Lab5: 图书管理系统

实验目的

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

实验需求

基本数据对象

所有的基本数据对象都被定义在`src/main/java/entities`包中，包括：

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

基本模块功能

- 接口`LibraryManagementSystem`中声明了图书管理系统的所有功能模块
- 类`LibraryManagementSystemImpl`中实现了接口`LibraryManagementSystem`的所有功能模块
- 主类`Main`实现了客户端的操作

数据库（表）设计

book表

```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`)
);
```

card表

```sql
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') )
);
```

borrow表

```sql
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
);
```

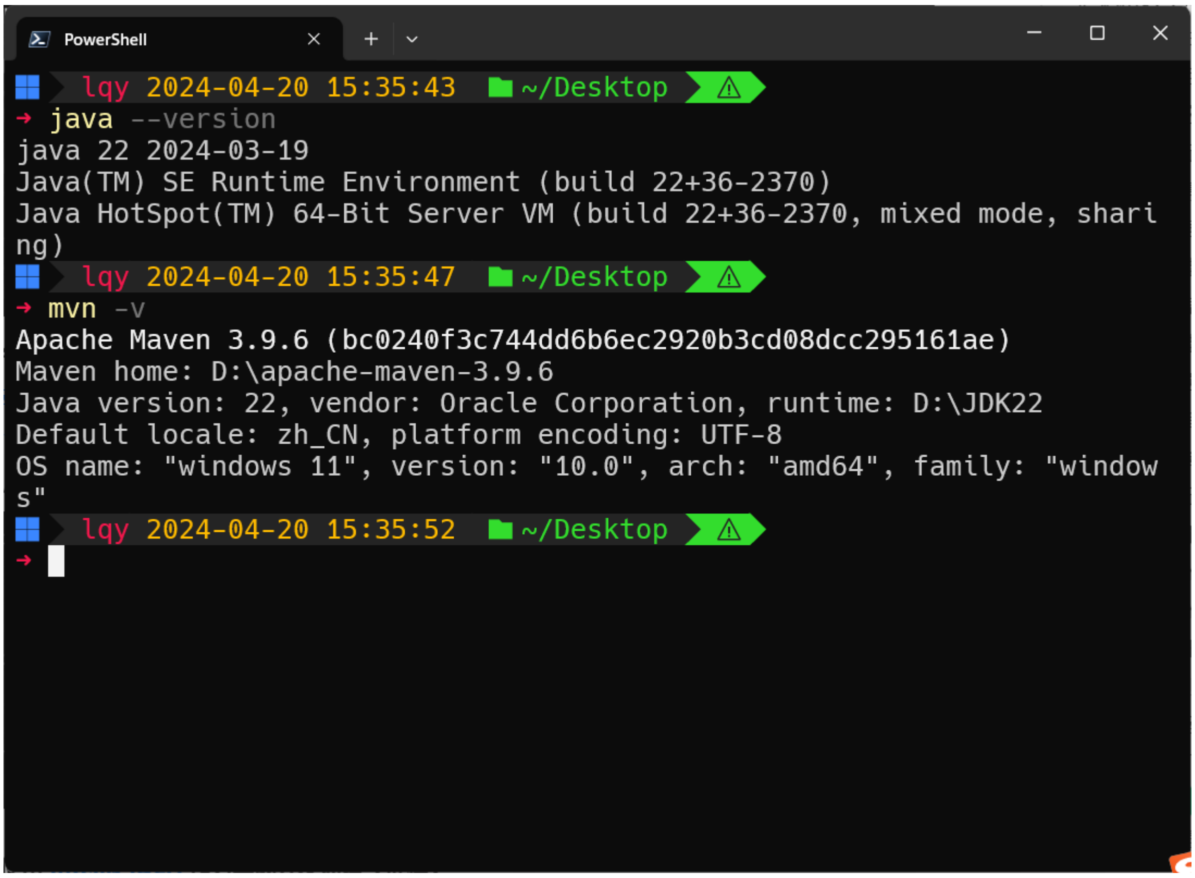
实现功能

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
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

功能	描述
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

实验环境

本次实验环境采用的是VSCode + JDK + Maven


```
 lqy 2024-04-20 15:35:43  ~/Desktop  ⚠
➜ java --version
java 22 2024-03-19
Java(TM) SE Runtime Environment (build 22+36-2370)
Java HotSpot(TM) 64-Bit Server VM (build 22+36-2370, mixed mode, shari
ng)
 lqy 2024-04-20 15:35:47  ~/Desktop  ⚠
➜ mvn -v
Apache Maven 3.9.6 (bc0240f3c744dd6b6ec2920b3cd08dcc295161ae)
Maven home: D:\apache-maven-3.9.6
Java version: 22, vendor: Oracle Corporation, runtime: D:\JDK22
Default locale: zh_CN, platform encoding: UTF-8
OS name: "windows 11", version: "10.0", arch: "amd64", family: "window
s"
 lqy 2024-04-20 15:35:52  ~/Desktop  ⚠
➜
```

本次实验采用框架ZJUGit

各系统模块设计

所有的函数声明都在`librarymanagementsystem/src/main/java/LibraryManagementSystem.java`中

所有的函数实现都在`librarymanagementsystem/src/main/java/LibraryManagementSystemImpl.java`中

以下是每个**待实现**模块的具体内容

`storeBook`

基本功能

图书入库模块，实现对一本书的入库

设计要点

1. 判断图书是否已经存在库中
2. 图书入库
3. 返回数据库的自增项`book_id`

模块代码

```java
@Override
    public ApiResult storeBook(Book book) {
        Connection conn = connector.getConn();
        try {
            // Check if the book is already in the database.
            {
                String sqlString = "select * from `book` where `category` = ? and `title` = ? and `press` = ? and `publish_year` = ? and `author` = ?";
                PreparedStatement stmt = conn.prepareStatement(sqlString);
                stmt.setString(1, book.getCategory());
                stmt.setString(2, book.getTitle());
                stmt.setString(3, book.getPress());
                stmt.setInt(4, book.getPublishYear());
                stmt.setString(5, book.getAuthor());
                ResultSet rs = stmt.executeQuery();
                if (rs.next()) {
                    return new ApiResult(false, "Book already exists.\n");
                }
                rs.close();
                stmt.close();
            }
            // Insert the book into the database.
            String sqlString = "insert into `book` (`category`, `title`, `press`, `publish_year`, `author`, `price`, `stock`) values (?, ?, ?, ?, ?, ?, ?)";
            PreparedStatement stmt = conn.prepareStatement(sqlString);
            stmt.setString(1, book.getCategory());
            stmt.setString(2, book.getTitle());
            stmt.setString(3, book.getPress());
            stmt.setInt(4, book.getPublishYear());
            stmt.setString(5, book.getAuthor());
            stmt.setDouble(6, book.getPrice());
            stmt.setInt(7, book.getStock());
            stmt.executeUpdate();
            // Get the book id.
            sqlString = "select `book_id` from `book` where `category` = ? and `title` = ? and `press` = ? and `publish_year` = ? and `author` = ?";
            stmt = conn.prepareStatement(sqlString);
            stmt.setString(1, book.getCategory());
            stmt.setString(2, book.getTitle());
            stmt.setString(3, book.getPress());
            stmt.setInt(4, book.getPublishYear());
            stmt.setString(5, book.getAuthor());
            ResultSet rs = stmt.executeQuery();
            rs.next();
            book.setBookId(rs.getInt("book_id"));
            rs.close();
            commit(conn);
            rs.close();
            stmt.close();
            return new ApiResult(true, "Insert book successfully with book_id = " + book.getBookId() + "\n");
```

```java
        } catch (Exception e) {
            rollback(conn);
            return new ApiResult(false, e.getMessage());
        }
    }
```

`incBookStock`

基本功能

增加或减少图书库存，保证最终图书库存非负

设计要点

1. 判断图书是否在库中
2. 判断修改后`stock`是否非负
3. `update`

模块代码

```java
@Override
    public ApiResult incBookStock(int bookId, int deltaStock) {
        Connection conn = connector.getConn();
        try {
            // Check if the book stock is larger than the deltaStock.
            {
                String sqlString = "SELECT `stock` FROM book WHERE book_id = ?";
                PreparedStatement stmt = conn.prepareStatement(sqlString);
                stmt.setInt(1, bookId);
                ResultSet rs = stmt.executeQuery();
                if (!rs.next()) {
                    return new ApiResult(false, "Book with book_id = " + bookId + " not found in the database.\n");
                }
                if (rs.getInt("stock") + deltaStock < 0) {
                    return new ApiResult(false, "Book stock less than 0.\n");
                }
                rs.close();
                stmt.close();
            }
            // Update the book stock.
            String sqlString = "update `book` set `stock` = `stock` + ? where `book_id` = ?";
            PreparedStatement stmt = conn.prepareStatement(sqlString);
            stmt.setInt(1, deltaStock);
            stmt.setInt(2, bookId);
            stmt.executeUpdate();
            commit(conn);
            stmt.close();
            return new ApiResult(true, "Change book stock successfully.\n");
        } catch (Exception e) {
            rollback(conn);
            return new ApiResult(false, e.getMessage());
        }
    }
```

`storeBook`

基本功能

实现批量图书入库，如果一本入库失败则全部回滚

设计要点

1. 对图书列表进行循环，分别判定每本书是否已经存在
2. 对每本书进行插入，如果一本失败则全部回滚

模块代码

```
@Override
    public ApiResult storeBook(List<Book> books) {
        Connection conn = connector.getConn();
        try {
            for (Book book : books) {
                // Check if the book is already in the database.
                String sqlString = "select * from `book` where `category` = ? and `title` = ? and `press` = ? and `publish_year` = ? and `author` = ?";
                PreparedStatement stmt = conn.prepareStatement(sqlString);
                stmt.setString(1, book.getCategory());
                stmt.setString(2, book.getTitle());
                stmt.setString(3, book.getPress());
                stmt.setInt(4, book.getPublishYear());
                stmt.setString(5, book.getAuthor());
                ResultSet rs = stmt.executeQuery();
                if (rs.next()) {
                    rollback(conn);
                    return new ApiResult(false, "Book already exists.\n");
                }
                // Insert the book into the database.
                sqlString = "insert into `book` (`category`, `title`, `press`, `publish_year`, `author`, `price`, `stock`) values (?, ?, ?, ?, ?, ?, ?)";
                stmt = conn.prepareStatement(sqlString);
                stmt.setString(1, book.getCategory());
                stmt.setString(2, book.getTitle());
                stmt.setString(3, book.getPress());
                stmt.setInt(4, book.getPublishYear());
                stmt.setString(5, book.getAuthor());
                stmt.setDouble(6, book.getPrice());
                stmt.setInt(7, book.getStock());
                stmt.executeUpdate();
                // Get the book id.
                sqlString = "select `book_id` from `book` where `category` = ? and `title` = ? and `press` = ? and `publish_year` = ? and `author` = ?";
                stmt = conn.prepareStatement(sqlString);
                stmt.setString(1, book.getCategory());
                stmt.setString(2, book.getTitle());
                stmt.setString(3, book.getPress());
                stmt.setInt(4, book.getPublishYear());
                stmt.setString(5, book.getAuthor());
```

```java
                rs = stmt.executeQuery();
                rs.next();
                book.setBookId(rs.getInt("book_id"));
            }
            commit(conn);
            return new ApiResult(true, "Insert " + books.size()
+ " books successfully.\n");
        } catch (Exception e) {
            rollback(conn);
            return new ApiResult(false, e.getMessage());
        }
    }
```

`removeBook`

基本功能

移除书本，如果书被借走尚未归还则无法移除

设计要点

1. 判定图书是否在库中
2. 判定图书是否被借走未还
3. 移除

模块代码

```java
@Override
    public ApiResult removeBook(int bookId) {
        Connection conn = connector.getConn();
        try {
            // Check if the book is in the database.
            {
                String sqlString = "select * from `book` where
`book_id` = ?";
                PreparedStatement stmt =
conn.prepareStatement(sqlString);
                stmt.setInt(1, bookId);
                ResultSet rs = stmt.executeQuery();
                if (!rs.next()) {
                    return new ApiResult(false, "Book with book_id = " + bookId + " not found in the database.\n");
                }
                rs.close();
                stmt.close();
            }
            // Check if the book is borrowed.
            {
                String sqlString = "select * from `borrow` where `book_id` = ? and `return_time` = 0";
                PreparedStatement stmt = conn.prepareStatement(sqlString);
                stmt.setInt(1, bookId);
                ResultSet rs = stmt.executeQuery();
                if (rs.next()) {
                    return new ApiResult(false, "Cannot remove book with book_id = " + bookId + ", it is borrowed.\n");
                }
                rs.close();
                stmt.close();
            }
            // Delete the book from the database.
            String sqlString = "delete from `book` where book_id = ?";
            PreparedStatement stmt = conn.prepareStatement(sqlString);
            stmt.setInt(1, bookId);
            stmt.executeUpdate();
            commit(conn);
            stmt.close();
            return new ApiResult(true, "Successfully remove book with book_id = " + bookId + ".\n");
        } catch (Exception e) {
            rollback(conn);
            return new ApiResult(false, e.getMessage());
        }
    }
```

`modifyBookInfo`

基本功能

修改图书信息，`book_id`和`stock`除外

设计要点

1. 判断图书是否在库中
2. 更新数据

模块代码

```java
@Override
    public ApiResult modifyBookInfo(Book book) {
        Connection conn = connector.getConn();
        try {
            // Check if the book is in the database.
            {
                String sqlString = "select * from `book` where `book_id` = ?";
                PreparedStatement stmt = conn.prepareStatement(sqlString);
                stmt.setInt(1, book.getBookId());
                ResultSet rs = stmt.executeQuery();
                if (!rs.next()) {
                    return new ApiResult(false, "Book with book_id = " + book.getBookId() + " not found in the database.\n");
                }
                rs.close();
                stmt.close();
            }
            // Update the book information.
            String sqlString = "update `book` set `category` = ?, `title` = ?, `press` = ?, `publish_year` = ?, `author` = ?, `price` = ? where `book_id` = ?";
            PreparedStatement stmt = conn.prepareStatement(sqlString);
            stmt.setString(1, book.getCategory());
            stmt.setString(2, book.getTitle());
```

```
                stmt.setString(3, book.getPress());
                stmt.setInt(4, book.getPublishYear());
                stmt.setString(5, book.getAuthor());
                stmt.setDouble(6, book.getPrice());
                stmt.setInt(7, book.getBookId());
                stmt.executeUpdate();
                commit(conn);
                stmt.close();
                return new ApiResult(true, "Successfully modify book information with book_id = " + book.getBookId() + "\n");
            } catch (Exception e) {
                rollback(conn);
                return new ApiResult(false, e.getMessage());
            }
        }
```

queryBook

基本功能

查询图书信息，实现

1. 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
2. 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
3. 如果两条记录排序条件的值相等，则按book_id升序排序

设计要点

1. 根据 `BookQueryConditions` 的条件获取查询条件
2. 根据设计需求对模糊查询、精确查询、范围查询进行SQL语句翻译
3. 转化为JDBC代码并执行查询
4. 将查询结果合并到 `BookQueryResults` 中并完成排序

模块代码

```
@Override
    public ApiResult queryBook(BookQueryConditions conditions) {
        Connection conn = connector.getConn();
```

```java
        try {
            String sqlString = "select * from `book`";
            String queryString = "";
            boolean isQueryCondition = false;
            // Get query conditions.
            if (conditions.getCategory() != null) {
                isQueryCondition = true;
                queryString = "`category` = '" + conditions.getCategory() + "'";
            }
            if (conditions.getTitle() != null) {
                if (isQueryCondition)
                    queryString += " and ";
                else
                    isQueryCondition = true;
                queryString += "`title` like '%" + conditions.getTitle() + "%'";
            }
            if (conditions.getPress() != null) {
                if (isQueryCondition)
                    queryString += " and ";
                else
                    isQueryCondition = true;
                queryString += "`press` like '%" + conditions.getPress() + "%'";
            }
            if (conditions.getMinPublishYear() != null) {
                if (isQueryCondition)
                    queryString += " and ";
                else
                    isQueryCondition = true;
                queryString += "`publish_year` >= " + Integer.toString(conditions.getMinPublishYear());
            }
            if (conditions.getMaxPublishYear() != null) {
                if (isQueryCondition)
                    queryString += " and ";
                else
                    isQueryCondition = true;
                queryString += "`publish_year` <= " + Integer.toString(conditions.getMaxPublishYear());
            }
            if (conditions.getAuthor() != null) {
                if (isQueryCondition)
                    queryString += " and ";
                else
                    isQueryCondition = true;
                queryString += "`author` like '%" + conditions.getAuthor() + "%'";
            }
            if (conditions.getMinPrice() != null) {
                if (isQueryCondition)
                    queryString += " and ";
                else
                    isQueryCondition = true;
                queryString += "`price` >= " + Double.toString(conditions.getMinPrice());
            }
            if (conditions.getMaxPrice() != null) {
                if (isQueryCondition)
                    queryString += " and ";
                else
                    isQueryCondition = true;
                queryString += "`price` <= " + Double.toString(conditions.getMaxPrice());
            }
            if (isQueryCondition)
                sqlString += " where " + queryString;
            Statement stmt = conn.createStatement();
            // Make query.
            ResultSet rs = stmt.executeQuery(sqlString);
            // Record results.
            List<Book> books = new ArrayList<>();
            while (rs.next()) {
                Book thisBook = new Book(
                    rs.getString("category"),
                    rs.getString("title"),
                    rs.getString("press"),
                    rs.getInt("publish_year"),
                    rs.getString("author"),
                    rs.getDouble("price"),
                    rs.getInt("stock"));
                thisBook.setBookId(rs.getInt("book_id"));
                books.add(thisBook);
            }
            String msg = "";
            if (!books.isEmpty()) {
                // Sort.
                if (conditions.getSortOrder() == SortOrder.DESC)
                    books.sort(conditions.getSortBy().getComparator().reversed().thenComparing(Book.SortColumn.BOOK_ID.getComparator()));
                else
                    books.sort(conditions.getSortBy().getComparator().thenComparing(Book.SortColumn.BOOK_ID.getComparator()));
                for (Book book : books)
                    msg += book.toString() + "\n";
            } else {
                msg = "No books founded for the given conditions.\n";
            }
            BookQueryResults results = new BookQueryResults(books);
            return new ApiResult(true, msg, results);
        } catch (Exception e) {
            return new ApiResult(false, e.getMessage());
        }
    }
```

`borrowBook`

基本功能

借书

1. 如果用户此前已经借过这本书尚未归还则无法借书
2. 并发控制

设计要点

1. 判断书是否在库中
2. 判断书是否库存大于0
3. 判断卡是否在库中
4. 判断是否借过未还
5. 更新库存，添加借书记录

模块设计

```java
@Override
    public ApiResult borrowBook(Borrow borrow) {
        Connection conn = connector.getConn();
        try {
            // Check if the book is in the database.
            {
                PreparedStatement stmt = conn.prepareStatement("select `stock` from `book` where `book_id` = ?");
                stmt.setInt(1, borrow.getBookId());
                ResultSet rs = stmt.executeQuery();
                if (!rs.next())
                    return new ApiResult(false, "Book with book_id = " + borrow.getBookId() + " not found in the database.\n");
                if (rs.getInt("stock") <= 0)
                    return new ApiResult(false, "Book with book_id = " + borrow.getBookId() + " is not available.\n");
                rs.close();
                stmt.close();
            }
            // Check if the card is in the database.
            {
                PreparedStatement stmt = conn.prepareStatement("select * from `card` where `card_id` = ?");
                stmt.setInt(1, borrow.getCardId());
                ResultSet rs = stmt.executeQuery();
                if (!rs.next())
```

```java
          return new ApiResult(false, "Card with card_id = " + borrow.getCardId() + " not found in the database.\n");
        rs.close();
        stmt.close();
      }
      // Check if the book is borrowed.
      {
        PreparedStatement stmt = conn.prepareStatement("select * from `borrow` where `book_id` = ? and `card_id` = ? and `return_time` = 0");
        stmt.setInt(1, borrow.getBookId());
        stmt.setInt(2, borrow.getCardId());
        ResultSet rs = stmt.executeQuery();
        if (rs.next())
          return new ApiResult(false, "The book with book_id = " + borrow.getBookId() + " is borrowed and not returned yet.\n");
        rs.close();
        stmt.close();
      }
      // Insert the borrow record.
      Statement stmt = conn.createStatement();
      String sqlString = "insert into `borrow` (`card_id`, `book_id`, `borrow_time`) values (" +
        Integer.toString(borrow.getCardId()) + ", " +
        Integer.toString(borrow.getBookId()) + ", " +
        Long.toString(borrow.getBorrowTime()) + ")";
      stmt.addBatch(sqlString);
      // Update the book stock.
      sqlString = "update `book` set `stock` = `stock` - 1 where `book_id` = " + borrow.getBookId();
      stmt.addBatch(sqlString);
      stmt.executeBatch();
      //commit(conn);
      return new ApiResult(true, "Successfully borrow book with book_id = " + borrow.getBookId() + " by card_id = " + borrow.getCardId() + " at date: " + borrow.getBorrowTime() + ".\n");
    } catch (Exception e) {
      rollback(conn);
```

```
            return new ApiResult(false, e.getMessage());
        }
    }
```

`returnBook`

基本功能

还书

设计要点

1. 判断书是否在库中
2. 判断卡是否在库中
3. 判断书是否被这张卡借了且未还

模块代码

```
@Override
    public ApiResult returnBook(Borrow borrow) {
        Connection conn = connector.getConn();
        try {
            // Check if the book is in the database.
            {
                String sqlString = "select * from `book` where `book_id` = ?";
                PreparedStatement stmt = conn.prepareStatement(sqlString);
                stmt.setInt(1, borrow.getBookId());
                ResultSet rs = stmt.executeQuery();
                if (!rs.next())
                    return new ApiResult(false, "Book not found in the database.\n");
                rs.close();
                stmt.close();
            }
            // Check if the card is in the database.
            {
                String sqlString = "select * from `card` where `card_id` = ?";
                PreparedStatement stmt =
conn.prepareStatement(sqlString);
                stmt.setInt(1, borrow.getCardId());
                ResultSet rs = stmt.executeQuery();
                if (!rs.next())
                    return new ApiResult(false, "Card not found
in the database.\n");
                rs.close();
                stmt.close();
            }
            // Check if the book is borrowed by the card.
            {
                String sqlString = "select `return_time` from
`borrow` where `card_id` = ? and `book_id` = ? and
`borrow_time` = ?";
                PreparedStatement stmt =
conn.prepareStatement(sqlString);
                stmt.setInt(1, borrow.getCardId());
                stmt.setInt(2, borrow.getBookId());
                stmt.setLong(3, borrow.getBorrowTime());
                ResultSet rs = stmt.executeQuery();
                if (!rs.next()) {
                    // The book is not borrowed by the card.
                    return new ApiResult(false, "The book is
not borrowed by the card.\n");
                }
                // The book is already returned.
                if (rs.getLong("return_time") != 0) {
                    return new ApiResult(false, "The book is
already returned.\n");
                }
                rs.close();
                stmt.close();
            }
            // Update the book stock.
            Statement stmt = conn.createStatement();
            String sqlString = "update `book` set `stock` =
`stock` + 1 where `book_id` = " + borrow.getBookId();
            stmt.addBatch(sqlString);
            // Update the borrow record.
            sqlString = "update `borrow` set `return_time` = " + Long.toString(borrow.getReturnTime()) +
                    " where `card_id` = " + Integer.toString(borrow.getCardId()) +
                    " and `book_id` = " + Integer.toString(borrow.getBookId()) +
                    " and `borrow_time` = " + Long.toString(borrow.getBorrowTime());
            stmt.addBatch(sqlString);
            stmt.executeBatch();
            commit(conn);
            stmt.close();
            return new ApiResult(true, "Successfully return book with book_id = " + borrow.getBookId() + " by card_id = " + borrow.getCardId() + "at date: " + borrow.getReturnTime() + ".\n");
        } catch (Exception e) {
            rollback(conn);
            return new ApiResult(false, e.getMessage());
        }
    }
```

`showBorrowHistory`

基本功能

查询一张卡的所有借书记录，按照借书时间递减、书号递增的方式排序

设计要点

1. 判断卡是否在库中
2. 查询对应借书记录
3. 连接查询每条借书记录的书号对应的书信息
4. 查询结果合并为 `BorrowHistories` 并返回

模块代码

```
@Override
    public ApiResult showBorrowHistory(int cardId) {
        Connection conn = connector.getConn();
        try {
            // Check if the card is in the database.
            {
                String sqlString = "select * from `card` where `card_id` = ?";
                PreparedStatement stmt = conn.prepareStatement(sqlString);
                stmt.setInt(1, cardId);
                ResultSet rs = stmt.executeQuery();
                if (!rs.next())
                    return new ApiResult(false, "Card not found in the database.\n");
                rs.close();
                stmt.close();
            }
            // Get the borrow history.
            String sqlString = "select `borrow`.`book_id`, " +
                    "`book`.`category`, `book`.`title`, `book`.`press`, `book`.`publish_year`, `book`.`author`, `book`.`price`, `book`.`stock`, " +
                    "`borrow`.`borrow_time`, `borrow`.`return_time` " +
                    "from `borrow` join `book` on `book`.`book_id` = `borrow`.`book_id` and `borrow`.`card_id` = ?";
            PreparedStatement stmt = conn.prepareStatement(sqlString);
            stmt.setInt(1, cardId);
            ResultSet rs = stmt.executeQuery();
            List<BorrowHistories.Item> items = new ArrayList<>();
            while (rs.next()) {
                Book thisBook = new Book(
                    rs.getString("category"),
                    rs.getString("title"),
                    rs.getString("press"),
                    rs.getInt("publish_year"),
                    rs.getString("author"),
                    rs.getDouble("price"),
                    rs.getInt("stock"));
                thisBook.setBookId(rs.getInt("book_id"));
```

```java
                Borrow thisBorrow = new Borrow(cardId, rs.getInt("book_id"));
                thisBorrow.setBorrowTime(rs.getLong("borrow_time"));
                thisBorrow.setReturnTime(rs.getLong("return_time"));
                BorrowHistories.Item thisItem = new BorrowHistories.Item(cardId, thisBook, thisBorrow);
                items.add(thisItem);
            }
            rs.close();
            stmt.close();
            // Record results.
            String msg = "";
            if (!items.isEmpty()) {
                // Sort.
                items.sort(Comparator.comparingLong(BorrowHistories.Item::getBorrowTime).reversed().thenComparingInt(BorrowHistories.Item::getBookId));
                for (BorrowHistories.Item item : items)
                    msg += item.toString() + "\n";
            } else {
                msg = "No borrow history found for the card with card_id = " + cardId + ".\n";
            }
            BorrowHistories histories = new BorrowHistories(items);
            return new ApiResult(true, msg, histories);
        } catch (Exception e) {
            return new ApiResult(false, e.getMessage());
        }
    }
```

`registerCard`

基本功能

注册卡

设计要点

1. 判断卡是否已经存在
2. 插入卡

模块代码

```java
@Override
    public ApiResult registerCard(Card card) {
        Connection conn = connector.getConn();
        try {
            // Check if the card is already in the database.
            {
                String sqlString = "select * from `card` where `name` = ? and `department` = ? and `type` = ?";
                PreparedStatement stmt = conn.prepareStatement(sqlString);
                stmt.setString(1, card.getName());
                stmt.setString(2, card.getDepartment());
                stmt.setString(3, card.getType().getStr());
                ResultSet rs = stmt.executeQuery();
                if (rs.next())
                    return new ApiResult(false, "Card already exists.");
                rs.close();
                stmt.close();
            }
            // Insert the card into the database.
            String sqlString = "insert into `card` (`name`, `department`, `type`) values (?, ?, ?)";
            PreparedStatement stmt = conn.prepareStatement(sqlString);
            stmt.setString(1, card.getName());
            stmt.setString(2, card.getDepartment());
            stmt.setString(3, card.getType().getStr());
            stmt.executeUpdate();
            sqlString = "select `card_id` from `card` where `name` = ? and `department` = ? and `type` = ?";
            stmt = conn.prepareStatement(sqlString);
            stmt.setString(1, card.getName());
            stmt.setString(2, card.getDepartment());
            stmt.setString(3, card.getType().getStr());
            ResultSet rs = stmt.executeQuery();
            rs.next();
            card.setCardId(rs.getInt("card_id"));
            commit(conn);
            rs.close();
            stmt.close();
            return new ApiResult(true, "Successfully register card with card_id: " + card.getCardId() + ".\n", card);
        } catch (Exception e) {
            rollback(conn);
            return new ApiResult(false, e.getMessage());
        }
    }
```

`removeCard`

基本功能

移除卡，如果该借书证还有未归还的图书，那么删除操作将失败

设计要点

1. 判断卡是否在库中
2. 判断是否有未还图书
3. 删除

模块代码

```
@Override
    public ApiResult removeCard(int cardId) {
        Connection conn = connector.getConn();
        try {
            // Check if the card is in the database.
            {
```

```
        String sqlString = "select * from `card` where `card_id` = ?";
        PreparedStatement stmt = conn.prepareStatement(sqlString);
        stmt.setInt(1, cardId);
        ResultSet rs = stmt.executeQuery();
        if (!rs.next())
          return new ApiResult(false, "Card not exists.");
        rs.close();
        stmt.close();
      }
      // Check if the card is borrowed by any book.
      {
        String sqlString = "select * from `borrow` where `card_id` = ? and `return_time` = 0";
        PreparedStatement stmt = conn.prepareStatement(sqlString);
        stmt.setInt(1, cardId);
        ResultSet rs = stmt.executeQuery();
        if (rs.next())
          return new ApiResult(false, "The card is borrowed by a book, cannot remove it.\n");
        rs.close();
        stmt.close();
      }
      // Delete the card from the database.
      String sqlString = "delete from `card` where `card_id` = ?";
      PreparedStatement stmt = conn.prepareStatement(sqlString);
      stmt.setInt(1, cardId);
      stmt.executeUpdate();
      commit(conn);
      stmt.close();
      return new ApiResult(true, "Successfully remove card with card_id: " + Integer.toString(cardId) + ".\n");
    } catch (Exception e) {
      rollback(conn);
      return new ApiResult(false, e.getMessage());
    }
    }
```

`showCards`

基本功能

列出所有借书证

设计要点

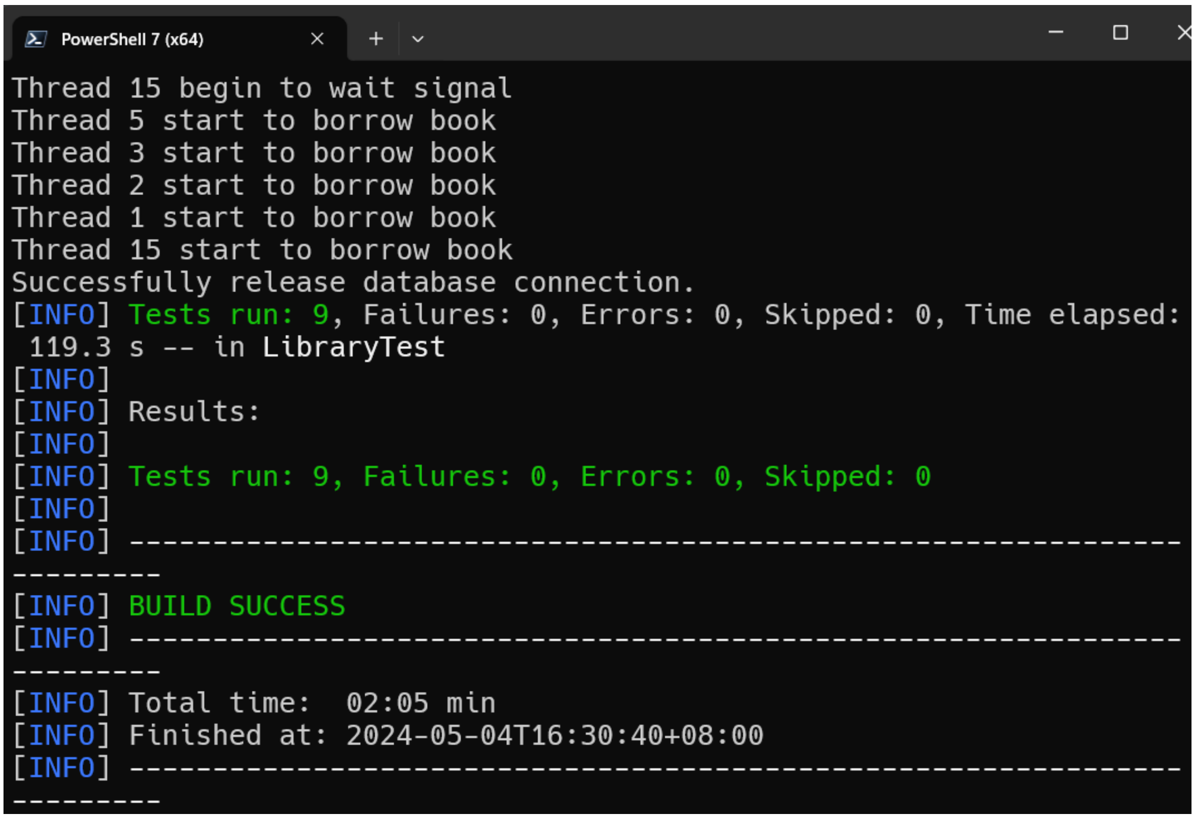
1. 查询
2. 排序
3. 返回`CardList`

模块代码

```java
@Override
    public ApiResult showCards() {
        Connection conn = connector.getConn();
        try {
            String sqlString = "select * from `card`";
            Statement stmt = conn.createStatement();
            ResultSet rs = stmt.executeQuery(sqlString);
            String msg = "";
            List<Card> cards = new ArrayList<>();
            while (rs.next()) {
                Card card = new Card(
                    rs.getInt("card_id"),
                    rs.getString("name"),
                    rs.getString("department"),
 Card.CardType.values(rs.getString("type")));
                msg += card.toString() + "\n";
                cards.add(card);
            }
            if (cards.isEmpty()) {
                msg = "No cards in this library.\n";
            } else {
                // Sort.
cards.sort(Comparator.comparing(Card::getCardId));
            }
            CardList cardList = new CardList(cards);
            rs.close();
            stmt.close();
            return new ApiResult(true, msg, cardList);
        } catch (Exception e) {
            return new ApiResult(false, e.getMessage());
        }
    }
```

系统验证测试

利用测试样例进行测试，最终结果为全部通过



```
Thread 15 begin to wait signal
Thread 5 start to borrow book
Thread 3 start to borrow book
Thread 2 start to borrow book
Thread 1 start to borrow book
Thread 15 start to borrow book
Successfully release database connection.
[INFO] Tests run: 9, Failures: 0, Errors: 0, Skipped: 0, Time elapsed:
 119.3 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:  02:05 min
[INFO] Finished at: 2024-05-04T16:30:40+08:00
[INFO] ----------------------------------------------------------------
---------
```

注：前端部分未实现

遇到的问题及解决方法

在完成整个图书管理系统的过程中，遇到了诸多问题

Java菜鸟

之前从来没有接触过Java，也没有写过这么大的项目，因此Java的初学与速通花费了一定时间

对高级的Java语法也花了很长时间进行理解与学习

总体设计

考虑到这个图书管理系统的每个函数在实现上其实都有一部分的重叠之处，比如针对书的操作都是基于书的查询，针对卡的操作都要基于对卡的查询，因此最开始设计了两个私有变量以及函数，分别为书的列表的卡的列表，并设计函数 `selectAllBook` 和 `selectAllCards`，希望通过这个操作将所有的信息搬到内存进行操作而加快访问速度

但是这样的做法会导致相当大的麻烦，比如数据量极大时对内存的开销是巨大的，没有很好发挥数据库服务端的功能等等，因此后期重新进行设计，把这部分的内容去除了

图书查询

图书查询不仅仅包含精确查询，还要实现模糊查询

专门重新学习了查询的语法结构，并对查询进行了调整

并发控制

这里注意到需要通过一个测试样例为 `parallelBorrowBookTest`，即并行借书

注意到这里是多线程借书，最终只有一个线程能够借书成功，但是调试的时候出现了各种问题，要么是16个线程全部借到书要么0个线程借到书

改到后期更夸张是出现3个线程借到书，完全想不到的问题 😠

后来不断调试，发现问题出在 `commit` 上

复杂关系

最开始的时候，本着由易到难的原则，我先选择了完成卡的模块，因为这里不涉及图书查询，内容更加少

但是后来发现`removeCard`这里需要完成对借书模块的实现，就需要书的模块，而后还需要借阅记录的模块，因此模块之间的关系是非常复杂的，也花费了很长时间才完全搞懂

需求分析有误

参考语雀文档中提出的需求，有部分需求是我没有完全明白意思因此而出错的，比如查询模块，最开始以为`BookQuereConditions`的`SortBy`字段是查询的字段，因此功能实现出错，最后需要对着测试样例的报错重新修改

小组合作

最开始我以为图书管理系统是最终夏季的大程，因此并没有马上开始着手去做，也叫了一位同学一起组队完成

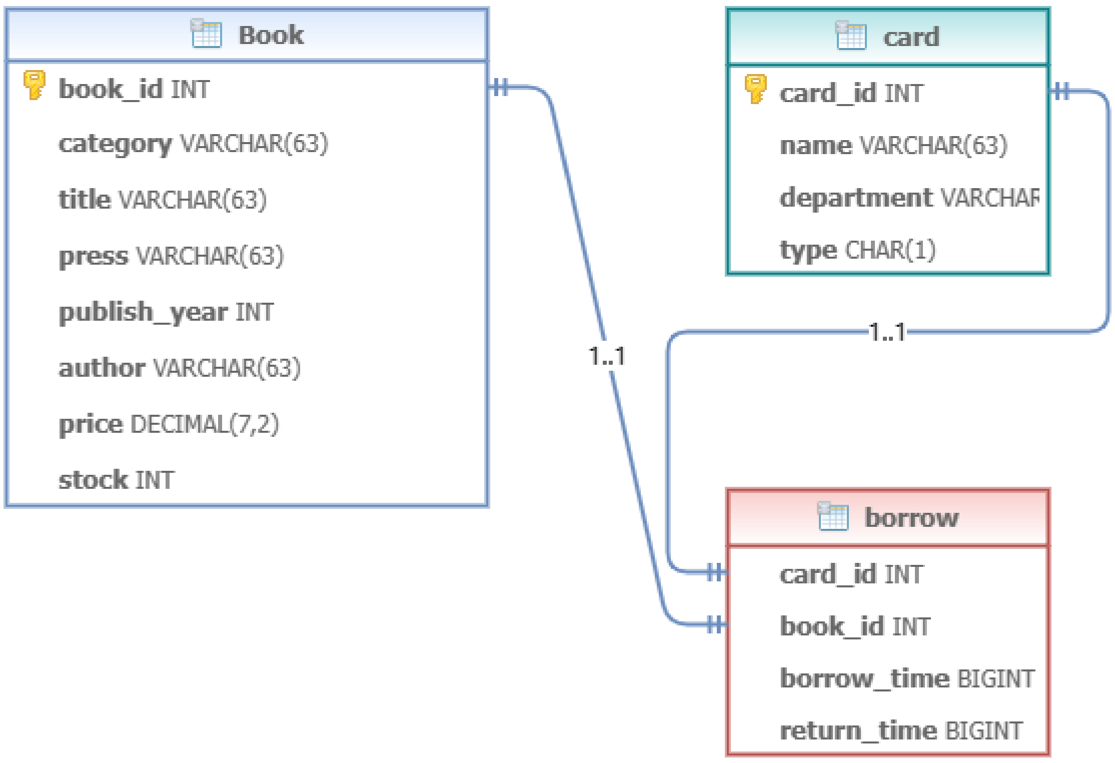
结果经过不断询问才得知这不是夏季大程，ddl迫近我不得不匆匆开始

但是小组组员摆烂行为让我非常不爽，在验收期限前一天晚上才开始参与

最终我们协商后提出solo完成自己的部分

思考题

E-R图



SQL注入

SQL注入通常来说就是使用不同寻常的查询条件来获取数据库其他信息的访问

> 例：`select * from user where id=();`，在括号填充查询条件
>
> 如果用户使用SQL注入如`select * from user where id=(0 or 1=1);`则会查出所有的信息，超过了用户允许的查询范围，会导致信息泄露

对于SQL注入的攻击防御，应该对查询条件进行一定的审查

并发控制

注意到这里使用的引擎为innoDB，默认的隔离级别为RR

对于RR来说，可以避免产生脏读与不可重复读的现象，但无法避免产生幻读，即实例所示，当A启动事务时对表进行更新，但尚未提交事务，此时B启动事务，对表的查询是A对表进行更新前的数据

对应策略：对表进行上锁，使得每次写只能由一个线程进行，而将其他线程通过锁挂起