='staff_return_book'),
path('staff_change_book_info/', app.views.staff_change_book_info,
name='staff_change_book_info'),
path('get_book_types/', app.views.get_book_types, name='get_book_types'),
path('staff_view_user_info/', app.views.staff_view_user_info,
name='staff_view_user_info'),
path('staff_view_user/', app.views.staff_view_user, name='staff_view_user'),

path('pressure_test/', app.views.pressure_test, name='pressure_test'),
]

```

代码 4.2.1 Django 中 URL 配置与视图之间的对应关系

#### 4.2.1 login\_view 视图函数

```

from django.contrib.auth.models import *
from django.core.paginator import *
from django.http import *
from django.shortcuts import render

from app.models import *

# Create your views here.

# 用户登录界面

def login_view(request):
    if request.user.is_authenticated:
        auth.logout(request)
        return render(request, "login.html")
    else:
        return render(request, "login.html")

```

---

#### 代码 4.2.2 login\_view 视图函数代码

如上代码 4.2.2 所示，当用户请求对应登录页面时（即访问相关的 URL），该函数会捕获到请求，即参数中的 request，然后使用 Django 自带的认证系统对用户进行认证，如果当前请求的用户身份是已经登录过的，则退出登录并返回登录页面，否则直接返回登录页面。

#### 4.2.2 logout\_view 视图函数

---

```
# 用户登出界面

def logout_view(request):
    auth.logout(request)
    return render(request, "logout.html")
```

---

#### 代码 4.2.3 logout\_view 视图函数代码

如上代码 4.2.3 所示，当用户请求登出页面时，使当前用户退出登录并返回登出页面即可。

#### 4.2.3 register\_view 视图函数

---

```
# 用户注册界面

def register_view(request):
    return render(request, "register.html")
```

---

#### 代码 4.2.4 register\_view 视图函数代码

如上代码 4.2.4 所示，当用户请求注册页面时，返回注册页面即可。

#### 4.2.4 register 视图函数

---

```
# 用户注册逻辑

def register(request):
    try:
        userid = request.POST['id']
        userpw = request.POST['pw']
        username = request.POST['name']
        usersex = request.POST['sex']

        if usersex == 'True':
            usersex = True
        else:
            usersex = False

        userage = request.POST['age']
        useremail = request.POST['email']
        userphone = request.POST['phone']
```

```

except:

    return render(request, "register.html", {'msgFail': 'GET Error, 注册失败, 请重新操作! '})

try:

    user = User.objects.create_user(username=userid, password=userpw, email=useremail)

    user.is_staff = False

    user.save()

    try:

        UserInfo.objects.create(userId=user, userName=username, userSex=usersex,
                               userAge=userage,
                               userPhone=userphone)

    except Exception as e:

        return render(request, "register.html", {'msgFail': f'注册失败, 请重新操作! 可能原因是:
{e}'})

    return render(request, "login.html", {'msgTrue': '注册成功, 请登录系统! '})

except Exception as e:

    return render(request, "register.html", {'msgFail': f'注册失败, 请重新操作! 可能原因是:
{e}'})

```

---

#### 代码 4.2.5 register 视图函数代码

如上代码 4.2.5 所示, 当用户点击注册时会进入此函数的逻辑, 首先系统会从 request 中捕获用户使用 POST 提交的表单数据, 并尝试使用相关数据在 Django 自带的认证系统 User 模型中创建对应的用户, 并设置 is\_staff 字段为 False, 以表示非管理员用户, 最后在普通用户信息表创建对应的用户信息, 若有某一步骤出现异常会被捕获并返回相应的提示页面。

#### 4.2.5 login 视图函数

---

```

# 用户登录逻辑

def login(request):

    try:

        userId = request.POST['id']

        userPw = request.POST['pw']

    except:

        return render(request, "login.html", {'msgFail': 'POST Error, 现在登出并请重新登录! '})

        user = auth.authenticate(request, username=userId, password=userPw)

        if user is not None and user.is_active:

            auth.login(request, user)

            return HttpResponseRedirect('/index')

        else:

            return render(request, "login.html", {'msgFail': '身份验证失败, 现在登出并请重新登录! '})

```

## 代码 4.2.6 login 视图函数代码

如上代码 4.2.6 所示，当用户点击登陆时，系统会从 Http 请求 request 中捕获用户以 POST 方式提交的表单信息，并用得到的用户 ID 和用户密码进行身份验证，只有当身份验证通过时才会跳转到主页，否则返回出错的提示信息。

## 4.2.6 view\_bookinfo 视图函数

```
# 查看书籍信息的视图逻辑

def view_bookinfo(request):
    if request.user.is_authenticated: # django 验证身份
        cur_user = request.user
        try:
            if cur_user.is_staff: # 验证管理员
                staff_flag = True
                username = StaffInfo.objects.get(staffId=cur_user).staffName
            else:
                staff_flag = False
                username = UserInfo.objects.get(userId=cur_user).userName
            cur_user_info = {"userId": cur_user.username, "userName": username} # cur_user 是
request 中的 user 类型
        except:
            return JsonResponse({'end': False, 'msg': '数据库中不存在所查找的数据!'})

        try:
            mode = request.GET['mode']
            startpage = request.GET['startpage']
            offset = request.GET['offset']
            booklikename = request.GET['booklikename']
            # 由于 Simple 和 Detail 在同一个视图处理 故这里的空输入判断移交到 js 脚本文件中进行判断
        except:
            return JsonResponse({'end': False, 'msg': 'GET Error!'})

        results = []
        if mode == 'Detail': # 详细显示某个书籍的具体信息
            try:
                get_book = Book.objects.get(bookId=booklikename)
                get_book_num_fk = BookNum.objects.get(bookId=get_book)
                book_detail = {"bookId": get_book.bookId,

```

```

        "bookName": get_book.bookName,
        "bookType": get_book.bookType.TypeName,
        "bookPublisher": get_book.bookPublisher,
        "bookAuthor": get_book.bookAuthor,
        "bookIntroduction": get_book.bookIntroduction,
        "bookPrice": get_book.bookPrice,
        "bookNum": get_book_num_fk.bookNum}

    return JsonResponse({'end': True, 'msg': book_detail})

except:

    return JsonResponse({'end': False, 'msg': '数据库中不存在所查找的数据!'})

elif mode == 'Simple': # 用于图书馆 library 界面所有书籍信息的展示

    cur_page = int(startpage)

    try:

        if booklikename == '':
            book_all = Book.objects.all().order_by('bookName')
            all_Items = Book.objects.all().count()

        else:
            if '书' in booklikename:
                book_all =
Book.objects.filter(bookName__contains=booklikename).order_by('bookName')
                all_Items = book_all.count()

            elif '作者' in booklikename:
                book_all =
Book.objects.filter(bookAuthor__contains=booklikename).order_by('bookName')
                all_Items = book_all.count()

            elif '出版社' in booklikename:
                book_all =
Book.objects.filter(bookPublisher__contains=booklikename).order_by('bookName')
                all_Items = book_all.count()

            else:
                book_all1 =
Book.objects.filter(bookName__contains=booklikename).order_by('bookName')
                book_all2 =
Book.objects.filter(bookAuthor__contains=booklikename).order_by('bookName')
                book_all3 =
Book.objects.filter(bookPublisher__contains=booklikename).order_by('bookName')
                book_all = []
                book_all.extend(book_all1)
                book_all.extend(book_all2)
                book_all.extend(book_all3)

```

```

book_all.extend(book_all3)

all_Items = book_all1.count() + book_all2.count() + book_all3.count()

except:

    return HttpResponse('数据库中不存在所查找的数据!')

try:

    book_pages = Paginator(book_all, int(offset)) # 创建分页器 用于页面显示时分页

    pagen = book_pages.page(cur_page)

    pagenum = book_pages.page_range[-1]

    after_range_num = 5

    before_range_num = 4

    book_results = pagen.object_list

    if cur_page - 1 > after_range_num:

        pagelist = book_pages.page_range[cur_page - 1 - after_range_num:cur_page - 1

+ before_range_num]

    else:

        pagelist = book_pages.page_range[0:cur_page - 1 + before_range_num]

    for book_result in book_results:

        book_num_fk = BookNum.objects.get(bookId=book_result)

        book_detail = {"bookName": book_result.bookName,

                      "bookId": book_result.bookId,

                      "bookType": book_result.bookType.TypeName,

                      "bookPublisher": book_result.bookPublisher,

                      "bookAuthor": book_result.bookAuthor,

                      "bookPrice": book_result.bookPrice,

                      "bookNum": book_num_fk.bookNum,

                      "bookIntroduction": book_result.bookIntroduction[:60]}

        results.append(book_detail)

    return render(request, 'library.html', {"currentUser": cur_user_info,

                                             "is_staff": staff_flag,

                                             "bookList": results,

                                             "bookLikeName": booklikename,

                                             "pageList": pagelist,

                                             "currentPage": cur_page,

                                             "nextPage": cur_page + 1,

                                             "prevPage": cur_page - 1,

                                             "endPage": pagenum,

                                             "curpageItems": len(results),

                                             "allpageItems": all_Items})

```

```

except:
    return HttpResponse('图书数据信息查找失败!')
else:
    return HttpResponseRedirect('/login_page') # 重定向

```

代码 4.2.7 view\_bookinfo 视图函数代码

如上代码 4.2.7 所示，此视图函数是用户显示图书信息的，根据用户提交请求的信息，首先进行身份验证，通过后系统会捕获用户以 GET 方式提交的请求中的相关参数，根据 mode 参数的值决定如何返回页面。

当 mode 参数为 “Detail” 时，说明是需要详细展示某个具体图书的详细信息，根据请求 URL 中 booklename 参数中的值在数据库查找对应符合条件的图书，若存在则返回对应的数据信息，以 Json 格式返回给 JavaScript 脚本文件进行处理后反馈到前端，否则返回出错的提示信息。

当 mode 参数为 “Simple” 时，说明是在图书库中查找指定的图书，并将所有找到的图书以缩略条目的形式进行展示，此时对检索得到的结果创建分页器，并将检索到的结果反馈到前端。这里在查找数据时检索了 Book 表中书名、出版社和图书作者所包含指定信息的条目，从而实现前端用户对书名、出版社和作者的模糊联合搜索。若在处理过程中出现任何错误异常均会被 Python 捕获到并返回对应的出错提示信息。

#### 4.2.7 staff\_add\_book 视图函数

```

# 管理员添加书籍的视图逻辑

def staff_add_book(request):
    if request.user.is_authenticated: # django 验证身份
        cur_user = request.user
        if cur_user.is_staff: # 验证管理员
            try:
                bookid = request.GET['bookid']
                bookname = request.GET['bookname']
                booktype = request.GET['booktype']
                bookpublisher = request.GET['bookpublisher']
                bookauthor = request.GET['bookauthor']
                bookintroduction = request.GET['bookintroduction']
                bookprice = request.GET['bookprice']
                booknum = request.GET['booknum']

                if bookid == '' or bookname == '' or booktype == '' or bookpublisher == '' \
                    or bookauthor == '' or bookintroduction == '' or bookprice == '' or
                booknum == '':
                    return JsonResponse({'end': True, 'msg': '请输入数据再进行操作!'})
            except:
                return JsonResponse({'end': False, 'msg': 'GET Error!'})
            try:
                booktype_fk = BookType.objects.get(TypeId=booktype)

```

```

book_added = Book.objects.create(bookId=bookid,
                                 bookName=bookname,
                                 bookType=booktype_fk,
                                 bookPublisher=bookpublisher,
                                 bookAuthor=bookauthor,
                                 bookIntroduction=bookintroduction,
                                 bookPrice=bookprice)

BookNum.objects.create(bookId=book_added, bookNum=booknum)

AddBook.objects.create(bookId=book_added,
                      staffId=StaffInfo.objects.get(staffId=cur_user),
                      addNum=booknum)

return JsonResponse({'end': True, 'msg': '添加书籍成功!'})

except:

    return JsonResponse({'end': False, 'msg': '添加书籍失败, 请注意输入格式并不要输入重复
书籍 ID, 请重试!'})

else:

    return JsonResponse({'end': False, 'msg': 'Permission Denied!'})

else:

    return JsonResponse({'end': False, 'msg': 'Permission Denied!'})

```

代码 4.2.8 staff\_add\_book 视图函数代码

如上代码 4.2.8 所示, 当管理员添加书籍时会进入此函数进行处理。首先验证用过户身份是否为管理员, 若不是则直接返回, 否则去捕获管理员以 GET 方式请求添加的书籍相关信息, 并尝试在 Book 表进行创建, 同时需要维护其他相关的表比如 BookNum、AddBook 中的对应信息, 成功添加或者出现错误均会返回对应的提示信息页面。

#### 4.2.8 staff\_alter\_book 视图函数

```

# 管理员修改书籍相关的属性的视图逻辑
# 主要是增加图书数量这一操作

def staff_alter_book(request):

    if request.user.is_authenticated:

        cur_user = request.user

        if cur_user.is_staff: # 验证为管理员则可以修改

            try:

                bookid = request.GET['bookid']

                addbooknum = request.GET['addbooknum']

                if bookid == '' or addbooknum == '':
                    return JsonResponse({'end': True, 'msg': '请输入数据再进行操作!'})


```

```

except:

    return JsonResponse({'end': False, 'msg': 'GET Error!'})

try: # try 查找数据

    book_fk = Book.objects.get(bookId=bookid)

    staffinfo_fk = StaffInfo.objects.get(staffId=cur_user)

    AddBook.objects.create(bookId=book_fk,
                          staffId=staffinfo_fk,
                          addNum=addbooknum)

except:

    return JsonResponse({'end': False, 'msg': '数据库中不存在所查找的数据!'})

try:

    try: # try 修改数据

        changebooknum = BookNum.objects.get(bookId__bookId=bookid)

        changebooknum.bookNum += int(addbooknum)

        changebooknum.save()

    except: # 否则新增对应的数据

        BookNum.objects.create(bookId=book_fk, bookNum=addbooknum)

    return JsonResponse({'end': True, 'msg': '添加图书数量成功!'})

except:

    return JsonResponse({'end': False, 'msg': '添加图书失败, 请重新操作!'})

else:

    auth.logout(request) # 非管理员退出

    return JsonResponse({'end': False, 'msg': 'Permission Denied!'})

else:

    return JsonResponse({'end': False, 'msg': 'Permission Denied!'})

```

代码 4.2.9 staff\_alter\_book 视图函数代码

如上代码 4.2.9 所示, 这是管理员用于增加图书数量的处理函数, 当管理员身份验证通过后, 从请求中捕获对应的参数信息, 并从 Book 表中查找对应的信息是否存在相关的数据, 若存在则执行相应的修改, 若出现错误均会被捕获到并返回出错的提示信息到前端。

#### 4.2.9 staff\_borrow\_book 视图函数

```

# 管理员负责用户借书

def staff_borrow_book(request):

    if request.user.is_authenticated:

        cur_user = request.user

        if cur_user.is_staff:

            try:

                bookid = request.GET['bookid'] # 用户所借书的信息

```

```

userid = request.GET['userid'] # 需要进行借书的用户
if bookid == '' or userid == '': # 用户没有输入数据时
    return JsonResponse({'end': True, 'msg': '请输入数据再进行操作!'})
except:
    return JsonResponse({'end': False, 'msg': 'GET Error!'})

try:
    booknum = BookNum.objects.get(bookId__bookId=bookid)
except:
    return JsonResponse({'end': False, 'msg': '数据库中不存在所查找的数据!'})

if booknum.bookNum == 0:
    return JsonResponse({'end': False, 'msg': '所借书数据库中无法提供更多的数量!'})

try:
    borrowbook_len = len(BorrowBook.objects.all())
    borrowbook = BorrowBook.objects.create(borrowId=borrowbook_len + 1,
                                           bookId=Book.objects.get(bookId=bookid),
                                           staffId=StaffInfo.objects.get(staffId=cur_user),
                                           userId=UserInfo.objects.get(userId__username=userid),
                                           hasReturned=False)

    booknum.bookNum -= 1
    booknum.save()

    return JsonResponse({'end': True, 'msg': '借书成功!'})
except:
    return JsonResponse({'end': False, 'msg': '借书失败, 请重新操作!'})

else:
    return JsonResponse({'end': False, 'msg': 'Permission Denied!'})

else:
    return JsonResponse({'end': False, 'msg': 'Permission Denied!'})

```

代码 4.2.10 staff\_borrow\_book 视图函数代码

如上代码 4.2.10 所示, 管理员负责借书操作时, 会进入此函数进行处理。首先验证管理员的身份, 通过身份验证后捕获前端输入的数据, 判断输入数据格式是否正确并到相关数据表中进行检索, 判断当前要借出的图书数量是否足以借出, 当没有问题时进行借出操作, 并修改相关表中的数据记录, 在 BorrowBook 表创建对应的借书即可, 返回对应的提示信息反馈到前端。

#### 4.2.10 staff\_return\_book 视图函数

---

```
# 管理员负责用户还书
```

```

def staff_return_book(request):
    if request.user.is_authenticated:
        cur_user = request.user
        if cur_user.is_staff:
            try:
                bookid = request.GET['bookid'] # 用户所还书的信息
                userid = request.GET['userid'] # 需要进行还书的用户
                if bookid == '' or userid == '':
                    return JsonResponse({'end': True, 'msg': '请输入数据再进行操作!'})
            except:
                return JsonResponse({'end': False, 'msg': 'GET Error!'})

            try:
                booknum = BookNum.objects.get(bookId__bookId=bookid)
                borrowbooks = BorrowBook.objects.filter(bookId__bookId=bookid,
                userId__userId__username=userid).exclude(hasReturned=True)

            except:
                return JsonResponse({'end': False, 'msg': '数据库中不存在所查找的数据!'})

            if len(borrowbooks) == 0:
                return JsonResponse({'end': False, 'msg': '不存在可以还的书!'}) # 无可还的书

            try:
                for borrowbook in borrowbooks:
                    if borrowbook.hasReturned:
                        continue
                    borrowbook.hasReturned = True
                    borrowbook.save()
                    booknum.bookNum += 1
                    booknum.save()

                return JsonResponse({'end': True, 'msg': '还书成功!'})
            except:
                return JsonResponse({'end': False, 'msg': '还书失败, 请重新操作!'})

        else:
            return JsonResponse({'end': False, 'msg': 'Permission Denied!'})

    else:
        return JsonResponse({'end': False, 'msg': 'Permission Denied!'})

```

代码 4.2.11 staff\_return\_book 视图函数代码

如上代码 4.2.11 所示，还书的逻辑与借书的逻辑基本一致，通过输入的用户 ID 和图书 ID 在借书记录

表中进行查询，从而得到所有用户 ID 所借指定 ID 的图书的记录，这部分记录就是要还的所有书籍，然后遍历这一记录，对每个记录判断是否已还，若存在未还的将其归还，修改 hasReturned 字段为 True，并将对应图书的库存数量增加，保存修改同步到数据库中，若中途出现错误，则会捕获到异常并将出错信息反馈到前端。

#### 4.2.11 get\_book\_types 视图函数

```
# 获取书籍类型

def get_book_types(request):
    if request.user.is_authenticated:
        book_type_id = []
        book_type_info = []
        types = BookType.objects.all()
        for type in types:
            book_type_id.append(type.TypeId)
            book_type_info.append(type.TypeName)
        return JsonResponse({'typeId': book_type_id, 'typeInfo': book_type_info})
    else:
        return HttpResponse('Permission Denied!')
```

代码 4.2.12 get\_book\_types 视图函数代码

如上代码 4.2.12 所示，当需要获取预设的所有图书类型时会进入这个函数，这个函数在身份验证通过后会查询 BookType 表中所有的记录来获取已有的图书类型，并将图书类型 ID 和具体的图书类型作为 Json 格式的响应返回给 JavaScript 脚本进行处理。

#### 4.2.12 staff\_view\_user\_info 视图函数

```
# 查看用户信息
# 当以非管理员用户身份登录时会以此视图进行跳转

def staff_view_user_info(request):
    if request.user.is_authenticated:
        cur_user = request.user
        if cur_user.is_staff: # 管理员
            try:
                userid = request.GET['userid']
            except:
                return HttpResponseRedirect('/login_page')
            cur_login_user = {'userId': cur_user.username,
                             'userName': StaffInfo.objects.get(staffId=cur_user).staffName}
            auth_user = User.objects.get(username=userid)
```

```

user_info_fk = UserInfo.objects.get(userId=auth_user)

borrow_books = BorrowBook.objects.filter(userId=user_info_fk).order_by('borrowId')

user_info = {'userid': auth_user.username,
            'username': user_info_fk.userName,
            'email': auth_user.email,
            'sex': user_info_fk.userSex,
            'age': user_info_fk.userAge,
            'pn': user_info_fk.userPhone,
            'time': user_info_fk.userRegisterTime.strftime('%Y-%m-%d')}

borrow_info_list = []

for i, borrow_book in enumerate(borrow_books):
    borrow_info_list.append({'bookName': borrow_book.bookId.bookName,
                            'itemNum': i + 1,
                            'staffId': borrow_book.staffId.staffId.username,
                            'hasReturn': borrow_book.hasReturned})

borrow_info_list.reverse()

return render(request, 'userdetail.html', {'userInfo': user_info,
                                            'borrowList': borrow_info_list,
                                            'currentUser': cur_login_user,
                                            'is_staff': True})

else: # 非管理员

    userid = cur_user.username

    cur_login_user = {'userId': cur_user.username,
                      'userName': UserInfo.objects.get(userId=cur_user).userName}

    auth_user = User.objects.get(username=userid)

    user_info_fk = UserInfo.objects.get(userId=auth_user)

    borrow_books = BorrowBook.objects.filter(userId=user_info_fk).order_by('borrowId')

    user_info = {'userid': auth_user.username,
                'username': user_info_fk.userName,
                'email': auth_user.email,
                'sex': user_info_fk.userSex,
                'age': user_info_fk.userAge,
                'pn': user_info_fk.userPhone,
                'time': user_info_fk.userRegisterTime.strftime('%Y-%m-%d')}

    borrow_info_list = []

    for i, borrow_book in enumerate(borrow_books):
        borrow_info_list.append({'bookName': borrow_book.bookId.bookName,
                                'itemNum': i + 1,
                                'staffId': borrow_book.staffId.staffId.username,

```

```

        'hasReturn': borrow_book.hasReturned})
borrow_info_list.reverse()

return render(request, 'userdetail.html', {'userInfo': user_info,
                                             'borrowList': borrow_info_list,
                                             'currentUser': cur_login_user,
                                             'is_staff': False})

else:

    return HttpResponseRedirect('/login_page')

```

代码 4.2.13 staff\_view\_user\_info 视图函数代码

如上代码 4.2.13 所示，此视图函数是用来浏览用户信息的，当用户的身份通过验证后，需要判断当前的用户身份是管理员用户还是普通用户，如果是管理员用户，首先从 Http 请求中捕获以 GET 方法获取的 userid 参数，并从 StaffInfo 表中查找当前管理员用户的信息，并通过获取的 userid 参数找到其对应的在 User 表中的条目，进而找到在 UserInfo 表中的数据，然后从 BorrowBook 表获取指定用户的借书记录，最后填充相应数据作为参数返回给“userdetail.html”页面进行渲染。

若当前用户是普通用户，和上述管理员构建数据的流程一样，不过是需要区分一下普通用户的用户信息表为 UserInfo，此函数的主要作用是实现当普通用户登录之后的页面显示逻辑，并当管理员需要查看某个用户的具体信息时，会访问到对应的 URL 以跳到此函数进行业务逻辑的处理。

#### 4.2.13 staff\_view\_user 视图函数

```

# 管理员查看用户
# 此视图逻辑用于在管理员界面浏览所有注册的用户信息

def staff_view_user(request):
    if request.user.is_authenticated:
        cur_user = request.user
        if cur_user.is_staff:
            try:
                username = request.GET['username']
                offset = request.GET['offset']
                startpage = request.GET['startpage']
            except:
                return HttpResponseRedirect('GET Error!')
            cur_page = int(startpage)
            if username == '':
                scan_user = UserInfo.objects.all().order_by('user_name')
            else:
                scan_user = UserInfo.objects.filter(user_name__contains=username)
            user_pagi = Paginator(scan_user, int(offset))

```

```

pageN = user_pagi.page(cur_page)
after_page_num = 5
before_page_num = 4
scan_user_list = pageN.object_list
results = []
for useri in scan_user_list:
    results.append({'userid': useri.userId.username,
                    'username': useri.userName,
                    'usersex': useri.userSex,
                    'userage': useri.userAge,
                    'userphone': useri.userPhone})

if cur_page - 1 > after_page_num:
    page_list = user_pagi.page_range[cur_page - 1 - after_page_num:cur_page - 1 +
before_page_num]

else:
    page_list = user_pagi.page_range[0:cur_page - 1 + before_page_num]
ret = {'userList': results, 'pageList': list(page_list), 'currentPage': cur_page}
return JsonResponse(ret)

else:
    return HttpResponse('Permission Denied!')

else:
    return HttpResponse('Permission Denied!')

```

代码 4.2.14 staff\_view\_user 视图函数代码

如上代码 4.2.14 所示，此函数的作用是在管理员中心界面以分页的方式显示用户的基本信息，当用户通过 Django 的身份认证并且确认为管理员用户后，从 Http 请求中获取以 GET 方法提交的参数字段的值，包括 username 参数、offset 参数、startpage 参数，后两个参数用于分页显示的处理，第一个 username 参数用于显示指定 username 用户的信息，若为空则说明要显示全部用户的信息，然后就是从 UserInfo 表中获取数据构建分页器，将查找得到的结果返回给前端进行渲染和展示。

#### 4.2.14 staff\_change\_book\_info 视图函数

```

# 管理员修改书籍的任意属性

def staff_change_book_info(request):
    if request.user.is_authenticated:
        cur_user = request.user
        if cur_user.is_staff:
            try:
                bookid = request.GET['bookid']
                bookname = request.GET['bookname']

```

```

booktype = request.GET['booktype']
bookpub = request.GET['bookpublisher']
bookauthor = request.GET['bookauthor']
bookintro = request.GET['bookintroduction']
bookprice = request.GET['bookprice']
booknum = request.GET['booknum']

if bookid == '' or bookname == '' or booktype == '' or bookpub == '' \
    or bookauthor == '' or bookintro == '' or bookprice == '' or booknum == '':
    return JsonResponse({'end': True, 'msg': '请输入数据再进行操作!'})

except:
    return JsonResponse({'end': False, 'msg': 'GET Error!'})

try:
    book_fk = Book.objects.get(bookId=bookid)
    booknum_fk = BookNum.objects.get(bookId__bookId=bookid)
    book_fk.bookName = bookname
    book_fk.bookType = BookType.objects.get(TypeId=booktype)
    book_fk.bookPublisher = bookpub
    book_fk.bookAuthor = bookauthor
    book_fk.bookIntroduction = bookintro
    book_fk.bookPrice = bookprice
    book_fk.save()
    booknum_fk.bookNum = booknum
    booknum_fk.save()

    return JsonResponse({'end': True, 'msg': '更新书籍信息成功!'})

except:
    return JsonResponse({'end': False, 'msg': '输入格式有误，更新操作失败，请重试!'})

else:
    return JsonResponse({'end': False, 'msg': 'Permission Denied!'})

else:
    return JsonResponse({'end': False, 'msg': 'Permission Denied!'})

```

代码 4.2.15 staff\_change\_book\_info 视图函数代码

如上代码 4.2.15 所示，这是用于管理员修改图书信息的视图函数，基本的逻辑是比较简明的，首先进行身份验证和管理员身份判断，然后从 Http 请求中捕获用户在前端页面输入的数据，判断数据是否合规，若合规则使用 bookid 从 Book 表中查找指定图书，利用 BookNum 表中所建的外键查找对应图书的数量信息，从 BookType 表中查找得到要修改的图书类型，然后修改对应图书的各个属性值，并将结果保存，若产生错误会将出错信息反馈到前端。

#### 4.2.15 index\_staff 视图函数

```
# 主界面 用于管理员

def index_staff(request):
    if request.user.is_authenticated:
        cur_user = request.user
        if cur_user.is_staff:
            staff_info = StaffInfo.objects.get(staffId=cur_user)
            types = []
            for type in BookType.objects.all():
                types.append({'typeNum': type.TypeId, 'typeInfo': type.TypeName})
            cur_user_info = {'userId': cur_user.username, 'username': staff_info.staffName}
            return render(request, 'indexstaff.html', {'currentUser': cur_user_info,
                                                         'is_staff': True,
                                                         'typeList': types})
        else:
            return HttpResponseRedirect('/staff_view_user_info') # 网站主页面为 index 当用户非管理员时会跳转到 user_info 视图导向的界面
    else:
        return HttpResponseRedirect('/login_page')
```

代码 4.2.16 index\_staff 视图函数代码

如上代码 4.2.16 所示，这是显示用户登录后的主页的视图函数逻辑，用户身份验证通过后判断是否为管理员，若为管理员则查找得到渲染“indexstaff.html”页面所需要的参数并传递给前端，显示管理员中心页面；否则重定向到 URL 为“/staff\_view\_user\_info”，执行上述 staff\_view\_user\_info 视图函数进行处理，以返回普通用户登录后的普通用户中心页面。若身份验证不通过则返回到登录页面，等待用户的重新登录操作。

#### 4.2.16 pressure\_test 视图函数

```
from faker import Faker
import random
import pandas as pd

# 百万条数据的压力测试

def pressure_test(request):
    isTest = False # 是否进行测试 设置为真时执行会添加数据
    if isTest:
        numTestEpoll = 600 # 每轮测试添加的数据数量 书籍表中的前 100 条书籍编号保留用于人工测试 后 100w 条
        # 用于随机生成添加
```

```

bookTypes = BookType.objects.all()

BOOKID = 980000 # 自定义开始添加数据的编号

fake = Faker('zh_CN') # 设置随机数据生成器

cur_user = request.user

try:

    for bookType in bookTypes: # 对每个书籍类型依次进行添加

        for i in range(numTestEpoll):

            testID = str(BOOKID)

            bookid = BOOKID

            BOOKID += 1

            bookname = fake.company_prefix()

            # bookname = '测试书' + testID

            booktype = bookType

            bookpublisher = fake.company()

            # bookpublisher = '测试出版社' + testID

            bookauthor = fake.name()

            # bookauthor = '测试作者' + testID

            bookintroduction = fake.sentence()

            # bookintroduction = '测试简介' + testID

            # bookprice = fake.random_int(min=20, max=200)

            bookprice = random.randint(20, 200)

            # booknum = fake.random_int(min=50, max=1000)

            booknum = random.randint(50, 1000)

            book_added = Book.objects.create(bookId=bookid,
                                              bookName=bookname,
                                              bookType=booktype,
                                              bookPublisher=bookpublisher,
                                              bookAuthor=bookauthor,
                                              bookIntroduction=bookintroduction,
                                              bookPrice=bookprice)

            BookNum.objects.create(bookId=book_added, bookNum=booknum)

            AddBook.objects.create(bookId=book_added,
                                  staffId=StaffInfo.objects.get(staffId=cur_user),
                                  addNum=booknum)

    return HttpResponse('Pressure Test Success')

except:

```

```

    return HttpResponse('Pressure Test Fail!')

else:

    return HttpResponse('Pressure Test Has Done!')

```

代码 4.2.17 pressure\_test 视图函数代码

如上代码 4.2.17 所示，最后一个函数是用于进行压力测试的，当用户发出了 URL 为 “/pressure\_test” 的请求之后会跳转到此函数进行处理，这个函数就是通过随机手段和循环方式以不断在 Book 表中新增图书，同时需要同步更新 BookNum 表和 AddBook 表，确保有关联的表之间数据的同步性。以此产生规模达到 1000000 本的图书数据量用于压力测试，在产生了需求的数据量之后此视图函数和对应的 URL 将不会再提供到前端。

## 5 压力测试

本图书管理系统还实现了压力测试，向数据库中的 Book 图书信息表添加了约 100 万本图书的数据量，同时与 Book 相关联的 AddBook 表和 BookNum 表也要添加相对应的每本图书的记录，最终 Book 表、AddBook 表和 BookNum 均含有 100 万条数据量，在前端图书库页面展示如下图 5.1 所示。



图 5.1 进行压力测试时图书库主页面

对达到 100 万规模的数据量，实际运行效果也是非常良好的，当用户输入书名、出版社或者作者进行模糊搜索时，对于任意的合法输入，点击检索后基本瞬间就可以检索到相关的结果并反馈到前端完成渲染。不会给系统的使用带来任何的卡顿或延时，压力测试水平表现优异，相关界面如下图 5.2、图 5.3、图 5.4、图 5.5 所示。

## 数据库原理实验报告

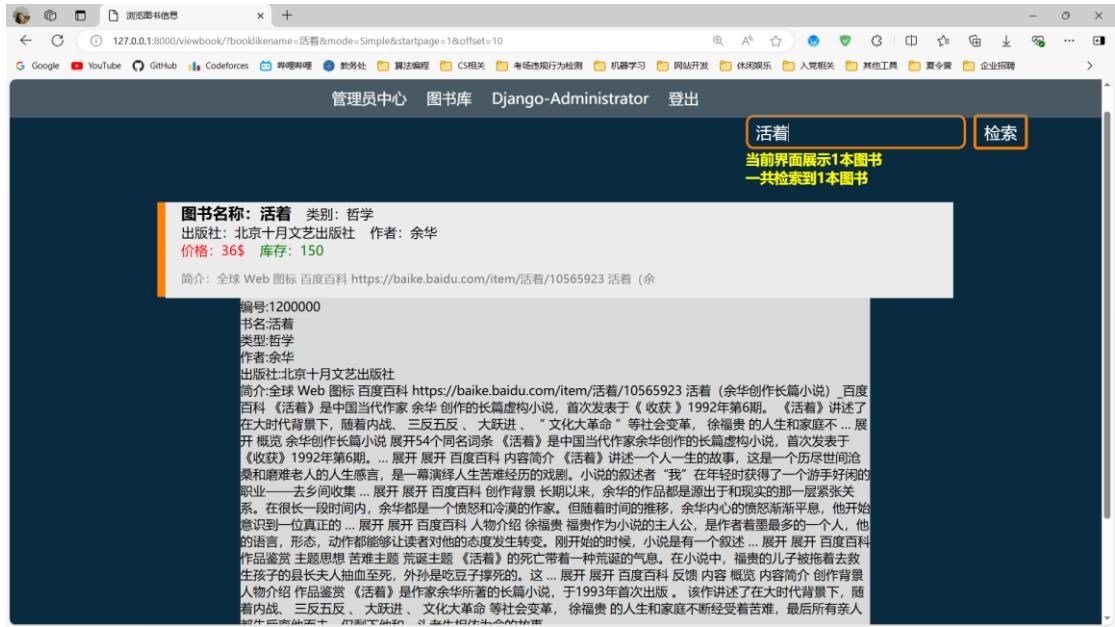


图 5.2 从百万本图书中检索指定书名唯一的图书



图 5.3 从百万本图书中检索多本重复书名的图书

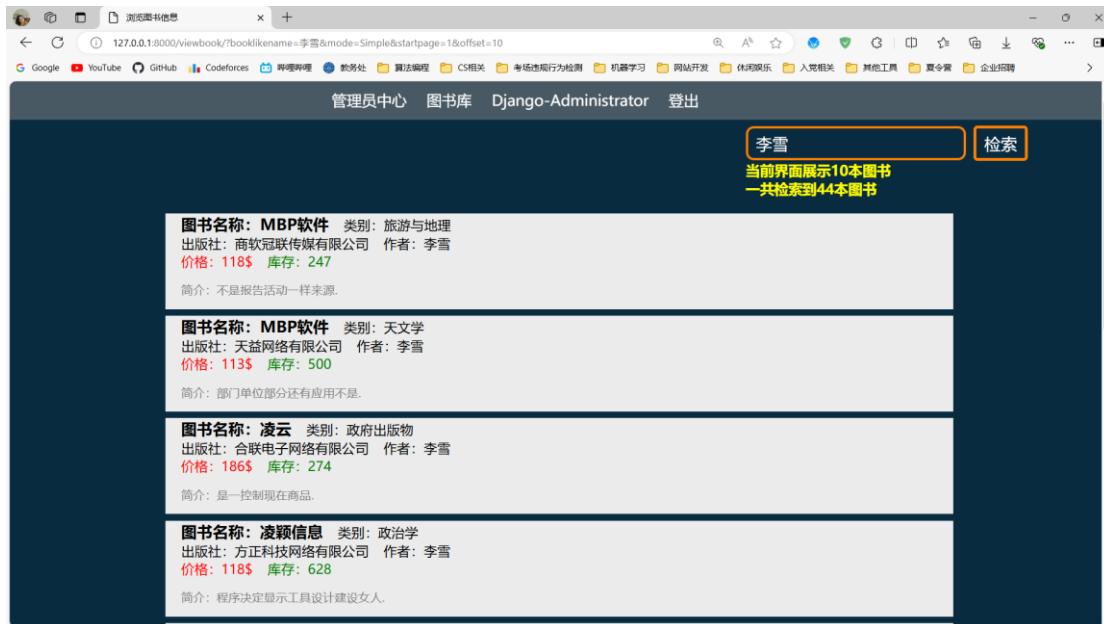


图 5.4 从百万本图书中检索指定作者的图书

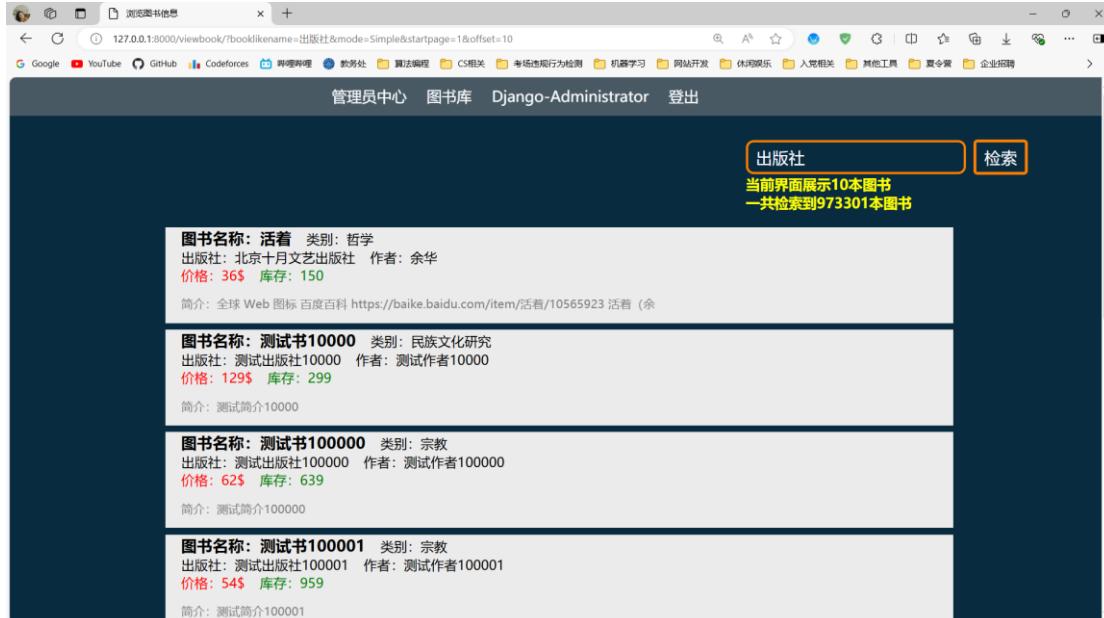


图 5.5 从百万本图书中检索含有指定出版社的图书

## 6 总结与展望

本次数据库的实验自己独立完成了所有的实验内容，其中实验一和实验五为基础实验，实验六为综合实验，在综合实验中自己设计并完成了一个小型的图书管理系统，数据库使用的实验要求的 OpenGauss 数据库，可视化界面是使用 Web 界面实现的，开发网站所用的框架为 Python 的 Django 框架，为提升页面的展示效果和网页 Web 的交互效果，还综合运用了 HTML 语言、CSS 层叠样式表、JavaScript 脚本语言等技术。

在本次实验中，自己设计的图书管理系统具有小型信息管理系统最基本的信息管理功能，普通用户可以通过注册成为系统的使用用户，在用户层面上实现了普通用户和管理员用户权限的划分，两类用户角色拥有不同的权限和功能，普通用户可以查看图书信息和查看个人信息，管理员用户可以对数据库中的数据进行修改，包括增加图书、修改图书信息、增加图书数量等功能，利用 Django 自带的 Admin 管理站点也可以对

所有创建的数据模型进行任意的修改操作。在核心功能图书的显示方面也是实现了压力测试，向数据库插入了百万条的图书条目，并提供了对图书书名、作者和出版社的联合模糊搜索，在提供便利性的同时也保证了系统运行的效率。同时 CSS 层叠样式表和 JavaScript 脚本语言的使用也让用户在使用本图书管理系统时拥有良好的反馈体验，对可能出现的错误都做了报错提示和异常处理，正确的操作也会有相对应的操作成功的提示信息反馈到前端页面中。同时，还增加了借书模块，在管理员的操作下可以为指定用户借出指定图书，也可以将指定图书进行归还。

不过此图书管理系统仍然存在不足和改进的空间，比如普通用户和管理员用户的权限划分过于严格，并且管理员用户的权限过于集中，导致普通用户可以使用的功能较少，期望后续设计更改的功能和更细的权限划分，在保证权限安全时适当增大普通用户的权限；整个图书管理系统的功能还是有所欠缺，期望后续向其中加入更多适用的功能，并且进一步精华当前已有的功能，比如图书馆的检索功能目前只支持书名、作者和出版社的简单联合模糊搜索，后续计划提供更精细的搜索功能，并提升检索效率；整个图书管理系统的页面布局和美化效果还可以进一步加以改进……

总而言之，通过这次的数据库实验，自己不仅熟悉了 SQL 语句的基本使用，也使用了 Django、HTML、JavaScript 以及 OpenGauss 数据库独立搭建一个小型图书管理系统的网站，提升了自己的编程能力，锻炼了自己的实践水平，也增强了自己对数据库知识的认识和理解，这也为我日后的学习和工作深造的过程提供更多的帮助，真是受益匪浅！

**致谢** 独立完成任一完整的项目系统都是不轻易的，这次的图书管理系统也是如此，从数据库的设计到前端页面的设计，从零开始构建项目到最终创建数十个文件完成项目，从一行一行单步调试到最终测试没有任何 bug，整个过程不能说是轻易的，也遇到了很多困难和问题，在这个过程中，互联网上很多前辈们所分享出的部署类似项目的经验和出现对应问题时的解决方案都让我遇到的很多问题迎刃而解，向这些热心在社区中分享自己开发项目的经验和注意事项的前辈们致以真挚的感谢！除了互联网上社区的资料给我提供了很大的帮助之外，身边的同学，还有数据库原理理论课程的任课老师也都给予了我很多帮助，以让我最终能够成功开发出这样一个虽然尚存不少改进空间但也较为成熟的图书管理系统，这次开发过程中学习到的知识和积累下的经验都将为我将来学习与工作过程推波助澜！最后，再向他们致以最真挚的感谢！

## 参考文献：

- [1] 王珊, 杜小勇, 陈红. 数据库系统概论. 第 6 版 [M]. 北京市:高等教育出版社,2023 年.
- [2] Django. Django 官方文档 [EB/OL]. <https://docs.djangoproject.com/zh-hans/4.2/>, 2024.6.5/2024.6.24.
- [3] 伏尔加河的卷毛. Django 连接 openGauss 数据库[EB/OL]. [https://blog.csdn.net/weixin\\_51005207/article/details/128044461](https://blog.csdn.net/weixin_51005207/article/details/128044461), 2022.12.10/2024.6.23.
- [4] -Learning-. django 视图层(views) [EB/OL]. <https://www.cnblogs.com/LearningOnline/p/9217474.html>, 2018.6.24/2024.6.24.
- [5] 彼岸花纽约. Django 表中的字段[EB/OL]. <https://www.cnblogs.com/zhouhai007/p/10279420.html>, 2019.1.16/2024.6.25.
- [6] WBOY. Django 框架中的 Model 详解[EB/OL]. <https://www.php.cn/faq/561003.html>, 2023.6.17/2024.6.24.
- [7] 景伟. 【Django】Django model 与数据库操作对应关系（转）[EB/OL]. <https://blog.csdn.net/jewely/article/details/105514965>, 2020.4.14/2024.6.26.
- [8] 未希. html 如何实现鼠标事件[EB/OL]. <https://www.kdun.com/ask/448993.html>, 2024.4.7/2024.6.25.

- [9] 神秘还作业. django 在 html 页面中写 js,Django 中简单添加 HTML、css、js 等文件[EB/OL]. [https://blog.csdn.net/weixin\\_36070797/article/details/117770977](https://blog.csdn.net/weixin_36070797/article/details/117770977), 2021.6.2/2024.6.24.
- [10] Mr.蔬菜. HTML 标签[EB/OL]. [https://blog.csdn.net/qq\\_67413159/article/details/123300273](https://blog.csdn.net/qq_67413159/article/details/123300273), 2022.3.6/2024.6.24.
- [11] 小半. Django 用户验证与权限管理[EB/OL]. <https://www.bmabk.com/index.php/post/191762.html>, 2023.12.24/2024.6.24.
- [12] the 6ix. django 打开的 html css\_python-django 加载 HTML/CSS 样式[EB/OL]. [https://blog.csdn.net/weixin\\_33971463/article/details/113057409](https://blog.csdn.net/weixin_33971463/article/details/113057409), 2021.1.19/2024.6.24.
- [13] 爱学习的苗总. django 分页(两种办法)[EB/OL]. [https://blog.csdn.net/qq\\_37605109/article/details/124514037](https://blog.csdn.net/qq_37605109/article/details/124514037), 2022.5.1/2024.6.24.