浙江大学

实验五 图书管理系统

学号：3220104519

姓名：蔡佳伟

一、实验目的

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

二、实验环境

操作系统：Windows11；

DBMS: MySQL

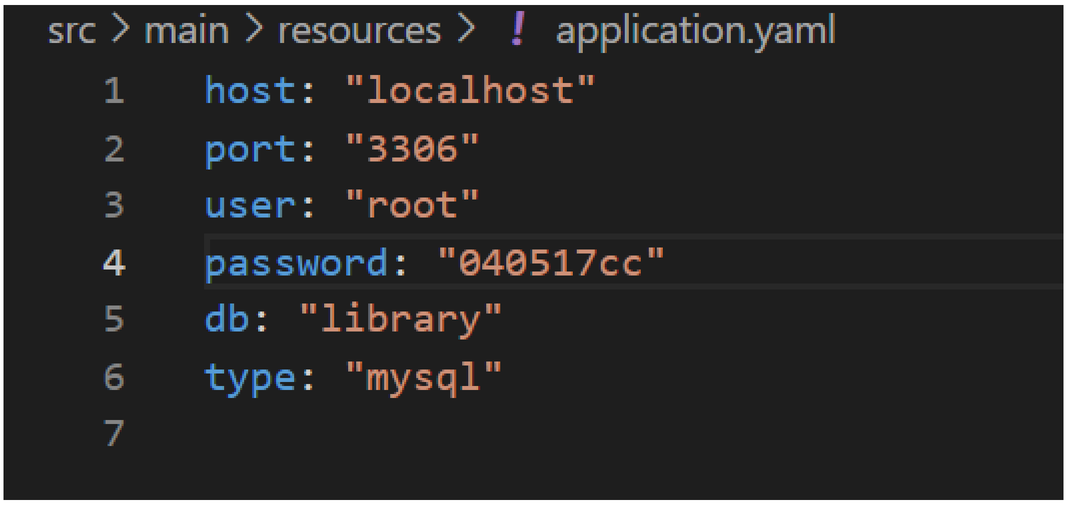
开发环境：VSCode+Java+Maven

三、实验流程

3.1 环境配置

依照实验流程，配置系统Java环境后安装插件Extension Pack for Java。

依照说明修改`.yaml`文件



```yaml
host: "localhost"
port: "3306"
user: "root"
password: "040517cc"
db: "library"
type: "mysql"

```

还需要在电脑本地的数据库root下创建一个library数据库。表格没有手动创建的情况下，我也成功通过了测试。

之后即可采用这些命令进行测试。

> **VSCode下运行测试用例**
>
> 在VSCode中的终端中输入相应的指令即可编译/运行/测试代码，以下是一些常用指令：
>
> - `mvn clean compile`：清理输出目录并编译项目主代码
> - `mvn exec:java -Dexec.mainClass="Main" -Dexec.cleanupDaemonThreads=false`：运行主代码
>   - **注意：**在**Windows**下，需要使用 `mvn exec:java -D"exec.mainClass"="Main" -D"exec.cleanupDaemonThreads"=false` 命令，来源参考Unknown lifecycle phase on Maven
> - `mvn -Dtest=LibraryTest clean test`：运行所有的测试
> - `mvn -Dtest=LibraryTest#子测试名 clean test`：运行某个特定的测试
>   - eg：`mvn -Dtest=LibraryTest#parallelBorrowBookTest clean test`

3.2 各模块功能具体实现

3.2.0 通用步骤

首先连接数据库，使用try尝试抛出异常并且在下方如果接住异常，就回滚且返回false，否则则返回true。关闭自动commit，在执行一系列操作后，需要commit之前的sql语句。需要注意的是，每次出现sql语句的修改提交，需要根据语句类型是修改相关还是查询使用`executeUpdate()/executeQuery()`提交。

3.2.1 storeBook

我们首先查询数据库中是否存在这本书（若类别，书名，出版社，年份，作者都相同的书，则认为这两本书相同），如果已经有了这本书，那就返回False。否则就插入。需要注意的是book_id的分配权我们交给了库内部，在插入时如果不插入这个属性，就会将这个元组的book_id设置为一个内部自增的计数器的值，插入后，我们需要查询这本书的id并将其设定为book实例的属性，返回给测试程序以便后续使用。

```
@Override
    public ApiResult storeBook(Book book) {
        Connection conn=connector.getConn();
        try{
            conn.setAutoCommit(false);
            Statement stmt=conn.createStatement();
            String sql="SELECT book_id FROM book WHERE "+"category = 
'"+book.getCategory()+
            "' AND title = '"+book.getTitle()+"' AND press = 
'"+book.getPress()+"' AND publish_year = "+
            book.getPublishYear()+" AND author = '"+book.getAuthor()+"';";
            ResultSet res=stmt.executeQuery(sql);
            if(res.next())
            {
                return new ApiResult(false,"The book already exists.");
            }
            try{
                PreparedStatement sql1 = conn.prepareStatement("INSERT INTO book 
(category, title, author, press, publish_year, price, stock) VALUES (?, ?, ?, ?, 
?, ?, ?);");
                sql1.setString(1, book.getCategory());
                sql1.setString(2, book.getTitle());
                sql1.setString(3, book.getAuthor());
                sql1.setString(4, book.getPress());
                sql1.setInt(5, book.getPublishYear());
                sql1.setDouble(6, book.getPrice());
                sql1.setInt(7, book.getStock());
                sql1.executeUpdate();
            } catch (SQLException sql_e) {
                System.out.println("Could not insert tuple. " + sql_e);
            }
            res = stmt.executeQuery(sql);
            if(res.next()){
                book.setBookId(res.getInt("book_id"));
            }
            stmt.close();
            commit(conn);
        }
        catch(Exception e)
        {
            rollback(conn);
            return new ApiResult(false,e.getMessage());
        }
        return new ApiResult(true, "storeBook(Book book) success");
    }
```

3.2.2 IncBookStock

首先我们连接数据库，查询数据库中是否有book_id等于bookId的书，如果没有就返回false。如果能找到这本书，我们就通过rs.getInt读取存量，并计算修改后存量。如果修改后变为负数，就需要报错。否则直接利用update语句修改存量并提交即可。

```
    @Override
    public ApiResult incBookStock(int bookId, int deltaStock) {
        Connection conn=connector.getConn();
        try{
            conn.setAutoCommit(false);
            PreparedStatement pstmt=null;
            String sql="SELECT stock "+"FROM book "+
            "WHERE book_id = ? ; ";
            pstmt=conn.prepareStatement(sql);
            pstmt.setInt(1,bookId);
            ResultSet res=pstmt.executeQuery();
            if(res.next())
            {
                int stock=res.getInt("stock");
                if(stock+deltaStock<0)
                {
                    throw new Exception("The stock can't benegative.");
                }
                else{
                    String sql1="UPDATE book "+"SET stock = ? "+
                    "WHERE book_id = ? ; ";
                    pstmt=conn.prepareStatement(sql1);
                    pstmt.setInt(1,stock+deltaStock);
                    pstmt.setInt(2,bookId);
                    pstmt.executeUpdate();
                    commit(conn);
                }
            }else{
                throw new Exception("The book doesn't exist in the library
system.");
            }
        }catch(Exception e)
        {
            rollback(conn);
            return new ApiResult(false,e.getMessage());
        }
        return new ApiResult(true, "incBookStock(int bookId,int deltaStock)
success");
    }
```

3.2.3 storeBook

图书批量入库模块和图书入库模块原理大致相同。我们遍历List，对于其中的每一本书我们都设置查询语句中的类别，书名，出版社，年份，作者并查询。只要其中有一本书，就要回滚整个操作并报错。

同时我们设置插入语句并通过addBatch添加到缓冲区里。最后循环结束时，如果没有书插入失败，我们一次性插入缓冲区所有语句，这样可以显著提高速度。插入结束后，同样需要对book_id进行处理。

```
@Override
    public ApiResult storeBook(List<Book> books) {
        Connection conn = connector.getConn();
        try{
            String sql = "SELECT book_id FROM book WHERE " + "category = ? and
title = ? and author = ? and press = ? and publish_year = ?;";
            PreparedStatement pstmt = conn.prepareStatement(sql);
            String sql1 = "INSERT INTO book (category, title, author, press,
publish_year, price, stock) VALUES (?, ?, ?, ?, ?, ?, ?);";
            PreparedStatement pstmt1 = conn.prepareStatement(sql1);

            for(Book mybook : books){
                pstmt.setString(1, mybook.getCategory());
                pstmt.setString(2, mybook.getTitle());
                pstmt.setString(3, mybook.getAuthor());
                pstmt.setString(4, mybook.getPress());
                pstmt.setInt(5, mybook.getPublishYear());
                ResultSet res = pstmt.executeQuery();
                if(res.next()){
                    rollback(conn);
                    return new ApiResult(false, "Some Books already exists");
                }
                pstmt1.setString(1, mybook.getCategory());
                pstmt1.setString(2, mybook.getTitle());
                pstmt1.setString(3, mybook.getAuthor());
                pstmt1.setString(4, mybook.getPress());
                pstmt1.setInt(5, mybook.getPublishYear());
                pstmt1.setDouble(6, mybook.getPrice());
                pstmt1.setInt(7, mybook.getStock());
                pstmt1.addBatch();
            }
            pstmt1.executeBatch();

            for(Book mybook : books){
                pstmt.setString(1, mybook.getCategory());
                pstmt.setString(2, mybook.getTitle());
                pstmt.setString(3, mybook.getAuthor());
                pstmt.setString(4, mybook.getPress());
                pstmt.setInt(5, mybook.getPublishYear());
                ResultSet res = pstmt.executeQuery();
                if(res.next()){
                    mybook.setBookId(res.getInt("book_id"));
                }
            }
            pstmt.close();
            pstmt1.close();
            commit(conn);
        }catch (Exception e) {
            rollback(conn);
            return new ApiResult(false, e.getMessage());
        }
        return new ApiResult(true, "Books stored successfully");
    }
```

3.2.4 removeBook

当输入书进行查询的时候，如果book_id查询不到或是有return_time=0（即没有归还）的记录，则不能删除这本书，分别返回对应的报错信息。否则就删除这本书。

```
@Override
    public ApiResult removeBook(int bookId) {
        Connection conn=connector.getConn();
        try{
            conn.setAutoCommit(false);
            PreparedStatement pstmt=null;
            String sql="SELECT * "+"FROM book "+
            "WHERE book_id = ? ; ";
            pstmt=conn.prepareStatement(sql);
            pstmt.setInt(1,bookId);
            ResultSet res=pstmt.executeQuery();
            if(!res.next())
            {
                throw new Exception("The book doesn't exist.");
            }
            sql="SELECT book_id "+"FROM borrow "+"WHERE book_id = ? "+
            "AND return_time = 0 ; ";
            pstmt=conn.prepareStatement(sql);
            pstmt.setInt(1,bookId);
            res=pstmt.executeQuery();
            if(res.next())
            {
                throw new Exception("There is someone who hasn't return this 
book.");
            }
            sql="DELETE FROM book "+"WHERE book_id = ? ;  ";
            pstmt=conn.prepareStatement(sql);
            pstmt.setInt(1,bookId);
            pstmt.executeUpdate();
            commit(conn);
        }catch(Exception e){
            rollback(conn);
            return new ApiResult(false,e.getMessage());
        }
        return new ApiResult(true, "removeBook(int bookId) success");
    }
```

3.2.5 modifyBookInfo

我们利用update对对应的书籍进行更新（即book_id相同）。注意我们这里只能更新category，title，press，publish_year，author，price而对于book_id，stock是不能修改的，修改后提交事务即可。

```
@Override
    public ApiResult modifyBookInfo(Book book) {
        Connection conn=connector.getConn();
        try{
            conn.setAutoCommit(false);
            PreparedStatement pstmt=null;
            String sql="SELECT * "+"FROM book "+"WHERE book_id = ? ;  ";
            pstmt=conn.prepareStatement(sql);
            pstmt.setInt(1,book.getBookId());
            ResultSet res=pstmt.executeQuery();
            if(!res.next())
            {
                throw new Exception("The book doesn't exist in the library.");
            }
            String sql1="UPDATE book "+"SET category = ?, title = ?, press = ?,
publish_year = ?, author = ?, price = ? "+
            "WHERE book_id = ? ;  ";
            pstmt = conn.prepareStatement(sql1);
            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.setDouble(6, book.getPrice());
            pstmt.setInt(7, book.getBookId());
            pstmt.executeUpdate();
            pstmt.close();
            commit(conn);
        }catch(Exception e)
        {
            rollback(conn);
            return new ApiResult(false,e.getMessage());
        }
        return new ApiResult(true, "modifyBookInfo(Book book) success");
    }
```

3.2.6 queryBook

首先我们连接数据库，设置sql语句为`String sql="SELECT book_id, category, title, press, publish_year, author, price, stock "+
             "FROM book WHERE category like ? AND title like ? AND press like ? AND publish_year >= ? "+
             "AND publish_year <= ? AND author like ? AND price >= ? AND price <= ? ;";`

这里我们放置了所有的查询条件，并利用PreparedStatement的机制进行参数设置。对于给出的查询条件的实例conditions，对于前面的模糊查询部分，如果conditions对应属性为空，我们直接将字符串设置为'%'表示通配符。否则填入对应的字符串，注意到这里是模糊查询，所以我们应该使用'%string%'的方式实现模糊查询。而对于后面的范围查询，如果conditions属性为空，我们就将范围填充为-9999999.0到9999999.0。如果属性不为空，直接填入即可。

最后我们执行查询语句，使用rs.next()遍历结果表格，并将利用rs.getInt()或者rs.getString()得到属性放到book实例中去，最后对List按照指定属性排序。排序用到的属性为`conditions.getSortBy().getComparator()`，同时控制升序或者降序排序。排序后返回答案即可。

```java
@Override
    public ApiResult queryBook(BookQueryConditions conditions) {
        Connection conn=connector.getConn();
        List<Book> results=new ArrayList<>();
        try{
            String sql="SELECT book_id, category, title, press, publish_year, 
author, price, stock "+
            "FROM book WHERE category like ? AND title like ? AND press like ? 
AND publish_year >= ? "+
            "AND publish_year <= ? AND author like ? AND price >= ? AND price <= 
? ;";
            PreparedStatement pstmt=conn.prepareStatement(sql);
            if(conditions.getCategory()!=null)
            {
                pstmt.setString(1,"%"+conditions.getCategory()+"%");
            }
            else{
                pstmt.setString(1,"%");
            }
            if(conditions.getTitle()!=null)
            {
                pstmt.setString(2,"%"+conditions.getTitle()+"%");
            }
            else{
                pstmt.setString(2,"%");
            }
            if(conditions.getPress()!=null)
            {
                pstmt.setString(3,"%"+conditions.getPress()+"%");
            }
            else{
                pstmt.setString(3,"%");
            }
            if(conditions.getMinPublishYear()!=null)
            {
                pstmt.setLong(4,conditions.getMinPublishYear());
            }
            else{
                pstmt.setLong(4,Long.MIN_VALUE);
            }
            if(conditions.getMaxPublishYear()!=null)
            {
                pstmt.setLong(5,conditions.getMaxPublishYear());
            }
            else{
                pstmt.setLong(5,Long.MAX_VALUE);
            }
            if(conditions.getAuthor()!=null)
            {
                pstmt.setString(6,"%"+conditions.getAuthor()+"%");
            }
            else{
                pstmt.setString(6,"%");
            }
            if(conditions.getMinPrice()!=null)
            {
                pstmt.setDouble(7,conditions.getMinPrice());
            }
            else{
                pstmt.setDouble(7,-9999999.0);
            }
            if(conditions.getMaxPrice()!=null)
            {
                pstmt.setDouble(8,conditions.getMaxPrice());
            }
            else{
                pstmt.setDouble(8,9999999.0);
            }
            ResultSet res=pstmt.executeQuery();
            while(res.next())
            {
                Book temp=new Book(res.getString("category"),
res.getString("title"), res.getString("press"),
                res.getInt("publish_year"), res.getString("author"),
res.getDouble("price"),
                res.getInt("stock"));
                temp.setBookId(res.getInt("book_id"));
                temp.setStock(res.getInt("stock"));
                results.add(temp);
            }
            if(conditions.getSortOrder()==SortOrder.ASC)
            {
                results.sort(conditions.getSortBy().getComparator());
            }
            else{
                results.sort(conditions.getSortBy().getComparator().reversed());
            }
            pstmt.close();
            commit(conn);
        }catch(Exception e)
        {
            rollback(conn);
            return new ApiResult(false,e.getMessage());
        }
        BookQueryResults ans=new BookQueryResults(results);
        return new ApiResult(true, "queryBook(BookQueryConditions conditions)
success",ans);
    }
```

3.2.7 borrowBook

在查询到这本书后，我们首先要查询这本书的存量一定要大于0，否则不能借阅。同时还要注意查询当前用户是否存在借阅了这本书却没有还的情况，这种情况也要rollback并返回报错信息。

值得注意的是，我们加上了`conn.setTransactionIsolation(Connection.TRANSACTION_SERIALIZABLE`这句话，这表明我们在查询时，将权限设置为串行化，对数据加锁防止其他用户对表进行操作，造成表的不一致性。

借书时，我们记得通过update语句更新余量，并且通过insert语句插入一条新的借书记录，最后提交事务即可。

```
@Override
    public ApiResult borrowBook(Borrow borrow) {
        Connection conn=connector.getConn();
        try{
            PreparedStatement pstmt=null;
            conn.setTransactionIsolation(Connection.TRANSACTION_SERIALIZABLE);
            conn.setAutoCommit(false);
            String sql="SELECT * "+"FROM borrow "+
            "WHERE card_id = ? AND book_id = ? AND return_time = 0 ; ";
            pstmt=conn.prepareStatement(sql);
            pstmt.setInt(1,borrow.getCardId());
            pstmt.setInt(2,borrow.getBookId());
            ResultSet res=pstmt.executeQuery();
            if(res.next())
            {
                throw new Exception("The customer borrowed this book but hasn't
return.");
            }
            sql="SELECT * "+"FROM book "+"WHERE book_id = ? ;  ";
            pstmt=conn.prepareStatement(sql);
            pstmt.setInt(1, borrow.getBookId());
            res=pstmt.executeQuery();
            int stock=0;
            if(res.next())
            {
                stock=res.getInt("stock");
            }
            else{
                throw new Exception("The book doesn't exist.");
            }
            if(stock<=0)
            {
                throw new Exception("The book is out of stock.");
            }
            else{
                sql="UPDATE book "+"SET stock = stock - 1 "+
                "WHERE book_id = ? AND stock = ? ;  ";
                pstmt=conn.prepareStatement(sql);
                pstmt.setInt(1, borrow.getBookId());
                pstmt.setInt(2, stock);
                pstmt.executeUpdate();
                String sql1="INSERT INTO borrow " +
                        "VALUES(?, ?, ?, ?);  ";
                pstmt=conn.prepareStatement(sql1);
                pstmt.setInt(1, borrow.getCardId());
                pstmt.setInt(2, borrow.getBookId());
                pstmt.setLong(3, borrow.getBorrowTime());
                pstmt.setLong(4, 0);
                pstmt.executeUpdate();
            }
            pstmt.close();
            commit(conn);
        }catch(Exception e){
            rollback(conn);
            return new ApiResult(false,e.getMessage());
        }
        return new ApiResult(true,"Borrow success");
    }
```

3.2.8 returnBook

我们先查询借书记录，即有没有这本书被借阅且未归还的情况，否则报错。如果存在，我们利用update语句对borrow记录进行更新，即补充归还记录return_time。需要注意的是，我们需要利用update语句对book进行更新，将存量加一，最后提交事务即可。

```
@Override
    public ApiResult returnBook(Borrow borrow) {
        Connection conn=connector.getConn();
        try{
            //conn.setTransactionIsolation(Connection.TRANSACTION_SERIALIZABLE);
            conn.setAutoCommit(false);
            PreparedStatement pstmt=null;
            String sql="SELECT * "+"FROM borrow "+
            "WHERE card_id = ? AND book_id = ? AND borrow_time = ? AND
return_time = 0 ; ";
            pstmt=conn.prepareStatement(sql);
            pstmt.setInt(1, borrow.getCardId());
            pstmt.setInt(2, borrow.getBookId());
            pstmt.setLong(3, borrow.getBorrowTime());
            ResultSet res=pstmt.executeQuery();
            if(!res.next())
            {
                throw new Exception("There is no borrow record or this book has
been returned.");
            }
            else{
                String sql1="UPDATE borrow "+"SET return_time = ? "+
                "WHERE card_id = ? AND book_id = ? AND borrow_time = ? ;  ";
                pstmt=conn.prepareStatement(sql1);
                pstmt.setLong(1, borrow.getReturnTime());
                pstmt.setInt(2, borrow.getCardId());
                pstmt.setInt(3, borrow.getBookId());
                pstmt.setLong(4, borrow.getBorrowTime());
                pstmt.executeUpdate();
                sql="SELECT * "+"FROM book "+"WHERE book_id = ? ; ";
                pstmt=conn.prepareStatement(sql);
                pstmt.setInt(1,borrow.getBookId());
                res=pstmt.executeQuery();
                int stock=0;
                if(res.next())
                {
                    stock=res.getInt("stock");
                }
                else{
                    throw new Exception("The book doesn't exist.");
                }
                sql="UPDATE book "+"SET stock = stock + 1 "+"WHERE book_id = ?
;";
                pstmt=conn.prepareStatement(sql);
                pstmt.setInt(1,borrow.getBookId());
                pstmt.executeUpdate();
            }
            commit(conn);
        }catch(Exception e){
            rollback(conn);
            return new ApiResult(false,e.getMessage());
        }
        return new ApiResult(true,"returnBook(Borrow borrow) success");
    }
```

3.2.9 showBorrowHistory

首先我们连接数据库，按照select语句查询对应cardId的用户的借书记录和书籍信息，按照borrow_time降序排列，如果相同再按book_id升序排列。得到结果后我们再用rs.next()遍历结果，利用rs.getInt(), rs.getString()等方法将结果存到实例中，并最后返回提交事务。

```
@Override
    public ApiResult showBorrowHistory(int cardId) {
        Connection conn=connector.getConn();
        BorrowHistories borrowHistories=null;
        try{
            conn.setAutoCommit(false);
            PreparedStatement pstmt=null;
            String sql="SELECT * "+"FROM borrow "+"WHERE card_id = ?  "+
            "ORDER BY borrow_time DESC, book_id ASC ; ";
            pstmt=conn.prepareStatement(sql);
            pstmt.setInt(1,cardId);
            ResultSet res=pstmt.executeQuery();
            List<BorrowHistories.Item> items=new ArrayList<BorrowHistories.Item>
();
            while(res.next())
            {
                int bookId=res.getInt("book_id");
                long borrowTime=res.getLong("borrow_time");
                long returnTime=res.getLong("return_time");
                sql="SELECT * "+"FROM book "+"WHERE book_id = ? ;  ";
                pstmt=conn.prepareStatement(sql);
                pstmt.setInt(1,bookId);
                ResultSet res1=pstmt.executeQuery();
                if(res1.next())
                {
                    String category=res1.getString("category");
                    String title=res1.getString("title");
                    String press=res1.getString("press");
                    int publishYear=res1.getInt("publish_year");
                    String author=res1.getString("author");
                    double price=res1.getDouble("price");
                    BorrowHistories.Item item=new BorrowHistories.Item();
                    item.setCardId(cardId);
                    item.setBookId(bookId);
                    item.setCategory(category);
                    item.setTitle(title);
                    item.setPress(press);
                    item.setPublishYear(publishYear);
                    item.setAuthor(author);
                    item.setPrice(price);
                    item.setBorrowTime(borrowTime);
                    item.setReturnTime(returnTime);
                    items.add(item);
                }
            }
            borrowHistories=new BorrowHistories(items);
            commit(conn);
        }catch(Exception e)
        {
            rollback(conn);
            return new ApiResult(false,e.getMessage());
        }
        return new ApiResult(true,"returnBook(Borrow borrow)
success",borrowHistories);
    }
```

3.2.10 registerCard

首先我们在card查询是否有相同的借书证，如果姓名、单位、身份都相同，才说明这两张证书相同。没有相同借书证，才利用insert语句设置name，department，type属性并进行更新，再查询card_id，并设置为card实例的属性。最后提交事务，返回id。否则则要报错。

```
@Override
    public ApiResult registerCard(Card card) {
        Connection conn=connector.getConn();
        try{
            PreparedStatement pstmt=conn.prepareStatement("SELECT card_id FROM
card WHERE name = ? AND department = ? AND type = ?;");
            pstmt.setString(1,card.getName());
            pstmt.setString(2,card.getDepartment());
            pstmt.setString(3,card.getType().getStr());
            ResultSet res=pstmt.executeQuery();
            if(res.next()){
                rollback(conn);
                return new ApiResult(false, "Card already exists");
            }
            PreparedStatement pstmt1=conn.prepareStatement("INSERT INTO card 
(name, department, type) VALUES (?, ?, ?);");
            pstmt1.setString(1,card.getName());
            pstmt1.setString(2,card.getDepartment());
            pstmt1.setString(3,card.getType().getStr());
            pstmt1.executeUpdate();
            res = pstmt.executeQuery();
            if(res.next()){
                card.setCardId(res.getInt("card_id"));
            }
            pstmt.close();
            pstmt1.close();
            commit(conn);
        }catch(Exception e){
            rollback(conn);
            return new ApiResult(false, e.getMessage());
        }
        return new ApiResult(true, "Register card success");
    }
```

3.2.11 removeCard

查询到借书证后，我们要先查询borrow中是否用使用该cardId借书并尚未归还的借书记录（即return_time=0)。如果有那我们不能删除这个借书证，需要返回报错。随后我们利用delete删除card_id结果为cardId的元组。同时记录pstmt.executeUpdate()的结果，如果结果为0，说明没有该cardId，也需要报错。

```
@Override
    public ApiResult removeCard(int cardId) {
        Connection conn=connector.getConn();
        try{
            PreparedStatement pstmt=conn.prepareStatement("SELECT * FROM borrow 
WHERE card_id = ? AND return_time = 0;");
            pstmt.setInt(1,cardId);
            ResultSet res=pstmt.executeQuery();
            if(res.next()){
                rollback(conn);
                return new ApiResult(false, "Card has borrowed books");
            }
            pstmt=conn.prepareStatement("DELETE FROM card WHERE card_id = ?;");
            pstmt.setInt(1, cardId);
            int res1=pstmt.executeUpdate();
            if(res1==0){
                rollback(conn);
                return new ApiResult(false, "Card not found");
            }
            pstmt.close();
            commit(conn);
        }catch(Exception e){
            rollback(conn);
            return new ApiResult(false, e.getMessage());
        }
```

```
            return new ApiResult(true, "Card removed successfully");
        }
```

3.2.12 showCards

我们用select语句查询并按card_id升序排序，随后利用rs.next()遍历结果，并读取对应值放到card实例中，放到结果list里，最后提交事务返回list。

```
@Override
    public ApiResult showCards() {
        Connection conn=connector.getConn();
        List<Card> results=new ArrayList<>();
        try{
            PreparedStatement pstmt=conn.prepareStatement("SELECT card_id, name, 
department, type FROM card order by card_id ASC ;");
            ResultSet res=pstmt.executeQuery();
            while(res.next()){
                Card card=new Card(res.getInt("card_id"), 
res.getString("name"),res.getString("department"), 
Card.CardType.values(res.getString("type")));
                results.add(card);
            }
            pstmt.close();
            commit(conn);
        }catch(Exception e){
            rollback(conn);
            return new ApiResult(false, e.getMessage());
        }
        return new ApiResult(true, new CardList(results));
    }
```

3.3 交互功能设计

我采用字符界面进行交互，通过在终端显示相关信息和进行输入和输出，实现了图书管理系统需要的功能。

在设计中，首先会提示输入数据进行选择功能，输入0会退出，否则进入相应步骤，进入步骤后需要输入密码（除了图书查询功能不需要输入密码），密码有5次输入机会，输入失败以后就退出该功能（但是不会退出图书管理系统，只有按0才会退出系统）。

密码输入正确后，需要根据提示进行对应的输入操作，每个操作完成后，数据库内会存储对应的记录（如添加书后，本地数据库library数据库中book表格会进行更新）。每个操作完成后会回到图书管理系统的初始界面，可以进行其他操作或退出。如果操作中出现问题，会输出提示（如查询借书记录时cardID下没有借书记录），但是不会退出系统，只是数据库没有更新。

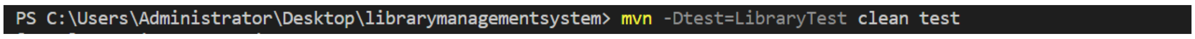
需要注意的是，第三个部分插入很多书籍，我使用了读入文件系统，我保存了1.txt文件在根目录下，测试时可以直接输入1.txt来进行测试。如果想自行设计文件，需要注意每七行是一本书的信息，同时.txt中不要有多余信息，否则会出错。

以下是自己填写的Main函数中部分代码：

```
LibraryManagementSystemImpl impl = new LibraryManagementSystemImpl(connector);
            Scanner temp = new Scanner(System.in);
            while (true) {
                System.out.println("你好！欢迎使用图书管理系统");
                System.out.println("输入1以存储一本新书");
                System.out.println("输入2以更改书的库存");
                System.out.println("输入3以批量入库图书");
                System.out.println("输入4以删除某书");
                System.out.println("输入5以更改书的信息");
                System.out.println("输入6以查询所需要的书");
                System.out.println("输入7以借书");
                System.out.println("输入8以还书");
                System.out.println("输入9以查询借书的历史记录");
                System.out.println("输入10以注册借阅卡");
                System.out.println("输入11以移去借阅卡");
                System.out.println("输入12以查看所有借阅卡信息");
                System.out.println("输入0以退出");
                int i;
                i = temp.nextInt();
                temp.nextLine();
                if (i == 0) {
                    System.out.println("退出成功");
                    break;
                }
                if (i != 6) {
                    System.out.println("你需要管理员权限,请输入密码");
                    String password = temp.nextLine();
                    int flag = 5;
                    while (!password.equals("01gg")) {
                        if (flag != 0) {
                            System.out.println("输入错误,您还有"+flag+ " 次机会" ) ;
                            password = temp.nextLine();
                            flag -= 1;
                        } else {
                            System.out.println("输入错误,退出系统");
                            break;
                        }
                    }
                    if (flag == 0) {
                        continue;
                    } else {
                        System.out.println("输入正确,成功进入系统");
                    }
                }
                if (i == 1) {
                    System.out.println("请输入书的类别");
                    String category = temp.nextLine();
                    System.out.println("请输入书的标题");
                    String title = temp.nextLine();
                    System.out.println("请输入书的出版社");
                    String press = temp.nextLine();
                    System.out.println("请输入书的出版年份");
                    int publishYear = temp.nextInt();
                    temp.nextLine();
                    System.out.println("请输入书的作者");
                    String author = temp.nextLine();
                    System.out.println("请输入书的价格");
                    double price = temp.nextDouble();
                    System.out.println("请输入书的库存");
```

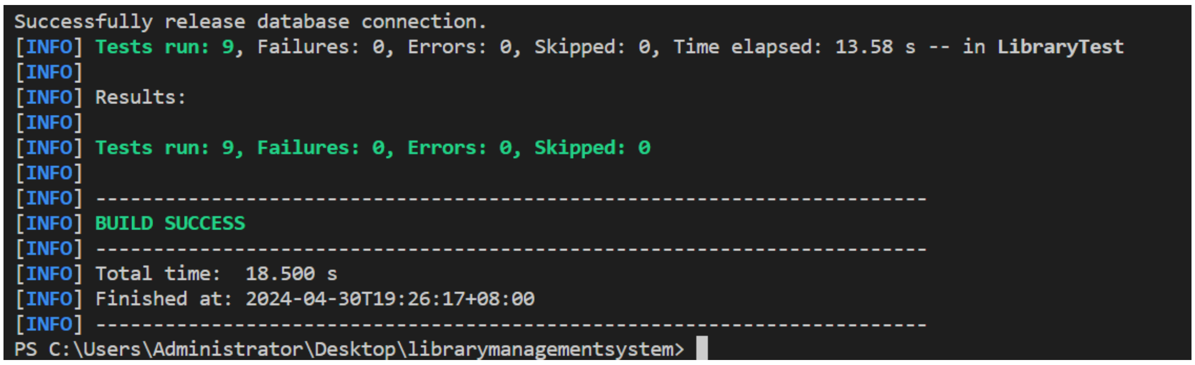
```
                    int stock = temp.nextInt();
                    Book book = new Book(category, title, press, publishYear,
author, price, stock);
                    impl.storeBook(book);
                } else if (i == 2) {
                    System.out.println("请输入书的ID");
                    int bookId = temp.nextInt();
                    System.out.println("请输入该书库存的更改量(输入变化量而不是最终
值)");
                    int deltaStock = temp.nextInt();
                    impl.incBookStock(bookId, deltaStock);
                } else if (i == 3) {
                    List<Book> books = new ArrayList<Book>();
                    System.out.println("请输入文件名称");
                    String filename = temp.nextLine();
                    try {
                        BufferedReader filein = new BufferedReader(new
FileReader(filename));
                        String str;
                        String category = null;
                        String title = null;
                        String press = null;
                        int publishYear = 0;
                        String author = null;
                        Double price = 0.0;
                        int stock = 0;
                        int cnt = 0;
                        while ((str = filein.readLine()) != null) {
                            if (cnt == 0)
                                category = str;
                            else if (cnt == 1)
                                title = str;
                            else if (cnt == 2)
                                press = str;
                            else if (cnt == 3)
                                publishYear = Integer.parseInt(str);
                            else if (cnt == 4)
                                author = str;
                            else if (cnt == 5)
                                price = Double.parseDouble(str);
                            else if (cnt == 6)
                                stock = Integer.parseInt(str);
                            cnt++;
                            if (cnt == 7)
                                cnt = 0;
                            if (cnt == 0) {
                                Book book = new Book(category, title, press,
publishYear, author, price, stock);
                                books.add(book);
                            }
                        }
                        filein.close();
                    } catch (IOException e) {
                    }
                    impl.storeBook(books);
                } else if (i == 4) {
                    System.out.println("请输入书的ID");
                    int bookId = temp.nextInt();
                    impl.removeBook(bookId);
                } else if (i == 5) {
                    System.out.println("请输入要修改的书的ID");
                    int bookID = temp.nextInt();
                    temp.nextLine();
                    System.out.println("请输入修改后书的类别");
                    String category = temp.nextLine();
                    System.out.println("请输入修改后书的标题");
                    String title = temp.nextLine();
                    System.out.println("请输入修改后书的出版社");
                    String press = temp.nextLine();
                    System.out.println("请输入修改后书的出版年份");
                    int publishYear = temp.nextInt();
                    temp.nextLine();
                    System.out.println("请输入修改后书的作者");
                    String author = temp.nextLine();
                    System.out.println("请输入修改后书的价格");
                    double price = temp.nextDouble();
                    Book book = new Book(category, title, press, publishYear,
author, price, 0);
                    book.setBookId(bookID);
                    impl.modifyBookInfo(book);
                } else if (i == 6) {
                    BookQueryConditions conditions = new BookQueryConditions();
                    System.out.println("请输入要找的书的类别(若没有请直接输入回车)");
                    String category = temp.nextLine();
                    System.out.println("请输入要找的书的标题(若没有请直接输入回车)");
                    String title = temp.nextLine();
                    System.out.println("请输入要找的书的出版社(若没有请直接输入回车)");
                    String press = temp.nextLine();
                    System.out.println("请输入要找的书的最小可能出版年份(若没有请输
入-1)");
                    Integer minPublishYear = temp.nextInt();
                    System.out.println("请输入要找的书的最大可能出版年份(若没有请输入
9999)");
                    Integer maxPublishYear = temp.nextInt();
                    temp.nextLine();
                    System.out.println("请输入要找的书的作者(若没有请直接输入回车)");
                    String author = temp.nextLine();
                    System.out.println("请输入要找的书的最低可能价格(若没有请输入-1)");
                    Double minPrice = temp.nextDouble();
                    System.out.println("请输入要找的书的最高可能价格(若没有请输入
9999)");
                    Double maxPrice = temp.nextDouble();
                    Book.SortColumn sortBy = Book.SortColumn.BOOK_ID;
                    System.out.println(
                            "按照书的ID排序请输入1,按照书的种类排序请输入2,按照书的标题排
序请输入3,按照书的出版社排序请输入4,按照书的出版年份排序请输入5,按照书的作者排序请输入6,按照书
的价格排序请输入7,按照书的库存排序请输入8");
                    int j = temp.nextInt();
                    if (j == 2) {
                        sortBy = Book.SortColumn.CATEGORY;
                    } else if (j == 3) {
                        sortBy = Book.SortColumn.TITLE;
                    } else if (j == 4) {
                        sortBy = Book.SortColumn.PRESS;
                    } else if (j == 5) {
                        sortBy = Book.SortColumn.PUBLISH_YEAR;
                    } else if (j == 6) {
                        sortBy = Book.SortColumn.AUTHOR;
                    } else if (j == 7) {
                        sortBy = Book.SortColumn.PRICE;
                    } else if (j == 8) {
                        sortBy = Book.SortColumn.STOCK;
                    }
                    SortOrder sortOrder = SortOrder.ASC;
                    System.out.println("按照升序排序请输入1,降序排序请输入2");
                    j = temp.nextInt();
                    if (j == 2) {
                        sortOrder = SortOrder.DESC;
                    }
                    if (category != "")
                        conditions.setCategory(category);
                    if (title != "")
                        conditions.setTitle(title);
                    if (press != "")
                        conditions.setPress(press);
                    conditions.setMinPublishYear(minPublishYear);
                    conditions.setMaxPublishYear(maxPublishYear);
                    if (author != "")
                        conditions.setAuthor(author);
                    conditions.setMinPrice(minPrice);
                    conditions.setMaxPrice(maxPrice);
                    conditions.setSortBy(sortBy);
                    conditions.setSortOrder(sortOrder);
                    ApiResult apiresult = impl.queryBook(conditions);
                    BookQueryResults bookqueryresults = (BookQueryResults)
apiresult.payload;
                    for (j = 0; j < bookqueryresults.getCount(); j++) {

 System.out.println(bookqueryresults.getResults().get(j));
                    }
                    if (bookqueryresults.getCount() == 0) {
                        System.out.println("未查询到想要的书籍");
                    }
                } else if (i == 7) {
                    System.out.println("请输入要借的书的ID");
                    int bookId = temp.nextInt();
                    System.out.println("请输入用来借书的卡的ID");
                    int cardId = temp.nextInt();
                    System.out.println("请输入借书时间");
                    long borrowtime = temp.nextLong();
                    Borrow borrow = new Borrow(bookId, cardId);
                    borrow.setBorrowTime(borrowtime);
                    impl.borrowBook(borrow);
                } else if (i == 8) {
                    System.out.println("请输入要还的书的ID");
                    int bookId = temp.nextInt();
                    System.out.println("请输入用来还书的卡的ID");
                    int cardId = temp.nextInt();
                    Borrow borrow = new Borrow(bookId, cardId);
                    System.out.println("请输入借书时间");
                    borrow.setBorrowTime(temp.nextLong());
                    System.out.println("请输入还书时间");
                    borrow.setReturnTime(temp.nextLong());
                    impl.returnBook(borrow);
                } else if (i == 9) {
                    System.out.println("请输入要查询记录的卡的ID");
                    int cardId = temp.nextInt();
                    ApiResult apiresult = impl.showBorrowHistory(cardId);
                    BorrowHistories borrowhistories = (BorrowHistories)
apiresult.payload;
                    int j;
                    int cnt = borrowhistories.getCount();
                    for (j = 0; j < cnt; j++) {
                        System.out.println(borrowhistories.getItems().get(j));
                    }
                    if (cnt == 0) {
                        System.out.println("此ID未查询到借书记录");
                    }
                } else if (i == 10) {
                    System.out.println("请输入要注册的卡的姓名");
                    String name = temp.nextLine();
                    System.out.println("请输入要注册的卡的部门");
                    String department = temp.nextLine();
                    System.out.println("请输入要注册的卡的类型(1为学生,2为教师)");
                    int j = temp.nextInt();
                    CardType type = CardType.Student;
                    if (j == 2) {
                        type = CardType.Teacher;
                    }
                    Card card = new Card(0, name, department, type);
                    impl.registerCard(card);
                } else if (i == 11) {
                    System.out.println("请输入要移除的卡的ID");
                    int cardId = temp.nextInt();
                    impl.removeCard(cardId);
                } else if (i == 12) {
                    System.out.println("卡信息展示如下");
                    ApiResult apiresult = impl.showCards();
                    CardList cardlist = (CardList) apiresult.payload;
                    int j;
                    int cnt = cardlist.getCount();
                    for (j = 0; j < cnt; j++) {

 System.out.println(cardlist.getCards().get(j).toString());
                    }
                }
            }
            temp.close();
```

3.4 程序测试及结果



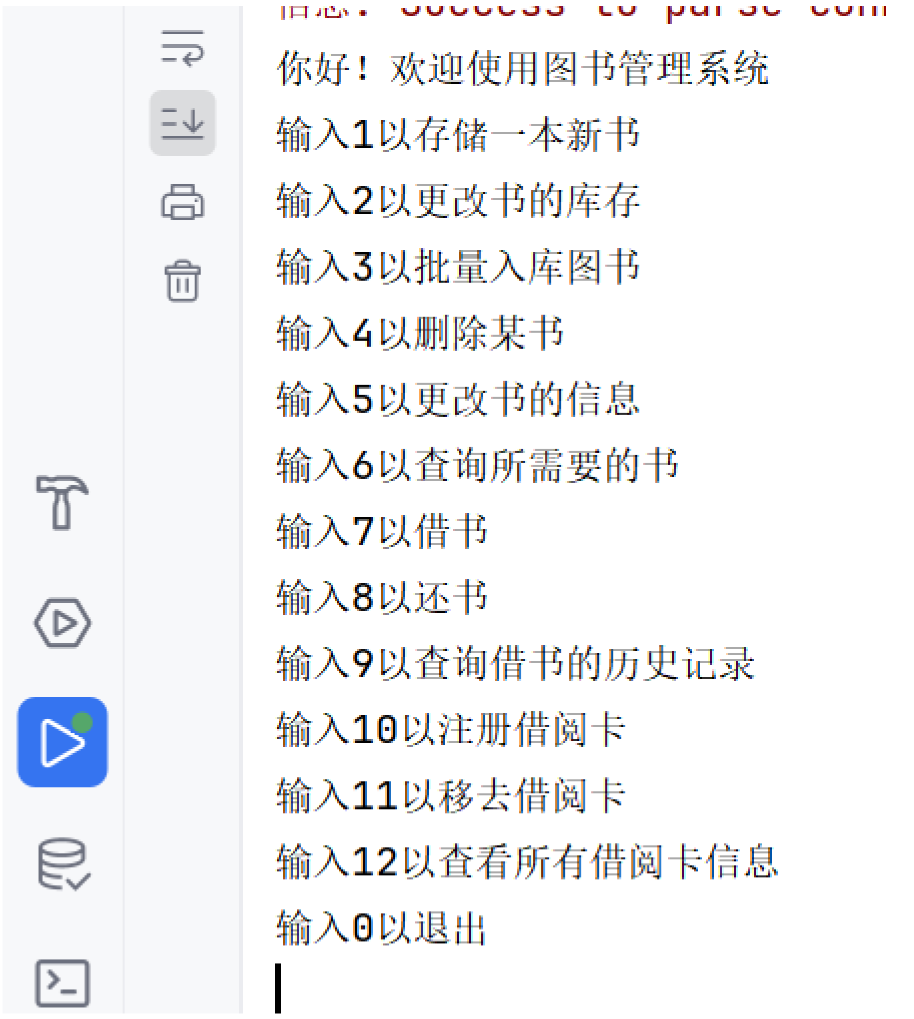
```
PS C:\Users\Administrator\Desktop\librarymanagementsystem> mvn -Dtest=LibraryTest clean test
```

如图使用`mvn -Dtest=LibraryTest clean test`命令进行所有测试点的测试，均通过，说明正确实现图书管理系统功能。



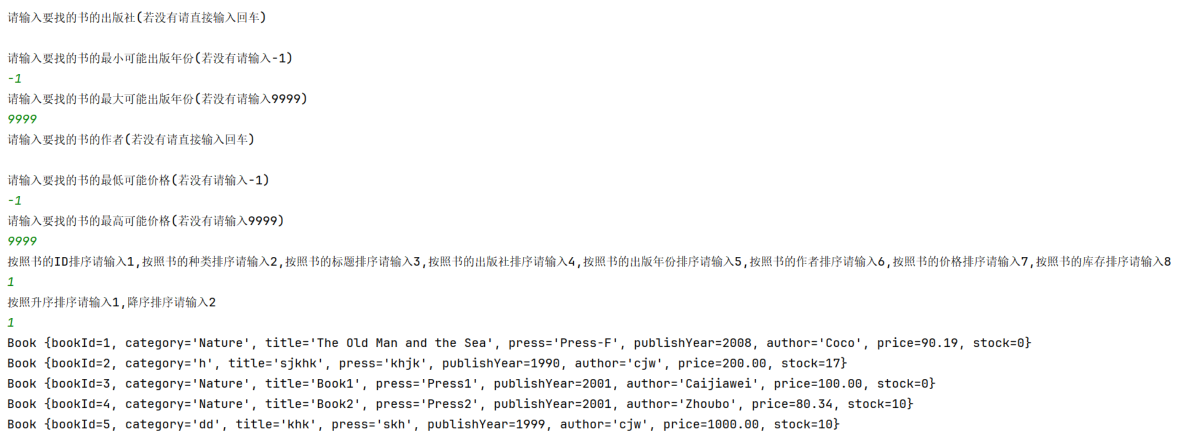
```
Successfully release database connection.
[INFO] Tests run: 9, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 13.58 s -- in LibraryTest
[INFO]
[INFO] Results:
[INFO]
[INFO] Tests run: 9, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  18.500 s
[INFO] Finished at: 2024-04-30T19:26:17+08:00
[INFO] ------------------------------------------------------------------------
PS C:\Users\Administrator\Desktop\librarymanagementsystem>
```

在前端测试中，如图成功运行了Main函数，进行一个简单测试，发现成功。（由于已经验收这里就简略些了TuT）



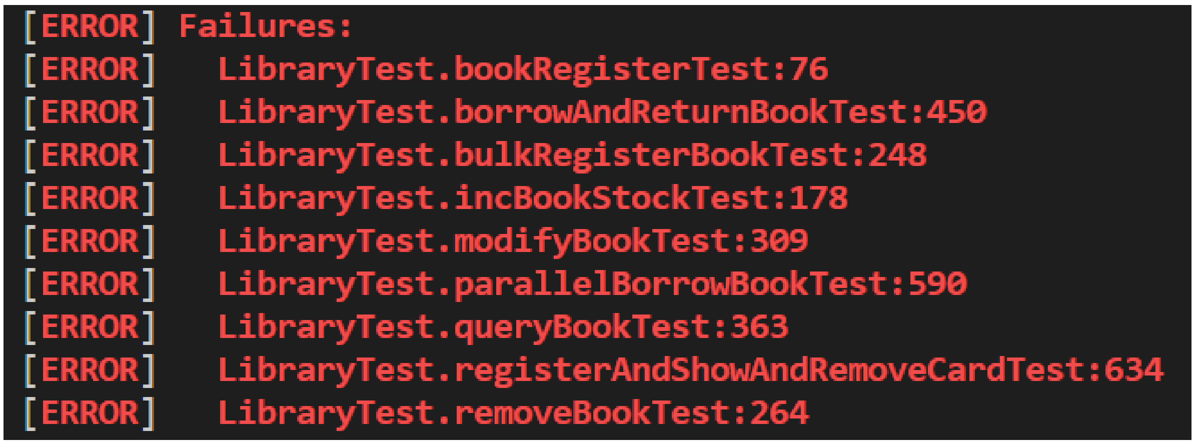
```
信息： Success to parse con
你好！欢迎使用图书管理系统
输入1以存储一本新书
输入2以更改书的库存
输入3以批量入库图书
输入4以删除某书
输入5以更改书的信息
输入6以查询所需要的书
输入7以借书
输入8以还书
输入9以查询借书的历史记录
输入10以注册借阅卡
输入11以移去借阅卡
输入12以查看所有借阅卡信息
输入0以退出
```

```
1
你需要管理员权限,请输入密码
01jj
输入错误,您还有5 次机会
01gg
输入正确,成功进入系统
请输入书的类别
dd
请输入书的标题
khk
请输入书的出版社
skh
请输入书的出版年份
1999
请输入书的作者
cjw
请输入书的价格
1000
请输入书的库存
10
```

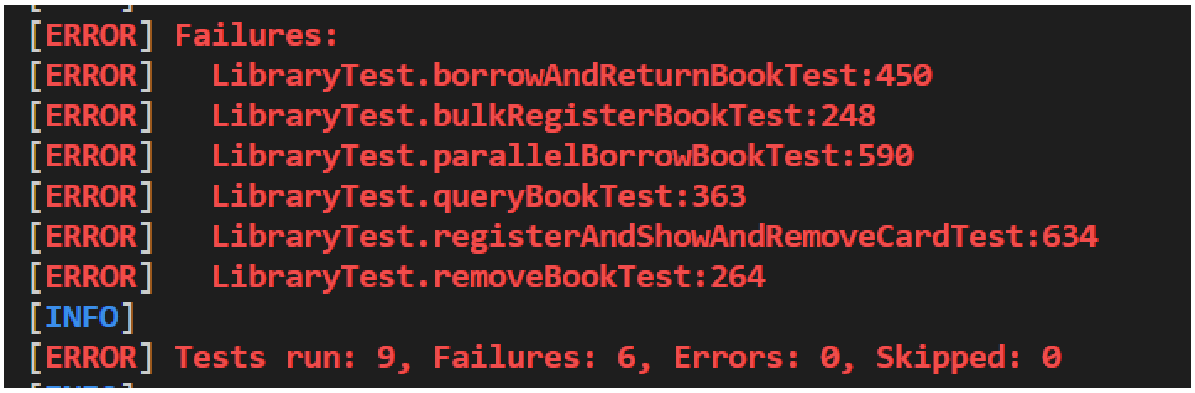


```
请输入要找的书的出版社(若没有请直接输入回车)

请输入要找的书的最小可能出版年份(若没有请输入-1)
-1
请输入要找的书的最大可能出版年份(若没有请输入9999)
9999
请输入要找的书的作者(若没有请直接输入回车)

请输入要找的书的最低可能价格(若没有请输入-1)
-1
请输入要找的书的最高可能价格(若没有请输入9999)
9999
按照书的ID排序请输入1,按照书的种类排序请输入2,按照书的标题排序请输入3,按照书的出版社排序请输入4,按照书的出版年份排序请输入5,按照书的作者排序请输入6,按照书的价格排序请输入7,按照书的库存排序请输入8
1
按照升序排序请输入1,降序排序请输入2
1
Book {bookId=1, category='Nature', title='The Old Man and the Sea', press='Press-F', publishYear=2008, author='Coco', price=90.19, stock=0}
Book {bookId=2, category='h', title='sjkhk', press='khjk', publishYear=1990, author='cjw', price=200.00, stock=17}
Book {bookId=3, category='Nature', title='Book1', press='Press1', publishYear=2001, author='Caijiawei', price=100.00, stock=0}
Book {bookId=4, category='Nature', title='Book2', press='Press2', publishYear=2001, author='Zhoubo', price=80.34, stock=10}
Book {bookId=5, category='dd', title='khk', press='skh', publishYear=1999, author='cjw', price=1000.00, stock=10}
```

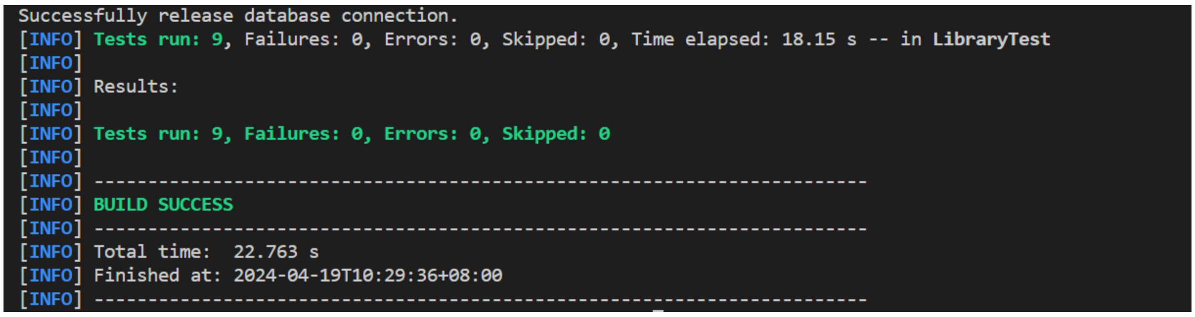
四、遇到的问题和方法



```
[ERROR] Failures:
[ERROR]   LibraryTest.bookRegisterTest:76
[ERROR]   LibraryTest.borrowAndReturnBookTest:450
[ERROR]   LibraryTest.bulkRegisterBookTest:248
[ERROR]   LibraryTest.incBookStockTest:178
[ERROR]   LibraryTest.modifyBookTest:309
[ERROR]   LibraryTest.parallelBorrowBookTest:590
[ERROR]   LibraryTest.queryBookTest:363
[ERROR]   LibraryTest.registerAndShowAndRemoveCardTest:634
[ERROR]   LibraryTest.removeBookTest:264
```



```
[ERROR] Failures:
[ERROR]   LibraryTest.borrowAndReturnBookTest:450
[ERROR]   LibraryTest.bulkRegisterBookTest:248
[ERROR]   LibraryTest.parallelBorrowBookTest:590
[ERROR]   LibraryTest.queryBookTest:363
[ERROR]   LibraryTest.registerAndShowAndRemoveCardTest:634
[ERROR]   LibraryTest.removeBookTest:264
[INFO]
[ERROR] Tests run: 9, Failures: 6, Errors: 0, Skipped: 0
```



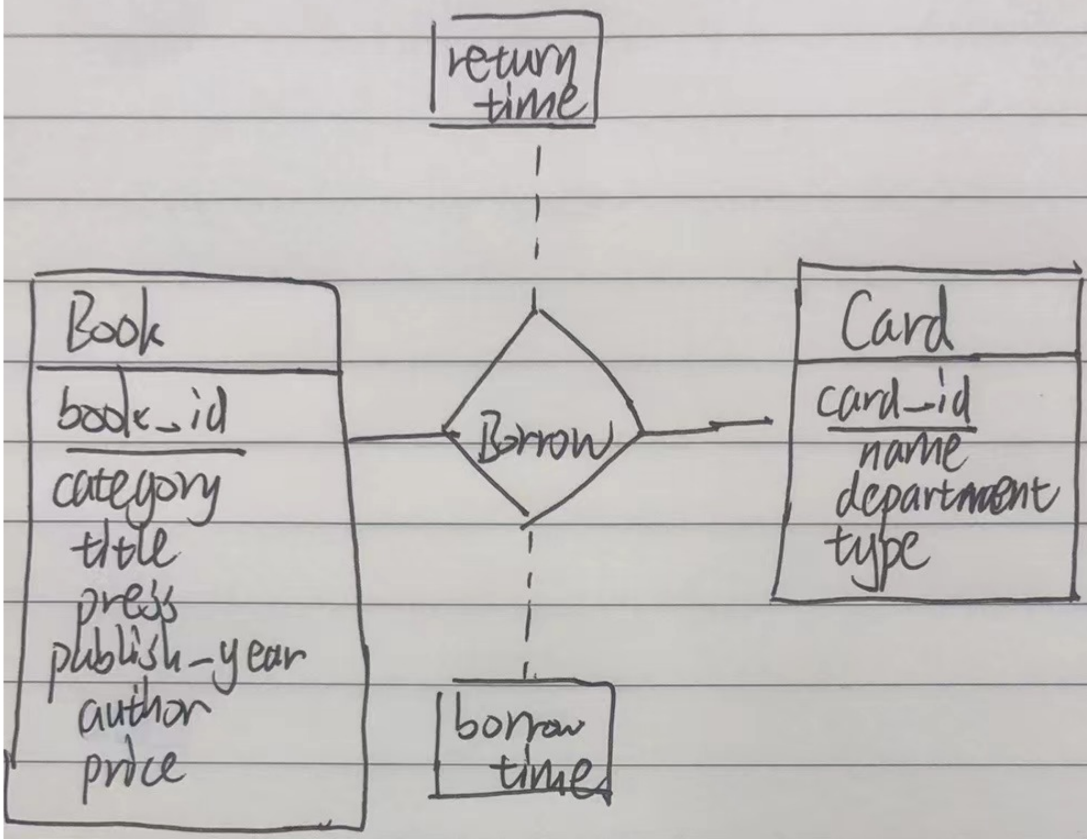
```
Successfully release database connection.
[INFO] Tests run: 9, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 18.15 s -- in LibraryTest
[INFO]
[INFO] Results:
[INFO]
[INFO] Tests run: 9, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  22.763 s
[INFO] Finished at: 2024-04-19T10:29:36+08:00
[INFO] ------------------------------------------------------------------------
```

在开始撰写图书管理系统后，我首先对JDBC非常不了解，也不知道整个程序的大体结构是什么样的。在网上查询和询问同学相关思路后，我逐渐了解了整体撰写的大致思路，也逐渐学习到了初步的java语法。一开始的报错问题很多，我逐步地对每个函数进行修改，反复测试，逐步定位到有错误的函数。一开始我还以为是环境的问题，因为都是每个Test中的ASSERT报错，也尝试去修改了环境，但是最后发现是函数本身的问题。

对于借书的加锁，一开始我也不了解，通过学习了解了JDBC中对操作加锁的方法，最终完成了这个函数。（`conn.setTransactionIsolation(Connection.TRANSACTION_SERIALIZABLE);`，可以通过这个语句，也可以在sql语句后加`for update`来加锁。）

五、思考题

1. 绘制该图书管理系统的E-R图


return time
Book
book_id
category
title
press
publish_year
author
price
Borrow
Card
card_id
name
department
type
borrow time

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注入攻击指的是通过构建特殊的输入语句作为参数传入Web应用程序，这些语句大多是SQL语法里的一些组合，通过执行SQL语句进而执行攻击者所需要的操作。主要原因是程序没有细致地过滤用户输入的数据，致使非法数据侵入系统。

我觉得可以类比在一个输入密码程序中，系统读入输入并且与系统内保存的密码作对比。系统内验证的操作是`cin>>str; if(str=="ABC")return true;`这种时候如果用户的输入是`true or xx`，系统判断就会变成`if(true or xx=="ABC");`可以预见到无论xx是否正确，都返回的是true值。

在图书管理系统中，例如对于借书系统，

解决方法是可以使用预编译语句，使用存储过程，检查数据类型，使用安全函数，多层验证等。

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？

MySQL中并发可能会引起的有三个问题，脏读（dirty read）——一个事务读取了另一个事务未提交的数据，不可重复读（unrepeatble read）——在一个事务内多次读取表中的数据，多次读取的结果不同，幻读（phantom read）——一个事务内读取到了别的事务插入或删除的数据，导致前后读取记录行数不同。

MySQL中有四个隔离级别，分别是读未提交（Read Uncommitted），读已提交（Read Committed），可重复读（Repeatable Read），串行化（Serializable）。MySQL的默认隔离级别即为可重复读。在默认隔离级别下，由于只能读取已经提交的数据，解决了脏读。

而解决不可重复读和幻读则是由于 MVCC 机制（Mutil-Version Concurrent Control(多版本并发控制)）。它对于数据库中的每一条数据，会存在多个版本，这是通过在每行记录后面保存两个隐藏的列来实现的。这两个列，一个保存 了行的创建时间，一个保存行的过期时间（或删除时间）。当然存储的并不是实际的时间值， 而是系统版本号（system version number）。每开始一个新的事务，系统版本号都会自动递增。事务开始时刻的系统版本号会作为事务的版本号，用来和查询到的每行记录的版本号进行比较。

SELECT: InnoDB 会根据以下两个条件检查每行记录： InnoDB 只查找版本早于当前事务版本的数据行（也就是，行的系统版本号小于或等于事务的系统版本号），这样可以确保事务读取的行，要么是在事务开始前已经存在的，要么是事务自身插入或者修改过的。 行的删除版本要么未定义，要么大于当前事务版本号。这可以确保事务读取到的行，在事务开始之前未被删除。 只有符合上述两个条件的记录，才能返回作为查询结果。

INSERT: InnoDB 为新插入的每一行保存当前系统版本号作为行版本号。

DELETE: InnoDB 为删除的每一行保存当前系统版本号作为行删除标识。

UPDATE: InnoDB 为插入一行新记录，保存当前系统版本号作为行版本号，同时保存当前系统版本号到原来的行作为行删除标识。