

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

基本数据对象

对象名称	类名	包含属性
书	Book	书号, 类别, 书名, 出版社, 年份, 作者, 价格, 剩余库存
借书证	Card	卡号, 姓名, 单位, 身份(教师或学生)
借书记录	Borrow	卡号, 书号, 借书日期, 还书日期

基本功能模块

图书入库模块

```
ApiResponse storeBook(Book book)
```

图书增加库存模块

```
ApiResponse incBookStock(int bookId, int deltaStock)
```

图书批量入库模块

```
ApiResponse storeBook(List<Book> books)
```

图书删除模块

```
ApiResponse removeBook(int bookId)
```

图书修改模块

```
ApiResponse modifyBookInfo(Book book)
```

图书查询模块

```
ApiResponse queryBook(BookQueryConditions conditions)
```

借书模块

```
ApiResponse borrowBook(Borrow borrow)
```

还书模块

```
ApiResponse returnBook(Borrow borrow)
```

借书记录查询模块

```
ApiResponse showBorrowHistory(int cardId)
```

借书证注册模块

```
ApiResponse registerCard(Card card)
```

删除借书证模块

```
ApiResponse removeCard(int cardId)
```

借书证查询模块

```
ApiResponse showCards()
```

数据库表定义

```
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
);
```

功能验证

1. 参考模拟场景

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
批量入库	输入图书导入文件的路径U, 然后从文件U中批量导入图书
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

2. 实现基于vue框架的前端

实验环境

- JDK21
- Apache Maven 3.9.2
- Node.js 10.5.0
- vscode

模块设计

基本功能模块

1. `ApiResult storeBook(Book book)` 图书入库模块
首先通过SELECT语句判断图书是否已经存在库中，如果已经存在，则返回false；否则，继续执行INSERT语句
2. `ApiResult incBookStock(int bookId, int deltaStock)` 图书增加库存模块
首先通过SELECT语句判断图书是否存在，如果不存在返回false，如果存在，则根据SELECT语句得到的图书库存和输入的库存变更量判断更新库存后数量是否非负，如果非负，则更新成功，否则更新失败，返回false
3. `ApiResult storeBook(List<Book> books)` 图书批量入库模块
和单一图书入库模块区别在于调用addBatch函数，批量处理INSERT语句
4. `ApiResult removeBook(int bookId)` 图书删除模块
首先对table borrow执行SELECT语句，判断目标图书是否被借用，如果有某本没被归还，则不能执行移除操作，返回false和Book not returned；接着对table book执行SELECT语句，判断库中是否存在图书，如果不存在，则返回false和Book not exists；如果以上两种情况都不存在，就执行DELETE语句
5. `ApiResult modifyBookInfo(Book book)` 图书修改模块
首先使用SELECT语句，判断图书是否存在并获取图书信息，接着根据输入的目标信息调用UPDATE语句更新图书信息
6. `ApiResult queryBook(BookQueryConditions conditions)` 图书查询模块
根据输入的条件信息，按要求的类别和升降顺序使用SELECT语句查询信息，构建bookQueryResults对象返回
7. `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，若用户此前借过这本书但尚未归还，那么借书操作将失败 首先使用SELECT语句查询table book，查询库中是否存在该图书以及图书的存量是否为正，如果图书不存在或者没有库存了，则借书操作失败；接着使用SELECT语句查询table card，查询该借书证是否已经注册过，如果没注册过，则借书失败；然后使用SELECT语句查询table borrow，查询借书证持有者是否借过该书并且没有归还，如果没有归还，则借书失败 如果并非以上几种情况，则说明借书操作可行，使用UPDATE函数更新图书库存，然后使用INSERT函数更新borrow table
8. `ApiResult returnBook(Borrow borrow)` 还书模块
使用UPDATE语句对table borrow和table book操作，更新图书存量和借书记录
9. `ApiResult showBorrowHistory(int cardId)` 借书记录查询模块
使用SELECT语句对table borrow和table book查询借书记录，按照借书时间递减、书号递增的方式排

序

10. `ApiResult registerCard(Card card)` 借书证注册模块

使用`SELECT`语句查询`table card`，判断是否重复注册；如果之前没注册过该借书证，则使用`INSERT`语句向`table card`插入借书证记录

11. `ApiResult removeCard(int cardId)` 删除借书证模块

首先使用`SELECT`语句查询使用该借书证借的书是否已经归还，如果未归还，则无法删除该借书证；否则使用`DELETE`语句删除借书证。

12. `ApiResult showCards()` 使用`SELECT`语句查询，最后构建`CardList`返回。

前端页面模块

1. `index.js`规定页面url``

```
const router = createRouter({
  history: createWebHistory(import.meta.env.BASE_URL),
  routes: [
    {
      path: '/',
      redirect: '/book'
    },
    {
      path: '/book',
      component: BookVue
    },
    {
      path: '/card',
      component: CardVue
    },
    {
      path: '/borrow',
      component: BorrowVue
    },
    {
      path: '/lookup-book', // add a new route for LookupBook.vue
      component: LookupBookVue
    }
  ]
})
```

2. `Book.vue`图书管理页面\

- `ConfirmTodo1`图书入库处理函数

使用`post`语句，请求头包括图书信息，在响应处理中更新前端图书对象各个域的值，其他函数类似

```
ConfirmTodo1(){//图书入库
  axios.post("/book",
    { //request body
      task: "图书入库",
      category: this.todo1Info.category,
```

```

        title: this.todo1Info.title,
        press: this.todo1Info.press,
        publishYear: this.todo1Info.publishYear,
        author: this.todo1Info.author,
        price: this.todo1Info.price,
        stock: this.todo1Info.stock
    })
    .then(response => {
        this.todo1Visible = false;
        if(response.data){
            ElMessage.success("图书入库成功");
        }else{
            ElMessage.error("图书入库失败");
        }
        this.todo1Info={
            category: '',
            title:'',
            press:'',
            publishYear:'',
            author:'',
            price:'',
            stock:''
        };
    })
},

```

- **ConfirmTodo2()** 批量入库

```

ConfirmTodo2(){// 批量入库
    axios.post("/book",
        { //request body
            task: "批量入库",
            url: this.todo2Info.url
        })
    .then(response => {
        this.todo2Visible = false;
        if(response.data){
            ElMessage.success("批量入库成功");
        }else{
            ElMessage.error("批量入库失败");
        }
        this.todo2Info={
            url:''
        };
    })
},

```

- **ConfirmTodo3** 增加库存

```
ConfirmTodo3(){// 增加库存
  axios.post("/book",
    { //request body
      task: "增加库存",
      bookId: this.todo3Info.bookId,
      deltaStock: this.todo3Info.deltaStock
    })
  .then(response => {
    this.todo3Visible = false;
    if(response.data){
      ElMessage.success("增加库存成功");
    }else{
      ElMessage.error("增加库存失败");
    }
    this.todo3Info={
      bookId:'',
      deltaStock:''
    };
  })
}
```

- **ConfirmTodo4**借阅图书

```
ConfirmTodo4(){// 借阅图书
  axios.post("/book",
    { //request body
      task: "借阅图书",
      bookId: this.todo4Info.bookId,
      cardId: this.todo4Info.cardId,
      borrowTime: this.todo4Info.borrowTime
    })
  .then(response => {
    this.todo4Visible = false;
    if(response.data){
      ElMessage.success("借阅图书成功");
    }else{
      ElMessage.error("借阅图书失败");
    }
    this.todo4Info={
      cardId:'',
      bookId:'',
      borrowTime:''
    };
  })
},
```

- **ConfirmTodo5**归还图书

```
ConfirmTodo5(){//归还图书
  axios.put("/book",
    { //request body
      task: "归还图书",
      bookId: this.todo5Info.bookId,
      cardId: this.todo5Info.cardId,
      borrowTime: this.todo5Info.borrowTime,
      returnTime: this.todo5Info.returnTime
    })
  .then(response => {
    this.todo5Visible = false;
    if(response.data){
      ElMessage.success("归还图书成功");
    }else{
      ElMessage.error("归还图书失败");
    }
    this.todo5Info={
      cardId:'',
      bookId:'',
      borrowTime:'',
      returnTime:''
    };
  })
},
```

- **ConfirmTodo6**修改图书信息

```
ConfirmTodo6(){//修改图书信息
  axios.put("/book",
    { //request body
      task: "修改图书信息",
      bookId : this.todo6Info.bookId,
      category: this.todo6Info.category,
      title: this.todo6Info.title,
      press: this.todo6Info.press,
      publishYear: this.todo6Info.publishYear,
      author: this.todo6Info.author,
      price: this.todo6Info.price,
      stock: this.todo6Info.stock
    })
  .then(response => {
    this.todo6Visible = false;
    if(response.data){
      ElMessage.success("图书信息修改成功");
    }else{
      ElMessage.error("图书信息修改失败");
    }
  }
  this.todo6Info= {
    bookId:'',
    category:'',
    title:'',
  }
```



```

        press:'',
        publishYear:'',
        author:'',
        price:'',
        stock:''
    };

    })
},

```

3. Borrow.vue借书记录查询页面\

使用后端传来的数据更新借书证卡片信息

```

methods: {
    async QueryBorrows() {
        this.tableData = [] // 清空列表
        let response = await axios.get('/borrow', { params: { cardID: this.toQuery
    } }) // 向/borrow发出GET请求，参数为cardID=this.toQuery
        let borrows = response.data // 获取响应负载
        borrows.forEach(borrow => { // 对于每一个借书记录
            this.tableData.push(borrow) // 将它加入到列表项中
        });
        this.isShow = true // 显示结果列表
    }
}

```

4. Card.vue借书证管理页面

这里以两种函数举例

- 修改借书证卡片函数

```

async modifyCard(id, name, department, type){
    try{
        const response = await axios.put("/card", {id, name, department, type});

        // const response = await axios.put("/card", {data:{id, name, department,
type}});
        if(response.data){
            ElMessage.success("借书证修改成功");
            return true;
        }else{
            ElMessage.error("借书证修改失败");
            return false;
        }
    }catch(error){
        ElMessage.error("借书证修改失败");
        console.error(error);
        return false;
    }
}

```

```

    },
    ConfirmModifyCard() {
      // TODO: YOUR CODE HERE
      this.modifyCard(this.toModifyInfo.id,
        this.toModifyInfo.name,
        this.toModifyInfo.department,
        this.toModifyInfo.type)
        .then(success =>{
          this.modifyCardVisible = false;
          if(success){
            this.QueryCards();
          }
        })
    },
  },

```

- 移除借书证卡片函数

```

async deleteCard(id){
  try{
    const response = await axios.delete("/card", {data: {id}});
    if(response.data){
      ElMessage.success("借书证删除成功");
      return true;
    }else{
      ElMessage.error("借书证删除失败");
      return false;
    }
  }catch(error){
    ElMessage.error("借书证删除失败");
    console.error(error);
    return false;
  }
},

ConfirmRemoveCard() {
  // TODO: YOUR CODE HERE
  this.deleteCard(this.toRemove).then(success =>{
    this.removeCardVisible = false;
    if(success){
      this.QueryCards();
    }
  })
},

```

5. `LookupBook.vue` 图书查询页面

使用前端页面输入的查询升降顺序和查询类别，与后端交互，得到的值更新图书的信息

```

async searchBooks(){
  this.books = []

```

```
const response = await axios.get('/book', {
  params: {
    filters: JSON.stringify(this.filters),
    selected: this.selected,
    order: this.order
  });
this.books = response.data;
this.isShow = true
this.filters={
  category:'',
  title:'',
  press:'',
  minPublishYear:'',
  maxPublishYear:'',
  author:'',
  minPrice:'',
  maxPrice:''
}
}
```

web响应处理部分

1. **BookHandler** 定义**handler**函数，对**GET**、**POST**和**PUT**处理头分别处理，并定义相关函数。使用Jackson处理json流。

```
public void handle(HttpExchange exchange) throws IOException{
    // 允许所有域的请求，cors处理，解决跨域问题
    Headers headers = exchange.getResponseHeaders();
    headers.add("Access-Control-Allow-Origin", "*");
    headers.add("Access-Control-Allow-Methods", "GET, POST, PUT, OPTIONS");
    headers.add("Access-Control-Allow-Headers", "Content-Type");
    // 解析请求的方法，看GET还是POST
    String requestMethod = exchange.getRequestMethod();
    if(requestMethod.equals("GET")){
        handleGetRequest(exchange);
    }else if(requestMethod.equals("POST")){
        handlePostRequest(exchange);
    }else if(requestMethod.equals("OPTIONS")){
        handleOptionsRequest(exchange);
    }else if(requestMethod.equals("PUT")){
        handlePutRequest(exchange);
    }else{
        exchange.sendResponseHeaders(405, -1);
    }
}
```

处理json的函数段

```
class Filters {
    private String category;
    private String title;
    private String press;
    private String minPublishYear;
    private String maxPublishYear;
    private String author;
    private String minPrice;
    private String maxPrice;
    public String getCategory(){return this.category;}
    public String getPress(){return this.press;}
    public String getTitle(){return this.title;}
    public String getAuthor(){return this.author;}
    public String getMinPublishYear(){return this.minPublishYear;}
    public String getMaxPublishYear(){return this.maxPublishYear;}
    public String getMinPrice(){return this.minPrice;}
    public String getMaxPrice(){return this.maxPrice;}
}
```

```
URI requestURI = exchange.getRequestURI();
String query = requestURI.getQuery();
Map<String, String> params = parseQuery(query);

// Now you can use the params map to get your parameters
String sortBy = params.get("selected");
String sortOrder = params.get("order");
String filtersJson = params.get("filters");

// Parse the filters JSON string into a Filters object
ObjectMapper mapper = new ObjectMapper();
Filters filters = mapper.readValue(filtersJson, Filters.class);
String category = filters.getCategory();
String title = filters.getTitle();
String press = filters.getPress();
String author = filters.getAuthor();
```

2. `CardHandler` 与 `BookHandler` 类似
3. `BorrowHandler` 与 `BookHandler` 类似

Application部分

首先获取数据库连接对象，接着为 `/card/`, `/borrow/`, `/bookurl` 注册JSON处理函数，然后启动服务器。

```
try {
    // parse connection config from "resources/application.yaml"
    ConnectConfig conf = new ConnectConfig();
    log.info("Success to parse connect config. " + conf.toString());
}
```

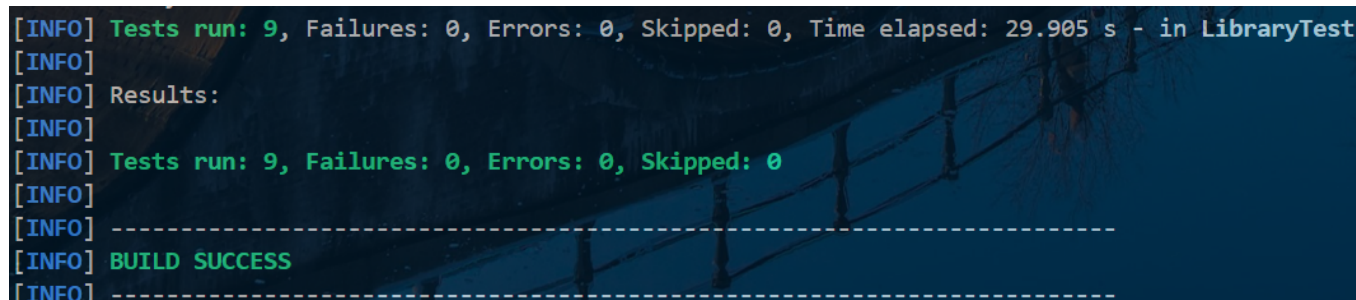
```
// connect to database
DatabaseConnector connector = new DatabaseConnector(conf);
boolean connStatus = connector.connect();
if (!connStatus) {
    log.severe("Failed to connect database.");
    System.exit(1);
}
/* do somethings */
// create library interfaces instance
LibraryManagementSystemImpl libImpl = new
LibraryManagementSystemImpl(connector);
// basic block
// create HTTP server, listen to the port 8000
HttpServer server = HttpServer.create(new InetSocketAddress(8000), 0);
//add handler, here bound to /card router
server.createContext("/card", new CardHandler(libImpl));
server.createContext("/borrow", new BorrowHandler(libImpl));
server.createContext("/book", new BookHandler(libImpl));

// start server
server.start();
// mark
System.out.println("Server is listening on port 8000");
// keep the server running until it is no longer needed
Runtime.getRuntime().addShutdownHook(new Thread(() -> {
    // release database connection handler
    if (connector.release()) {
        log.info("Success to release connection.");
    } else {
        log.warning("Failed to release connection.");
    }
}));
} catch (Exception e) {
    e.printStackTrace();
}
```

验证测试

测试点通过情况

如图，测试通过



```
[INFO] Tests run: 9, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 29.905 s - in LibraryTest
[INFO]
[INFO] Results:
[INFO]
[INFO] Tests run: 9, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
```

前端交互测试

验收已展示功能，这里不一一列举

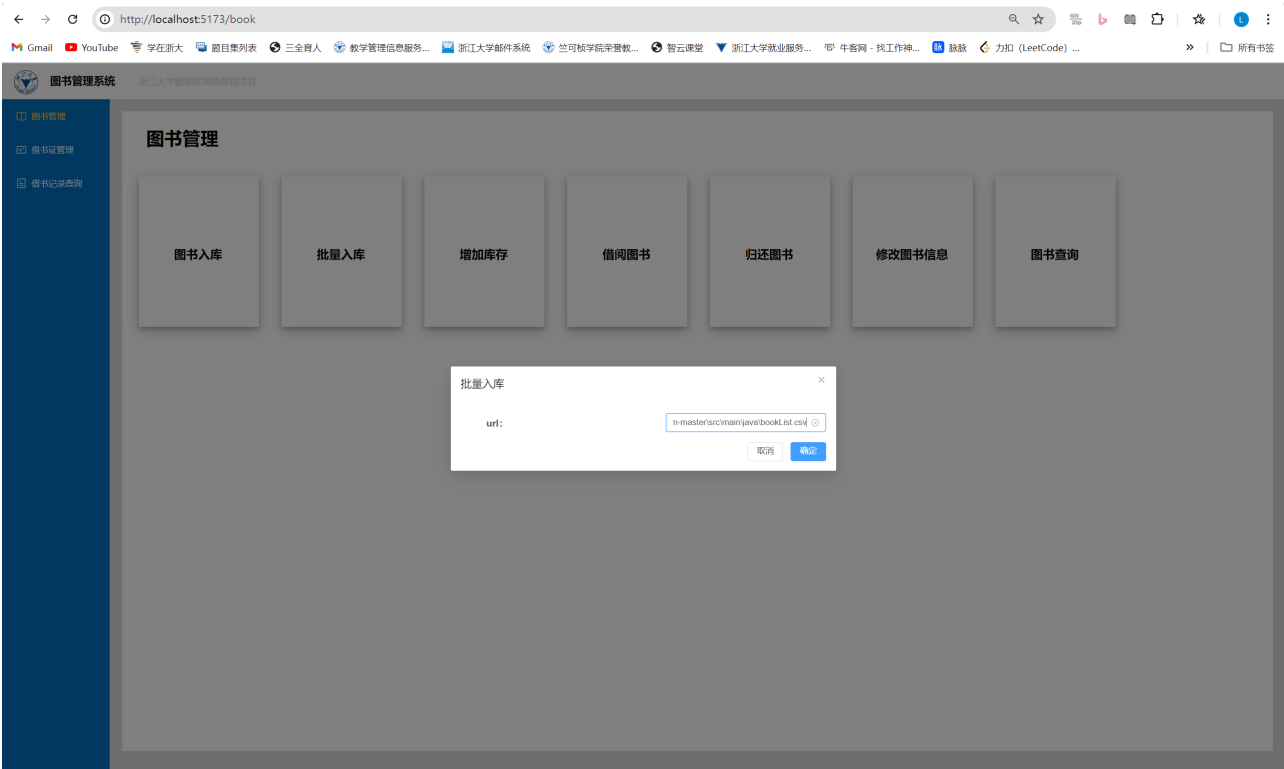
1. 图书管理界面



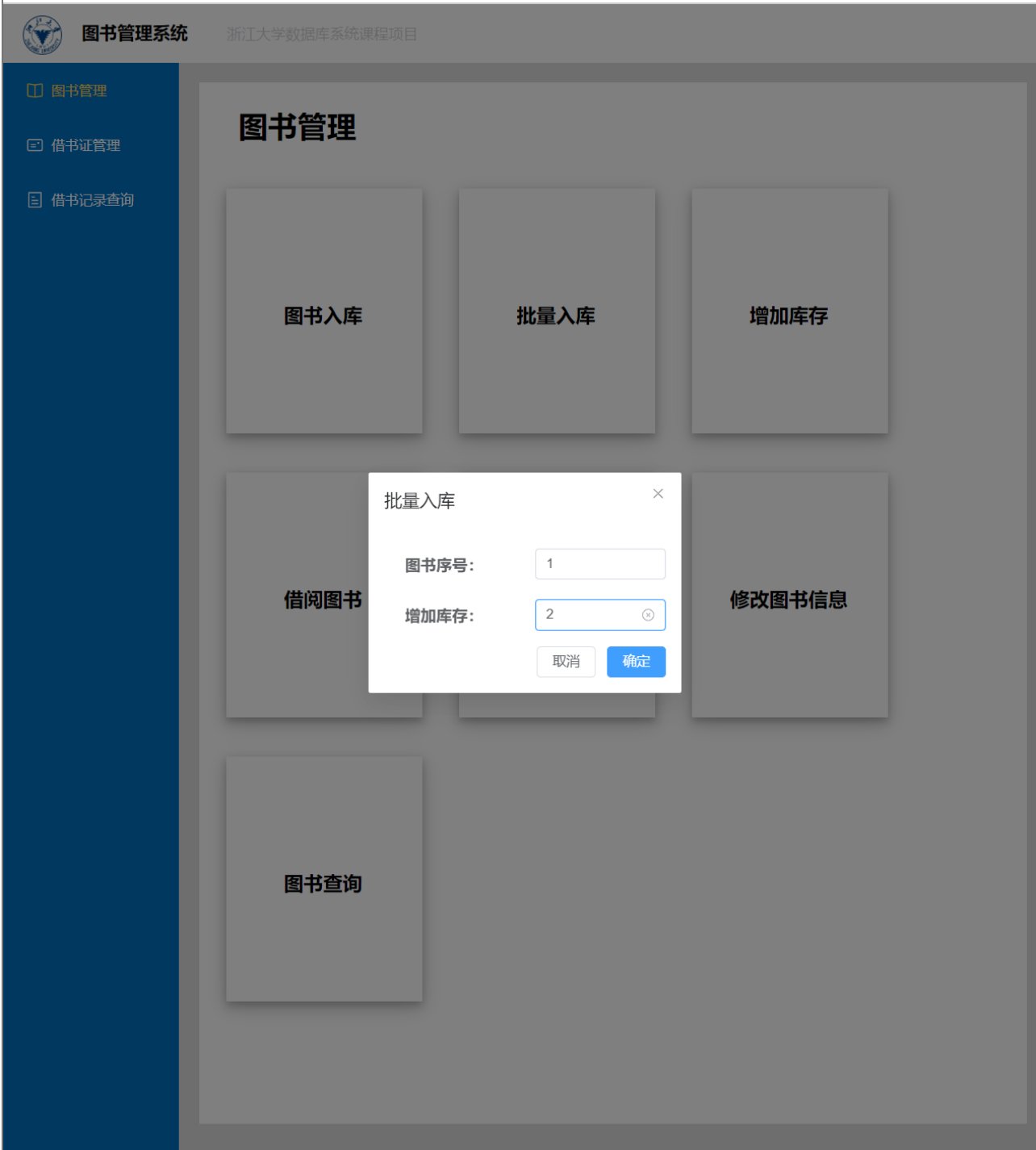
图书入库选项卡



批量入库选项卡



增加库存



图书记录查询



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

图书查询

筛选条件

种类

标题

出版社

最小出版年份

最大出版年份

作者

最低价格

最高价格

排序参照

种类


顺序

升序

查询

bookId	category	title	press	publishYear	author	price	stock
2	1	1	1	1	1	1	1
1	Autobiography	Compiler Designs	Press-F	2019	Erica	120.88	0

2. 借书证界面



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

借书证管理

No. 1

姓名: User00000

部门: Law

类型: Student

No. 2

姓名: User00001

部门: Environmental Science

类型: Teacher

No. 3

姓名: User00002

部门: Civil Engineering

类型: Teacher

No. 4

姓名: User00003

部门: Environmental Science

类型: Student

No. 5

姓名: User00004

部门: Civil Engineering

类型: Teacher

No. 6

姓名: User00005

部门: Law

类型: Teacher

No. 7

姓名: User00006

部门: Civil Engineering

类型: Teacher

No. 8

姓名: User00007

部门: Environmental Science

类型: Student

No. 9

姓名: User00008

部门: Management

类型: Student

3. 借书记录查询



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

借书记录查询

1

查询

借书证ID	图书ID	借出时间	归还时间
1	2	1	0

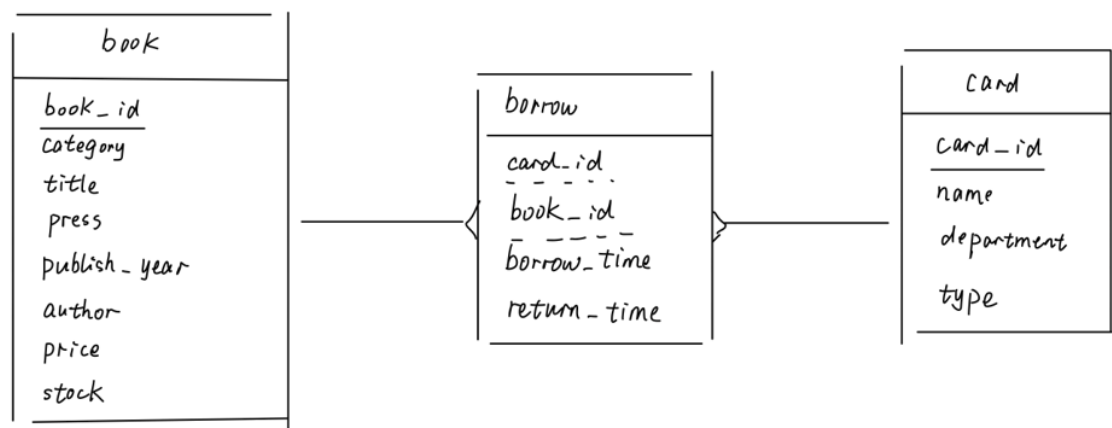
问题与方案

为了要让前后端持续交互，使用如下函数块释放数据库连接

```
Runtime.getRuntime().addShutdownHook(new Thread(() -> {
    // release database connection handler
    if (connector.release()) {
        log.info("Success to release connection.");
    } else {
        log.warning("Failed to release connection.");
    }
}));
```

思考题

1. 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？ SQL注入攻击是一种代码注入技术，攻击者通过输入恶意的SQL代码，来操控后端数据库的行为。例如，对于一个查询是这样构建的："SELECT * FROM borrow WHERE bookID = '' + cardID = + '';"，攻击者可以输入"a'; DROP TABLE book; SELECT * FROM Card WHERE 'x' = 'x"，这将导致删除borrow表。在图书管理系统中，任何接受用户输入作为查询参数的模块都可能受到SQL注入攻击，例如搜索图书、借阅图书、注册新的借书卡等。防止SQL注入的方法主要有：使用预编译语句)、使用存储过程、输入验证和转义等。
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？下面是一个在该场景下可能会出现非预期结果的例子：在RR隔离级别下，InnoDB使用多版本并发控制(MVCC)来保证事务的隔离性。在这种情况下，每个事务都会看到一个一致的快照版本的数据。因此，即使在并发访问时，也能保证数据的一致性。但是，如果两个事务都试图修改同一条记录，可能会出现数据不一致的情况。例如，如果两个用户同时试图借阅最后一本书，可能会出现库存为负数的情况。为了避免这种情况，可以在修改数据前加锁，确保在任何时候只有一个事务能够修改数据。