实验5 图书管理系统

郑乔尹 3210104169

实验目的

1. 设计并实现一个精简的图书管理系统，具有**入库**、**查询**、**借书**、**还书**、**借书证管理**等基本功能
2. 通过本次设计来加深对数据库的了解和使用，同时提高自身的系统编程能力

实验平台

1. 数据库平台：SQL Server 2022
2. 操作系统：Windows11 22H2
3. 开发工具：JetBrain IntelliJ IDEA
4. 实验环境：Java 17.0.6
5. SQL server端口：1433（默认端口）

实验内容和要求

1. 基于JDBC开发一个图书管理系统，要求实现如下功能

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

功能	描述
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. 以下是数据表的定义

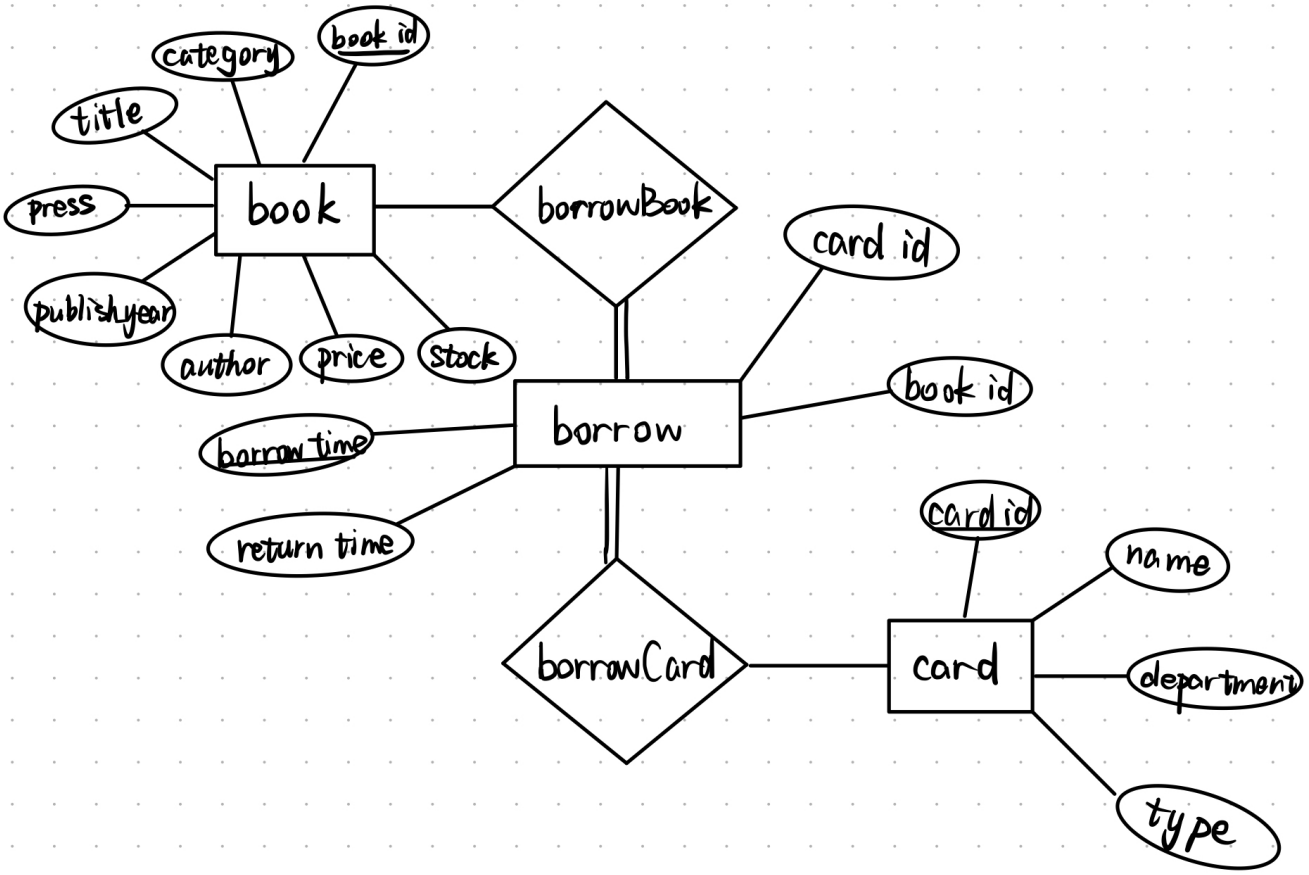
```SQL
create table `book` (
    `book_id` int not null auto_increment,
    `category` varchar(63) not null,
    `title` varchar(63) not null,
    `press` varchar(63) not null,
    `publish_year` int not null,
    `author` varchar(63) not null,
    `price` decimal(7, 2) not null default 0.00,
    `stock` 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
);
```

实验过程

E-R Diagram



参数化查询

- 通过以下形式对输入的数据进行处理以避免SQL注入攻击（以`storeBook`为例）：

JAVA

```java
@Override
public ApiResult storeBook(Book book) {
    String checkBook = "SELECT * FROM book WHERE category = ? AND
title = ? AND press = ? AND publish_year = ? AND author = ?";
    String insertBook = "INSERT INTO book (category, title, press,
publish_year, author, price, stock) VALUES (?, ?, ?, ?, ?, ?, ?)";
    Connection connection = connector.getConn();
    try {
        PreparedStatement bookStatement =
connection.prepareStatement(checkBook);
        bookStatement.setString(1, book.getCategory());
        bookStatement.setString(2, book.getTitle());
        bookStatement.setString(3, book.getPress());
        bookStatement.setInt(4, book.getPublishYear());
        bookStatement.setString(5, book.getAuthor());
        ResultSet resultSet = bookStatement.executeQuery();
        if(resultSet.next()) { //找到了一样的
            return new ApiResult(false, "Error! Already Exists!");
        }
    } catch (SQLException e) {
        throw new RuntimeException(e);
    }

    try {
        PreparedStatement bookStatement =
connection.prepareStatement(insertBook,
Statement.RETURN_GENERATED_KEYS);// 补全占位符，获取自动主键
        bookStatement.setString(1, book.getCategory());
        bookStatement.setString(2, book.getTitle());
        bookStatement.setString(3, book.getPress());
        bookStatement.setInt(4, book.getPublishYear());
        bookStatement.setString(5, book.getAuthor());
        bookStatement.setDouble(6, book.getPrice());
        bookStatement.setDouble(7, book.getStock());
        int newRow = bookStatement.executeUpdate();
        if (newRow == 0) {
            throw new SQLException("Creating book failed, no rows
affected.");
        }
```

```
        try (ResultSet generatedKeys =
bookStatement.getGeneratedKeys()) {//获取主键，赋值回book
            if (generatedKeys.next()) {
                book.setBookId(generatedKeys.getInt(1));
            } else {
                throw new SQLException("Creating book failed, no ID
obtained.");
            }
        }
        commit(connection);
        return new ApiResult(true, "Stored successfully");
    } catch (SQLException e) {
        rollback(connection);
        e.printStackTrace();
        return new ApiResult(false, "Error storing book: " +
e.getMessage());
    // throw new RuntimeException(e);
    }
}
```

1. 单本图书入库

功能要求

- 对单本图书进行入库，要求实现查重功能，对`category`，`title`，`press`，`publish year`，`author`三个字段进行查重操作，即对于这五个字段相同的书籍，视为重复书籍，不进行入库
- 对于不重复的书籍，分配book id，入库

原理

- 采用如下SQL语句

> 由于`book id`设为自动分配的主键，会自动生成当前最后一个连续值

书籍查重

```sql
SELECT * FROM book WHERE category = ? AND title = ? AND press = ? AND
publish_year = ? AND author = ?
```

插入书籍

```sql
INSERT INTO book (category, title, press, publish_year, author,
price, stock) VALUES (?, ?, ?, ?, ?, ?, ?)
```

2. 减少库存

功能要求

- 对于输入的两个参数`bookID`，`deltaStock`，先对书库中的书籍进行查询，如果书籍不存在，则减少库存失败；若书籍存在，需要先对书籍现在的库存进行检测，若书籍的库存不足以减少这么多书籍，则减少库存失败，反之更新库存。

原理

书籍存在性检测

```sql
SELECT * FROM book WHERE book_id = ?
```

书籍库存检查

- 在上述查询出来的结果集中，对列`stock`进行查询
- 检查其剩余库存

```java
int currentStock = resultSet.getInt("stock");
if (currentStock + deltaStock < 0) {
    rollback(connection);
    return new ApiResult(false, "No such books to be incline!");
} else {
    rollback(connection);
    return new ApiResult(false, "No such book_id!");
}
catch (SQLException e) {
    throw new RuntimeException(e);
}
```

减少库存

```sql
UPDATE book SET stock = stock + ? WHERE book_id = ?
```

3. 图书批量入库

要求

- 一次加入多本书籍，同时，对这些插入的书籍也进行和单本一样的查重，一旦有重复，则插入失败，整个List无重复则插入成功

原理

- 对书籍进行查重以及插入的SQL语句

```sql
SELECT * FROM book WHERE category = ? AND title = ? AND press = ? AND
publish_year = ? AND author = ?
INSERT INTO book (category, title, press, publish_year, author,
price, stock) VALUES (?, ?, ?, ?, ?, ?, ?)
```

- 对列表中的书籍，使用循环进行插入，每插入一本，使用查重语句进行查询，若查询结果集非空，则可以返回false，终止插入，回滚更改，否则继续处理下一条数据。
- 若所有数据均无重复，则实现正确插入。

4. 移除书籍

要求

- 从书库中移除书籍，要求移除的书籍必须是已经归还的

原理

查询书籍存在性以及归还状态

```SQL
SELECT * FROM borrow WHERE book_id = ? and return_time = 0
```

- 该SQL语句查询了书库中未归还的需移除书籍，若结果集非空，则归还失败

移除书籍

```SQL
DELETE FROM book WHERE book_id = ?
```

5. 更改图书信息

要求

- 对书库中的对应书籍进行信息修改

原理

- 查询和更新同时进行，使用一条语句，结果集非空则可更新

```SQL
update book set category = ? , title = ? , press = ? , publish_year =
? , author = ? , price = ? WHERE book_id = ?
```

6. 书籍查询

要求

- 对于输入的查询条件，对书库中的书籍进行查询，支持类别点查(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，同时指定

一个排序方式，默认为`book id`升序，返回查询结果。

原理

- 使用`StringBuilder`，简化对SQL语句条件的添加，并利用`List`对相应追加条件的待定值`?`进行保存和赋值：

```java
StringBuilder queryBuilder = new StringBuilder("select * from book
where 1=1"); // 1=1 占位
List<Object> queryParams = new ArrayList<>();

if(conditions.getCategory() != null){
    queryBuilder.append(" AND category = ?"); //追加条件
    queryParams.add(conditions.getCategory());
}
```

- 模糊查询，使用`LIKE`：

> ? 将被替换为形如%String%的形式，%代表多个或没有字符

```sql
SELECT * FROM book WHERE title LIKE ?
```

7. 借书

要求

- 对于给定的借书人和借阅书籍以及借阅时间，在书库中查找书籍，若书籍尚有库存，且该借书人不存在借此书不还的情况，则借书成功，生成借书记录，更新表`borrow`
- 存在批量操作，多个线程对库存同时出现影响的情况，需要解决这种情况

原理

库存更新锁

- 通过加锁，来实现一个线程对一本书籍进行库存操作时，其他线程无法对相同的书籍库存进行更改，从而避免了并发借书的问题
- SQL语句如下，添加了行级锁：

```SQL
SELECT * FROM book WITH (UPDLOCK, ROWLOCK) WHERE book_id = ? AND
stock > 0
```

插入借书记录

```SQL
INSERT INTO borrow (card_id, book_id, borrow_time, return_time)
values (?, ?, ?, ?)
```

更新库存

```SQL
UPDATE book SET stock = stock - 1 WHERE book_id = ?
```

8. 还书

要求

- 对于当前未还的存在书籍，执行还书操作，将库存加一

原理

- 查询是否存在以及当前是否存在未还记录

```SQL
SELECT * FROM borrow WHERE card_id = ? AND book_id = ? AND
return_time = 0
```

- 若存在未还记录，更新该书库存

```SQL
UPDATE book SET stock = stock + 1 WHERE book_id = ?
```

- 检查还书时间是否合法：

```java
try {
    PreparedStatement checkStatement =
connection.prepareStatement(checkBorrowBook);
    checkStatement.setInt(1, borrow.getCardId());
    checkStatement.setInt(2, borrow.getBookId());
    ResultSet resultSet = checkStatement.executeQuery();
    if (!resultSet.next()) {
        rollback(connection);
        return new ApiResult(false, "No book to be returned!");
    } else {
    tmp = resultSet.getLong("borrow_time");
        if (tmp >= borrow.getReturnTime()) { //在此检测时间合法性
            rollback(connection);
            return new ApiResult(false, "Return time error!");
        }
    }
} catch (SQLException e) {
    throw new RuntimeException(e);
}
```

9. 查询用户历史记录

要求

- 对给定的用户`Card ID`，显示其全部借阅记录，要求包含书籍信息并按照借阅时间**降序**、`book id`**升序**排序

原理

- 先通过`borrow`表查询`book id`，再通过此`book id`在`book`表中查询书籍信息

1. 查询`book id`

```sql
SELECT * FROM borrow WHERE card_id = ? order by borrow_time DESC,
book_id ASC
```

2. 查询书籍信息

```SQL
SELECT * FROM book WHERE book_id = ?
```

10. 注册用户卡

要求

- 给出用户的姓名、部门、用户种类(S, T)信息，生成一个`card id`，需要查重

原理

- 查重，新建卡，自动生成`card id`

```SQL
select * from card where name = ? and department = ? and type = ?
insert into card (name, department, type) values (?, ?, ?)
```

11. 删除用户卡

要求

- 对于存在的用户卡，若不存在未归还书籍，则可对其用户卡进行删除

原理

- 查询对应`card id`的借书记录中，是否有未还书籍

```SQL
SELECT * FROM borrow WHERE card_id = ? AND return_time = 0
```

- 若无未归还书籍，移除卡

```SQL
DELETE FROM card WHERE card_id = ?
```

12. 展示所有用户卡

要求

- 以`card id`升序展示所有用户卡

原理

- 查询语句

```SQL
SELECT * FROM card order by ASC
```

GUI

基于Java Swing

- 以下展示部分控件的原理，其余控件均为相似操作

多界面跳转

- 通过JPanel实现：

> 以其中一个跳转界面按钮为例

```JAVA
storeBookButton.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        CardLayout cardLayout = (CardLayout) cards.getLayout();
        cardLayout.show(cards, "storeBook");
    }
});
```

错误消息弹窗

- 利用JOptionPane，对函数返回中的错误信息进行调用，正确执行时不弹窗:

```java
ApiResult result = library.removeCard(cardID);
if(!result.ok) {
    JOptionPane.showMessageDialog(registerCard.this, result.message,
"Remove Failure", JOptionPane.ERROR_MESSAGE);
}
```

按钮

- 添加按钮

```java
JButton removeBookButton = new JButton("Remove Book");
removeBookButton.setBounds(100, 370, 170, 30);
add(removeBookButton);
```

- 添加监视器

```java
removeBookButton.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        int bookID = Integer.parseInt(BookID.getText());
        ApiResult result = library.removeBook(bookID);
        if(!result.ok) {
        JOptionPane.showMessageDialog(storeBookPanel.this,
result.message, "Remove Failure", JOptionPane.ERROR_MESSAGE);
        }
        //clear
        BookID.setText("");
        refreshBook();
    }
});
```

文本框

```java
JLabel Title = new JLabel("Title");
Title.setBounds(100, 130, 70, 30);
add(Title);
JTextField bookTitle = new JTextField(70);
bookTitle.setBounds(170,130,100,30);
add(bookTitle);
```

列表

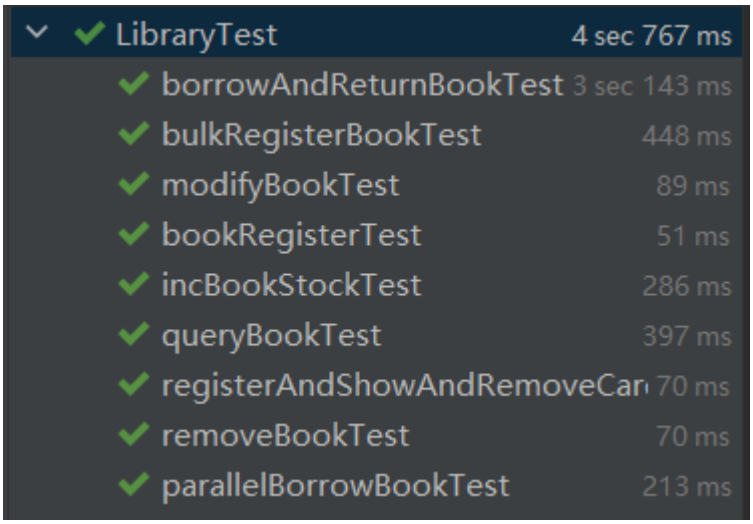
```java
public void refreshBook(BookQueryConditions conditions) {
    DefaultTableModel tableModel = (DefaultTableModel)
bookTable.getModel();
    tableModel.setRowCount(0);
    ApiResult result = library.queryBook(conditions);
    if(result.ok) {
        BookQueryResults queryResults = (BookQueryResults)
result.payload;
        List<Book> books = queryResults.getResults();
        for(Book book:books) {
            tableModel.addRow(new Object[]{
                book.getBookId(),
                book.getCategory(),
                book.getTitle(),
                book.getPress(),
                book.getPublishYear(),
                book.getAuthor(),
                book.getPrice(),
                book.getStock()
            });
        }
    }
}
```

下拉选框

```java
String[] options = {"Book ID", "Category", "Title", "Press",
"PublishYear", "Author", "Price"};
JComboBox<String> sortOptions = new JComboBox<>(options);
sortOptions.setBounds(170, 490, 100, 30);
add(sortOptions);
```

实验成果

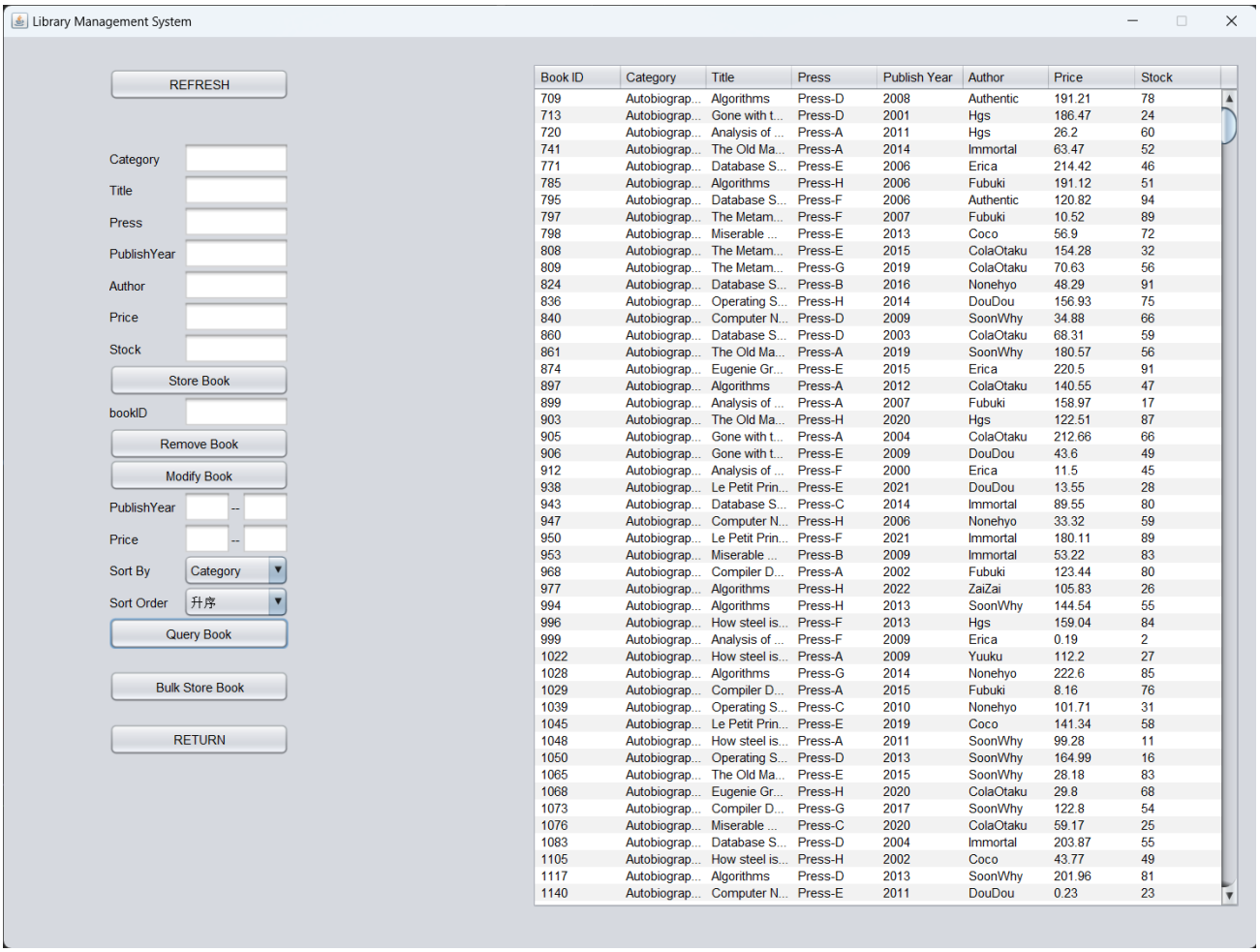
- 测试通过情况



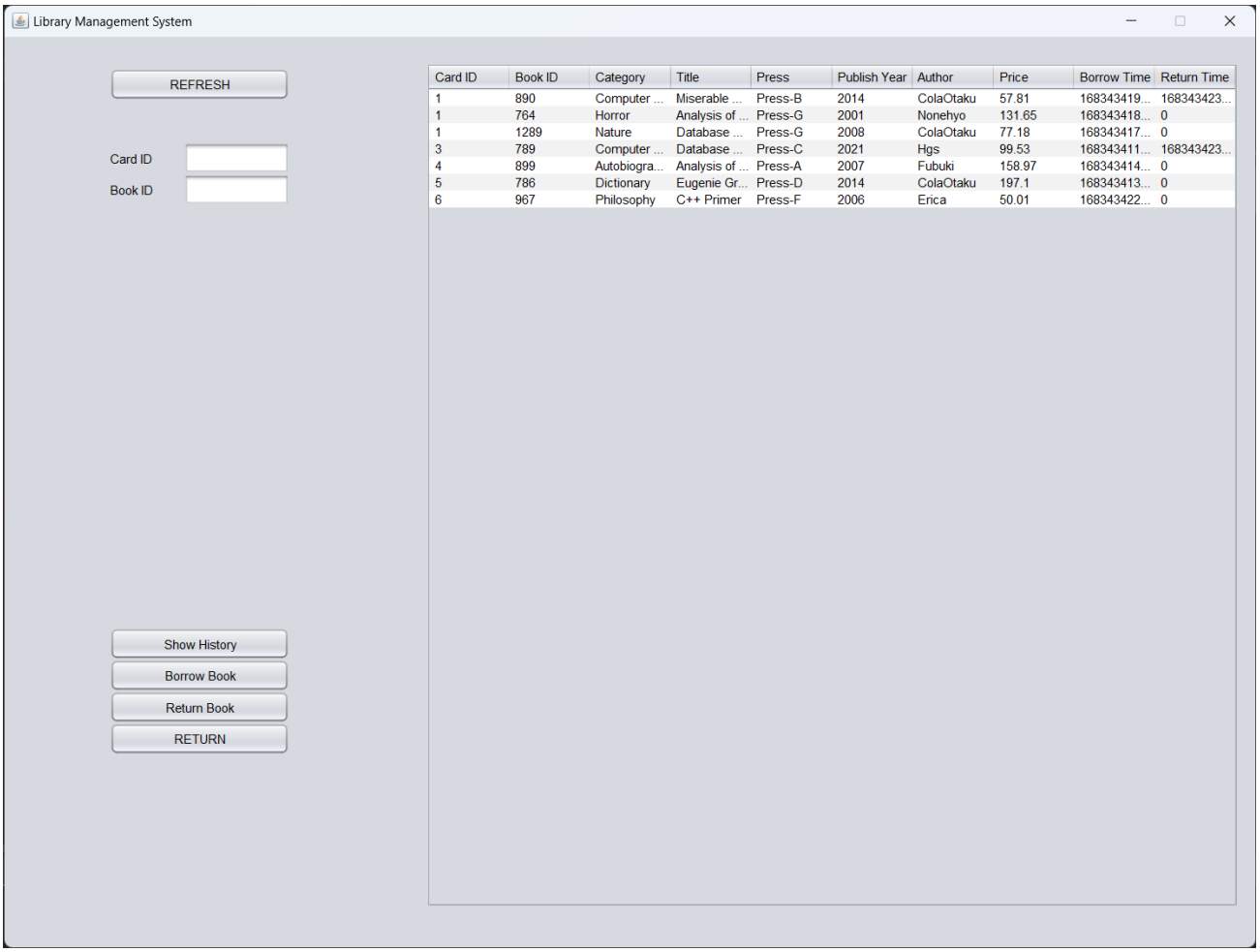
- GUI功能在验收时详细演示

GUI

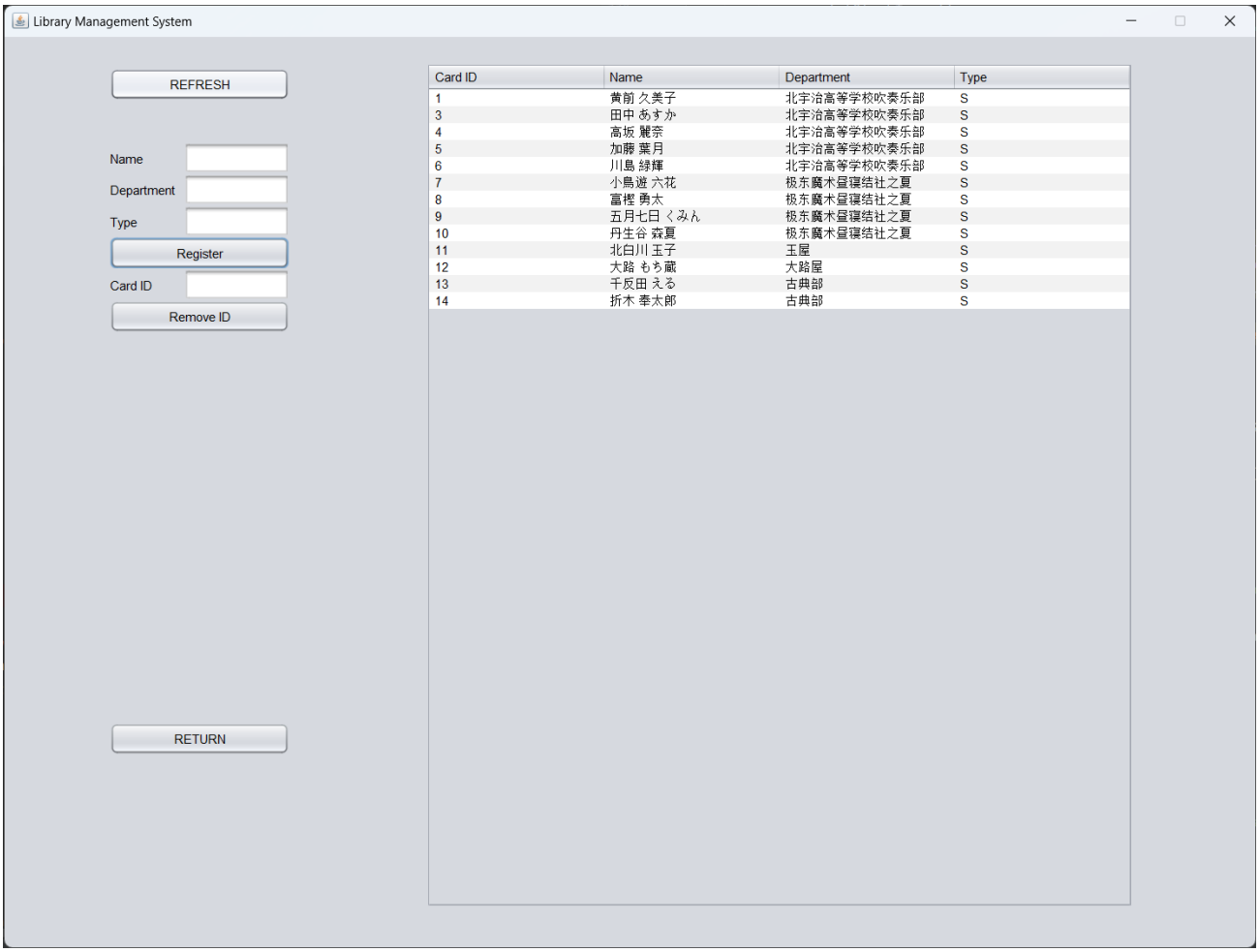
- 图书相关操作


Library Management System
REFRESH
Category
Title
Press
PublishYear
Author
Price
Stock
Store Book
bookID
Remove Book
Modify Book
PublishYear
--
Price
--
Sort By
Category
Sort Order
升序
Query Book
Bulk Store Book
RETURN
Book ID Category Title Press Publish Year Author Price Stock
709 Autobiograp... Algorithms Press-D 2008 Authentic 191.21 78
713 Autobiograp... Gone with t... Press-D 2001 Hgs 186.47 24
720 Autobiograp... Analysis of ... Press-A 2011 Hgs 26.2 60
741 Autobiograp... The Old Ma... Press-A 2014 Immortal 63.47 52
771 Autobiograp... Database S... Press-E 2006 Erica 214.42 46
785 Autobiograp... Algorithms Press-H 2006 Fubuki 191.12 51
795 Autobiograp... Database S... Press-F 2006 Authentic 120.82 94
797 Autobiograp... The Metam... Press-F 2007 Fubuki 10.52 89
798 Autobiograp... Miserable ... Press-E 2013 Coco 56.9 72
808 Autobiograp... The Metam... Press-E 2015 ColaOtaku 154.28 32
809 Autobiograp... The Metam... Press-G 2019 ColaOtaku 70.63 56
824 Autobiograp... Database S... Press-B 2016 Nonehyo 48.29 91
836 Autobiograp... Operating S... Press-H 2014 DouDou 156.93 75
840 Autobiograp... Computer N... Press-D 2009 SoonWhy 34.88 66
860 Autobiograp... Database S... Press-D 2003 ColaOtaku 68.31 59
861 Autobiograp... The Old Ma... Press-A 2019 SoonWhy 180.57 56
874 Autobiograp... Eugenie Gr... Press-E 2015 Erica 220.5 91
897 Autobiograp... Algorithms Press-A 2012 ColaOtaku 140.55 47
899 Autobiograp... Analysis of ... Press-A 2007 Fubuki 158.97 17
903 Autobiograp... The Old Ma... Press-H 2020 Hgs 122.51 87
905 Autobiograp... Gone with t... Press-A 2004 ColaOtaku 212.66 66
906 Autobiograp... Gone with t... Press-E 2009 DouDou 43.6 49
912 Autobiograp... Analysis of ... Press-F 2000 Erica 11.5 45
938 Autobiograp... Le Petit Prin... Press-E 2021 DouDou 13.55 28
943 Autobiograp... Database S... Press-C 2014 Immortal 89.55 80
947 Autobiograp... Computer N... Press-H 2006 Nonehyo 33.32 59
950 Autobiograp... Le Petit Prin... Press-F 2021 Immortal 180.11 89
953 Autobiograp... Miserable ... Press-B 2009 Immortal 53.22 83
968 Autobiograp... Compiler D... Press-A 2002 Fubuki 123.44 80
977 Autobiograp... Algorithms Press-H 2022 ZaiZai 105.83 26
994 Autobiograp... Algorithms Press-H 2013 SoonWhy 144.54 55
996 Autobiograp... How steel is... Press-F 2013 Hgs 159.04 84
999 Autobiograp... Analysis of ... Press-F 2009 Erica 0.19 2
1022 Autobiograp... How steel is... Press-A 2009 Yuuku 112.2 27
1028 Autobiograp... Algorithms Press-G 2014 Nonehyo 222.6 85
1029 Autobiograp... Compiler D... Press-A 2015 Fubuki 8.16 76
1039 Autobiograp... Operating S... Press-C 2010 Nonehyo 101.71 31
1045 Autobiograp... Le Petit Prin... Press-E 2019 Coco 141.34 58
1048 Autobiograp... How steel is... Press-A 2011 SoonWhy 99.28 11
1050 Autobiograp... Operating S... Press-D 2013 SoonWhy 164.99 16
1065 Autobiograp... The Old Ma... Press-E 2015 SoonWhy 28.18 83
1068 Autobiograp... Eugenie Gr... Press-H 2020 ColaOtaku 29.8 68
1073 Autobiograp... Compiler D... Press-G 2017 SoonWhy 122.8 54
1076 Autobiograp... Miserable ... Press-C 2020 ColaOtaku 59.17 25
1083 Autobiograp... Database S... Press-D 2004 Immortal 203.87 55
1105 Autobiograp... How steel is... Press-H 2002 Coco 43.77 49
1117 Autobiograp... Algorithms Press-D 2013 SoonWhy 201.96 81
1140 Autobiograp... Computer N... Press-E 2011 DouDou 0.23 23

- 借书相关操作


Library Management System
REFRESH
Card ID
Book ID
Show History
Borrow Book
Return Book
RETURN
Card ID Book ID Category Title Press Publish Year Author Price Borrow Time Return Time
1 890 Computer ... Miserable ... Press-B 2014 ColaOtaku 57.81 168343419... 168343423...
1 764 Horror Analysis of ... Press-G 2001 Nonehyo 131.65 168343418... 0
1 1289 Nature Database ... Press-G 2008 ColaOtaku 77.18 168343417... 0
3 789 Computer ... Database ... Press-C 2021 Hgs 99.53 168343411... 168343423...
4 899 Autobiogra... Analysis of ... Press-A 2007 Fubuki 158.97 168343414... 0
5 786 Dictionary Eugenie Gr... Press-D 2014 ColaOtaku 197.1 168343413... 0
6 967 Philosophy C++ Primer Press-F 2006 Erica 50.01 168343422... 0

- 借书卡相关操作



实验心得

- 通过此次实验，我了解了如何使用JDBC开发一个图书管理系统，了解了如何通过添加行级锁解决并发问题的方法。同时，这个实验锻炼了我的系统编程能力，进一步的巩固了我对课上所学的SQL语句的认识与运用。
- 同时，我对思考题中提到的SQL注入攻击，并发访问的正确性保证都做了初步的了解：
  1. SQL注入攻击
     + 攻击者通过应用程序的输入字段插入SQL语句从而使之在数据库中执行意外的语句，比如对于一个登录功能，可以有以下注入攻击：

> 以下是正常语句

```SQL
SELECT * FROM Users WHERE Username = '[username]' AND Password =
'[password]'
```

> 以下是攻击语句

```SQL
SELECT * FROM Users WHERE Username = '"" OR ""=""' AND Password =
'[password]'
```

- 因为 `""=""` 总是为真，所以这个查询将返回所有用户，在图书管理系统中，每一个接受输入信息的位置都可能遭受SQL注入攻击，所以我采用了参数化查询。

2. 高并发情况

- 我为借书操作添加了行级锁从而解决了默认隔离级别下的并发借书问题。