图书管理系统数据库实验报告

作者姓名

2025 年 4 月 14 日

1 实验目的

本实验旨在设计并实现一个图书管理系统的数据库，该系统能够有效地管理图书、借阅者卡片以及借阅记录，并通过一系列的测试用例来验证系统的正确性和性能。

2 项目文件结构

本项目采用Maven项目结构，主要包含以下几个模块：

- `src/main/java`: 包含Java源代码。
- `src/main/resources`: 包含配置文件，如数据库连接信息。
- `src/test/java`: 包含测试代码。
- `librarymanagementsystem-frontend`: 前端代码。
- `report`: 实验报告及相关文档。

在 `src/main/java` 目录下，主要模块如下：

- `config`: 配置文件，如跨域配置。
- `entities`: 实体类，如书籍、借阅者卡片等。
- `queries`: 查询条件和结果类。
- `library.system`: 核心系统类和接口。
- `library.system.controller`: 控制器层，负责处理前端请求。
- `utils`: 工具类。

3 文件作用说明

3.1 config/CorsConfig.java

文件定义了跨域资源共享（CORS）的配置，以允许前端应用从不同的源访问API，确保前后端协同工作时不会遇到跨域问题。在实验中最初发现前后端不能完成正常通信。前端发送的请求会被后端404拒绝，原因是没有配置spring boot的跨域规则。

功能说明：

- 使用`@Configuration`注解标记为配置类。
- 通过`@Bean`注解定义了一个`CorsFilter` Bean，用于处理跨域请求。
- 配置允许所有来源（`"*"`）、所有请求头（`"*"`）和所有HTTP方法（`"*"`），并允许凭据（`setAllowCredentials(true)`）。

代码关键点：

- `CorsConfiguration`用于定义跨域规则。
- `UrlBasedCorsConfigurationSource`用于将跨域规则绑定到特定的URL路径（`"**"`表示所有路径）。
- `CorsFilter`用于实际拦截请求并应用跨域规则。

设计思路：

- 确保前后端分离时，前端可以无限制地访问后端API。
- 在开发和测试阶段，允许所有来源的访问以简化调试。

Listing 1: `config/CorsConfig.java`

```
1 package config;
2
3 import org.springframework.context.annotation.Bean;
4 import org.springframework.context.annotation.Configuration;
5 import org.springframework.web.cors.CorsConfiguration;
6 import org.springframework.web.cors.UrlBasedCorsConfigurationSource;
7 import org.springframework.web.filter.CorsFilter;
8
9 @Configuration
```

```java
public class CorsConfig {
    private static final long MAX_AGE = 24 * 60 * 60; // 1 day

    @Bean
    public CorsFilter corsFilter() {
        UrlBasedCorsConfigurationSource source = new
            UrlBasedCorsConfigurationSource();
        CorsConfiguration config = new CorsConfiguration();
        config.addAllowedHeader("*");
        config.addAllowedMethod("*");
        config.addAllowedOrigin("http://localhost:5173");
        config.setMaxAge(MAX_AGE);
        source.registerCorsConfiguration("/**", config);
        return new CorsFilter(source);
    }
}
```

3.2 src/main/java/library/system/LibraryManagementSystem.java

文件定义了图书管理系统的接口，声明了系统需要实现的各种功能，如注册图书、借书、还书等。通过PreparedStatement预编译SQL语句，避免SQL注入攻击。使用stmt.executeUpdate()执行SQL语句，与数据库交互。 **功能说明：**

- 定义了系统的核心功能接口，包括：
  - storeBook(Book book)：注册新图书。
  - borrowBook(Borrow borrow)：借书。
  - returnBook(Borrow borrow)：还书。
  - storeBook(List<Book> books)：批量注册图书。
  - queryBook(BookQueryConditions conditions)：查询图书。

代码关键点：

- 使用ApiResult作为返回值，统一处理成功和失败的响应。
- BookQueryResults用于封装查询结果，包含状态、消息和图书列表。

设计思路：

- 通过接口定义系统功能，便于后续实现和扩展。
- 使用统一的返回类型（ApiResult和BookQueryResults）简化前端处理逻辑。

Listing 2: library.system/LibraryManagementSystem.java

```java
public interface LibraryManagementSystem {

    ApiResult storeBook(Book book);
    ApiResult incBookStock(int bookId, int deltaStock);
    ApiResult storeBook(List<Book> books);
    ApiResult removeBook(int booApiResultkId);
    ApiResult queryBook(BookQueryConditions conditions);
    ApiResult borrowBook(Borrow borrow);
    ApiResult returnBook(Borrow borrow);
    ApiResult showBorrowHistory(int cardId);
    ApiResult registerCard(Card card);
    ApiResult removeCard(int cardId);
    ApiResult showCards();
    ApiResult modifyCardInfo(Card card);
}
```

3.3 src/main/java/library/system/LibraryManagementSystemImpl.java

文件实现了LibraryManagementSystem接口，提供了具体的功能实现。在进行并行请求测试的时候使用FOR UPDATE锁定记录，避免数据不一致的问题。

功能说明：

- 实现了LibraryManagementSystem接口的所有方法。
- 完成了图书注册、图书批量注册、并发借阅、借阅和还书等功能。

设计思路：

- 使用PreparedStatement预编译SQL语句，避免SQL注入攻击。
- 使用统一的返回类型

Listing 3: library.system/LibraryManagementSystemImpl.java

```java
import org.springframework.stereotype.Component;
```

```
2  import org.springframework.stereotype.Service;
3
4  @Component
5  @Service
6  public class LibraryManagementSystemImpl implements LibraryManagementSystem
       {
7
8      private final DatabaseConnector connector;
9
10     public LibraryManagementSystemImpl(DatabaseConnector connector) {
11         this.connector = connector;
12     }
13
14     @Override
15     public ApiResult storeBook(Book book) {
16         Connection conn = connector.getConn();
17         try {
18             PreparedStatement checkStmt = conn.prepareStatement(
19                     "SELECT COUNT(*) FROM book WHERE category = ? AND title
                           = ? AND press = ? AND author = ? AND publish_year =
                           ?");
20             checkStmt.setString(1, book.getCategory());
21             checkStmt.setString(2, book.getTitle());
22             checkStmt.setString(3, book.getPress());
23             checkStmt.setString(4, book.getAuthor());
24             checkStmt.setInt(5, book.getPublishYear());
25
26             ResultSet rs = checkStmt.executeQuery();
27             rs.next();
28             int count = rs.getInt(1);
29
30             if (count > 0) {
31                 return new ApiResult(false, "Book already exists in the
                       library system.");
32             }
33
34             PreparedStatement insertStmt = conn
35                     .prepareStatement(
36                             "INSERT INTO book (category, title, press,
```

```
                                publish_year, author, price, stock) VALUES
                                (?,?,?,?,?,?,?)",
                            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 affectedRows = insertStmt.executeUpdate();
            if (affectedRows == 1) {
                ResultSet generatedKeys = insertStmt.getGeneratedKeys();
                if (generatedKeys.next()) {
                    book.setBookId(generatedKeys.getInt(1));
                }
                commit(conn);
                return new ApiResult(true, "Book stored successfully");
            } else {
                rollback(conn);
                return new ApiResult(false, "Failed to store book");
            }

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

3.4 src/main/java/library/system/controller/BookController.java

文件是控制器层的一部分，处理与图书相关的HTTP请求，如注册新图书、查询图书等。

功能说明：

- 定义了处理图书相关请求的RESTful API。
- 使用`@RestController`注解标记为控制器。
- 使用`@RequestMapping("/books")`定义了API的基路径。

设计思路：

- 通过@RequestBody和@RequestParam注解解析请求参数。
- 将业务逻辑委托给LibraryManagementSystem接口实现类，保持控制器的简洁性。

Listing 4: library.system.controller/BookController.java

```java
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;

@RestController
@CrossOrigin(origins = "http://localhost:5173")
@RequestMapping("/api")
public class BookController {
    @Resource
    private LibraryManagementSystem library;
    private static final Logger log = Logger.getLogger(BookController.class
        .getName());

    @PostConstruct
    public void init() {
        try {
            ConnectConfig conf = new ConnectConfig();
            DatabaseConnector connector = new DatabaseConnector(conf);
            boolean connStatus = connector.connect();
            if (!connStatus) {
                throw new SQLException("Failed to connect to database.");
            }
            this.library = new LibraryManagementSystemImpl(connector);
        } catch (Exception e) {
            throw new RuntimeException("Failed to initialize the controller
                .", e);
        }
        resetDatabase();

    }

    public BookController() {
        init();
    }
```

```
32
33      @GetMapping("/book")
34      public BookQueryResults queryBooks(@ModelAttribute BookQueryConditions
            queryConditions) {
35          try {
36              ApiResult result = library.queryBook(queryConditions);
37              if (!result.ok) {
38                  throw new RuntimeException("Failed" + result.message);
39              }
40              return (BookQueryResults) result.payload;
41          } catch (Exception e) {
42              throw new RuntimeException("Failed to query books.", e);
43          }
44      }
45  }
```

4 实验结果

4.1 测试用例验证

最终的测试用例验证了系统的正确性和性能，成功通过了LibraryTest中的所有测试函数

```
1       [INFO] Tests run: 9, Failures: 0, Errors: 0, Skipped: 0, Time elapsed:
            18.576 s - in LibraryTest
2       [INFO]
3       [INFO] Results:
4       [INFO]
5       [INFO] Tests run: 9, Failures: 0, Errors: 0, Skipped: 0
6       [INFO]
```