浙江大学

数据库系统实验报告

作业名称：图书管理系统

姓　　名：汪珉凯

学　　号：3220100975

电子邮箱：3220100975@zju.edu.cn

联系电话：18157421318

指导老师：孙建伶

2024 年 4 月 23 日

实验名称

一、 实验目的

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

二、 系统需求

2.1 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

2.2 基本功能模块

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

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

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

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

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

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

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

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

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

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

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

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

2.3 数据库(表)设计

```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
);
```

2.4 系统功能验证

系统功能验证测试分为功能性测试和正确性测试。

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

当实现了前端，图书管理系统表现为一个完整可使用的程序时，进行功能性测试。功能性测试通过验收时随机运行模拟场景的结果进行评判，以软件使用时的交互友好程度、效率、正确性等指标来评定功能性测试部分的得分。

三、 实验环境

我使用了 IntelliJ IDEA 作为本次实验后端的开发环境。

四、 系统设计及实现

1 绘制该图书管理系统的 E-R 图

2 系统各函数的设计思路和实现

2.1storeBook(Book book)

2.1.1 使用 PreparedStatement 插入新书籍记录到数据库。

2.1.2 插入前先检查书籍是否存在，避免重复添加。

2.1.3 设置事务，确保操作的原子性。

```
        @Override//done
        public ApiResult storeBook(Book book) {
            Connection conn= connector.getConn();//connection
            try {
                conn.setAutoCommit(false); // Disable auto-commit to handle transaction manually
                //check if the book exists?
                String sql_s = "SELECT * FROM book WHERE category = ? AND title = ? AND press = ? AND publish_year = ? AND author = ?";//sql-statement
                PreparedStatement statement = conn.prepareStatement(sql_s);//connect to the db
                statement.setString(1, book.getCategory());
                statement.setString(2, book.getTitle());
                statement.setString(3, book.getPress());
                statement.setInt(4, book.getPublishYear());
                statement.setString(5, book.getAuthor());//5 arguments
                ResultSet resultSet = statement.executeQuery();//query
                if (resultSet.next()) {//already exist
                    return new ApiResult(false, "A same book already exists.");
                } else {
                    String sql_i = "INSERT INTO book(category,title,press,publish_year,author,price,stock) VALUES(?,?,?,?,?,?,?)";//prepared statement
                    PreparedStatement insertStmt = conn.prepareStatement(sql_i, Statement.RETURN_GENERATED_KEYS);
                    insertStmt.setString(1, book.getCategory());
                    insertStmt.setString(2, book.getTitle());
                    insertStmt.setString(3, book.getPress());
                    insertStmt.setInt(4, book.getPublishYear());
                    insertStmt.setString(5, book.getAuthor());
                    insertStmt.setDouble(6, book.getPrice());
                    insertStmt.setInt(7, book.getStock());//
                    int rows = insertStmt.executeUpdate();
                    if (rows > 0) {//if insertion succeeded
                        conn.commit();
                        // Retrieve the generated keys for each inserted book
                        ResultSet generatedKeys = insertStmt.getGeneratedKeys();
                        int id=-1;
```

```
                    if (generatedKeys.next()) {
                        id = generatedKeys.getInt(1);
                        book.setBookId(id); //update book_id
                    }
                    return new ApiResult(true, "Storing book succeeded.");
                } else {//if insertion failed
                    conn.rollback();
                    return new ApiResult(false, "Storing book failed.");
                }
            }
        } catch (SQLException e) {//query failed
            e.printStackTrace();
            try {
                conn.rollback();
            } catch (SQLException ee) {
                //rollback failed
                e.printStackTrace();
            }
            return new ApiResult(false, "Storing book failed.");
        } finally {
            try {
                conn.setAutoCommit(true);
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
    }
```

2.2incBookStock(int bookId, int deltaStock)

2.2.1 通过 PreparedStatement 查询指定书籍的当前库存。

2.2.2 根据 deltaStock 增减库存，确保库存不会降到负数。

2.2.3 更新操作也使用了事务处理。

```
    @Override//done
    public ApiResult incBookStock(int bookId, int deltaStock) {
        Connection conn= connector.getConn();//connection
        try {
            conn.setAutoCommit(false);
            String selSql = "SELECT * FROM book WHERE book_id = ?";
            PreparedStatement selStatement = conn.prepareStatement(selSql);
            selStatement.setInt(1, bookId);
            ResultSet resultSet = selStatement.executeQuery();
            if (resultSet.next()) {
                int stock = resultSet.getInt("stock");
                if (stock + deltaStock < 0) {//库存不够
```

```
                    return new ApiResult(false, "There aren't so many stocks.");
                } else {
                    stock += deltaStock;//new stock
                    String setSql = "UPDATE book SET stock = ? WHERE book_id = ?";
                    PreparedStatement upStatement = conn.prepareStatement(setSql);
                    upStatement.setInt(1, stock);
                    upStatement.setInt(2, bookId);//update new stock
                    int rows = upStatement.executeUpdate();
                    if (rows > 0) {
                        return new ApiResult(true, "IncBookStock succeeded.");
                    } else {
                        return new ApiResult(false, "IncBookStock failed.");
                    }
                }

            } else {
                return new ApiResult(false, "There isn't such a book");
            }

        } catch (SQLException e) {
            e.printStackTrace();
            try {
                conn.rollback();
            } catch (SQLException ee) {
                ee.printStackTrace();
            }
            return new ApiResult(false, "IncBookStock failed.");
        } finally {
            try {
                conn.setAutoCommit(true);
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
    }
```

2.3storeBook(List<Book> books)

2.3.1 该方法批量插入多本书籍，使用 PreparedStatement 的批处理功能。

2.3.2 与单本插入类似，先检查是否存在，然后再执行插入。

2.3.3 使用事务来保证多个插入操作的完整性。

```
    @Override//done
    public ApiResult storeBook(List<Book> books) {
        try {
            connector.getConn().setAutoCommit(false);
```

```
5.                 String insSql = "INSERT INTO book(category,title,press,publish_year,author,price,stock)
   VALUES(?,?,?,?,?,?,?)";
6.                 PreparedStatement Stmt = connector.getConn().prepareStatement(insSql, Statement.RETURN
   _GENERATED_KEYS);
7.                 for (Book book : books) {
8.                     Stmt.setString(1, book.getCategory());
9.                     Stmt.setString(2, book.getTitle());
10.                    Stmt.setString(3, book.getPress());
11.                    Stmt.setInt(4, book.getPublishYear());
12.                    Stmt.setString(5, book.getAuthor());
13.                    Stmt.setDouble(6, book.getPrice());
14.                    Stmt.setInt(7, book.getStock());
15.                    Stmt.addBatch();
16.                }
17.                int[] rows = Stmt.executeBatch();
18.                for (int i : rows) {
19.                    if (i == 0) {
20.                        try {
21.                            connector.getConn().rollback();
22.                        } catch (SQLException ex) {
23.                            ex.printStackTrace();
24.                        }
25.                        return new ApiResult(false, "Storing book failed.");
26.                    }
27.                }
28.                ResultSet keys = Stmt.getGeneratedKeys();
29.                int i = 0;
30.                while (keys.next()) {
31.                    books.get(i).setBookId(keys.getInt(1));
32.                    i++;
33.                }
34.                connector.getConn().commit();
35.                return new ApiResult(true, "Storing book succeeded.");
36.            } catch (Exception e) {
37.                try {
38.                    connector.getConn().rollback();
39.                } catch (SQLException ee) {
40.                    ee.printStackTrace();
41.                }
42.                return new ApiResult(false, "Storing book  failed.");
43.            } finally {
44.                try {
45.                    connector.getConn().setAutoCommit(true);
46.                } catch (SQLException eee) {
```

```
                eee.printStackTrace();
            }
        }
    }
```

2.4removeBook(int bookId)

2.4.1 先检查指定的书籍是否存在。

2.4.2 如果存在，则通过 PreparedStatement 执行删除操作。

2.4.3 使用事务确保删除操作的一致性。

```
    @Override
    public ApiResult removeBook(int bookId) {
        Connection conn= connector.getConn();
        try {
            conn.setAutoCommit(false);
            //检查这本书是否存在
            String check_book_exists = "SELECT * FROM book WHERE book_id = ?";
            PreparedStatement stmt = conn.prepareStatement(check_book_exists);
            stmt.setInt(1, bookId);
            ResultSet resultSet = stmt.executeQuery();
            if (!resultSet.next()) {
                return new ApiResult(false, "Book does not exist.");
            }

            //检查这本书是否正在被外借
            String check_book_unreturned = "SELECT * FROM borrow WHERE book_id = ? AND (return_time<borrow_time)";
            PreparedStatement stmt2 = conn.prepareStatement(check_book_unreturned);
            stmt2.setInt(1, bookId);
            ResultSet resultSet2 = stmt2.executeQuery();
            if (resultSet2.next()) {
                return new ApiResult(false, "Book unreturned.");
            }

            String delSQL = "DELETE FROM book WHERE book_id = ?";
            PreparedStatement delStmt = conn.prepareStatement(delSQL);
            delStmt.setInt(1, bookId);
            int rows = delStmt.executeUpdate();
            if (rows > 0) {
                conn.commit();
                return new ApiResult(true, "Book deleted.");
            }
            return new ApiResult(false, "Return book failed.");
        }
        catch(SQLException e) {
```

```
            try {
                conn.rollback();
            }catch(SQLException ee) {
                ee.printStackTrace();
            }
            return new ApiResult(false, "Removing book failed.");
        }
        finally{
            try {
                conn.setAutoCommit(true);
            }catch(SQLException eee) {
                eee.printStackTrace();
            }
        }
    }
```

2.5modifyBookInfo(Book book)

2.5.1 通过 PreparedStatement 查询要修改的书籍是否存在。

2.5.2 如果存在，则更新书籍的相关信息（如标题、作者等）。

2.5.3 事务处理确保数据更新的完整性。

```
    @Override
    public ApiResult modifyBookInfo(Book book) {
        Connection conn= connector.getConn();
        try {
            conn.setAutoCommit(false);
            //先搜索是否有符合条件的书籍
            String checkSQL = "SELECT * FROM book WHERE book_id = ?";
            PreparedStatement stmt = conn.prepareStatement(checkSQL);
            stmt.setInt(1, book.getBookId());
            ResultSet resultSet = stmt.executeQuery();
            if (!resultSet.next()) {
                return new ApiResult(false, "Book does not exist.");
            }

            //update the information of the book
            String updateSQL="UPDATE book SET ";
            int flag=0;//if flag=1,then update is needed
            if(book.getCategory() != null) {
                updateSQL += "category = ?";
                flag++;
            }
            if(book.getTitle() != null) {
                updateSQL += (flag > 0 ? "," : "") + "title = ?";
                flag++;
```

```
25.             }
26.             if(book.getPress() != null) {
27.                 updateSQL += (flag > 0 ? "," : "") + "press = ?";
28.                 flag++;
29.             }
30.             if(book.getPublishYear() != -1) {
31.                 updateSQL += (flag > 0 ? "," : "") + "publish_year = ?";
32.                 flag++;
33.             }
34.             if(book.getAuthor() != null) {
35.                 updateSQL += (flag >0 ? "," : "") + "author = ?";
36.                 flag++;
37.             }
38.             if(book.getPrice() != -1) {
39.                 updateSQL += (flag > 0 ? "," : "") + "price = ?";
40.                 flag++;
41.             }
42.             if(flag == 0) {
43.                 return new ApiResult(false,"no update is needed");
44.             }
45.
46.             updateSQL+="WHERE book_id = ?";
47.             PreparedStatement stmt2 = conn.prepareStatement(updateSQL);
48.             int index=1;
49.             if(book.getCategory() != null) {
50.                 stmt2.setString(index++, book.getCategory());
51.             }
52.             if(book.getTitle() != null) {
53.                 stmt2.setString(index++, book.getTitle());
54.             }
55.             if(book.getPress() != null) {
56.                 stmt2.setString(index++, book.getPress());
57.             }
58.             if(book.getPublishYear() != -1) {
59.                 stmt2.setInt(index++, book.getPublishYear());
60.             }
61.             if(book.getAuthor() != null) {
62.                 stmt2.setString(index++, book.getAuthor());
63.             }
64.             if(book.getPrice() != -1) {
65.                 stmt2.setDouble(index++, book.getPrice());
66.             }
67.             stmt2.setInt(index, book.getBookId());
68.             int rows = stmt2.executeUpdate();
```

```
            if (rows > 0) {
                conn.commit();
                return new ApiResult(true, "Book updated.");
            }else {
                return new ApiResult(false, "Book update failed.");
            }
        }catch (SQLException e) {
            try {
                conn.rollback();
            }catch (SQLException ee) {
                ee.printStackTrace();
            }
            return new ApiResult(false, "Modifying book failed.");
        }finally {
            try {
                conn.setAutoCommit(true);
            }catch (SQLException eee) {
                eee.printStackTrace();
            }
        }
    }
```

2.6queryBook(BookQueryConditions conditions)

2.6.1 根据提供的查询条件构建动态 SQL 查询。

2.6.2 使用 PreparedStatement 执行查询，保证安全性。

2.6.3 处理结果集，返回符合条件的所有书籍信息。

```
    @Override//done
    public ApiResult queryBook(BookQueryConditions conditions) {
        Connection conn= connector.getConn();
        try {
            StringBuilder QuerySQL = new StringBuilder();//构造一个可变字符串
            QuerySQL.append("SELECT * FROM book WHERE 1=1 ");//不断往这个 query 语句中加入筛选条件
            List<Object> params = new ArrayList<>();//往这个向量中不停加入带筛选的参数
            if (conditions.getCategory() != null) {//根据种类筛选
                QuerySQL.append("AND category = ?");
                params.add(conditions.getCategory());
            }
            if (conditions.getTitle() != null) {
                QuerySQL.append(" AND title LIKE ?");
                params.add("%" + conditions.getTitle() + "%");
            }
            if (conditions.getPress() != null) {
                QuerySQL.append(" AND press LIKE ?");
                params.add("%" + conditions.getPress() + "%");
```

```
        }
        if (conditions.getMinPublishYear() != null) {
            QuerySQL.append(" AND publish_year >= ?");
            params.add(conditions.getMinPublishYear());
        }
        if (conditions.getMaxPublishYear() != null) {
            QuerySQL.append(" AND publish_year <= ?");
            params.add(conditions.getMaxPublishYear());
        }
        if (conditions.getAuthor() != null) {
            QuerySQL.append(" AND author LIKE ?");
            params.add("%" + conditions.getAuthor() + "%");
        }
        if (conditions.getMinPrice() != null) {
            QuerySQL.append(" AND price >= ?");
            params.add(conditions.getMinPrice());
        }
        if (conditions.getMaxPrice() != null) {
            QuerySQL.append(" AND price <= ?");
            params.add(conditions.getMaxPrice());
        }
        boolean flag = false;//是否对查询结果进行排序
        if (conditions.getSortBy() != null) {//排序特征
            flag = true;
            QuerySQL.append(" ORDER BY ").append(conditions.getSortBy());
        }
        if (conditions.getSortBy() != null && conditions.getSortOrder() != null) {//排序方式
            QuerySQL.append(' ').append(conditions.getSortOrder().getValue());
        }
        if(flag){//如果有进行排序
            QuerySQL.append(" ,book_id ASC");
        }else{//没有排序指标则默认是依靠 book_id 排序
            QuerySQL.append(" ORDER BY book_id ASC");
        }
        System.out.println(connector.toString());
        PreparedStatement stmt = conn.prepareStatement(QuerySQL.toString());
        for (int i = 0; i < params.size(); i++) {
            stmt.setObject(i+1, params.get(i));
        }//把不同的参数从向量中取出来，并加入到查询的条件中去
        ResultSet resultSet = stmt.executeQuery();
        List<Book> bookList = new ArrayList<>();
        while (resultSet.next()) {
            Book tempbook = new Book();//临时存放结果
            tempbook.setBookId(resultSet.getInt("book_id"));
```

```
                tempbook.setCategory(resultSet.getString("category"));
                tempbook.setTitle(resultSet.getString("title"));
                tempbook.setPress(resultSet.getString("press"));
                tempbook.setPublishYear(resultSet.getInt("publish_year"));
                tempbook.setAuthor(resultSet.getString("author"));
                tempbook.setPrice(resultSet.getDouble("price"));
                tempbook.setStock(resultSet.getInt("stock"));
                bookList.add(tempbook);//把每一本被查询到的书存入向量中去
            }
            //正常的返回结果是一个查询结果对象

            BookQueryResults bookQueryResults = new BookQueryResults(bookList);
            return new ApiResult(true, bookQueryResults);
        } catch (SQLException e) {
            e.printStackTrace();
            try {
                conn.rollback();
            } catch (SQLException ee) {
                // 处理异常
                e.printStackTrace();
            }
            return new ApiResult(false, "QueryBook failed.");
        } finally {
            try{
                conn.setAutoCommit(true);
            }catch (SQLException ee){
                ee.printStackTrace();
            }
        }
    }
```

2. 7borrowBook(Borrow borrow)

2. 7. 1 检查书籍库存是否足够。

2. 7. 2 如果足够，则记录借书信息，并更新库存。

2. 7. 3 事务处理确保借书记录和库存更新的原子性。

```
    @Override
    public ApiResult borrowBook(Borrow borrow) {
        Connection conn= connector.getConn();
        try {
            synchronized(Lock){
                conn.setAutoCommit(false);

                //先看是不是借了书没还
```

```
9.                  String selSQL="SELECT * FROM borrow WHERE card_id = ? AND book_id = ? AND return_time <borrow_time";
10.                 PreparedStatement stmt = conn.prepareStatement(selSQL);
11.                 stmt.setInt(1, borrow.getCardId());
12.                 stmt.setInt(2, borrow.getBookId());
13.                 ResultSet resultSet = stmt.executeQuery();
14.                 if(resultSet.next()){
15.                     //还有没还的
16.                     return new ApiResult(false, "Still books not returned.");
17.                 }
18.                 else{
19.                     //先验证cardid 合理性
20.                     String checkSQL="SELECT * FROM card WHERE card_id = ?";
21.                     PreparedStatement checkStmt = conn.prepareStatement(checkSQL);
22.                     checkStmt.setInt(1, borrow.getCardId());
23.                     ResultSet resultSet1 = checkStmt.executeQuery();
24.                     if(!resultSet1.next()){
25.                         return new ApiResult(false, "Card-id does not exist.");
26.                     }
27.
28.                     //找出符合的书
29.                     String selSQL2="SELECT * FROM book WHERE book_id = ? AND stock>0";
30.                     PreparedStatement selectStmt = conn.prepareStatement(selSQL2);
31.                     selectStmt.setInt(1, borrow.getBookId());
32.                     ResultSet resultSet2 = selectStmt.executeQuery();
33.                     if(!resultSet2.next()){
34.                         return new ApiResult(false, "Borrow book failed.");
35.                     }else {
36.                         String upstockSQL = "UPDATE book SET stock = stock - 1 WHERE book_id = ? AND NOT EXISTS(SELECT * FROM borrow WHERE card_id = ? AND book_id = ? AND return_time <borrow_time) AND stock>0";
37.                         PreparedStatement upstockStmt = conn.prepareStatement(upstockSQL);
38.                         upstockStmt.setInt(1, borrow.getBookId());
39.                         upstockStmt.setInt(2, borrow.getCardId());
40.                         upstockStmt.setInt(3, borrow.getBookId());
41.                         int upstock = upstockStmt.executeUpdate();
42.                         if(upstock == 0){
43.                             return new ApiResult(false, "Borrow book failed.");
44.                         }
45.                         String insertSQL = "INSERT INTO borrow(card_id,book_id,borrow_time) VALUES (?,?,?)";
46.                         PreparedStatement insertStmt = conn.prepareStatement(insertSQL);
47.                         insertStmt.setInt(1, borrow.getCardId());
48.                         insertStmt.setInt(2, borrow.getBookId());
```

```
                        insertStmt.setLong(3,borrow.getBorrowTime());
                        int insert = insertStmt.executeUpdate();
                        if(insert == 0){
                            return new ApiResult(false, "Borrow book failed.");
                        }
                        else {
                            conn.commit();
                            return new ApiResult(true, "Borrowed the book successfully.");
                        }
                    }
                }
            }catch (SQLException e1){
                try {
                    conn.rollback();
                }catch (SQLException e2){
                    e2.printStackTrace();
                }
                e1.printStackTrace();
                return new ApiResult(false,"borrowBook failed.");
            }finally {
                try {
                    conn.setAutoCommit(true);
                }catch (SQLException e3){
                    e3.printStackTrace();
                }
            }
        }
```

2.8returnBook(Borrow borrow)

2.8.1 更新借阅记录，标记书籍已归还。

2.8.2 同时更新库存，增加归还的书籍数量。

2.8.3 使用事务处理确保操作的完整性。

```
        @Override
        public ApiResult returnBook(Borrow borrow) {
            Connection conn= connector.getConn();
            try {
                conn.setAutoCommit(false);
                String SelSQL="SELECT * FROM borrow WHERE card_id = ? AND book_id = ? AND (return_time < borrow_time||return_time = 0) AND borrow_time < ?";
                //是否有满足的结束条件
                PreparedStatement selStmt = conn.prepareStatement(SelSQL);
                selStmt.setInt(1, borrow.getCardId());
                selStmt.setInt(2, borrow.getBookId());
```

```
            selStmt.setLong(3,borrow.getReturnTime());
            ResultSet selResult = selStmt.executeQuery();
            if(selResult.next()){
                String upSQL= "UPDATE borrow SET return_time = ? WHERE card_id =? AND book_id = ? AND (return_time < borrow_time||return_time = 0)";
                PreparedStatement upstmt = conn.prepareStatement(upSQL);
                upstmt.setLong(1, borrow.getReturnTime());
                upstmt.setInt(2, borrow.getCardId());
                upstmt.setInt(3, borrow.getBookId());
                int rows = upstmt.executeUpdate();//执行更新操作
                if(rows == 0){//如果操作失败
                    return new ApiResult(false, "Return book failed.");
                }else {
                    //更新库存
                    String upstockSQL= "UPDATE book SET stock = stock + 1 WHERE book_id = ?";
                    PreparedStatement upstmt2 = conn.prepareStatement(upstockSQL);
                    upstmt2.setInt(1, borrow.getBookId());
                    int rows2 = upstmt2.executeUpdate();
                    if(rows2 == 0){
                        //库存更新失败
                        conn.rollback();
                        return new ApiResult(false, "Update book stock failed.");
                    }else {
                        conn.commit();
                        return new ApiResult(true, "Return book successfully.");
                    }
                }
            }else {
                return new ApiResult(false, "Illegal book.");
            }
        }catch (SQLException e1){
            try {
                conn.rollback();
            }catch (SQLException e2){
                e2.printStackTrace();
            }
            e1.printStackTrace();
            return new ApiResult(false,"returnBook failed.");
        }finally {
            try {
                conn.setAutoCommit(true);
            }catch (SQLException e3){
                e3.printStackTrace();
            }
```

```
        }
    }
```

2.9showBorrowHistory(int cardId)

2.9.1 查询指定读者卡的所有借阅记录。

2.9.2 处理结果集，返回所有相关的借阅信息。

```
    @Override
    public ApiResult showBorrowHistory(int cardId) {
        Connection conn= connector.getConn();
        try {
            conn.setAutoCommit(false);
            String bookSQL = "SELECT * FROM borrow WHERE card_id = ? ORDER BY borrow_time DESC, book_id ASC";
            PreparedStatement bookStmt = conn.prepareStatement(bookSQL);
            bookStmt.setInt(1, cardId);
            ResultSet bookResult = bookStmt.executeQuery();

            List <BorrowHistories.Item> records= new ArrayList<>();
            while(bookResult.next()){
                Borrow borrow = new Borrow();
                borrow.setCardId(cardId);
                int bookId = bookResult.getInt("book_id");
                borrow.setBookId(bookId);
                borrow.setReturnTime(bookResult.getLong("return_time"));
                borrow.setBorrowTime((bookResult.getLong("borrow_time")));

                Book book = new Book();
                String bookSQL2 = "SELECT * FROM book WHERE book_id = ?";
                PreparedStatement bookStmt2 = conn.prepareStatement(bookSQL2);
                bookStmt2.setInt(1, bookId);
                ResultSet bookResult2 = bookStmt2.executeQuery();
                if(bookResult2.next()){
                    book.setBookId(bookResult2.getInt("book_id"));
                    book.setPrice(bookResult2.getDouble("price"));
                    book.setAuthor(bookResult2.getString("author"));
                    book.setTitle(bookResult2.getString("title"));
                    book.setCategory(bookResult2.getString("category"));
                    book.setPress(bookResult2.getString("press"));
                    book.setPublishYear(bookResult2.getInt("publish_year"));
                }
                BorrowHistories.Item item = new BorrowHistories.Item(cardId,book,borrow);
                records.add(item);
            }

```

```
            return new ApiResult(true,new BorrowHistories(records));
        }
        catch (SQLException e1){
            e1.printStackTrace();
            try {
                conn.rollback();
            }
            catch (SQLException e2){
                e2.printStackTrace();
            }
            return new ApiResult(false,"showBorrowHistory failed.");
        }
        finally {
            try {
                conn.setAutoCommit(true);
            }
            catch (SQLException e3){
                e3.printStackTrace();
            }
        }
    }
```

2.10registerCard(Card card)

2.10.1 检查读者卡是否已被注册。

2.10.2 如果未注册，则插入新的读者卡信息。

2.10.3 事务处理保证数据一致性。

```
    @Override
    public ApiResult registerCard(Card card) {
        Connection conn= connector.getConn();
        try {
            conn.setAutoCommit(false);
            String selSQL = "SELECT * FROM card WHERE name = ? AND department = ? AND type = ? ";
            PreparedStatement stmt = conn.prepareStatement(selSQL);
            stmt.setString(1, card.getName());
            stmt.setString(2, card.getDepartment());
            stmt.setString(3,card.getType().getStr());
            ResultSet resultSet = stmt.executeQuery();

            if(resultSet.next()){//已经被注册了
                return new ApiResult(false, "Already registered.");
            }

            else {
                String insertSQL = "INSERT INTO card(name,department,type) VALUES (?,?,?)";
```

```
                PreparedStatement inStmt = conn.prepareStatement(insertSQL,Statement.RETURN_GENERATED_KEYS);
                inStmt.setString(1, card.getName());
                inStmt.setString(2, card.getDepartment());
                inStmt.setString(3,card.getType().getStr());
                int rows = inStmt.executeUpdate();
                if(rows>0){
                    ResultSet rs = inStmt.getGeneratedKeys();
                    rs.next();
                    int cardId = rs.getInt(1);
                    card.setCardId(cardId);
                    conn.commit();
                    return new ApiResult(true, cardId+"Registered.");
                }
                else {
                    try {
                        conn.rollback();
                    }catch (SQLException e){
                        e.printStackTrace();
                    }
                    return new ApiResult(false, "Not registered.");
                }
            }
        }catch (SQLException e0){
            try {
                conn.rollback();
            }catch (SQLException e1){
                e1.printStackTrace();
            }
            e0.printStackTrace();
            return new ApiResult(false,"registerCard failed.");
        }finally {
            try {
                conn.setAutoCommit(true);
            }catch (SQLException e2){
                e2.printStackTrace();
            }
        }
    }
```

2.11removeCard(int cardId)

2.11.1 检查读者是否有未归还的书籍。

2.11.2 如果没有未归还书籍，删除读者卡信息。

2.11.3 使用事务处理确保删除操作的一致性。

```java
        @Override
        public ApiResult removeCard(int cardId) {
            Connection conn= connector.getConn();
            try {
                conn.setAutoCommit(false);
                //先看有没有该用户还没还的书
                String borrowSQL = "SELECT * FROM borrow WHERE card_id = ? AND (return_time < borrow_time || return_time = 0 )";
                PreparedStatement borrowStmt = conn.prepareStatement(borrowSQL);
                borrowStmt.setInt(1, cardId);
                ResultSet borrowResult = borrowStmt.executeQuery();
                if(borrowResult.next()){
                    return new ApiResult(false, "Books not returned yet.");
                }

                //再看 card-id 是否有效
                String checkSQL = "SELECT * FROM card WHERE card_id = ?";
                PreparedStatement checkStmt = conn.prepareStatement(checkSQL);
                checkStmt.setInt(1, cardId);
                ResultSet checkResult = checkStmt.executeQuery();
                if(!checkResult.next()){
                    return new ApiResult(false, "Card-ID does not exist.");
                }

                String deleteSQL = "DELETE FROM card WHERE card_id = ?";
                PreparedStatement deleteStmt = conn.prepareStatement(deleteSQL);
                deleteStmt.setInt(1, cardId);
                int rows = deleteStmt.executeUpdate();
                if(rows>0){
                    conn.commit();
                    return new ApiResult(true, cardId+"Removed.");
                }else {
                    return new ApiResult(false, "Card remove failed.");
                }

            }catch (SQLException e){
                try {
                    conn.rollback();
                }catch (SQLException e3){
                    e3.printStackTrace();
                }
                return new ApiResult(false,"removeCard failed.");
            }finally {
                try {
```

```
                conn.setAutoCommit(true);
            }catch (SQLException e2){
                e2.printStackTrace();
            }
        }
    }
```

2. 12showCards()

2. 12. 1 查询并返回所有注册的读者卡信息。

2. 12. 2 处理结果集，以列表形式返回所有卡信息。

```
    @Override
    public ApiResult showCards() {
        Connection conn= connector.getConn();
        try {
            conn.setAutoCommit(false);
            String selectSQL = "SELECT * FROM card ORDER BY card_id";
            PreparedStatement selectStmt = conn.prepareStatement(selectSQL);
            ResultSet resultSet = selectStmt.executeQuery();
            List <Card> cards = new ArrayList<>();
            while(resultSet.next()){
                Card card = new Card();
                card.setCardId(resultSet.getInt("card_id"));
                card.setName(resultSet.getString("name"));
                card.setDepartment(resultSet.getString("department"));
                card.setType(Card.CardType.values(resultSet.getString("type")));
                cards.add(card);
            }
            return new ApiResult(true, new CardList(cards));
        }
        catch (SQLException e){
            try {
                conn.rollback();
            }
            catch (SQLException e1){
                e1.printStackTrace();
            }
            e.printStackTrace();
            return new ApiResult(false,"showCards failed.");
        }
        finally {
            try {
                conn.setAutoCommit(true);
            }
            catch (SQLException e2){
```

```
                e2.printStackTrace();
            }
        }
    }
```

3.程序运行结果场景以及截图说明（即实验指导文档中的系统功能验证）

我还没来得及做前端部分的内容 ORZ.

五、 遇到的问题及解决方法

思考题 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？

答：SQL 注入攻击是一种利用 Web 应用程序未正确过滤用户输入数据，从而在数据库中执行恶意 SQL 语句的攻击方式。攻击者通过在输入字段中注入恶意的 SQL 代码，使得应用程序误以为是合法的数据库查询语句，从而执行恶意操作，如删除数据、修改数据、或者获取敏感信息。

举例来说，假设一个网站的登录页面，其后台使用类似以下的 SQL 语句进行查询：

SELECT * FROM users WHERE username = '输入的用户名' AND password = '输入的密码';

攻击者可以通过在用户名或密码输入框中输入恶意的 SQL 代码，比如：' OR '1'='1

那么最终执行的 SQL 语句就会变成：

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

这将导致查询返回所有用户的数据，因为'1'='1'这个条件始终为真。

在图书管理系统中，可能会受到 SQL 注入攻击的模块包括但不限于：

用户登录模块：如上述例子中的登录功能。

图书搜索模块：如果搜索功能不正确地过滤用户输入，就容易受到攻击。

用户注册模块：如果注册信息插入数据库的过程存在漏洞，攻击者可能进行注入攻击。

要解决 SQL 注入攻击，可以采取以下措施：

使用参数化查询：使用参数化查询可以防止恶意 SQL 代码的注入，因为参数值会被视为数据而不是 SQL 代码的一部分。

输入验证和过滤：对用户输入的数据进行验证和过滤，确保只有合法的数据被传递给数据库。

最小权限原则：确保数据库用户具有最小的权限，限制其对数据库的操作范围，从而减少攻击面。

定期更新和审查代码：定期审查应用程序的代码，确保没有潜在的安全漏洞存在，并及时更新修复已发现的漏洞。

思考题 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？

答：我采用了行级锁来处理这个问题，即通过在执行 SQL 语句时使用 synchronized 关键字来确保对关键代码块的互斥访问。这意味着同一时刻只有一个线程可以执行借书操作，避免了并发访问导致的问题。

我遇到的问题：由于第一次使用 IDEA 开发软件，我并不熟悉如何根据报错来调试自己的代码，debug 更是束手无策。这使我的开始时写代码事倍功半。而且这也是我第一次接触 Java 类型的语言，感觉到比较陌生。

解决方法：我目前唯一学习的 OOP 语言是 C++，我试着用 C++的语法去思考每个模块的写法，就会感觉好很多。我也根据 error 的类型，通缩搜索引擎找到可能出问题的原因，后面也开始熟悉起来。此外，IDEA 自带的预测代码 TAB 补全功能帮助我省去了写那些枯燥乏味的结构化的代码（比如连接、断开等），让我把更多心思放到实现具体函数内容中去。

六、 总结

第一次做 DB 的大型实验，跌跌撞撞，感觉最难的部分是 JDBC 环境的配置以及如何通过 Java 语言来实现数据库的种种操作。由于时间关系，我只完成了后端部分，希望以后有时间能够再做一些前端的工作，使这次的实验结果更加完善。