

浙江大学

**Database System Lab 5**

学号：3230104980

姓名：姜思妤

指导老师：苗晓晔

# 目录

实验目的 .....

实验需求 .....

实验环境 .....

实验模块设计思路与实现 .....

接口实现 .....

后端实现 .....

前端实现 .....

系统验证测试 .....

正确性测试 .....

功能性测试 .....

遇到的问题及解决方法 .....

思考题 .....

2

2

2

3

3

4

7

10

10

10

18

19

## 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。

## 实验需求

- 提供一个基于 MySQL 的精简图书管理程序，该图书管理程序应具备较好的可扩展性、鲁棒性和安全性，并且在高并发场景下仍能正确运行。
- 完成类 LibraryManagementSystemImpl 中各功能模块的函数，并通过所有测试样例。
- 使用提供的前端框架，正确完成图书管理系统的前端页面，使其成为一个用户能真正使用的图书管理系统。

## 实验环境

Java	后端核心语言，提供基础运行时支持
Node.js	前端框架运行环境，支持 Vue 3 开发
Vue.js	前端框架，用于构建用户界面
Element Plus	基于 Vue 3 的 UI 组件库，提供表格、表单、弹窗等交互组件
PostgreSQL	关系型数据库，用于存储图书、卡证、借阅记录等数据
IDE	IntelliJ IDEA 2024.3.3
构建工具	Maven，Java 项目管理与依赖管理工具
包管理工具	npm，前端依赖管理工具

# 实验模块设计思路与实现

## 接口实现

本图书管理系统通过三个表搭建，以下是其 E-R 图：

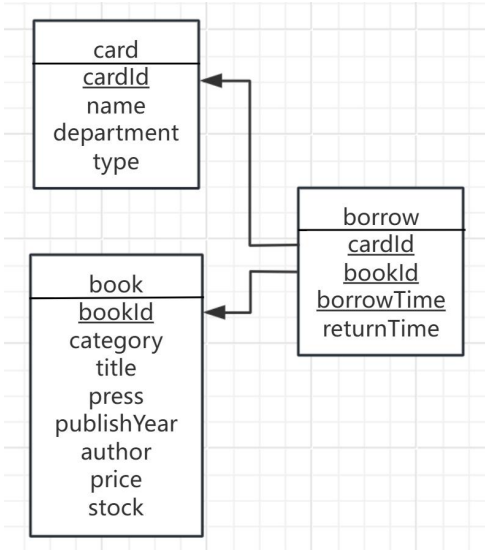


图 1 图书管理系统 E-R 图

## 书籍管理模块

**实现功能：**书籍的存储、查询、库存调整、删除及信息修改。

**设计思路：**

1. 唯一性校验：通过 `category, title, press, publishYear, author` 组合字段校验书籍是否重复。
2. 事务管理：通过 `Connection.commit()`和 `rollback()`保证操作的原子性。
3. 动态查询：通过 `BookQueryConditions` 动态拼接 SQL，支持多条件模糊查询（如书名、出版社、作者等）及排序。
4. 库存管理：`incBookStock` 方法通过事务锁（`for update`）防止并发修改导致的数据不一致。
5. 异常处理：统一捕获 `SQLException` 并回滚事务，确保数据一致性。

```
1. // 动态 SQL 构建
2. StringBuilder querySql = new StringBuilder("SELECT * FROM book WHERE 1=1");
3. if (conditions.getTitle() != null) {
4.     querySql.append(" AND title LIKE ?");
5.     params.add("%" + conditions.getTitle() + "%");
6. }
7. // 执行预处理防止 SQL 注入，提升执行效率
8. PreparedStatement stmt = conn.prepareStatement(querySql.toString());
```

## 借阅管理模块

**实现功能：**处理借书、还书以及借阅历史查询

### 设计思路：

1. 借书流程： 检查库存是否充足（`SELECT ... FOR UPDATE` 加锁防止超借）；检查是否存在未归还记录，避免重复借阅；减少库存并插入借阅记录。
2. 还书流程： 验证借阅记录有效性（`returnTime > borrowTime`）；更新归还时间并恢复库存。
3. 事务隔离： 通过 `conn.setAutoCommit(false)` 显式控制事务边界。

```
1. // 借书时的库存检查与更新 (borrowBook 方法)
2. String stockSql = "SELECT stock FROM book WHERE bookId = ? FOR UPDATE";
3. PreparedStatement stmt = conn.prepareStatement(stockSql);
4. stmt.setInt(1, borrow.getBookId());
5. ResultSet rs = stmt.executeQuery();
```

### 借书证管理模块

实现功能：借书证的新建、删除、修改及查询。

#### 设计思路：

1. 唯一性约束： 通过 `name, department, type` 组合字段防止重复注册。
2. 级联删除： 删除借书证时，先删除关联的借阅记录（`DELETE FROM Borrow`），再删除借书证。
3. 数据回填： 使用 `RETURN_GENERATED_KEYS` 获取自增 ID 并回填到实体对象。

```
1. // 卡证注册时获取自增 ID (registerCard 方法)
2. PreparedStatement stmt = conn.prepareStatement(insertSql, PreparedStatement.
    RETURN_GENERATED_KEYS);
3. ResultSet rs = stmt.getGeneratedKeys();
4. if (rs.next()) {
5.     card.setCardId(rs.getInt(1));
6. }
```

## 后端实现

### 整体架构

网络层： 通过 `HttpServer` 监听 HTTP 请求，处理连接与路由。

业务逻辑层： `LibraryManagementSystemImpl` 封装数据库操作，实现核心业务逻辑。

数据交互层：使用 `fastjson2` 解析请求体为 JSON 对象，并通过 `ApiResult` 统一封装响应。

实体层： `Book`、`Borrow`、`Card` 等实体类定义数据模型。

### 请求路由与处理

1. RESTful： 通过不同 HTTP 方法（GET/POST/PUT/DELETE）和路径（`/books`、`/cards`、`/borrows`）区分功能。  
示例： `POST /cards` 注册新卡， `GET /books` 查询图书。

2. 统一处理器基类： `BaseHandler` 实现公共逻辑（CORS 处理、请求解析、响应封装），子类专注业务逻辑。
3. 动态路径参数： 通过解析 URI 路径（如 `/cards/{cardId}`）获取资源 ID。

```
1. // 示例: CardHandler 删除卡片逻辑
2. String path = exchange.getRequestURI().getPath();
3. String cardId = path.substring(path.lastIndexOf('/') + 1);
4. ApiResult result = library.removeCard(Integer.parseInt(cardId));
```

## BaseHandler 基类

### 请求处理入口：

统一处理所有请求，设置 CORS 头，处理 OPTIONS 预检请求，并按 HTTP 方法路由。这样实现，能使子类无需关心跨域和路由逻辑，专注业务实现。

```
1. public void handle(HttpExchange exchange) throws IOException {
2.     try {
3.         setCorsHeaders(exchange); // 设置跨域头
4.         if ("OPTIONS".equals(exchange.getRequestMethod())) {
5.             handleOptions(exchange); // 处理预检请求
6.             return;
7.         }
8.         // 根据 HTTP 方法路由到子类实现
9.         switch (exchange.getRequestMethod()) {
10.            case "GET" -> handleGet(exchange);
11.            case "POST" -> handlePost(exchange);
12.            case "PUT" -> handlePut(exchange);
13.            case "DELETE" -> handleDelete(exchange);
14.            default -> sendResponse(exchange, 405, "Method Not Allowed");
15.        }
16.    } catch (Exception e) {
17.        log.severe("Request handling error: " + e.getMessage());
18.        sendResponse(exchange, 500, "Internal Server Error");
19.    }
20. }
```

## 公共工具方法

### CORS 头设置

```
1. protected void setCorsHeaders(HttpExchange exchange) {
2.     Headers headers = exchange.getResponseHeaders();
3.     headers.add("Access-Control-Allow-Origin", "*");
4.     headers.add("Access-Control-Allow-Methods", "GET, POST, PUT, DELETE, OPTIONS");
5.     headers.add("Access-Control-Allow-Headers", "Content-Type");
6. }
```

## 请求体读取

```
1.     protected String readRequestBody(HttpExchange exchange) throws IOException {
2.         try (BufferedReader reader = new BufferedReader(new InputStreamReader(ex
change.getRequestBody())) {
3.             return reader.lines().collect(Collectors.joining());
4.         }
5.     }
```

## 响应发送

```
1.     protected void sendResponse(HttpExchange exchange, int statusCode, String re
sponse) throws IOException {
2.         exchange.getResponseHeaders().set("Content-Type", "application/json");
3.         exchange.sendResponseHeaders(statusCode, response.getBytes().length);
4.         try (OutputStream os = exchange.getResponseBody()) {
5.             os.write(response.getBytes());
6.         }
7.     }
```

## 抽象方法定义

强制子类实现具体的业务逻辑

```
1.     protected abstract void handleGet(HttpExchange exchange) throws IOException;
2.     protected abstract void handlePost(HttpExchange exchange) throws IOException;
3.     protected abstract void handlePut(HttpExchange exchange) throws IOException;
4.     protected abstract void handleDelete(HttpExchange exchange) throws IOException;
```

## 数据序列化与反序列化

1. JSON 数据绑定： 使用 [fastjson2](#) 将请求体自动转换为实体对象。  
示例： [POST /cards](#) 请求体解析为 [Card](#) 对象。
2. 动态条件查询： [BookHandler](#) 通过 [BookQueryConditions](#) 解析 URL 参数，构建动态 SQL 查询条件。
3. 统一响应格式： [ApiResult](#) 封装操作状态、消息及数据，通过 [sendApiResult](#) 方法返回 JSON 响应。

```
1.     // 图书查询条件解析 (BookHandler 的 handleGet 方法)
2.     Map<String, String> params = parseQueryParams(exchange.getRequestURI().getQuery());
3.     BookQueryConditions conditions = new BookQueryConditions();
4.     if (params.containsKey("title")) conditions.setTitle(params.get("title"));
```

## 资源管理

1. 数据库连接生命周期： 在 [main](#) 方法中通过 [Runtime.getRuntime\(\).addShutdownHook](#) 确保服务关闭时释放连接。
2. 事务控制： 业务逻辑层（如 [LibraryManagementSystemImpl](#)）通过 [commit\(\)](#)

- 和 `rollback()` 保证数据一致性。
3. 批处理优化: `storeBook(List<Book>)` 方法使用 `PreparedStatement.addBatch()` 提升批量插入效率。

## 前端实现

### 图书管理模块 (Book.vue)

#### 动态查询:

1. 参数收集: 通过多个 `el-input` 组件绑定查询条件变量 (如 `toQueryCategory`、`toQueryTitle` 等)。
2. 动态构建请求: 使用 `axios.get` 发送请求时, 将查询条件作为 `params` 对象传递, 后端通过 `BookQueryConditions` 解析参数。
3. 排序支持: 提供 `el-select` 组件选择排序字段 (`sortBy`) 和顺序 (`sortOrder`), 映射到后端枚举值。

```
1. // 查询参数构建 (Book.vue 的 QueryBooks 方法)
2. await axios.get("/books", {
3.   params: {
4.     category: this.toQueryCategory,
5.     title: this.toQueryTitle,
6.     press: this.toQueryPress,
7.     minPublishYear: this.toQueryMinPublishYear,
8.     // ...其他参数
9.     sortBy: this.toQuerySortBy,
10.    sortOrder: this.toQuerySortOrder
11.  }
12. });
13.
14. // 后端动态 SQL 构建 (LibraryManagementSystemImpl.java 的 queryBook 方法)
15. if (conditions.getTitle() != null) {
16.   querySql.append(" AND title LIKE ?");
17.   params.add("%" + conditions.getTitle() + "%");
18. }
```

#### 批量操作:

1. 文件选择: 隐藏 `input[type=file]` 元素, 通过按钮触发文件选择。
2. 文件解析: 使用 `FileReader` 读取文件内容并解析为 `JSON` 数组。
3. 批量请求: 将解析后的图书列表通过 `POST /books` 批量提交, 后端通过 `storeBook(List<Book>)` 处理。

```
1. // 文件处理逻辑 (Book.vue 的 handleBatchImport 方法)
2. const reader = new FileReader();
3. reader.onload = async (e) => {
4.   const content = e.target.result;
5.   const books = JSON.parse(content);
6.   await axios.post("/books", books);
7. }
```

```

7.     };
8.
9.     // 后端批量插入 (LibraryManagementSystemImpl.java 的 storeBook 方法)
10.    for (Book book : booksToAdd) {
11.        insertStmt.setString(1, book.getCategory());
12.        insertStmt.addBatch(); // 批量添加
13.    }
14.    int[] batchResult = insertStmt.executeBatch();

```

### 交互优化:

1. 动态禁用按钮: 通过 `disabled` 绑定校验方法 (如 `isValidNewBookInfo()`) 。
2. 字段校验逻辑: 检查必填字段和非负数值, 例如价格和库存必须大于 0。
3. 错误提示: 使用 `ElMessage.error` 显示后端返回的错误信息。

```

1.    // 新增图书校验逻辑 (Book.vue 的 isValidNewBookInfo 方法)
2.    isValidNewBookInfo() {
3.        return (
4.            this.newBookInfo.title != "" &&
5.            this.newBookInfo.price > 0 &&
6.            this.newBookInfo.stock > 0
7.        );
8.    }
9.
10.   // 后端业务校验 (LibraryManagementSystemImpl.java 的 storeBook 方法)
11.   if (rs.next()) {
12.       return new ApiResult(false, "Book already exists"); // 唯一性校验
13.   }

```

### 借阅记录模块 (Borrow.vue)

**时间处理与展示:** 将时间戳转换为易读格式, 区分未归还记录。

1. 时间格式化方法: 使用 `formatDate` 方法将时间戳转为本地时间字符串。
2. 模板渲染: 在 `el-table-column` 中通过插槽自定义时间显示逻辑。

```

1.    // 时间格式化方法 (Borrow.vue)
2.    formatDate(timestamp) {
3.        return new Date(timestamp).toLocaleString();
4.    }
5.
6.    // 表格列渲染 (Borrow.vue 模板)
7.    <el-table-column prop="returnTime" label="归还时间">
8.        <template #default="{ row }">
9.            {{ row.returnTime !== 0 ? formatDate(row.returnTime) : '未归还' }}
10.        </template>
11.    </el-table-column>

```



**归还操作检验：**防止非法归还时间，确保业务逻辑正确性。

1. 时间选择器：使用 `el-date-picker` 选择归还时间，绑定到 `returnTimeH`。
2. 时间戳转换：将选择的日期转换为时间戳，并通过计算属性 `isReturnTimeValid` 校验。

```
1. // 时间转换与校验 (Borrow.vue)
2. convertToTimestamp(val) {
3.   this.returnBookInfo.returnTime = new Date(val).getTime();
4. },
5. isReturnTimeValid() {
6.   return this.returnBookInfo.returnTime > this.returnBookInfo.borrowTime;
7. }
8.
9. // 后端校验 (LibraryManagementSystemImpl.java 的 returnBook 方法)
10. if (returnTime <= borrowTime) {
11.   return new ApiResult(false, "Return time must be after borrow time");
12. }
```

## 借书证管理模块 (Card.vue)

**卡片化展示与模糊搜索：**直观展示卡证信息，支持快速定位

1. 循环渲染：使用 `v-for` 遍历 `cards` 数组，动态生成卡证卡片。
2. 模糊搜索：通过 `v-show="card.name.includes(toSearch)"` 实现客户端本地过滤。

```
1. <!-- 卡片渲染逻辑 (Card.vue 模板) -->
2. <div v-for="card in cards" v-show="card.name.includes(toSearch)">
3.   <p>{{ card.name }}</p>
4.   <p>{{ card.type === 'Student' ? '学生' : '教师' }}</p>
5. </div>
6.
7. <!-- 搜索框绑定 -->
8. <el-input v-model="toSearch" placeholder="姓名检索" />
```

**类型映射与表单处理：**统一前后端类型标识（如 `S/T` 转为学生/教师）

1. 前端显示转换：在模板中通过三元表达式直接映射类型。
2. 表单提交处理：将用户选择的类型（`S/T`）转换为后端需要的格式。

```
1. // 类型转换 (Card.vue 的 ConfirmModifyCard 方法)
2. const typeToSend = this.toModifyInfo.type === 'S' ? 'S' : 'T';
3. await axios.put(`/cards/${this.toModifyInfo.id}`, {
4.   type: typeToSend
5. });
6. // 后端枚举处理 (Card.java 的 CardType 枚举)
7. public enum CardType {
8.   STUDENT("S"), TEACHER("T");
9.   private final String str;
10.   // ...
11. }
```

级联删除与校验：删除卡证时确保无归还记录

1. 前置校验：后端检查 `Borrow` 表中是否存在未归还记录。
2. 级联删除：先删除关联的借阅记录，再删除卡证。
3. 具体实现：在后续的具体实现上，如果存在未归还书籍记录，直接禁止删除。

```
1. // 后端删除逻辑 (LibraryManagementSystemImpl.java 的 removeCard 方法)
2. String borrowSql = "SELECT 1 FROM Borrow WHERE cardId = ? AND returnTime = 0";
3. String deleteBorrowSql = "DELETE FROM Borrow WHERE cardId = ?";
4. String deleteCardSql = "DELETE FROM Card WHERE cardId = ?";
```

## 系统验证测试

### 正确性测试

通过 `LibraryTest.java` 来测试接口部分实现的正确性，并由图中可以得到，程序通过了所有的测试。

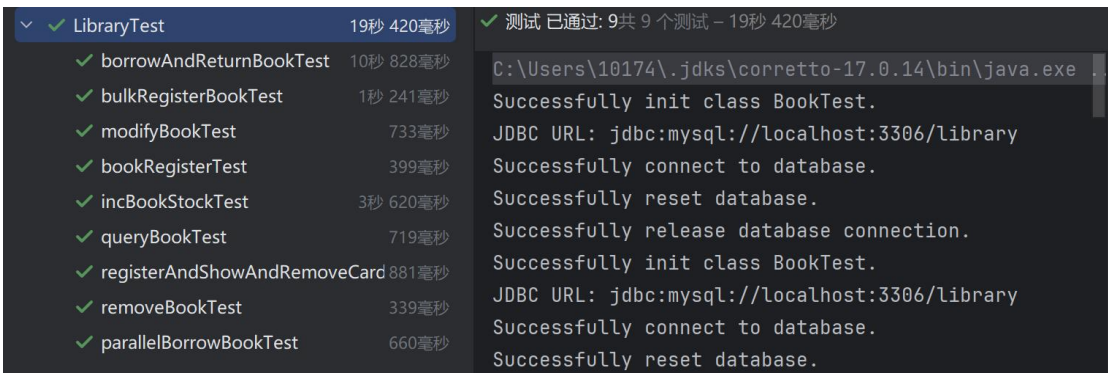


图 2 程序通过测试

### 功能性测试

#### 图书管理功能

##### 1. 展示功能

首先，是默认页面，展示数据库内所有的书籍的信息。

图书信息查询

+ 添加书本

批量导入

输入书本类别

输入书名

输入出版社

输入最小出版年份

输入最大出版年份

输入作者

输入最小价格

输入最大价格

选择排序字段

选择排序方式

查询

书本ID	类别	书名	出版社	出版年份	作者	价格	库存	操作
1	Nature	C++ Primer	Press-A	2007	Immorta	66.96	0	<div>修改</div> <div>删除</div>
2	悬疑	白夜行	教育出版	社	1980	东野圭吾	28.99	3 <div><div>修改</div><div>删除</div><div>借阅</div></div>
3	文学	美丽新世界	浙大出版	社	1755	阿道司	16.85	6 <div><div>修改</div><div>删除</div><div>借阅</div></div>
4	漫画	三毛流浪记	人民出版	社	2200	漫画家	16.85	2 <div><div>修改</div><div>删除</div><div>借阅</div></div>

(a) 图书查询界面

bookId	category	title	press	publishYear	author	price	stock
1	1 Nature	C++ Primer	Press-A	2007	Immortal	66.96	8
2	2 悬疑	白夜行	教育出版社	1980	东野圭吾	28.99	3
5	3 文学	美丽新世界	浙大出版社	1755	阿道司	16.85	6
4	4 漫画	三毛流浪记	人民出版社	2200	漫画家	16.85	2

(b) 数据库内图书信息

图 3 所有图书信息展示功能

第二，是按顺序和信息查询书籍信息，其中书本类型是精确搜索，书名、出版社和作者是模糊搜索实现。

小说

输入书名

输入出版社

输入最小出版年份

输入最大出版年份

输入作者

输入最小价格

输入最大价格

书本编号

降序

查询

书本ID

类别

书名

出版社

出版年份

作者

价格

库存

操作

5	小说	三体	重庆出版社	2008	刘慈欣	45	10	修改 删除 借阅
2	小说	白夜行	教育出版社	1980	东野圭吾	88.99	2	修改 删除 借阅

(a) 书本类别精确搜索（精确）

(b) 书本类别精确搜索（模糊）

输入书本类别

三

输入出版社

输入最小出版年份

输入最大出版年份

输入作者

输入最小价格

输入最大价格

出版日期

升序

查询

书本ID	类别	书名	出版社	出版年份	作者	价格	库存	操作
5	小说	三体	重庆出版社	2008	刘慈欣	45	10	修改 删除 借阅
4	漫画	三毛流浪记	人民出版社	2200	漫画家	16.85	2	修改 删除 借阅

(c) 书本名称模糊查询

输入书本类别

输入书名

出版社

输入最小出版年份

输入最大出版年份

输入作者

输入最小价格

输入最大价格

标题

升序

查询

书本ID	类别	书名	出版社	出版年份	作者	价格	库存	操作
5	小说	aaa三体	重庆出版社	2008	刘慈欣	45	10	修改 删除 借阅
4	漫画	bbb三毛流浪记	人民出版社	2200	漫画家	16.85	2	修改 删除 借阅
2	小说	ccc白夜行	教育出版社	1980	东野圭吾	88.99	2	修改 删除 借阅
6	编程	jjjava核心技术	机械工业出版社	2020	Cay S. Horstmann	149	5	修改 删除 借阅
3	文学	mmm美丽新世界	浙大出版社	1755	阿道司	16.85	6	修改 删除 借阅

(d) 出版社模糊查询

标题 ▾

升序 ▾

查询

书本ID	类别	书名	出版社	出版年份	作者	价格	库存	操作
1	Nature	C++ Primer	Press-A	2007	Immortal	66.96	0	修改 删除
6	编程	Java核心技术	机械工业出版社	2020	Cay S. Horstmann	149	5	修改 删除 借阅
5	小说	三体	重庆出版社	2008	刘慈欣	45	10	修改 删除 借阅
4	漫画	三毛流浪记	人民出版社	2200	漫画家	16.85	2	修改 删除 借阅

(e) 年份范围查询

标题 ▾

升序 ▾

查询

书本ID	类别	书名	出版社	出版年份	作者	价格	库存	操作
5	小说	三体	重庆出版社	2008	刘慈欣	45	10	修改 删除 借阅

(f) 作者名称模糊查询

库存 ▾

降序 ▾

查询

书本ID	类别	书名	出版社	出版年份	作者	价格	库存	操作
2	小说	白夜行	教育出版社	1980	东野圭吾	88.99	2	修改 删除 借阅
1	Nature	C++ Primer	Press-A	2007	Immortal	66.96	0	修改 删除

(g) 价格范围查询

图 4 所有图书信息展示功能

## 2. 删除功能

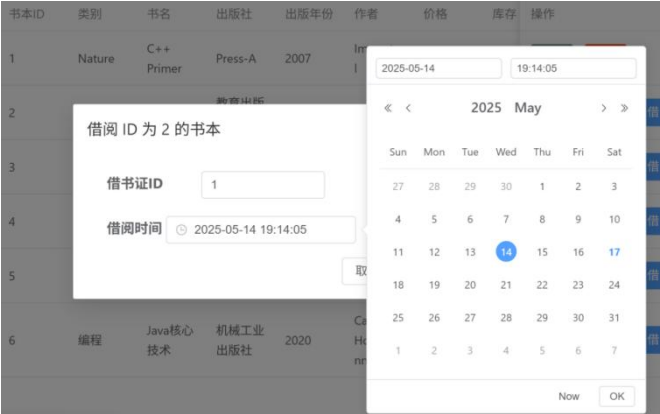
删除功能通过书本信息边上的红色按键实现，并如果该书存在图书未归还的记录，则删除失败。



图 5 删除图书弹窗

3. 借阅功能

借阅功能通过书本信息边上的蓝色按钮实现。



(a) 借阅弹窗与时间选择组件



(b) 借阅成功弹窗



(c) 重复借阅弹窗

图.6 图书借阅功能

4. 修改信息功能

修改功能通过书本信息边上的绿色按钮实现。

修改信息(书本 ID: 2)

修改信息(书本 ID: 2)

标题: 白夜行

出版社: 教育出版社

出版年份: 1980

作者: 东野圭吾

价格: 88.99

类别: 小说

库存: 2

取消 确定

(a) 修改信息弹窗



(b) 修改成功弹窗

图.7 修改图书功能


5. 添加书籍功能

新建单本书功能，并如果该书本以及存在的话，会出现弹窗提示。



图 8 新建书本弹窗

批量导入功能：



```
[
  {
    "category": "小说",
    "title": "三体",
    "press": "重庆出版社",
    "publishYear": 2008,
    "author": "刘慈欣",
    "price": 45.00,
    "stock": 10
  },
  {
    "category": "编程",
    "title": "Java核心技术",
    "press": "机械工业出版社",
    "publishYear": 2020,
    "author": "Cay S. Horstmann",
    "price": 149.00,
    "stock": 5
  }
]
```

(a) 批量导入弹窗

(b) .txt 文件内容



(c) 成功导入弹窗

图 9 批量导入

借书证管理功能

1. 卡片式展示

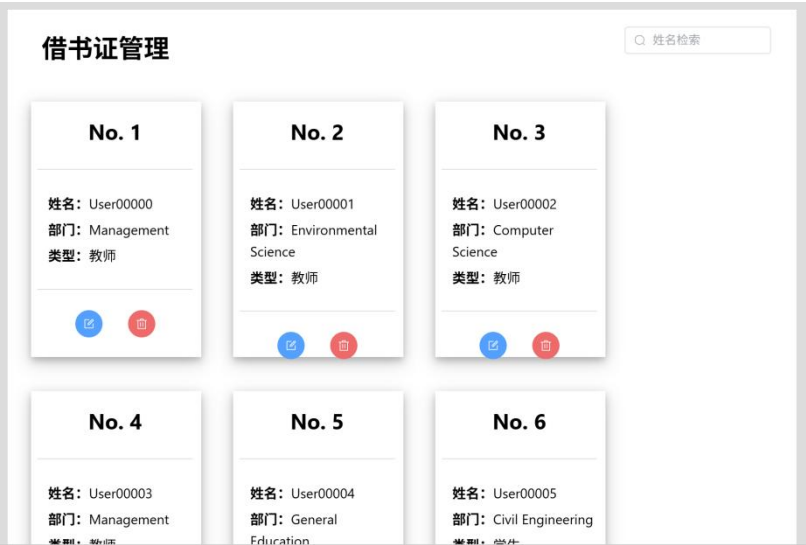


图 10 卡片式借书证展示页面

2. 新增借书证



图 11 新建借书证弹窗



图 12 卡片式新建按钮

3. 修改借书证信息




(a) 修改借书证窗口



(b) 修改成功弹窗

图 13 修改借书证

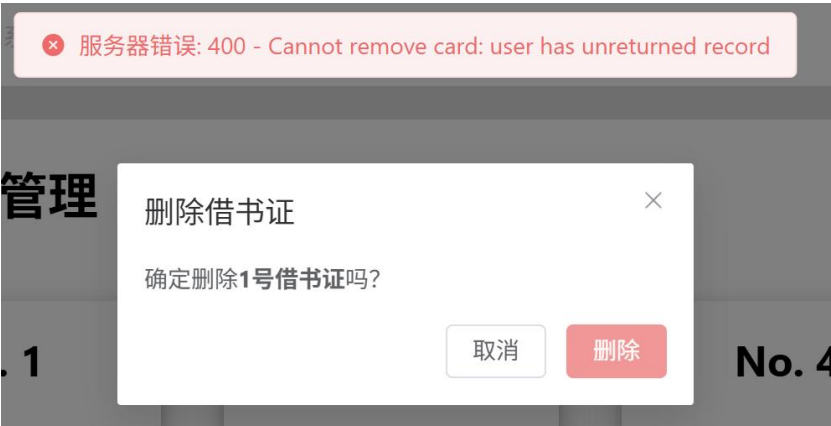
4. 借书证删除



(a) 删除借书证窗口



(b) 删除成功弹窗



(b) 删除失败情况

图 14 删除借书证



5. 名字检索



图 15 通过姓名模糊检索借书证

借书记录管理功能

根据借书证 ID 进行搜索，如果属于未归还状态时，就会出现蓝色的归还按钮。

借书记录查询

借书证ID	图书ID	借出时间	归还时间	归还操作
1	2	2025/5/14 19:14:05	未归还	<input type="button" value="归还"/>
1	1	2025/5/17 16:20:03	未归还	<input type="button" value="归还"/>

图 16 借阅记录查询

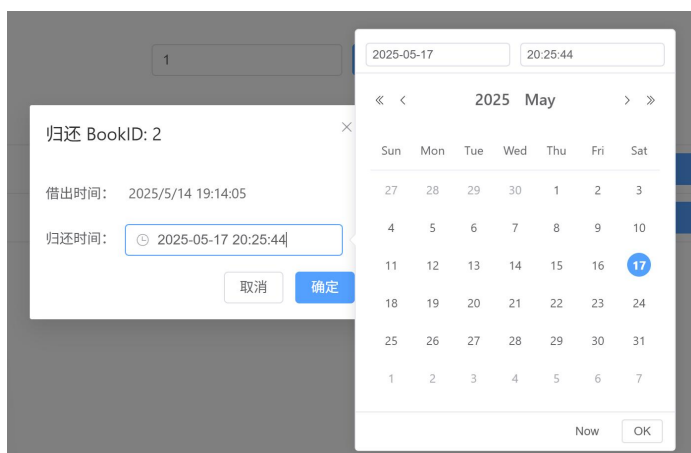
归还书籍，要求归还时间必须要晚于借出时间，否则无法选中确定按钮

归还 BookID: 2

借出时间: 2025/5/14 19:14:05

归还时间:

(a) 归还时间检查



(b) 时间选择组件



(c) 还书成功弹窗

图 17 还书操作

归还成功后，则归还按键消失

## 借书记录查询

1

查询

借书证ID	图书ID	借出时间	归还时间	归还操作
1	2	2025/5/14 19:14:05	2025/5/17 20:25:44	
1	1	2025/5/17 16:20:03	未归还	归还

图 18 还书后的借阅记录

## 遇到的问题及解决方法

1. 在前期写代码时，未加入足够的错误信息输出，导致后期在 debug 时，经常处于一个盲人摸象的状态

解决方法：在代码可能出错处，加入足够的报错信息，帮助调整代码。

2. 在网页上无法正确显示借书卡片

解决方法：一开始对于 API 返回结果的数据结构不够清晰理解，错误使用卡片数据。但后方返回的 ApiResponse 对象，实际的数据其实是在 payload 字段中。

3. 批量插入书籍时，无法正确获取所有自增 ID，导致索引越界异常

解决方法：因为默认情况下，MySQL 的 JDBC 驱动在批处理插入时不会返回所有生成的主键，导致获取自增 ID 时索引越界。故需要修改数据库的连接 URL。具体解决方法是，把 DatabaseConnector.java 文件里的

```
String url = conf.getType().url(conf.getHost(), conf.getPort(), conf.getDB());
```

改为

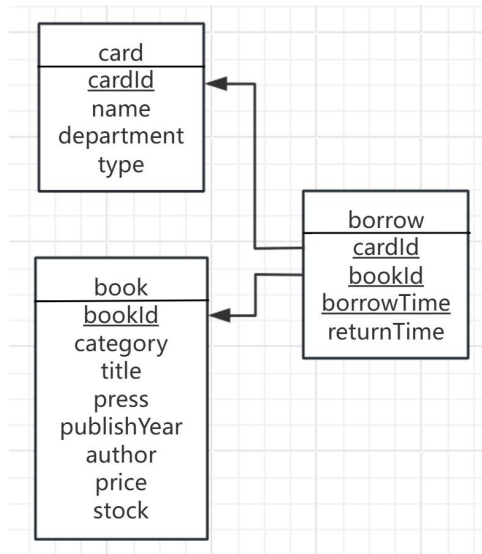
```
String baseUrl = conf.getType().url(conf.getHost(), conf.getPort(), conf.getDB());
```

```
String url = baseUrl + (baseUrl.contains("?") ? "&" : "?") + "rewriteBatchedStatements=true";
```

即可正常实现批量导入功能。需要注意的是，如果修改了这里后，会导致原测试无法通过，故而需要把这里改回原样，才能正常通过后端实现测试。

## 思考题

### 1. 绘制该图书管理系统的 E-R 图。



### 2. 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？

原理：SQL 注入通过用户输入的数据篡改 SQL 语句逻辑，从而执行非预期的数据库操作。

例如，在登录功能中，若 SQL 语句直接拼接用户输入，

```
SELECT * FROM users WHERE username = '${userInput}' AND password = '${password}'
```

当用户输入 'OR '1'='1' 作为用户名时，SQL 变为：

```
SELECT * FROM users WHERE username = '' OR '1'='1' AND password = 'xxx'
```

此时条件恒为真，攻击者可绕过身份验证。

而在图书管理系统中，图书查询模块，借书证管理模块，借阅记录查询模块等易受攻击，但是通过过滤特殊字符或是参数化查询，能一定程度上避免攻击。

### 3. 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？

快照读指事务内多次读取同一数据时，结果基于事务开始时的快照，保证一致性。

当前读指读取最新数据时加锁，阻塞其他事务修改。

解决方法：通过显示加锁，或是通过 RR 级别的行锁，防止其他事务修改已锁定的行，直到当前事务提交。

而在本次实验中，也是通过 select...for update 来解决的。

### 4. 在实际应用（例如电商系统）中，“秒杀”、“团购”是频繁出现的一些活动。“秒杀”活动通常伴随着高并发、访问量激增等特点。当并发请求数过多时，“秒杀”系统又是如何防止库存超卖的呢？

通过检索资料，得到：

悲观锁：使用 SELECT ... FOR UPDATE 锁定库存行，确保串行化扣减

乐观锁：通过版本号控制。在库存表中增加 version 字段，更新时校验版本号

分布式锁：通过 Redis 锁，使用 SETNX 命令或 Redisson 实现分布式锁，限制同一商品仅一个请求处理扣减。

异步队列：通过请求排队，将秒杀请求放入消息队列（如 Kafka），由消费者顺序处理，缓解瞬时压力。

缓存预扣：通过 Redis 原子操作，利用 DECR 命令预扣库存，成功后异步同步至数据库。