浙江大学

数据库系统实验报告

作业名称：	图书管理系统
姓　　名：	李谞远
学　　号：	3220102970
电子邮箱：	497410282@qq.com
联系电话：	15268453576
指导老师：	苗晓晔

2024 年 3 月 30 日

图书管理系统

一、 实验目的

设计并实现一个精简的图书管理程序，要求具有图书入库、查询、借书、还书、借书证管理等功能。

二、 系统需求

基本数据对象：

所有基本数据对象都被定义在 entities 包中，以下是这些对象的基本信息，其它信息请参考类内的注释。

书：类名为 Book，属性有书号，类别，书名，出版社，年份，作者，价格，剩余库存。

借书证：类名为 Card，属性有卡号，姓名，单位，身份(教师或学生)。

借书记录：类名为 Borrow，属性有卡号，书号，借书日期，还书日期。

基本功能模块：

图书管理系统中所有应具备的功能模块都在接口 LibraryManagementSystem 中被声明，以下是 LibraryManagementSystem 中声明的模块，有关各模块的详细注释和说明已经在接口 LibraryManagementSystem 中给出，请自行阅读：

● ApiResult storeBook(Book book)：图书入库模块。向图书库中注册(添加)一本新书，并返回新书的书号。如果该书已经存在于图书库中，那么入库操作将失败。当且仅当书的<类别，书名，出版社，年份，作者>均相同时，才认为两本书相同。请注意，book_id 作为自增列，应该插入时由数据库生成。插入完成后，需要根据数据库生成的 book_id 值去更新 book 对象里的 book_id。

● ApiResult incBookStock(int bookId, int deltaStock)：图书增加库存模块。为图书库中的某一本书增加库存。其中库存增量 deltaStock 可正可负，若为负数，则需要保证最终库存是一个非负数。

● ApiResult storeBook(List<Book> books)：图书批量入库模块。批量入库图书，如果有一本书入库失败，那么就需要回滚整个事务(即所有的书都不能被入库)。

● ApiResult removeBook(int bookId)：图书删除模块。从图书库中删除一本书。如果还有人尚未归还这本书，那么删除操作将失败。

● ApiResult modifyBookInfo(Book book)：图书修改模块。修改已入库图书的基本信息，该接口不能修改图书的书号和存量。

● ApiResult queryBook(BookQueryConditions conditions)：图书查询模块。根据提供的查询条件查询符合条件的图书，并按照指定排序方式排序。查询条件包括：类别点查(精确查询)，书名点查(模糊查询)，出版社点查(模糊查询)，年份范围查，作者点查(模糊查询)，价格范围差。如果两条记录排序条件的值相等，则按 book_id 升序排序。

● ApiResult borrowBook(Borrow borrow)：借书模块。根据给定的书号、卡号和借书时间添加一条借书记录，然后更新库存。若用户此前已经借过这本书但尚未归还，那么借书操作将失败。

● ApiResult returnBook(Borrow borrow)：还书模块。根据给定的书号、卡号和还书时间，查询对应的借书记录，并补充归还时间，然后更新库存。

● ApiResult showBorrowHistory(int cardId)：借书记录查询模块。查询某个用户的借书记录，按照借书时间递减、书号递增的方式排序。

● ApiResult registerCard(Card card)：借书证注册模块。注册一个借书证，若借书证已经存在，则该操作将失败。当且仅当<姓名，单位，身份>均相同时，才认为两张借书证相同。

● ApiResult removeCard(int cardId)：删除借书证模块。如果该借书证还有未归还的图书，那么删除操作将失败。

● ApiResult modifyCardInfo(Card card)：借书证修改模块。修改已存在的借书证的基本信息，该接口不能修改借书证的卡号。

● ApiResult showCards()：借书证查询模块。列出所有的借书证。

数据库（表）设计：

以下是该图书管理系统的数据表定义：

```
create table `book` (
    `book_id` int not null auto_increment,
    `category` varchar(63) not null,
    `title` varchar(63) not null,
    `press` varchar(63) not null,
    `publish_year` int not null,
    `author` varchar(63) not null,
    `price` decimal(7, 2) not null default 0.00,
    `stock` int not null default 0,
    primary key (`book_id`),
    unique (`category`, `press`, `author`, `title`, `publish_year`)
);
create table `card` (
    `card_id` int not null auto_increment,
    `name` varchar(63) not null,
    `department` varchar(63) not null,
    `type` char(1) not null,
    primary key (`card_id`),
    unique (`department`, `type`, `name`),
    check ( `type` in ('T', 'S') )
);
create table `borrow` (
    `card_id` int not null,
    `book_id` int not null,
    `borrow_time` bigint not null,
    `return_time` bigint not null default 0,
    primary key (`card_id`, `book_id`, `borrow_time`),
    foreign key (`card_id`) references `card`(`card_id`) on delete
cascade on update cascade,
    foreign key (`book_id`) references `book`(`book_id`) on delete
cascade on update cascade
);
```

系统功能验证：

系统功能验证测试分为功能性测试和正确性测试。

● 正确性测试通过测试用例进行评判，以验收时通过的测试用例数量占总测试用例数量的百分比来评定正确性测试部分的得分。

● 功能性测试通过验收时随机运行模拟场景的结果进行评判，以软件使用时的交互友好程度、效率、正确性等指标来评定功能性测试部分的得分。

功能性测试的参考模拟场景如下：

功能	描述
图书入库	输入<书号,类别,书名,出版社,年份,作者,价格,初始库存>,入库一本新书 B
增加库存	将书 B 的库存增加到 X，然后减少到 1
修改图书信息	随机抽取 N 个字段，修改图书 B 的图书信息
添加借书证	输入<姓名,单位,身份>，添加一张新的借书证 C
查询借书证	列出所有的借书证
借书	用借书证 C 借图书 B，再借一次 B，然后再借一本书 K
还书	用借书证 C 还掉刚刚借到的书 B
借书记录查询	查询 C 的借书记录
图书查询	从查询条件<类别点查(精确查询)，书名点查(模糊查询)，出版社点查(模糊查询)，年份范围查，作者点查(模糊查询)，价格范围差>中随机选取 N 个条件，并随机选取一个排序列和顺序

三、 实验环境

操作系统：Windows 11

开发环境：VSCode + Java + Maven + Vue.js

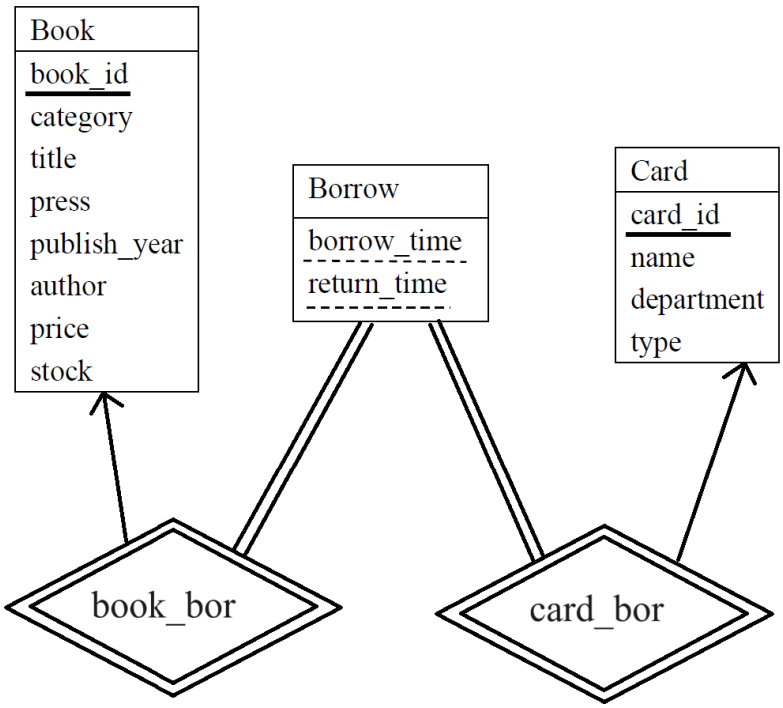
数据库系统：MySQL Server 8.3.0

Java 和 Maven 版本截图如下：

```
C:\Users\ycyaw>java -version
java version "21.0.1" 2023-10-17 LTS
Java(TM) SE Runtime Environment (build 21.0.1+12-LTS-29)
Java HotSpot(TM) 64-Bit Server VM (build 21.0.1+12-LTS-29, mixed mode, sharing)

C:\Users\ycyaw>mvn -v
Apache Maven 3.9.6 (bc0240f3c744dd6b6ec2920b3cd08dcc295161ae)
Maven home: D:\Maven\apache-maven-3.9.6
Java version: 21.0.1, vendor: Oracle Corporation, runtime: C:\Program Files\Java\jdk-21
Default locale: zh_CN, platform encoding: UTF-8
OS name: "windows 11", version: "10.0", arch: "amd64", family: "windows"
```

四、 系统设计及实现

1. 绘制该图书管理系统的 E-R 图（思考题 1）



2. 系统各函数的设计思路和实现

● ApiResult storeBook(Book book)：图书入库模块。

设计思路：先用 SELECT 语句在入库前查询是否有相同图书，如果没有则用 INSERT 语句往 book 表中插入该图书，插入后获取自增列 book_id。

代码实现：

```
1.  public ApiResult storeBook(Book book) {
2.      Connection conn = connector.getConn();
3.      PreparedStatement pStmt = null;
4.      ResultSet rSet = null;
5.      try {
6.          String category = book.getCategory();
7.          String title = book.getTitle();
8.          String press = book.getPress();
9.          int publishYear = book.getPublishYear();
10.         String author = book.getAuthor();
11.         double price = book.getPrice();
12.         int stock = book.getStock();
13.
14.         /* check if there are same books */
15.         String sameBookCheck = "SELECT * FROM book WHERE category = ? AND ti
tle = ? AND press = ? AND publish_year = ? AND author = ?";
16.
17.         pStmt = conn.prepareStatement(sameBookCheck);
```

```
18.         pStmt.setString(1, category);
19.         pStmt.setString(2, title);
20.         pStmt.setString(3, press);
21.         pStmt.setInt(4, publishYear);
22.         pStmt.setString(5, author);
23.         rSet = pStmt.executeQuery();
24.         if (rSet.next()) {
25.             return new ApiResult(false, "图书添加失败：存在相同图书");
26.         }
27.
28.         String storeBookQuery = "INSERT INTO book (category, title, press, p
ublish_year, author, price, stock) VALUES (?, ?, ?, ?, ?, ?, ?)";
29.
30.         pStmt = conn.prepareStatement(storeBookQuery, PreparedStatement.RETU
RN_GENERATED_KEYS);
31.         pStmt.setString(1, category);
32.         pStmt.setString(2, title);
33.         pStmt.setString(3, press);
34.         pStmt.setInt(4, publishYear);
35.         pStmt.setString(5, author);
36.         pStmt.setDouble(6, price);
37.         pStmt.setInt(7, stock);
38.         pStmt.executeUpdate();
39.
40.         rSet = pStmt.getGeneratedKeys();
41.         if (rSet.next()) {
42.             int bookId = rSet.getInt(1);
43.             book.setBookId(bookId);
44.         }
45.
46.         commit(conn);
47.     } catch (Exception e) {
48.         rollback(conn);
49.         return new ApiResult(false, e.getMessage());
50.     } finally {
51.         try {
52.             if (rSet != null) {
53.                 rSet.close();
54.             }
55.             if (pStmt != null) {
56.                 pStmt.close();
57.             }
58.         } catch (SQLException e) {
59.             e.printStackTrace();
```

```
60.         }
61.     }
62.     return new ApiResult(true, "图书添加成功");
63. }
```

● ApiResult incBookStock(int bookId, int deltaStock):图书增加库存模块。

设计思路：先用 SELECT 语句查询图书是否存在，然后获取图书库存，看修改后的库存是否会为负，如果条件都满足，则用 UPDATE 语句更新图书库存。

代码实现：

```
1.  public ApiResult incBookStock(int bookId, int deltaStock) {
2.      Connection conn = connector.getConn();
3.      PreparedStatement pStmt = null;
4.      ResultSet rSet = null;
5.      try {
6.          String selectStockQuery = "SELECT stock FROM book WHERE book_id = ?"
;
7.          pStmt = conn.prepareStatement(selectStockQuery);
8.          pStmt.setInt(1, bookId);
9.          rSet = pStmt.executeQuery();
10.
11.         int currentStock = 0;
12.         if (rSet.next()) {
13.             currentStock = rSet.getInt("stock");
14.         }
15.         else {
16.             return new ApiResult(false, "库存修改失败：图书不存在");
17.         }
18.
19.         if (currentStock + deltaStock < 0) {
20.             return new ApiResult(false, "库存修改失败：库存为负");
21.         }
22.
23.         String incBookStockQuery = "UPDATE book SET stock = stock + ? WHERE
book_id = ?";
24.
25.         pStmt = conn.prepareStatement(incBookStockQuery);
26.         pStmt.setInt(1, deltaStock);
27.         pStmt.setInt(2, bookId);
28.         pStmt.executeUpdate();
29.
30.         commit(conn);
31.     } catch (Exception e) {
32.         rollback(conn);
33.         return new ApiResult(false, e.getMessage());
```

```
    } finally {
        try {
            if (rSet != null) {
                rSet.close();
            }
            if (pStmt != null) {
                pStmt.close();
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
    return new ApiResult(true, "库存修改成功");
}
```

● ApiResult storeBook(List<Book> books)：图书批量入库模块。

设计思路：类似单本图书入库，先查询图书是否已经在书库中，与单本图书入库不同的是，利用批处理操作（addBatch 和 executeBatch），如果所有图书均合法，则一次性执行所有的插入操作。

代码实现：

```
public ApiResult storeBook(List<Book> books) {
    Connection conn = connector.getConn();
    PreparedStatement pStmt = null;
    PreparedStatement pStmt_i = null;
    ResultSet rSet = null;
    try {
        String sameBookCheck = "SELECT * FROM book WHERE category = ? AND title = ? AND press = ? AND publish_year = ? AND author = ?";
        pStmt = conn.prepareStatement(sameBookCheck);

        String storeBookQuery = "INSERT INTO book (category, title, press, publish_year, author, price, stock) VALUES (?, ?, ?, ?, ?, ?, ?)";
        pStmt_i = conn.prepareStatement(storeBookQuery, PreparedStatement.RETURN_GENERATED_KEYS);

        for (Book book : books) {
            String category = book.getCategory();
            String title = book.getTitle();
            String press = book.getPress();
            int publishYear = book.getPublishYear();
            String author = book.getAuthor();
            double price = book.getPrice();
            int stock = book.getStock();

```

```
22.             /* check if there are same books */
23.             pStmt.setString(1, category);
24.             pStmt.setString(2, title);
25.             pStmt.setString(3, press);
26.             pStmt.setInt(4, publishYear);
27.             pStmt.setString(5, author);
28.             rSet = pStmt.executeQuery();
29.
30.             if (rSet.next()) {
31.                 rollback(conn);
32.                 return new ApiResult(false, "图书添加失败：存在相同图书");
33.             }
34.
35.             pStmt_i.setString(1, category);
36.             pStmt_i.setString(2, title);
37.             pStmt_i.setString(3, press);
38.             pStmt_i.setInt(4, publishYear);
39.             pStmt_i.setString(5, author);
40.             pStmt_i.setDouble(6, price);
41.             pStmt_i.setInt(7, stock);
42.             pStmt_i.addBatch();
43.         }
44.
45.         pStmt_i.executeBatch();
46.         rSet = pStmt_i.getGeneratedKeys();
47.         int index = 0;
48.         while (rSet.next()) {
49.             int bookId = rSet.getInt(1);
50.             books.get(index++).setBookId(bookId);
51.         }
52.
53.         commit(conn);
54.     } catch (Exception e) {
55.         rollback(conn);
56.         return new ApiResult(false, e.getMessage());
57.     } finally {
58.         try {
59.             if (rSet != null) {
60.                 rSet.close();
61.             }
62.             if (pStmt != null) {
63.                 pStmt.close();
64.             }
65.             if (pStmt_i != null) {
```

```
                pStmt_i.close();
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
    return new ApiResult(true, "图书添加成功");
}
```

● ApiResult removeBook(int bookId)：图书删除模块。

设计思路：先用 SELECT 语句查询图书是否被出借，以及图书是否存在，如果条件满足，使用 DELETE 语句删除该图书。

代码实现：

```
public ApiResult removeBook(int bookId) {
    Connection conn = connector.getConn();
    PreparedStatement pStmt = null;
    ResultSet rSet = null;
    try {
        String bookBorrowedCheck = "SELECT * FROM borrow WHERE book_id = ? AND return_time = 0";
        pStmt = conn.prepareStatement(bookBorrowedCheck);
        pStmt.setInt(1, bookId);
        rSet = pStmt.executeQuery();
        if (rSet.next()) {
            return new ApiResult(false, "图书删除失败：图书处于出借状态");
        }

        String bookExistCheck = "SELECT * FROM book WHERE book_id = ?";
        pStmt = conn.prepareStatement(bookExistCheck);
        pStmt.setInt(1, bookId);
        rSet = pStmt.executeQuery();
        if (!rSet.next()) {
            return new ApiResult(false, "图书删除失败：图书不存在");
        }

        String removeBookQuery = "DELETE FROM book WHERE book_id = ?";
        pStmt = conn.prepareStatement(removeBookQuery);
        pStmt.setInt(1, bookId);
        pStmt.executeUpdate();

        commit(conn);
    } catch (Exception e) {
        rollback(conn);
        return new ApiResult(false, e.getMessage());
```

```
    } finally {
        try {
            if (rSet != null) {
                rSet.close();
            }
            if (pStmt != null) {
                pStmt.close();
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
    return new ApiResult(true, "图书删除成功");
}
```

● ApiResult modifyBookInfo(Book book)：图书修改模块。

设计思路：先用 SELECT 语句查询图书是否存在，同时修改后的信息不能和已经存在的图书相同。

代码实现：

```
public ApiResult modifyBookInfo(Book book) {
    Connection conn = connector.getConn();
    PreparedStatement pStmt = null;
    ResultSet rSet = null;
    try {
        String bookExistCheck = "SELECT * FROM book WHERE book_id = ?";
        pStmt = conn.prepareStatement(bookExistCheck);
        pStmt.setInt(1, book.getBookId());
        rSet = pStmt.executeQuery();
        if (!rSet.next()) {
            return new ApiResult(false, "图书修改失败：图书不存在");
        }

        String sameBookCheck = "SELECT * FROM book WHERE category = ? AND title = ? AND press = ? AND publish_year = ? AND author = ? AND book_id <> ?";
        pStmt = conn.prepareStatement(sameBookCheck);
        pStmt.setString(1, book.getCategory());
        pStmt.setString(2, book.getTitle());
        pStmt.setString(3, book.getPress());
        pStmt.setInt(4, book.getPublishYear());
        pStmt.setString(5, book.getAuthor());
        pStmt.setInt(6, book.getBookId());
        rSet = pStmt.executeQuery();
        if (rSet.next()) {
            return new ApiResult(false, "图书修改失败：存在相同图书");
```

```
25.         }
26.
27.         String modifyBookInfoQuery = "UPDATE book SET category = ?, title =
?, press = ?, publish_year = ?, author = ?, price = ? WHERE book_id = ?";
28.         pStmt = conn.prepareStatement(modifyBookInfoQuery);
29.         pStmt.setString(1, book.getCategory());
30.         pStmt.setString(2, book.getTitle());
31.         pStmt.setString(3, book.getPress());
32.         pStmt.setInt(4, book.getPublishYear());
33.         pStmt.setString(5, book.getAuthor());
34.         pStmt.setDouble(6, book.getPrice());
35.         pStmt.setInt(7, book.getBookId());
36.         pStmt.executeUpdate();
37.
38.         commit(conn);
39.     } catch (Exception e) {
40.         rollback(conn);
41.         return new ApiResult(false, e.getMessage());
42.     } finally {
43.         try {
44.             if (rSet != null) {
45.                 rSet.close();
46.             }
47.             if (pStmt != null) {
48.                 pStmt.close();
49.             }
50.         } catch (SQLException e) {
51.             e.printStackTrace();
52.         }
53.     }
54.     return new ApiResult(true, "图书修改成功");
55. }
```

● ApiResult queryBook(BookQueryConditions conditions)：图书查询模块。

设计思路：先将 BookQueryConditions 类中的各个条件限制提取出来，根据各个条件是否为 null，拼接出一个完整的 SELECT 查询语句，精确查询使用等于符号，模糊查询使用 LIKE 关键字，同时利用 ORDER BY，来对查询结果相应的关键字进行排序，最后执行查询语句得到对应结果。

代码实现：

```
1.  public ApiResult queryBook(BookQueryConditions conditions) {
2.      Connection conn = connector.getConn();
3.      PreparedStatement pStmt = null;
4.      ResultSet rSet = null;
5.      BookQueryResults bookQueryResults;
```

```
6.      try {
7.          String category = conditions.getCategory();
8.          String title = conditions.getTitle();
9.          String press = conditions.getPress();
10.         Integer tminPublishYear = conditions.getMinPublishYear();
11.         Integer tmaxPublishYear = conditions.getMaxPublishYear();
12.         int minPublishYear = -2147483647, maxPublishYear = 2147483647;
13.         String author = conditions.getAuthor();
14.         Double tminPrice = conditions.getMinPrice();
15.         Double tmaxPrice = conditions.getMaxPrice();
16.         double minPrice = -2147483647, maxPrice = 2147483647;
17.         if(tminPublishYear != null) minPublishYear = tminPublishYear.intValu
e();
18.         if(tmaxPublishYear != null) maxPublishYear = tmaxPublishYear.intValu
e();
19.         if(tminPrice != null) minPrice = tminPrice.doubleValue();
20.         if(tmaxPrice != null) maxPrice = tmaxPrice.doubleValue();
21.
22.         String selectBookQuery = "SELECT * FROM book WHERE publish_year >= ?
 AND publish_year <= ? AND price >= ? AND price <= ?";
23.         int index = 5;
24.         int categoryIndex = 0, titleIndex = 0, pressIndex = 0, authorIndex =
 0;
25.         if (category != null) {
26.             selectBookQuery += " AND category = ?";
27.             categoryIndex = index++;
28.         }
29.         if (title != null) {
30.             selectBookQuery += " AND title LIKE ?";
31.             title = "%" + title + "%";
32.             titleIndex = index++;
33.         }
34.         if (press != null) {
35.             selectBookQuery += " AND press LIKE ?";
36.             press = "%" + press + "%";
37.             pressIndex = index++;
38.         }
39.         if (author != null) {
40.             selectBookQuery += " AND author LIKE ?";
41.             author = "%" + author + "%";
42.             authorIndex = index++;
43.         }
44.         selectBookQuery += " ORDER BY " + conditions.getSortBy().getValue()
+ " " + conditions.getSortOrder().getValue();
```

```
45.         if (conditions.getSortBy().getValue() != "book_id") {
46.             selectBookQuery += ", book_id ASC";
47.         }
48.         pStmt = conn.prepareStatement(selectBookQuery);
49.         pStmt.setInt(1, minPublishYear);
50.         pStmt.setInt(2, maxPublishYear);
51.         pStmt.setDouble(3, minPrice);
52.         pStmt.setDouble(4, maxPrice);
53.         if (categoryIndex > 0) pStmt.setString(categoryIndex, category);
54.         if (titleIndex > 0) pStmt.setString(titleIndex, title);
55.         if (pressIndex > 0) pStmt.setString(pressIndex, press);
56.         if (authorIndex > 0) pStmt.setString(authorIndex, author);
57.         rSet = pStmt.executeQuery();
58.
59.         List<Book> books = new ArrayList<Book>();
60.         while (rSet.next()) {
61.             int bookId = rSet.getInt("book_id");
62.             String tempCategory = rSet.getString("category");
63.             String tempTitle = rSet.getString("title");
64.             String tempPress = rSet.getString("press");
65.             int tempPublishYear = rSet.getInt("publish_year");
66.             String tempAuthor = rSet.getString("author");
67.             double tempPrice = rSet.getDouble("price");
68.             int tempStock = rSet.getInt("stock");
69.             Book book = new Book(tempCategory, tempTitle, tempPress, tempPub
lishYear, tempAuthor, tempPrice, tempStock);
70.             book.setBookId(bookId);
71.             books.add(book);
72.         }
73.         bookQueryResults = new BookQueryResults(books);
74.
75.         commit(conn);
76.     } catch (Exception e) {
77.         rollback(conn);
78.         return new ApiResult(false, e.getMessage());
79.     } finally {
80.         try {
81.             if (rSet != null) {
82.                 rSet.close();
83.             }
84.             if (pStmt != null) {
85.                 pStmt.close();
86.             }
87.         } catch (SQLException e) {
```

```
            e.printStackTrace();
        }
    }
    return new ApiResult(true, null, bookQueryResults);
}
```

● ApiResult borrowBook(Borrow borrow)：借书模块。

设计思路：先用 SELECT 语句确认借书的借书证和所借图书均存在，然后判断该借书证之前是否借过这本书，如果条件均满足，则用 UPDATE 语句更新库存。

代码实现：

```
public ApiResult borrowBook(Borrow borrow) {
    Connection conn = connector.getConn();
    PreparedStatement pStmt = null;
    ResultSet rSet = null;
    try {
        int cardId = borrow.getCardId();
        int bookId = borrow.getBookId();
        long borrowTime = borrow.getBorrowTime();

        String cardCheck = "SELECT * FROM card WHERE card_id = ?";
        pStmt = conn.prepareStatement(cardCheck);
        pStmt.setInt(1, cardId);
        rSet = pStmt.executeQuery();
        if (!rSet.next()) {
            return new ApiResult(false, "借书失败：借书证不存在");
        }

        String bookCheck = "SELECT * FROM book WHERE book_id = ?";
        pStmt = conn.prepareStatement(bookCheck);
        pStmt.setInt(1, bookId);
        rSet = pStmt.executeQuery();
        if (!rSet.next()) {
            return new ApiResult(false, "借书失败：该书不存在");
        }

        String userCheck = "SELECT * FROM borrow WHERE book_id = ? AND card_
id = ? AND return_time = 0";
        pStmt = conn.prepareStatement(userCheck);
        pStmt.setInt(1, bookId);
        pStmt.setInt(2, cardId);
        rSet = pStmt.executeQuery();
        if (rSet.next()) {
            return new ApiResult(false, "借书失败：该书尚未归还");
        }
```

```
34.
35.         String bookDecQuery = "UPDATE book SET stock = stock - 1 WHERE book_
id = ? AND stock > 0";
36.         pStmt = conn.prepareStatement(bookDecQuery);
37.         pStmt.setInt(1, bookId);
38.         int affectedRows = pStmt.executeUpdate();
39.         if (affectedRows == 0) {
40.             rollback(conn);
41.             return new ApiResult(false, "借书失败：库存不足");
42.         }
43.
44.         String insertBorrowQuery = "INSERT INTO borrow (card_id, book_id, bo
rrow_time, return_time) VALUES (?, ?, ?, 0)";
45.         pStmt = conn.prepareStatement(insertBorrowQuery);
46.         pStmt.setInt(1, cardId);
47.         pStmt.setInt(2, bookId);
48.         pStmt.setLong(3, borrowTime);
49.         pStmt.executeUpdate();
50.
51.         commit(conn);
52.     } catch (Exception e) {
53.         rollback(conn);
54.         return new ApiResult(false, e.getMessage());
55.     } finally {
56.         try {
57.             if (rSet != null) {
58.                 rSet.close();
59.             }
60.             if (pStmt != null) {
61.                 pStmt.close();
62.             }
63.         } catch (SQLException e) {
64.             e.printStackTrace();
65.         }
66.     }
67.     return new ApiResult(true, "借书成功");
68. }
```

● ApiResult returnBook(Borrow borrow)：还书模块。

设计思路：先用 SELECT 语句确认还书的借书证和所借图书均存在，然后查询该借书记录，确保用户借了该书且未归还，同时导入的还书时间晚于借书时间，进行以上查询后，使用 UPDATE 语句更新库存和借书记录。

代码实现：

```
1.  public ApiResult returnBook(Borrow borrow) {
```

```
2.      Connection conn = connector.getConn();
3.      PreparedStatement pStmt = null;
4.      ResultSet rSet = null;
5.      try {
6.          int cardId = borrow.getCardId();
7.          int bookId = borrow.getBookId();
8.          long returnTime = borrow.getReturnTime();
9.
10.         String cardCheck = "SELECT * FROM card WHERE card_id = ?";
11.         pStmt = conn.prepareStatement(cardCheck);
12.         pStmt.setInt(1, cardId);
13.         rSet = pStmt.executeQuery();
14.         if (!rSet.next()) {
15.             return new ApiResult(false, "还书失败：借书证不存在");
16.         }
17.
18.         String bookCheck = "SELECT * FROM book WHERE book_id = ?";
19.         pStmt = conn.prepareStatement(bookCheck);
20.         pStmt.setInt(1, bookId);
21.         rSet = pStmt.executeQuery();
22.         if (!rSet.next()) {
23.             return new ApiResult(false, "还书失败：该书不存在");
24.         }
25.
26.         String userCheck = "SELECT * FROM borrow WHERE book_id = ? AND card_
id = ? AND return_time = 0";
27.         pStmt = conn.prepareStatement(userCheck);
28.         pStmt.setInt(1, bookId);
29.         pStmt.setInt(2, cardId);
30.         rSet = pStmt.executeQuery();
31.         long borrowTime = 0;
32.         if (rSet.next()) {
33.             borrowTime = rSet.getLong("borrow_time");
34.         }
35.         else {
36.             return new ApiResult(false, "还书失败：用户未借该书");
37.         }
38.         if (borrowTime >= returnTime) {
39.             return new ApiResult(false, "还书失败：还书时间早于借书时间");
40.         }
41.
42.         String bookIncQuery = "UPDATE book SET stock = stock + 1 WHERE book_
id = ?";
43.         pStmt = conn.prepareStatement(bookIncQuery);
```

```
44.         pStmt.setInt(1, bookId);
45.         pStmt.executeUpdate();
46.
47.         String returnQuery = "UPDATE borrow SET return_time = ? WHERE card_i
d = ? AND book_id = ? AND return_time = 0";
48.         pStmt = conn.prepareStatement(returnQuery);
49.         pStmt.setLong(1, returnTime);
50.         pStmt.setInt(2, cardId);
51.         pStmt.setInt(3, bookId);
52.         pStmt.executeUpdate();
53.
54.         commit(conn);
55.     } catch (Exception e) {
56.         rollback(conn);
57.         return new ApiResult(false, e.getMessage());
58.     } finally {
59.         try {
60.             if (rSet != null) {
61.                 rSet.close();
62.             }
63.             if (pStmt != null) {
64.                 pStmt.close();
65.             }
66.         } catch (SQLException e) {
67.             e.printStackTrace();
68.         }
69.     }
70.     return new ApiResult(true, "还书成功");
71. }
```

● ApiResult showBorrowHistory(int cardId)：借书记录查询模块。

设计思路：使用 SELECT 语句从 borrow 中查询对应借书证的借书记录，获取一条借书记录后，从书库中查询该书，将信息全部归纳到 Item 类中，形成一个 List<Item>列表，最后构造 BorrowHistories 返回。

代码实现：

```
1.  public ApiResult showBorrowHistory(int cardId) {
2.      Connection conn = connector.getConn();
3.      PreparedStatement pStmt = null;
4.      ResultSet rSet = null, resBook = null;
5.      BorrowHistories borrowHistories;
6.      try {
7.          String borrowQuery = "SELECT * FROM borrow WHERE card_id = ? ORDER B
Y borrow_time DESC, book_id ASC";
8.          pStmt = conn.prepareStatement(borrowQuery);
```

```
9.          pStmt.setInt(1, cardId);
10.         rSet = pStmt.executeQuery();
11.
12.         List<Item> items = new ArrayList<Item>();
13.         while (rSet.next()) {
14.             int bookId = rSet.getInt("book_id");
15.             String bookQuery = "SELECT * FROM book WHERE book_id = ?";
16.             pStmt = conn.prepareStatement(bookQuery);
17.             pStmt.setInt(1, bookId);
18.             resBook = pStmt.executeQuery();
19.
20.             if (resBook.next()) {
21.                 Book book = new Book(resBook.getString("category"), resBook.
getString("title"), resBook.getString("press"), resBook.getInt("publish_year"), r
esBook.getString("author"), resBook.getDouble("price"), resBook.getInt("stock"));

22.                 book.setBookId(bookId);
23.
24.                 Borrow borrow = new Borrow(bookId, cardId);
25.                 borrow.setBorrowTime(rSet.getLong("borrow_time"));
26.                 borrow.setReturnTime(rSet.getLong("return_time"));
27.
28.                 Item item = new Item(cardId, book, borrow);
29.                 items.add(item);
30.             }
31.         }
32.         borrowHistories = new BorrowHistories(items);
33.
34.         commit(conn);
35.     } catch (Exception e) {
36.         rollback(conn);
37.         return new ApiResult(false, e.getMessage());
38.     } finally {
39.         try {
40.             if (rSet != null) {
41.                 rSet.close();
42.             }
43.             if (resBook != null) {
44.                 resBook.close();
45.             }
46.             if (pStmt != null) {
47.                 pStmt.close();
48.             }
49.         } catch (SQLException e) {
```

```
            e.printStackTrace();
        }
    }
    return new ApiResult(true, null, borrowHistories);
}
```

● ApiResult registerCard(Card card)：借书证注册模块。

设计思路：先用 SELECT 语句在注册前查询是否有相同信息的借书证，如果没有则用 INSERT 语句往 card 表中插入该借书证，插入后获取自增列 card_id。

代码实现：

```
public ApiResult registerCard(Card card) {
    Connection conn = connector.getConn();
    PreparedStatement pStmt = null;
    ResultSet rSet = null;
    try {
        String name = card.getName();
        String department = card.getDepartment();
        String type = card.getType().getStr();

        String sameCardCheck = "SELECT * FROM card WHERE name = ? AND department = ? AND type = ?";
        pStmt = conn.prepareStatement(sameCardCheck);
        pStmt.setString(1, name);
        pStmt.setString(2, department);
        pStmt.setString(3, type);
        rSet = pStmt.executeQuery();
        if (rSet.next()) {
            return new ApiResult(false, "借书证新建失败：存在相同借书证");
        }

        String storeCardQuery = "INSERT INTO card (name, department, type) VALUES (?, ?, ?)";
        pStmt = conn.prepareStatement(storeCardQuery, PreparedStatement.RETURN_GENERATED_KEYS);
        pStmt.setString(1, name);
        pStmt.setString(2, department);
        pStmt.setString(3, type);
        pStmt.executeUpdate();

        rSet = pStmt.getGeneratedKeys();
        if (rSet.next()) {
            int cardId = rSet.getInt(1);
            card.setCardId(cardId);
        }
```

```
32.
33.         commit(conn);
34.     } catch (Exception e) {
35.         rollback(conn);
36.         return new ApiResult(false, e.getMessage());
37.     } finally {
38.         try {
39.             if (rSet != null) {
40.                 rSet.close();
41.             }
42.             if (pStmt != null) {
43.                 pStmt.close();
44.             }
45.         } catch (SQLException e) {
46.             e.printStackTrace();
47.         }
48.     }
49.     return new ApiResult(true, "借书证新建成功");
50. }
```

● ApiResult removeCard(int cardId)：删除借书证模块。

设计思路：先用 SELECT 查询借书证是否存在，以及该借书证是否存在未归还的图书，存在则无法删除借书证，如果满足条件则用 DELETE 删除该借书证。

代码实现：

```
1.  public ApiResult removeCard(int cardId) {
2.      Connection conn = connector.getConn();
3.      PreparedStatement pStmt = null;
4.      ResultSet rSet = null;
5.      try {
6.          String bookBorrowedCheck = "SELECT * FROM borrow WHERE card_id = ? A
ND return_time = 0";
7.          pStmt = conn.prepareStatement(bookBorrowedCheck);
8.          pStmt.setInt(1, cardId);
9.          rSet = pStmt.executeQuery();
10.         if (rSet.next()) {
11.             return new ApiResult(false, "借书证删除失败：有未归还的图书");
12.         }
13.
14.         String cardExistCheck = "SELECT * FROM card WHERE card_id = ?";
15.         pStmt = conn.prepareStatement(cardExistCheck);
16.         pStmt.setInt(1, cardId);
17.         rSet = pStmt.executeQuery();
18.         if (!rSet.next()) {
19.             return new ApiResult(false, "不存在该借书证");
```

```
        }

        String removeCardQuery = "DELETE FROM card WHERE card_id = ?";
        pStmt = conn.prepareStatement(removeCardQuery);
        pStmt.setInt(1, cardId);
        pStmt.executeUpdate();

        commit(conn);
    } catch (Exception e) {
        rollback(conn);
        return new ApiResult(false, e.getMessage());
    } finally {
        try {
            if (rSet != null) {
                rSet.close();
            }
            if (pStmt != null) {
                pStmt.close();
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
    return new ApiResult(true, "借书证删除成功");
}
```

● ApiResult modifyCardInfo(Card card)：借书证修改模块。

设计思路：先用 SELECT 查询借书证是否存在，因为借书证的信息和所借图书没有关系，所以如果借书证存在，就直接更新借书证信息即可。

代码实现：

```
public ApiResult modifyCardInfo(Card card) {
    Connection conn = connector.getConn();
    PreparedStatement pStmt = null;
    ResultSet rSet = null;
    try {
        String cardExistCheck = "SELECT * FROM card WHERE card_id = ?";
        pStmt = conn.prepareStatement(cardExistCheck);
        pStmt.setInt(1, card.getCardId());
        rSet = pStmt.executeQuery();
        if (!rSet.next()) {
            return new ApiResult(false, "不存在该借书证");
        }

```

```
        String modifyCardInfoQuery = "UPDATE card SET name = ?, department = ?, type = ? WHERE card_id = ?";
        pStmt = conn.prepareStatement(modifyCardInfoQuery);
        pStmt.setString(1, card.getName());
        pStmt.setString(2, card.getDepartment());
        pStmt.setString(3, card.getType().getStr());
        pStmt.setInt(4, card.getCardId());
        pStmt.executeUpdate();

        commit(conn);
    } catch (Exception e) {
        rollback(conn);
        return new ApiResult(false, e.getMessage());
    } finally {
        try {
            if (rSet != null) {
                rSet.close();
            }
            if (pStmt != null) {
                pStmt.close();
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
    return new ApiResult(true, "借书证修改成功");
}
```

● ApiResult showCards()：借书证查询模块。

设计思路：利用 SELECT 语句和 ORDER 关键字，直接查询 card 表中的借书证并按 card_id 排序。

代码实现：

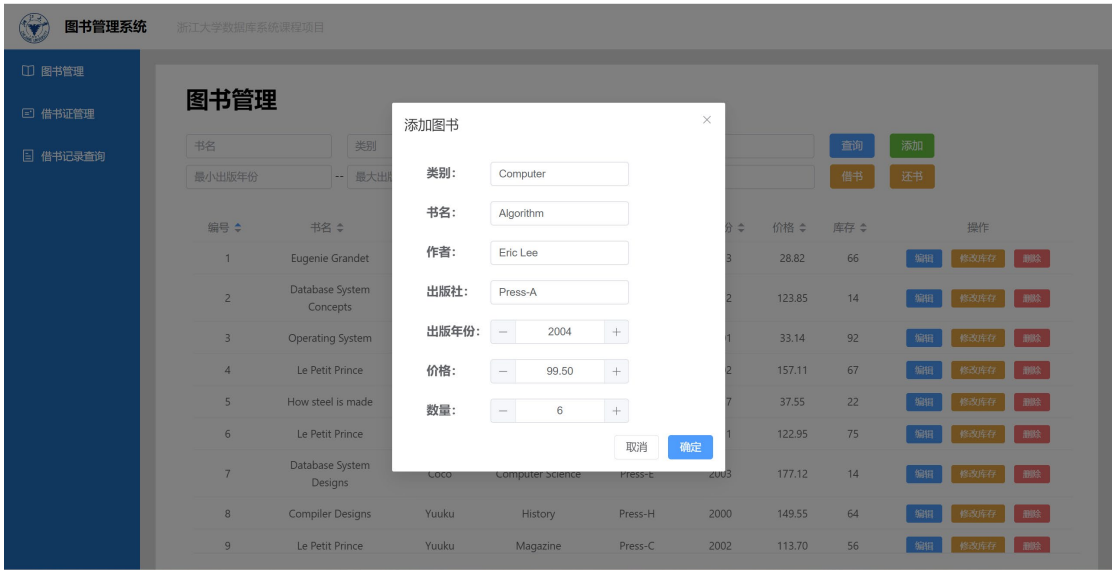
```
public ApiResult showCards() {
    Connection conn = connector.getConn();
    PreparedStatement pStmt = null;
    ResultSet rSet = null;
    CardList cardList;
    try {
        String cardQuery = "SELECT * FROM card ORDER BY card_id ASC";
        pStmt = conn.prepareStatement(cardQuery);
        rSet = pStmt.executeQuery();
        List<Card> cards = new ArrayList<Card>();
        while (rSet.next()) {
            int cardId = rSet.getInt("card_id");
```

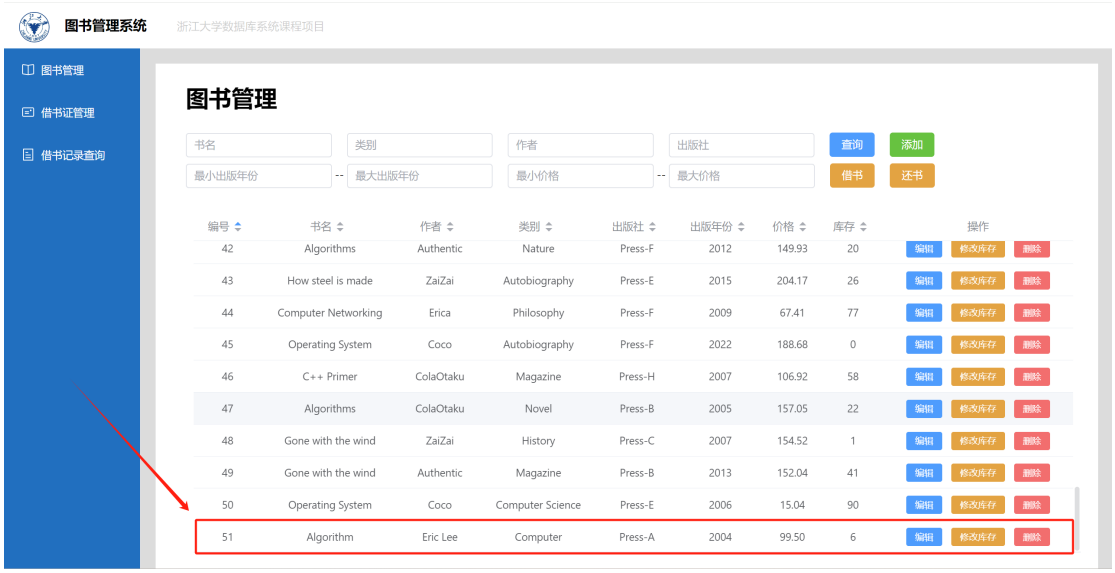
```
            String name = rSet.getString("name");
            String department = rSet.getString("department");
            String tmptype = rSet.getString("type");
            Card.CardType type = Card.CardType.values(tmptype);
            Card card = new Card(cardId, name, department, type);
            cards.add(card);
        }
        cardList = new CardList(cards);

        commit(conn);
    } catch (Exception e) {
        rollback(conn);
        return new ApiResult(false, e.getMessage());
    } finally {
        try {
            if (rSet != null) {
                rSet.close();
            }
            if (pStmt != null) {
                pStmt.close();
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
    return new ApiResult(true, null, cardList);
}
```

3. 程序运行结果场景以及截图说明（即实验指导文档中的系统功能验证）
   程序运行并通过了框架内置的所有单元测试：

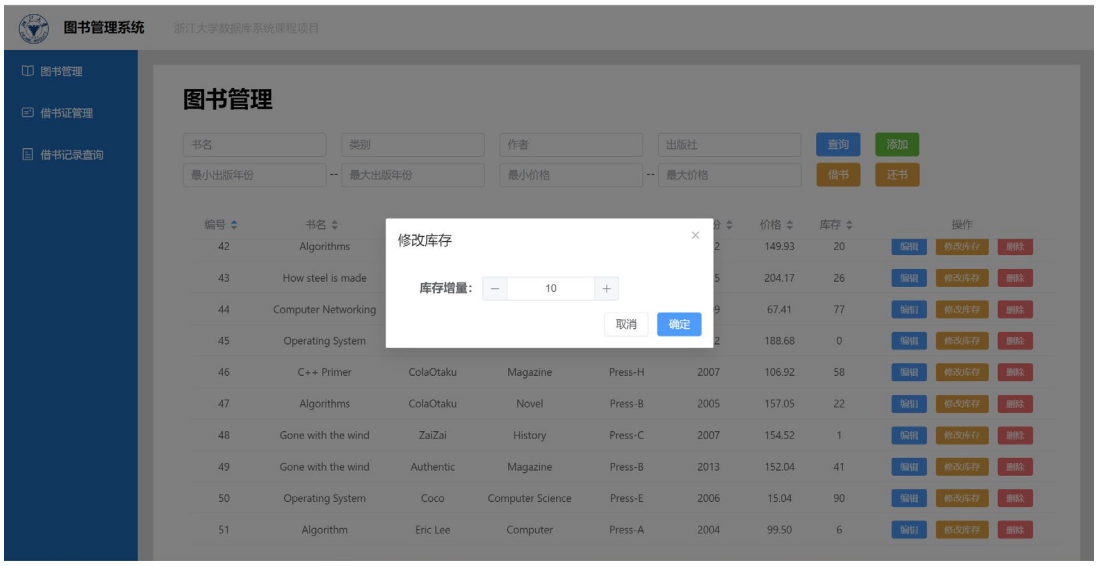
基于前端框架，完成了前端，以下系统功能验证的各部分由前端截图展示：

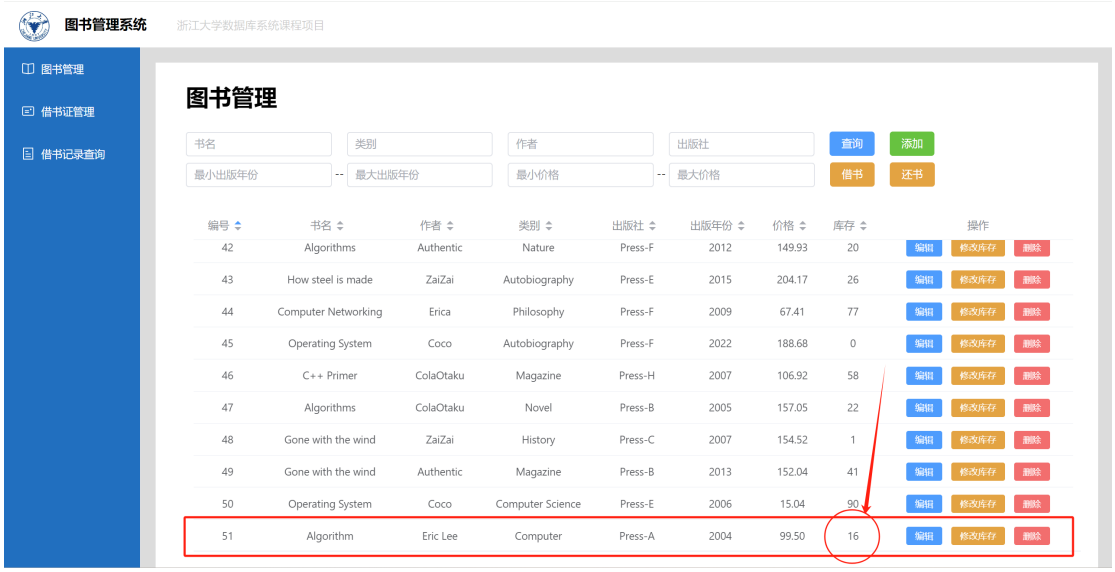
a. 图书入库：输入新书各项信息，入库一本新书。

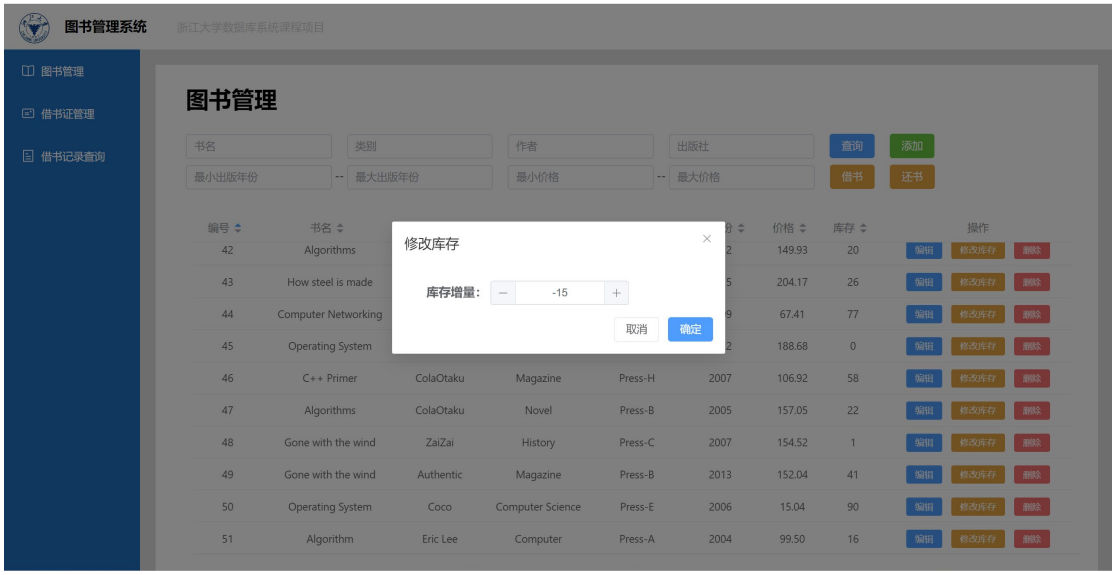


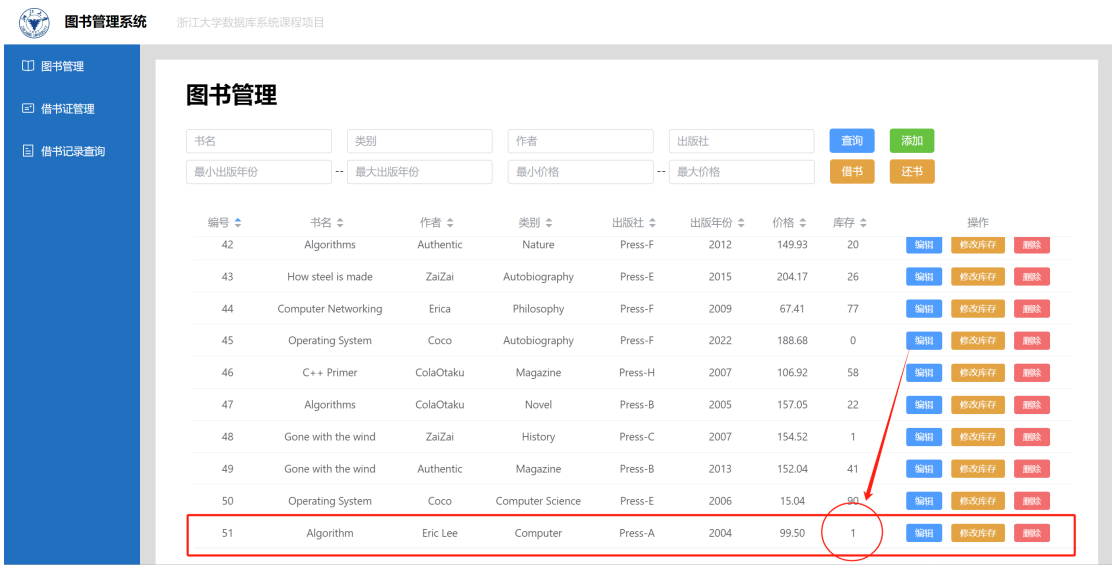


b. 增加库存：给上述入库图书增加 10 本库存，再减少到 1 本。

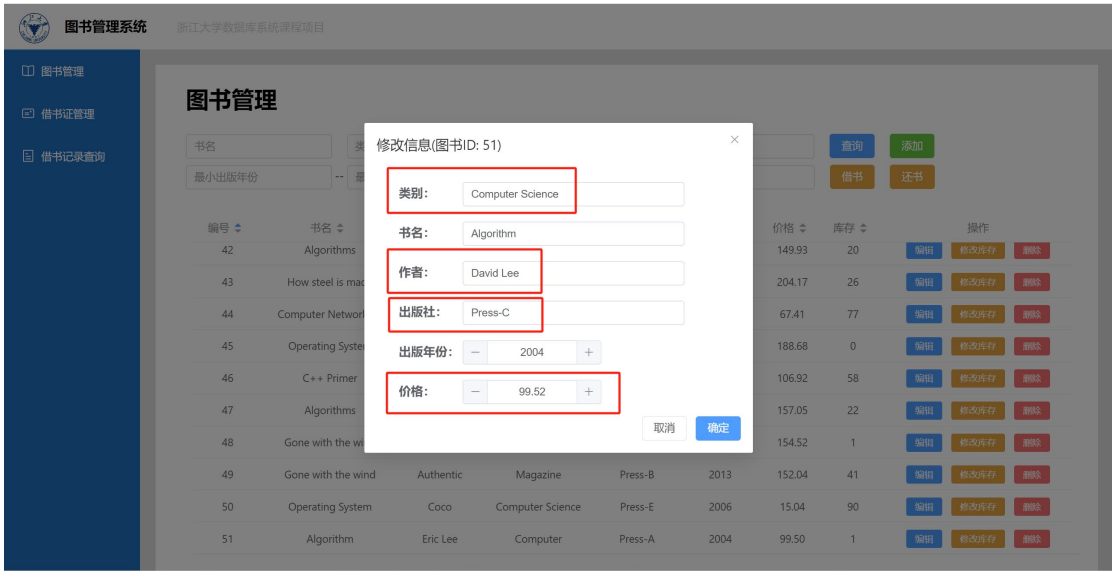


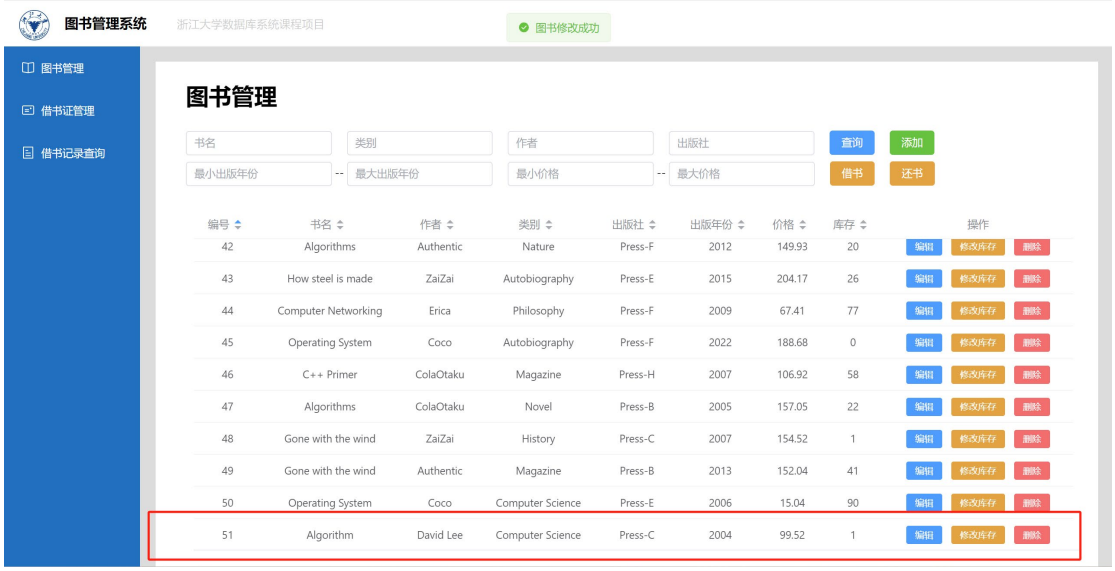

图书管理系统
浙江大学数据库系统课程项目
图书管理
借书证管理
借书记录查询
图书管理
书名
类别
作者
出版社
查询
添加
最小出版年份
最大出版年份
最小价格
最大价格
借书
还书
编号 书名 作者 类别 出版社 出版年份 价格 库存 操作
42 Algorithms Authentic Nature Press-F 2012 149.93 20 编辑 修改库存 删除
43 How steel is made ZaiZai Autobiography Press-E 2015 204.17 26 编辑 修改库存 删除
44 Computer Networking Erica Philosophy Press-F 2009 67.41 77 编辑 修改库存 删除
45 Operating System Coco Autobiography Press-F 2022 188.68 0 编辑 修改库存 删除
46 C++ Primer ColaOtaku Magazine Press-H 2007 106.92 58 编辑 修改库存 删除
47 Algorithms ColaOtaku Novel Press-B 2005 157.05 22 编辑 修改库存 删除
48 Gone with the wind ZaiZai History Press-C 2007 154.52 1 编辑 修改库存 删除
49 Gone with the wind Authentic Magazine Press-B 2013 152.04 41 编辑 修改库存 删除
50 Operating System Coco Computer Science Press-E 2006 15.04 90 编辑 修改库存 删除
51 Algorithm Eric Lee Computer Press-A 2004 99.50 16 编辑 修改库存 删除


图书管理系统
浙江大学数据库系统课程项目
图书管理
借书证管理
借书记录查询
图书管理
修改库存
库存增量：
-15
取消
确定

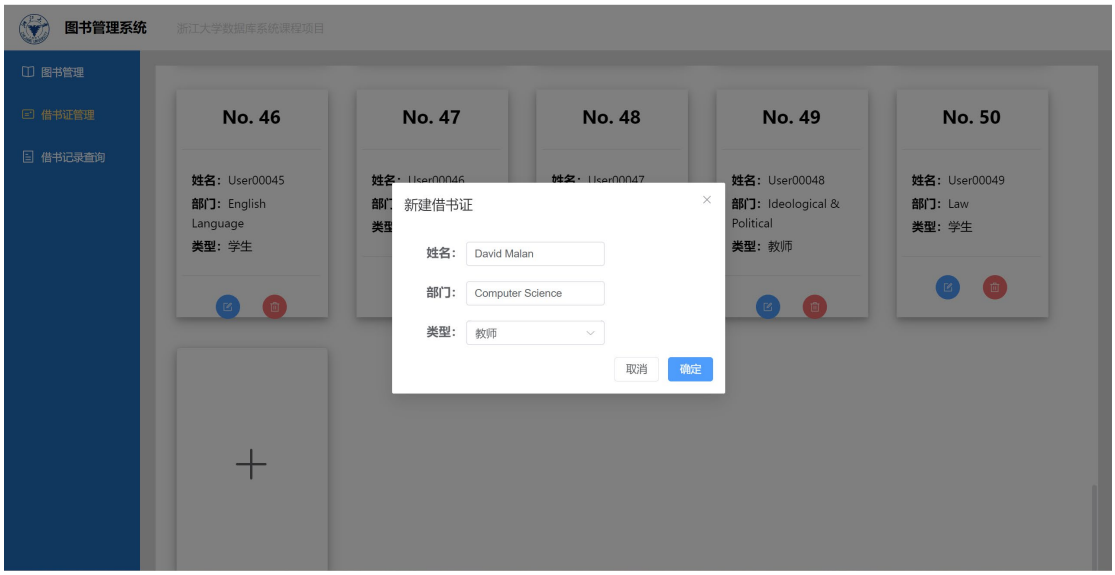

图书管理系统
浙江大学数据库系统课程项目
图书管理
借书证管理
借书记录查询
图书管理
书名
类别
作者
出版社
查询
添加
最小出版年份
最大出版年份
最小价格
最大价格
借书
还书
编号 书名 作者 类别 出版社 出版年份 价格 库存 操作
42 Algorithms Authentic Nature Press-F 2012 149.93 20 编辑 修改库存 删除
43 How steel is made ZaiZai Autobiography Press-E 2015 204.17 26 编辑 修改库存 删除
44 Computer Networking Erica Philosophy Press-F 2009 67.41 77 编辑 修改库存 删除
45 Operating System Coco Autobiography Press-F 2022 188.68 0 编辑 修改库存 删除
46 C++ Primer ColaOtaku Magazine Press-H 2007 106.92 58 编辑 修改库存 删除
47 Algorithms ColaOtaku Novel Press-B 2005 157.05 22 编辑 修改库存 删除
48 Gone with the wind ZaiZai History Press-C 2007 154.52 1 编辑 修改库存 删除
49 Gone with the wind Authentic Magazine Press-B 2013 152.04 41 编辑 修改库存 删除
50 Operating System Coco Computer Science Press-E 2006 15.04 90 编辑 修改库存 删除
51 Algorithm Eric Lee Computer Press-A 2004 99.50 1 编辑 修改库存 删除

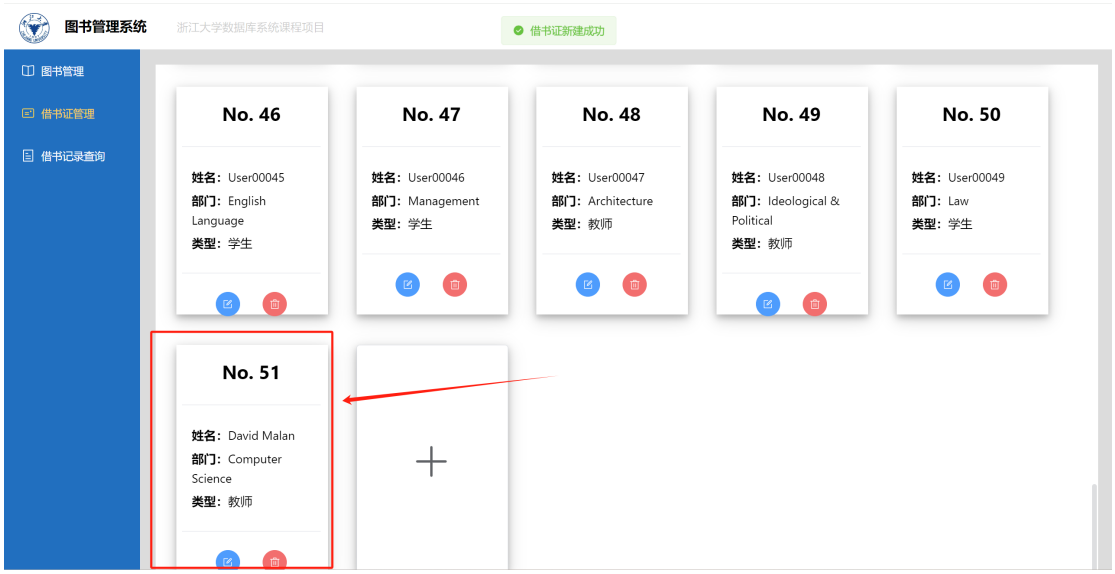
c. 修改图书信息：随机几个字段，修改刚插入的图书信息。



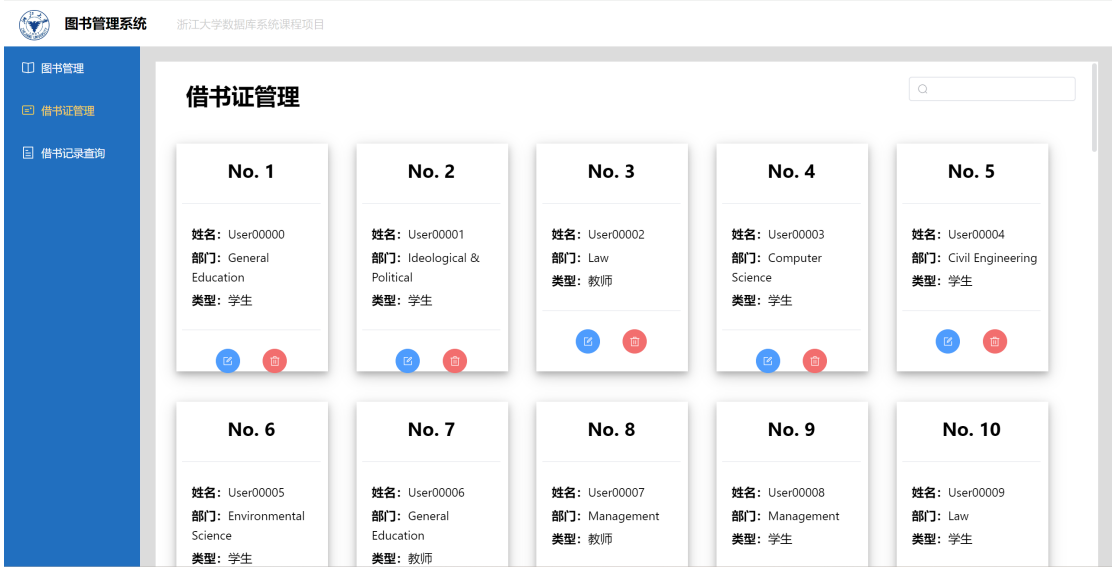


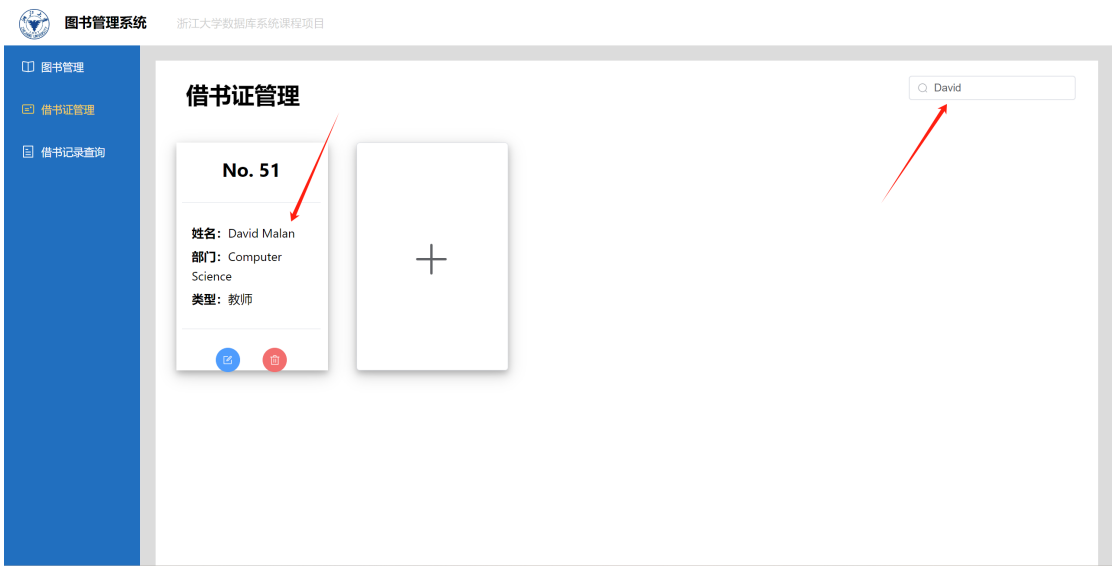
d. 添加借书证：输入新借书证各项信息，添加一张新的借书证。





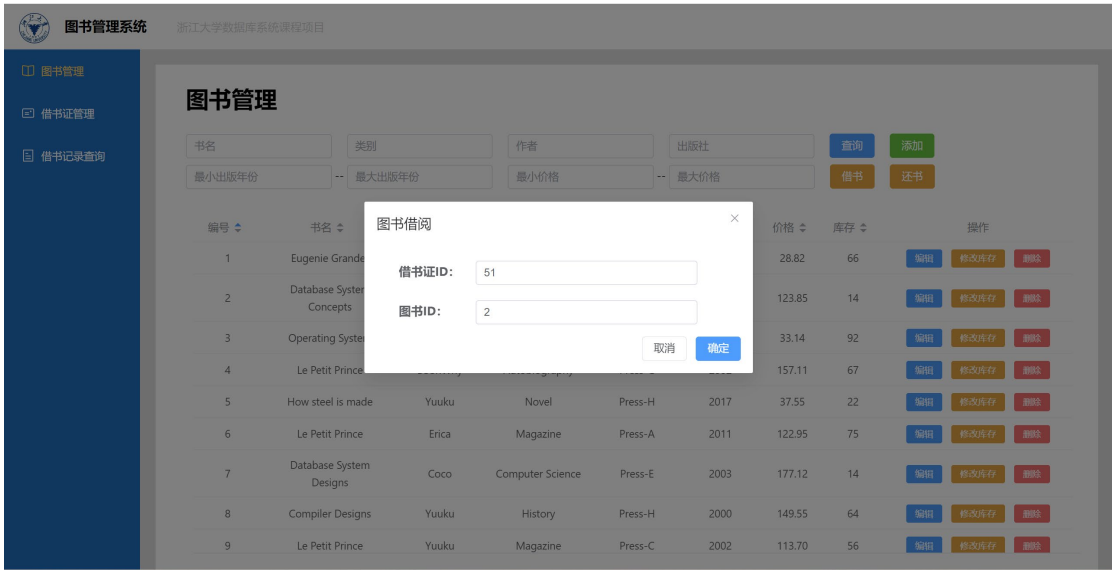
e. 查询借书证：根据搜索条件列出相应借书证，为空则列出所有借书证。



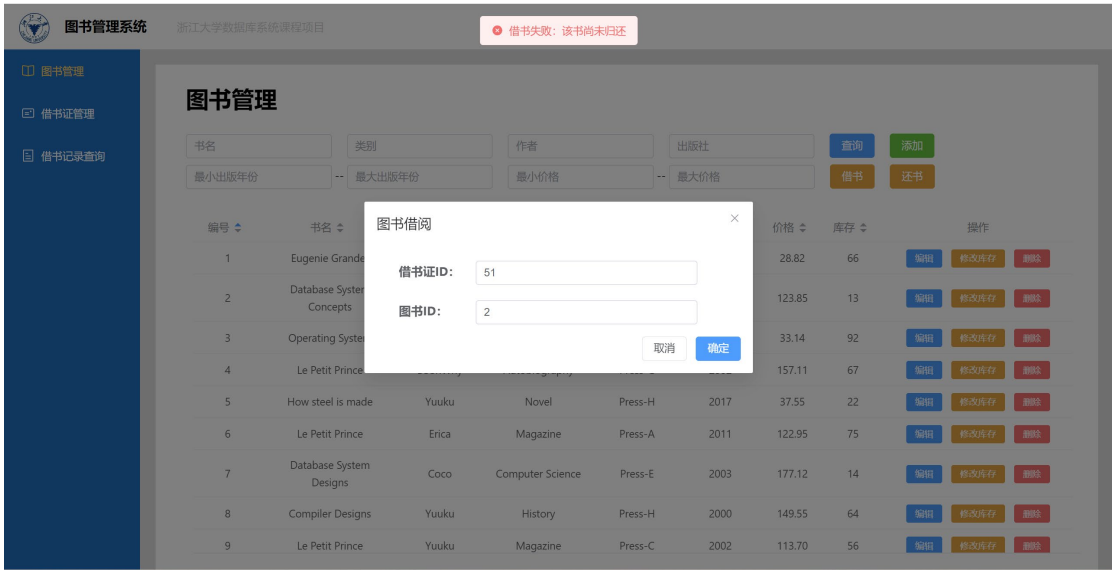


f. 借书：用借书证 C 借一本书 B，再借一次 B，然后再借一本书 C。

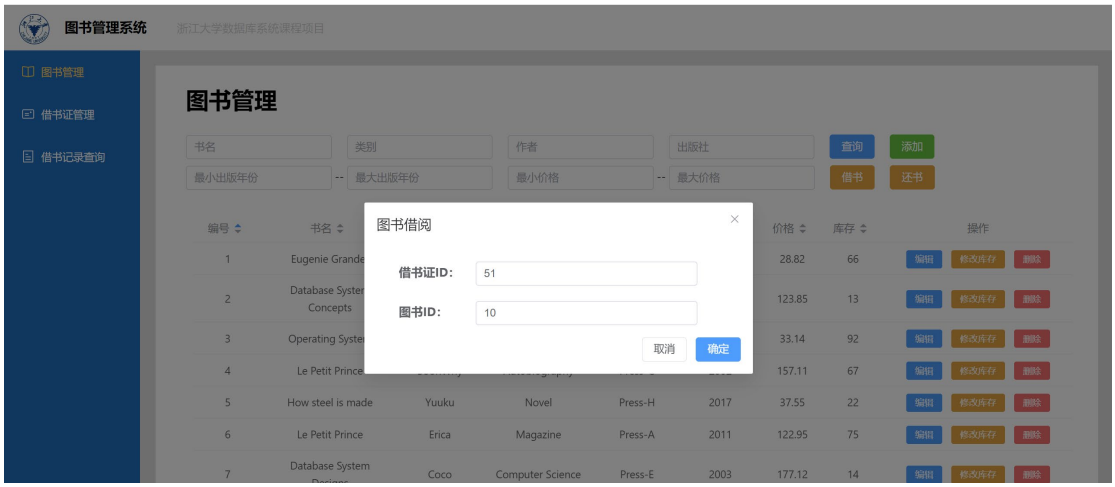
第一次借书 B，借书成功：



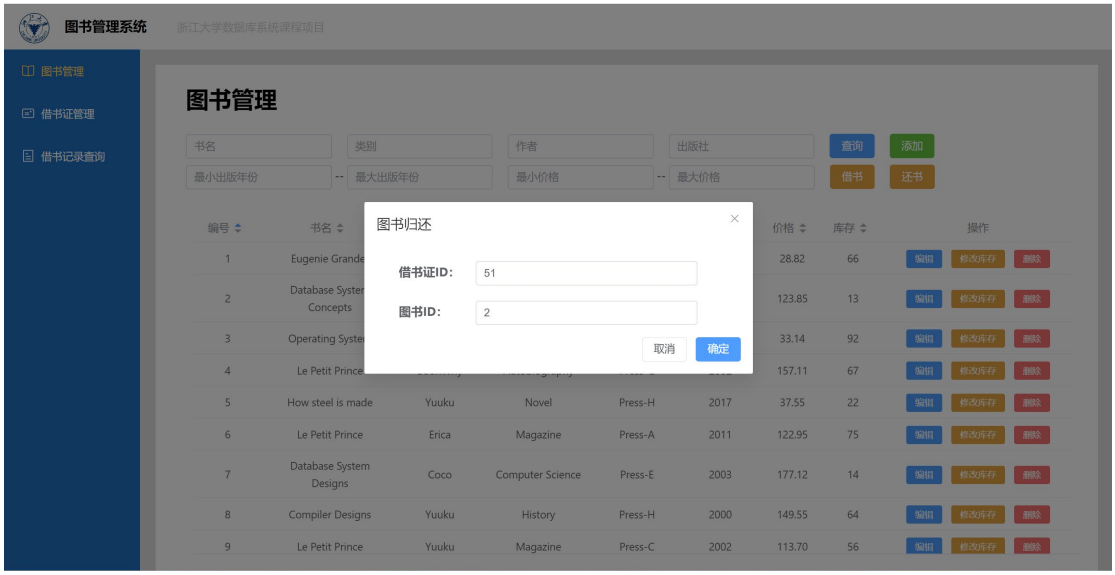
第二次借书 B，借书失败：

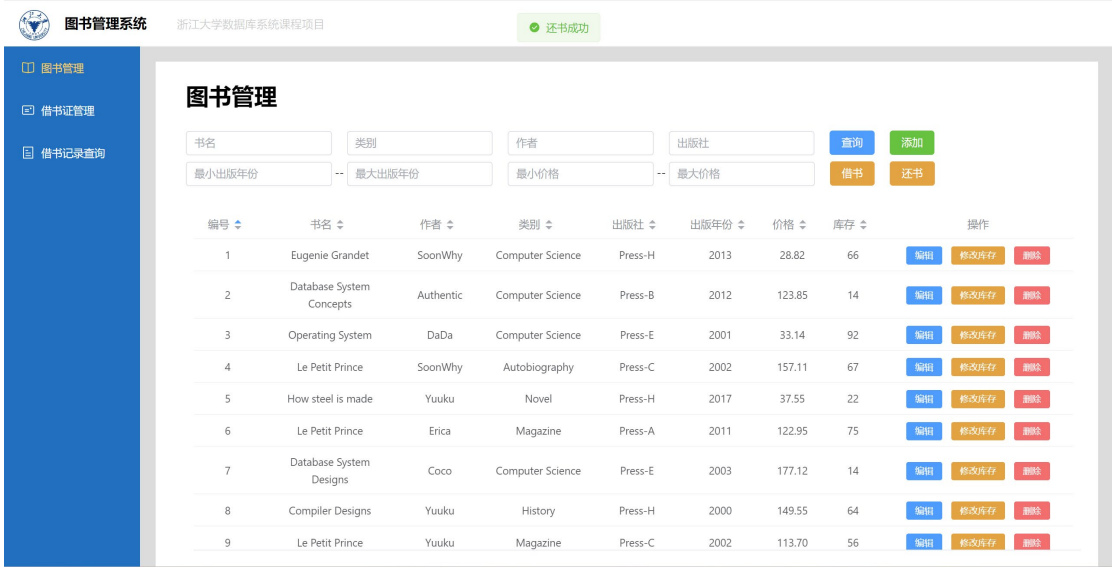


再借一本书 K：



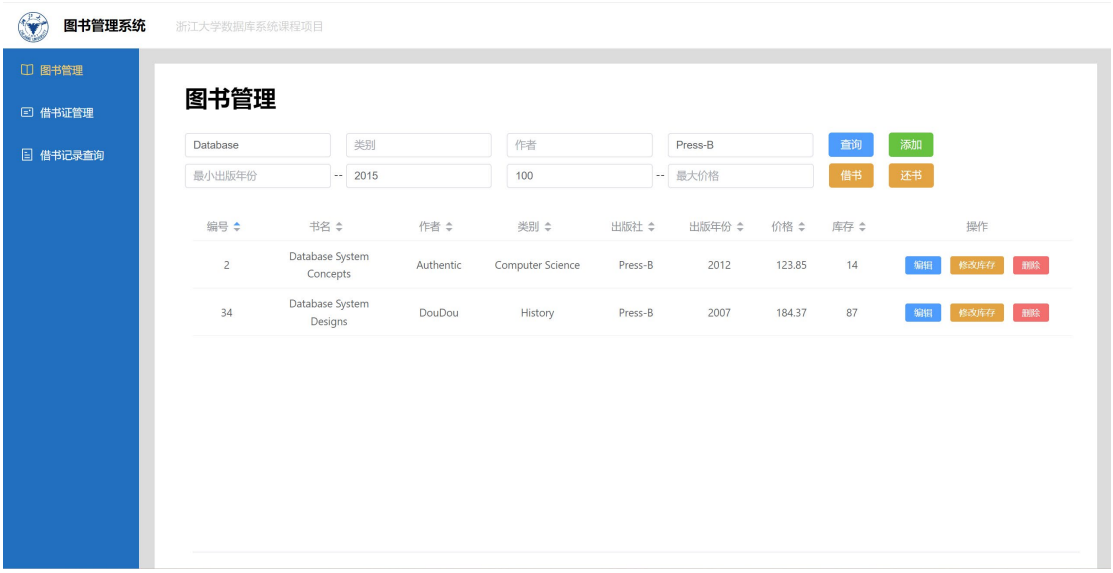
g. 还书：用借书证 C 还掉刚刚借到的书 B。



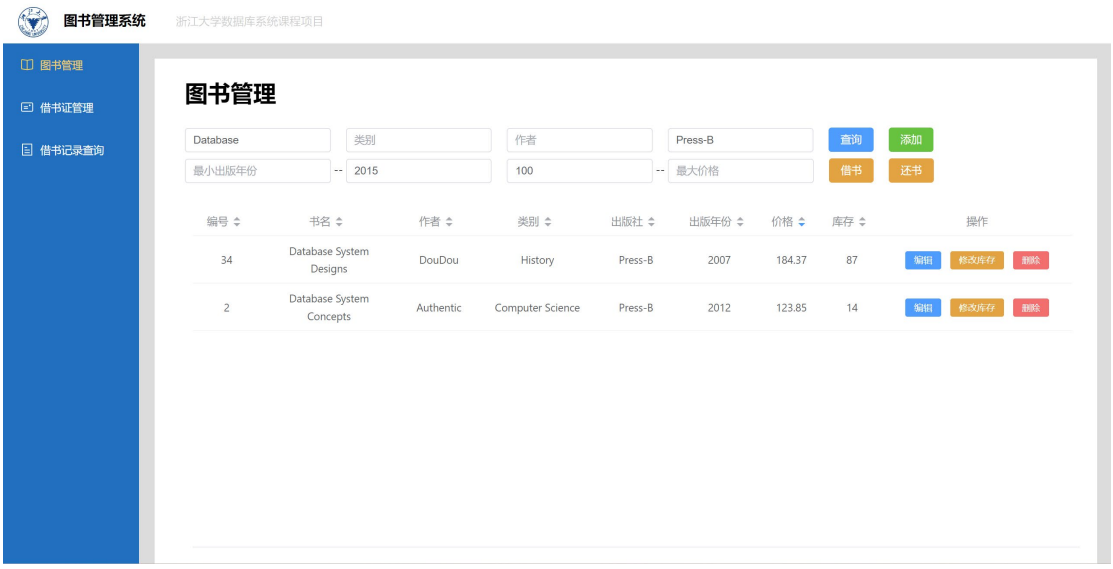


h. 借书记录查询：查询 C 的借书记录。

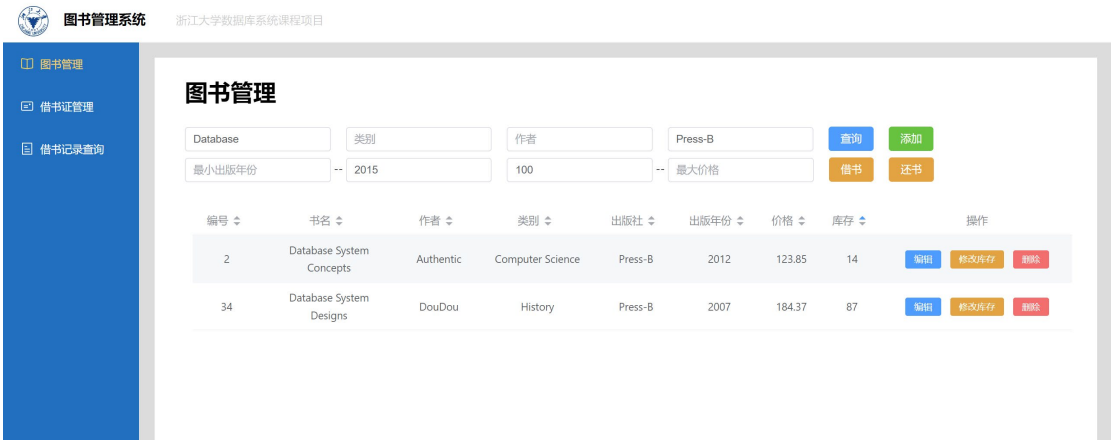
i. 图书查询：从查询条件<类别点查(精确查询)，书名点查(模糊查询)，出版社点查(模糊查询)，年份范围查，作者点查(模糊查询)，价格范围差>中随机选取若干条件，并随机选取一个排序列和顺序。



按价格降序后：



按库存升序后：



五、 遇到的问题及解决方法

1. 描述 SQL 注入攻击的原理，并简要举例。在图书管理系统中，哪些模块可能会遭受 SQL 注入攻击？如何解决？（思考题 2）

答：

假如我用构建字符串的形式来组成查询语句，如"SELECT * FROM book WHERE title = '" + title + "'"，然后用户进行 SQL 注入攻击，即输入 X' or 'Y' = 'Y，那么查询语句变成"SELECT * FROM book WHERE title = 'X' or 'Y' = 'Y'"，WHERE 后面的条件为恒真，那么所有记录都将被显示。同理该攻击可用于 UPDATE 等，造成数据被篡改。

在图书管理系统中，各个查询、修改、删除模块如果使用字符串拼接的方式，都可能遭受 SQL 注入攻击，解决方法是使用 SQL 预处理语句 Prepared Statement。

使用 PreparedStatement，可以写带参数的 SQL 查询、更新、删除语句，通过使用相同的 SQL 语句和不同的参数值构建语句。同时，PreparedStatement 用来执行 SQL 语句的时候，数据库系统会对 SQL 语句进行预编译处理，预处理语句将被预先编译好，这条预编译的 SQL 语句支持重用，这样一来，它比 Statement 对象生成的语句速度更快。

2. 在 InnoDB 的默认隔离级别（RR, Repeated Read）下，当出现并发访问时，如何保证借书结果的正确性？（思考题 3）

答：

一般的 SELECT…FROM…WHERE…语句都是快照读，不会对数据加锁，因此并发访问执行 SELECT 查询库存时会得到一样的结果。为了防止并发借书导致库存为负，我们在 UPDATE 时，加上条件判断 stock>0 即可。这种方法可行，是因为在 MySQL 在 InnoDB 默认隔离级别下，使用 MVCC（多版本并发控制）来处理并发事务，UPDATE 使用的是当前读，可以读取到其他事务已经做出但尚未提交的事务，因此可以获取到最新的数据，当另一个事务借书并导致某本书 stock=0 时，在该事务中执行 UPDATE，会判断 stock>0 条件不成立，不进行借书操作。

3. 一直无法通过借还书的单元测试，如何解决？

答：

反复查看还书部分代码，但没有看出问题，前往观察借还书单元测试部分的代码，发现其设置了一些 corner case，比如还书时间早于借书时间，这种情况可能发生在管理员手动输入还书时间且输入错误的时候，在相应还书模块中加上对这种情况的特判，成功通过了这部分单元测试。

4. 后端得到的数据，如何传向前端，以及前端输入的数据如何传向后端？

答：

前端文档中提出，在该前端框架中，前后端通过 JSON 来传递数据，一开始尝试写函数来将获取的数据组成 JSON 格式，虽然可行但并不通用。经过搜索，发现 java 提供了相应的轮子，于是 import 对应的包 JSONObject，调用相应函数，就非常方便的进行了 JSON 的解析。

六、 总结

这是我第一次利用框架独立完成一个数据库小项目，由于第一次接触 Java 和 Vue 两种全新的语言，整个过程充满了挑战和收获。

首先，通过对图书管理系统中的基本数据对象和功能模块的分析，我清晰地了解了图书管理系统的架构和各个模块之间的交互关系，这为后续的开发工作提供了良好的基础，让我能够有针对性地进行代码编写和测试。

其次，遇到的问题和解决方法让我更加深入地理解了上课提到的一些内容。例如，SQL 注入攻击让我意识到了数据安全在系统开发中的重要性，同时我学会了如何使用预处理语句来防范 SQL 注入攻击。此外，在解决借还书功能单元测试失败的问题时，我发现了一些边界情况下的问题，并加以修复，这提醒了我在开发过程中要对各种情况进行充分考虑和测试。

同时，前端的编写也很有意思，自己几乎从零开始做出了一个可交互的界面，不断完善前后端的通信，给我带来了很多的成就感，也了解到了很多新知识。

总的来说，通过这次图书管理系统的实现，我不仅学会了许多新的知识和技术，还提高了问题分析和解决能力。相信这些经验会给我将来的学习和工作带来很大帮助。