# 一、项目概述

本课程设计的目标是开发一个功能全面的图书管理系统。该系统将能够实现图书信息的输入、修改、删除，图书销售的记录与统计，以及图书销售情况的查询和报表生成。系统的核心功能将集中在销售统计上，旨在帮助管理人员深入了解图书的销售动态，并据此制定相应的库存和营销策略。

# 二、系统功能

本系统将采用MySQL作为数据库，使用Navicat作为数据库管理工具，以便于数据库的管理和维护。前端系统将基于现有的Element-Admin开发框架和Vue框架进行搭建，以实现一个响应式和用户友好的界面。后端系统则采用Express框架，以提供稳定和高效的服务。以下为系统基本功能。

1. **添加图书基础信息：**用户可以通过前端界面输入图书的详细信息，包括书号、书名、作者、出版社等。
2. **添加图书采购信息：**用于添加图书的采购详情，包括采购日期、采购数量和价格等。
3. **添加图书销售信息：**记录图书的销售情况，包括销售日期、销售数量和价格等。
4. **图书基础信息查询：**用户可以通过输入书号、书名、作者或出版社查询图书的详细信息，还可对图书信息根据书号、书名等信息进行排序查看，也可以对已输入的图书信息进行编辑或删除操作。
5. **图书数量信息查询：**用户可以查看所有已输入的图书的书号、书名、作者和数量信息，并根据相关信息进行排序。
6. **图书销售信息查询：**用户可以查看所有已销售的图书的信息，包括书名、作者、销售数量、销售金额、销售日期，并根据相关信息进行排序。
7. **每月图书销售情况查询：**用户可以输入月份，系统能够自动生成该月图书的销售情况报表，包括日期、书名、月销售总量等信息，并根据相关信息进行排序。

# 三、全局ER图、关系模式及数据库创建

## 4.1 ER图的设计

根据课程设计题目要求，设计了两个实体“图书基本信息”和“图书销售信息”，但两个实体系统过于单调，在老师的指导下，新增了两个实体“图书存储信息”和“图书采购信息”，绘制出的ER图如图1所示。

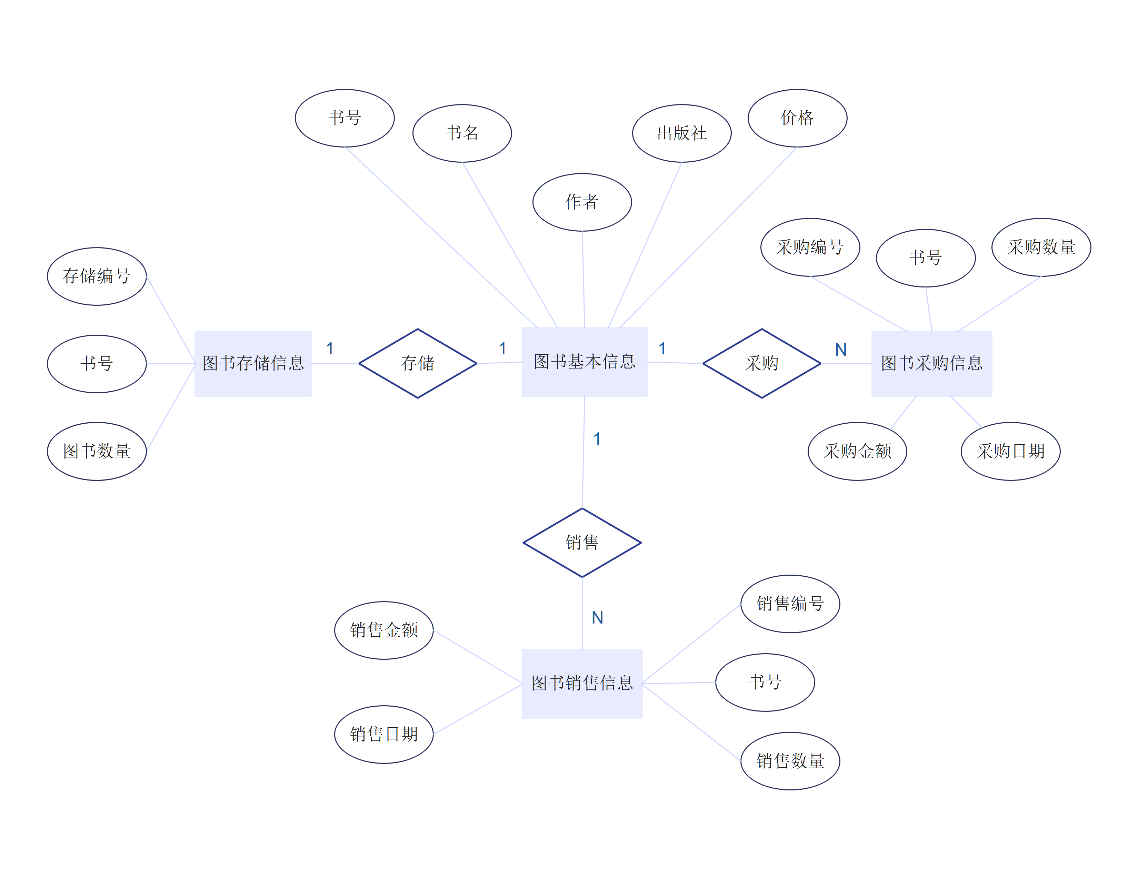


图1 ER图

## 4.2、关系模式

对于表“图书基本信息”，以“书号（ISBN）”作为主键，而其他表都与图书信息有关，故以表“图书基本信息”中的“书号（ISBN）”作为外键，并定义了外键约束，确保一个图书的基本信息被更改或删除时，所有依赖这些信息的记录也会相应地更新或删除，从而维护了数据库的一致性和完整性，关系模式如图2所示。

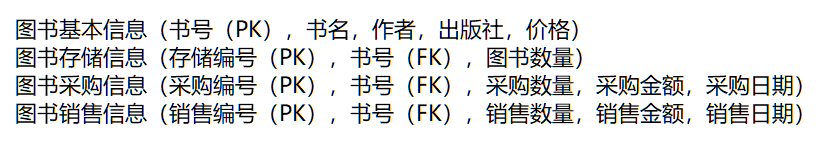


图2 关系模式

## 4.3、数据库和数据表的创建

根据ER图和关系模式创建数据库和相关数据表，如下代码所示。

1. -- 创建数据库，如果数据库已存在，此语句会失败

2. CREATE DATABASE

3. IF

4. NOT EXISTS LibraryManagement;-- 使用该数据库

5. USE LibraryManagement;-- 创建图书基本信息表

6. CREATE TABLE

7. IF

8. NOT EXISTS BookBaseInfo (

9. ISBN INT PRIMARY KEY COMMENT '书号，主键',-- 书号作为主键

10. Title VARCHAR ( 100 ) NOT NULL COMMENT '书名',-- 书名

11. Author VARCHAR ( 100 ) NOT NULL COMMENT '作者',-- 作者

12. Publisher VARCHAR ( 100 ) NOT NULL COMMENT '出版社',-- 出版社

13. Price DECIMAL ( 10, 2 ) NOT NULL COMMENT '价格，保留两位小数' -- 价格

14.

15. ) COMMENT = '存储图书的基本信息';-- 创建图书存储信息表

16. CREATE TABLE

17. IF

18. NOT EXISTS BookStorageInfo (

19. StorageID INT AUTO\_INCREMENT PRIMARY KEY COMMENT '存储编号，主键',-- 存储编号作为主键，自增

20. ISBN INT NOT NULL COMMENT '书号，外键',-- 书号，外键

21. BookQuantity INT NOT NULL COMMENT '图书数量',-- 图书数量

22. FOREIGN KEY ( ISBN ) REFERENCES BookBaseInfo ( ISBN ) ON DELETE CASCADE ON UPDATE CASCADE -- 外键约束，关联到图书基本信息表的书号

23.

24. ) COMMENT = '存储图书的存储信息';-- 创建图书采购信息表

25. CREATE TABLE

26. IF

27. NOT EXISTS BuyInfo (

28. BuyID INT AUTO\_INCREMENT PRIMARY KEY COMMENT '采购编号，主键',-- 采购编号作为主键，自增

29. ISBN INT NOT NULL COMMENT '书号，外键',-- 书号，外键

30. BuyQuantity INT NOT NULL COMMENT '采购数量',-- 采购数量

31. BuyAmount DECIMAL ( 10, 2 ) NOT NULL COMMENT '采购金额，保留两位小数',-- 采购金额

32. BuyDate DATE NOT NULL COMMENT '采购日期',-- 采购日期

33. FOREIGN KEY ( ISBN ) REFERENCES BookBaseInfo ( ISBN ) ON DELETE CASCADE ON UPDATE CASCADE -- 外键约束，关联到图书基本信息表的书号

34.

35. ) COMMENT = '存储图书的采购信息';-- 创建图书采购信息表

36. CREATE TABLE

37. IF

38. NOT EXISTS SaleInfo (

39. SaleID INT AUTO\_INCREMENT PRIMARY KEY COMMENT '销售编号，主键',-- 销售编号作为主键，自增

40. ISBN INT NOT NULL COMMENT '书号，外键',-- 书号，外键

41. SaleQuantity INT NOT NULL COMMENT '销售数量',-- 销售数量

42. SaleAmount DECIMAL ( 10, 2 ) NOT NULL COMMENT '销售金额，保留两位小数',-- 销售金额

43. SaleDate DATE NOT NULL COMMENT '销售日期',-- 销售日期

44. FOREIGN KEY ( ISBN ) REFERENCES BookBaseInfo ( ISBN ) ON DELETE CASCADE ON UPDATE CASCADE -- 外键约束，关联到图书基本信息表的书号

45.

46. ) COMMENT = '存储图书的销售信息';-- 创建图书销售信息表

47.

# 四、视图、存储过程、用户管理及触发器

## 4.1 视图

为实现数据库的可视化，创建了四个视图，“图书采购”，“图书基本信息”，“图书数量”，“图书销售”，用于展示数据库关键信息，便于管理员查看。

1. **图书采购视图：**包含“书名”、“作者”、“采购数量”、“采购金额”和“采购日期”相关信息，便于用户查看图书的采购信息。
2. **图书基本信息视图：**包含“书号”、“书名”、“作者”、“出版社”和“价格”相关信息，便于用户查看看图书的基本信息。
3. **图书数量视图：**包含“书号”、“书名”、“作者”和“数量”相关信息，便于用户查看当前图书的数量信息。
4. **图书销售视图：**包含“书名”、“作者”、“销售数量”、“销售金额”和“销售日期”相关信息，便于用户查看各种图书的销售信息。

## 存储过程

为便于数据管理，提高数据库操纵效率，实现系统的主要功能，建立了四个存储过程“get\_book\_buy”、“get\_book\_num”、“get\_book\_sale”和“get\_book\_sale\_month”。如下代码所示。

1. DELIMITER //

2.

3. CREATE PROCEDURE get\_book\_num(IN book\_id INT)

4. BEGIN

5. SELECT `书号`, `书名`, `作者`, `数量`

6. FROM 图书数量

7. WHERE `书号` = book\_id;

8. END //

9.

10. DELIMITER ;

11.

12. DELIMITER //

13. CREATE PROCEDURE get\_book\_buy(IN rq date)

14. BEGIN

15. SELECT \*

16. FROM 图书采购

17. WHERE `采购日期` = rq;

18. END //

19.

20. DELIMITER ;

21.

22. DELIMITER //

23. CREATE PROCEDURE get\_book\_sale(IN rq date)

24. BEGIN

25. SELECT \*

26. FROM 图书销售

27. WHERE `销售日期` = rq;

28. END //

29.

30. DELIMITER ;

31.

32. DELIMITER //

33.

34. CREATE PROCEDURE get\_book\_sale\_month(IN yearMonth VARCHAR(8))

35. BEGIN

36. SELECT `书名`,

37. `作者`,

38. SUM(`销售数量`) AS 总销售数量,

39. SUM(`销售金额`) AS 总销售金额

40. FROM `图书销售`

41. WHERE DATE\_FORMAT(`销售日期`, '%Y-%m') = yearMonth

42. GROUP BY `书名`, `作者`;

43. END //

44.

45. DELIMITER ;

46.

## 4.3 用户管理

为确保数据库安全性和数据库管理的高效性，故添加一个数据库管理员“libraryadmin123”，使其只拥有对数据库“librarymanagement”的管理权限，提高数据库的安全性。代码如下所示。

1. CREATE USER `libraryadmin123`@`%` IDENTIFIED WITH mysql\_native\_password BY 'admin123';

2.

3.

4. GRANT Alter, Alter Routine, Create, Create Routine, Create Temporary Tables, Create View, Delete, Drop, Event, Execute, Grant Option, Index, Insert, Lock Tables, References, Select, Show View, Trigger, Update ON `librarymanagement`.\* TO `libraryadmin123`@`%`;

5.

## 4.4 触发器

为提示数据库操作的简便型，维护数据库的一致性，保证数据的正确性、数据的自动更新，故添加了3个触发器用于维护数据，使得添加新的图书信息会自动添加一条对应的数量为0的图书存储信息，添加采购、销售信息时，会将对应的图书存储信息加减相关值以维护数据的一致性。代码如下所示。

1. create trigger buy\_book after insert

2. on buyinfo for each row

3. update bookstorageinfo set BookQuantity=BookQuantity+new.BuyQuantity where bookstorageinfo.ISBN=new.ISBN;

4.

5.

6. DROP TRIGGER buy\_book

7.

8.

9.

10. create trigger sale\_book after insert

11. on saleinfo for each row

12. update bookstorageinfo set BookQuantity=BookQuantity-new.SaleQuantity where bookstorageinfo.ISBN=new.ISBN;

13.

14.

15.

16. DELIMITER $$

17.

18. CREATE TRIGGER AfterBookInsert

19. AFTER INSERT ON bookbaseinfo

20. FOR EACH ROW

21. BEGIN

22. INSERT INTO bookstorageinfo (StorageID, ISBN, BookQuantity)

23. VALUES (NULL, NEW.ISBN, 0);

24. END$$

25.

26. DELIMITER ;

27.

# 5、系统的可视化UI设计

为实时展示数据，便于用户查看数据和对相关数据进行管理，使用vscode作为开发工具，nodejs作为编程语言，npm作为包管理工具，利用vue和express技术搭建了web应用对系统进行管理。

## 5.1 前端

采用现有的element-admin框架作为模板，使用vue技术对其进行二次开发，搭建其前端的展示页面，页面美观简洁，便于操作管理。相关代码如下所示。

### 5.1.1 main.js

**Main.js是整个Vue项目的入口文件，用于初始化Vue实例、挂载根组件、配置全局插件和组件、引入全局样式和配置文件、集成路由和状态管理**。通过这些配置，确保整个Vue应用能够正常运行，并具备良好的可扩展性和可维护性。通过main.js中的相关代码配置Vue实例，包括路由、HTTP请求库（axios）、以及Element UI插件，并启动Vue应用。这样，当浏览器加载这个文件时，Vue应用就会被初始化并渲染到页面上。

1. main.js

2. //main.js

3. import Vue from 'vue'

4. import App from './App.vue'

5. import router from './router'

6. import './plugins/element.js'

7.

8. Vue.config.productionTip = false

9.

10. import axios from 'axios' //用来请求接口的。前后端都可以用axios请求接口。没有安装axios的需要安装下npm install axios

11. Vue.prototype.$http = axios.create({

12.   baseURL: 'http://localhost:3001/api'//填写后端接口，后面有后端接口的样例

13. })//可以在任意页面使用this.$http访问axios实例

14.

15. new Vue({

16.   router,

17.   render: h => h(App)

18. }).$mount('#app')

19.

20.

### 5.1.2 App.vue

App.vue文件是整个 Vue 应用的根组件，是项目结构和样式的起点，是vue页面资源的首加载项，是主组件，页面入口文件，所有页面都是在App.vue下进行切换的，也是整个项目的关键，app.vue负责构建定义及页面组件归集。App.vue 作为Vue应用的根组件，定义了整个应用的布局结构，包括侧边导航栏和主内容区域。它使用了Element UI组件库来创建一个具有导航功能的侧边栏，以及一个用于显示页面主要内容的区域。通过<router-view>，它可以动态地根据路由变化显示不同的组件，实现单页面应用的页面切换。

1. App.vue

2. //App.vue

3. <template>

4.   <el-container style="height: 100vh; border: 1px solid #eee">

5.   <el-aside width="200px" style="background-color: rgb(238, 241, 246)">

6.     <!-- <el-menu :default-openeds="openeds" :default-active="active"> -->

7.     <el-menu router :default-openeds="['1', '2']">

8.       <el-submenu index='1' @click.native="handleOpen('1')">

9.           <template slot="title">

10. <i class="el-icon-edit"></i>添加信息

11. </template>

12.         <!-- <el-menu-item index="/articles/create">新建文章</el-menu-item>

13.         <el-menu-item index="/articles/index">文章列表</el-menu-item> -->

14.         <el-menu-item index="/Book/add\_base\_info">添加图书基础信息</el-menu-item>

15.         <el-menu-item index="/Book/add\_buy\_info">添加图书采购信息</el-menu-item>

16.         <el-menu-item index="/Book/add\_sale\_info">添加图书销售信息</el-menu-item>

17.       </el-submenu>

18.       <el-submenu index='2' @click.native="handleOpen('2')">

19.         <template slot="title">

20.           <i class="el-icon-document"></i>查看信息

21.           </template>

22.         <el-menu-item index="/Book/base\_info">图书基础信息</el-menu-item>

23.         <el-menu-item index="/Book/storage\_info">图书数量信息</el-menu-item>

24.         <el-menu-item index="/Book/sale\_info">图书销售情况</el-menu-item>

25.         <el-menu-item index="/Book/sale\_info\_month">每月图书销售情况</el-menu-item>

26.       </el-submenu>

27.     </el-menu>

28.   </el-aside>

29.

30.

31.   <el-container>

32.     <el-header style="text-align: right; font-size: 12px">

33.       <el-dropdown>

34.         <i class="el-icon-setting" style="margin-right: 15px"></i>

35.         <el-dropdown-menu slot="dropdown">

36.           <!-- <el-dropdown-item>查看</el-dropdown-item>

37.           <el-dropdown-item>新增</el-dropdown-item>

38.           <el-dropdown-item>删除</el-dropdown-item>

39.           <el-dropdown-item>图书</el-dropdown-item> -->

40.           </el-dropdown-menu>

41.       </el-dropdown>

42.       <span>图书销售管理系统</span>

43.     </el-header>

44.

45.     <el-main>

46.       <router-view></router-view>

47.     </el-main>

48.   </el-container>

49. </el-container>

50. </template>

51.

52. <style>

53. html,body{

54.   padding: 0;

55.   margin: 0;

56. }

57.   .el-header {

58.     background-color: #B3C0D1;

59.     color: #333;

60.     line-height: 60px;

61.   }

62.

63.   .el-aside {

64.     color: #333;

65.   }

66. </style>

67.

68. <script>

69.   export default {

70.     data() {

71.       return {

72.         openeds: ['2'],

73.       }

74.     },

75.     methods: {

76.     handleOpen(index) {

77.       this.openeds = [index]; // 点击时只展开对应的下拉框

78.       console.log("this.openeds => ", this.openeds);

79.

80.     },

81.   },

82.   };

83. </script>

84.

85.

### 5.1.3 index.js

Index.js文件位于根目录下的src目录下的router目录下，是Vue Router的配置文件，它定义了整个应用的路由规则。Vue Router是一个Vue插件，用于构建单页面应用（SPA），允许用户在不同页面间导航而无需重新加载整个页面。通过相关路由配置，Vue Router能够根据URL的变化渲染对应的组件，实现页面的动态加载和更新。

1. index.js

2. import Vue from 'vue'

3. import VueRouter from 'vue-router'

4. import BuyInfo from '../views/BuyInfo.vue'

5. import SaleInfo from '../views/SaleInfo.vue'

6. import BookSaleInfo from '../views/BookSaleInfo.vue'

7. import BookSaleInfoMonth from '../views/BookSaleInfoMonth.vue'

8. import AddBookBaseInfo from '../views/AddBookBaseInfo.vue'

9. import BookStorageInfo from '../views/BookStorageInfo.vue'

10. import BookBaseInfo from '../views/BookBaseInfo.vue'

11.

12. Vue.use(VueRouter)

13.

14.   const routes = [

15.   {

16.     path: '/',

17.     name: 'Home',

18.     redirect: '/Book/base\_info'

19.   },

20.   {

21.     path: '/Book/add\_base\_info',

22.     name: 'book-add\_base\_info',

23.     component: AddBookBaseInfo

24.   },

25.   {

26.     path: '/Book/base\_info',

27.     name: 'book-base\_info',

28.     component: BookBaseInfo

29.   },

30.   {

31.     path: '/Book/storage\_info',

32.     name: 'book-storage\_info',

33.     component: BookStorageInfo

34.   },

35.   {

36.     path: '/Book/add\_buy\_info',

37.     name: 'book-add\_buy\_info',

38.     component: BuyInfo

39.   },

40.   {

41.     path: '/Book/add\_sale\_info',

42.     name: 'book-add\_sale\_info',

43.     component: SaleInfo

44.   },

45.   {

46.     path: '/Book/sale\_info',

47.     name: 'book-book\_sale\_info',

48.     component: BookSaleInfo

49.   },

50.   {

51.     path: '/Book/sale\_info\_month',

52.     name: 'book-book\_sale\_info\_month',

53.     component: BookSaleInfoMonth

54.   }

55. ]

56.

57. const router = new VueRouter({

58.   routes

59. })

60.

61. export default router

62.

63.

64.

### 5.1.4 ./src/views/目录

在Vue项目中，src 目录下的 views 目录通常用于存放应用中的页面组件。这些页面组件对应于应用中的不同页面或视图，它们是Vue单文件组件（.vue 文件），包含了HTML模板、JavaScript逻辑和CSS样式。每个.vue文件代表应用中的一个页面或视图，在Vue Router的配置中，每个路由通常指向一个组件，这些组件就是放在views目录下的。当用户访问特定的URL时，对应的组件会被加载和渲染。views目录有助于保持项目的组织结构清晰。通过将所有页面组件放在一个单独的目录中，开发者可以更容易地找到和管理这些组件。将页面组件放在views目录下，可以让开发和维护变得更加简单，因为开发者可以专注于页面级别的逻辑和布局，而不需要在多个文件或目录中分散注意力。总的来说，views目录是Vue项目结构中的一个重要部分，它有助于组织和管理页面级别的组件，使得项目结构更加模块化和清晰。以其中一个页面的代码举例。

1. AddBookBaseInfo.vue

2. <template>

3.   <el-form @submit.native.prevent="SaveBook" ref="form" :model="book" label-width="100px">

4.   <el-form-item label="书号">

5.     <el-input type="number" v-model="book.isbn"></el-input>

6.   </el-form-item>

7.   <el-form-item label="书名">

8.     <el-input type="textarea" v-model="book.title"></el-input>

9.   </el-form-item>

10.   <el-form-item label="作者">

11.     <el-input type="textarea" v-model="book.author"></el-input>

12.   </el-form-item>

13.   <el-form-item label="出版社">

14.     <el-input type="textarea" v-model="book.publisher"></el-input>

15.   </el-form-item>

16.   <el-form-item label="价格">

17.     <el-input type="floatValue" v-model="book.price"></el-input>

18.   </el-form-item>

19.   <el-form-item>

20.     <el-button type="primary" native-type="submit">立即添加</el-button>

21.     <el-button>取消</el-button>

22.   </el-form-item>

23. </el-form>

24. </template>

25. <script>

26.   export default {

27.     data() {

28.       return {

29.         book: {}

30.       }

31.     },

32.     methods: {

33.       SaveBook() {

34.         this.$http.post('add\_books',this.book).then(res => {

35.           console.log("res.data => ", res.data);

36.           console.log("res.status => ", res.status);

37.

38.           if (res.status == 200) {

39.             this.$message({

40.               message: "图书信息添加成功",

41.               type: "success"

42.             });

43.           } else {

44.             this.$message({

45.               message: "图书信息添加失败，请检查书号是否重复",

46.               type: "error"

47.             });

48.           }

49.

50.         });

51.       }

52.     }

53.   }

54. </script>

55.

56.

## 5.2 后端

Express是目前流行的基于Node.js运行环境的Web应用程序开发框架，它简洁且灵活，为Web应用程序提供了强大的功能。Express提供了一个轻量级模块，类似于jQuery（封装的工具库），它把Node.js的HTTP模块的功能封装在一个简单易用的接口中，用于扩展HTTP模块的功能，能够轻松地处理服务器的路由、响应、Cookie和HTTP请求的状态。它有着如下的框架特性。

1. 提供了方便简洁的路由定义方式
2. 对获取HTTP请求参数进行了简化处理
3. 对模版引擎支持程度高，方便渲染动态HTML页面
4. 提供了中间件机制有效控制HTTP请求
5. 拥有大量第三方中间件对功能进行扩展

### 5.2.1 config.js

作为express连接数据库的配置文件，记录了连接数据库所需的配置信息，并提供了一个函数来执行SQL查询，该模块可以在Node.js应用程序中被重用，以简化数据库操作。

1. Config.js

2. // 连接数据库，提供数据库操作方法

3. const mysql = require('mysql')

4.

5. const client = mysql.createConnection({

6.     host: "localhost",

7.     user: "libraryadmin123",

8.     password: "admin123",

9.     database: "librarymanagement"

10. })

11.

12. // query:执行数据库语句的方法

13. function connectMySQL(sql, arr, callback) {

14.     console.log("connectSQL arr, arr => ", arr);

15.

16.     client.query(sql, arr, function(err, result) {

17.         console.log("connectMySQL sql => ", sql);

18.

19.         if (err) {

20.             console.log(err);

21.             return;

22.         }

23.         callback(result);

24.     })

25. }

26.

27. module.exports = connectMySQL

28.

### 5.2.2 router.js

该文件定义了一个 Express 应用的路由处理器，它负责处理不同的 HTTP 请求，并根据请求的路径和方法执行相应的逻辑。每个路由都使用 connectMySQL 函数来与 MySQL 数据库进行交互，执行相应的 SQL 语句，并根据操作结果向客户端返回相应的响应，是 Express 应用的核心部分，它定义了应用的 API 接口和业务逻辑，极大地简化了路由配置。

1. Router.js

2. const express = require('express')

3. const router = express.Router()

4. const url = require('url')

5. const connectMySQL = require('./config');

6. const { modelNames } = require('mongoose');

7.

8. // 删除文章

9. router.delete('/articles/:id', async (req, res) => {

10.   try {

11.     console.log("dele req => ", req.params.id);

12.     \_id = req.params.id

13.     const sql = "DELETE FROM articles WHERE \_id = ?"

14.

15.     connectMySQL(sql, [\_id], function(result){

16.         console.log("sql => ", sql);

17.

18.         if (result.affectedRows > 0) {

19.             res.send({

20.                 status: 200,

21.                 msg: "删除成功"

22.             })

23.         } else {

24.             res.send({

25.                 msg: "删除失败"

26.             })

27.         }

28.     })

29.   } catch (err) {

30.     console.log("err => ", err);

31.

32.     res.status(500).send('Error fetching articles');

33.   }

34. });

35.

36. router.post('/add\_books', async (req, res) => {

37.     console.log("add\_books req => ", req.body);

38.

39.   try {

40.     const { isbn, title, author, publisher, price } = req.body;

41.

42.     const sql = "INSERT INTO bookbaseinfo (isbn, title, author, publisher, price) VALUES (?, ?, ?, ?, ?)"

43.

44.     connectMySQL(sql, [isbn, title, author, publisher, price], function(result){

45.         console.log(sql);

46.         console.log("result.affectRow => ", result.affectedRows);

47.

48.         if (result.affectedRows > 0) {

49.             res.send({

50.                 status: 200

51.             })

52.         } else {

53.             res.send({

54.                 status: 500

55.             })

56.         }

57.     })

58.   } catch (err) {

59.     res.status(500).send('Error creating article');

60.   }

61. });

62.

63. router.post('/update\_book\_baseinfo', async (req, res) => {

64.     console.log("update\_book\_baseinfo /articles");

65.

66.     console.log("req => ", req.body);

67.

68.   try {

69.     const { title, author, publisher, price, isbn } = req.body;

70.

71.

72.     // const sql = "INSERT INTO articles (title, body) VALUES (?, ?)"

73.     const sql = "update bookbaseinfo set Title=?, Author=?, Publisher=?, Price=? where isbn=?"

74.

75.     connectMySQL(sql, [title, author, publisher, price, isbn], function(result){

76.         console.log(result);

77.         console.log("result.affectRow => ", result.affectedRows);

78.

79.         if (result.affectedRows > 0) {

80.             res.send({

81.                 status: 200

82.             })

83.         } else {

84.             res.send({

85.                 msg: "修改错误"

86.             })

87.         }

88.     })

89.   } catch (err) {

90.     res.status(500).send('Error creating article');

91.   }

92. });

93.

94. router.get('/get\_books', async (req, res) => {

95.     try {

96.       const sql = "select \* from 图书基本信息"

97.

98.       connectMySQL(sql, null, function(result){

99.         console.log("get\_books  => ", result);

100.

101.         const transformedData = result.map(item => ({

102.                 isbn: item.书号,

103.                 title: item.书名,

104.                 author: item.作者,

105.                 publisher: item.出版社,

106.                 price: item['价格（元）']

107.             }));

108.             console.log("transformedData => ", transformedData);

109.

110.           if (result.length > 0) {

111.               res.send(transformedData)

112.           } else {

113.               res.send({

114.                   msg: "获取数据失败"

115.               })

116.           }

117.       })

118.     } catch (err) {

119.       console.log("err => ", err);

120.

121.       res.status(500).send('Error fetching articles');

122.     }

123.   });

124.

125. router.delete('/dele\_book/:id', async (req, res) => {

126.   try {

127.     console.log("dele req => ", req.params.id);

128.     isbn = req.params.id

129.     console.log("dele isbn => ", isbn);

130.

131.     const sql = "DELETE FROM bookbaseinfo WHERE isbn = ?"

132.

133.     connectMySQL(sql, [isbn], function(result){

134.         console.log("dele\_book sql => ", sql);

135.         console.log("result => ", result);

136.

137.

138.         if (result.affectedRows > 0) {

139.             res.send({

140.                 status: 200,

141.                 msg: "删除成功"

142.             })

143.         } else {

144.             res.send({

145.                 msg: "删除失败"

146.             })

147.         }

148.     })

149.   } catch (err) {

150.     console.log("err => ", err);

151.

152.     res.status(500).send('Error fetching articles');

153.   }

154. });

155.

156. router.get('/get\_books\_num', async (req, res) => {

157.     try {

158.       const sql = "select \* from 图书数量"

159.

160.       connectMySQL(sql, null, function(result){

161.         console.log("get\_books\_num  => ", result);

162.

163.         const transformedData = result.map(item => ({

164.                 isbn: item.书号,

165.                 title: item.书名,

166.                 author: item.作者,

167.                 num: item.数量,

168.             }));

169.             console.log("transformedData => ", transformedData);

170.

171.           if (result.length > 0) {

172.               res.send(transformedData)

173.           } else {

174.               res.send({

175.                   msg: "获取数据失败"

176.               })

177.           }

178.       })

179.     } catch (err) {

180.       console.log("err => ", err);

181.

182.       res.status(500).send('Error fetching articles');

183.     }

184. });

185.

186. router.post('/buy\_books', async (req, res) => {

187.     console.log("add\_books req => ", req.body);

188.

189.   try {

190.     const { isbn, BuyQuantity, BuyAmount, BuyDate } = req.body;

191.

192.     const sql = "INSERT INTO buyinfo (isbn, BuyQuantity, BuyAmount, BuyDate) VALUES (?, ?, ?, ?)"

193.

194.     connectMySQL(sql, [isbn, BuyQuantity, BuyAmount, BuyDate], function(result){

195.         console.log(sql);

196.         console.log("result.affectRow => ", result.affectedRows);

197.

198.         if (result.affectedRows > 0) {

199.             res.send({

200.                 status: 200

201.             })

202.         } else {

203.             res.send({

204.                 msg: "添加失败"

205.             })

206.         }

207.     })

208.   } catch (err) {

209.     res.status(500).send('Error creating article');

210.   }

211. });

212.

213. router.post('/sale\_books', async (req, res) => {

214.     console.log("add\_books req => ", req.body);

215.

216.   try {

217.     const { isbn, SaleQuantity, SaleAmount, SaleDate } = req.body;

218.

219.     const sql = "INSERT INTO saleinfo (isbn, SaleQuantity, SaleAmount, SaleDate) VALUES (?, ?, ?, ?)"

220.

221.     connectMySQL(sql, [isbn, SaleQuantity, SaleAmount, SaleDate], function(result){

222.         console.log(sql);

223.         console.log("result.affectRow => ", result.affectedRows);

224.

225.         if (result.affectedRows > 0) {

226.             res.send({

227.                 status: 200

228.             })

229.         } else {

230.             res.send({

231.                 msg: "添加失败"

232.             })

233.         }

234.     })

235.   } catch (err) {

236.     res.status(500).send('Error creating article');

237.   }

238. });

239.

240. router.get('/get\_books\_sale', async (req, res) => {

241.     try {

242.         const sql = "select \* from 图书销售"

243.

244.         connectMySQL(sql, null, function(result){

245.         console.log("get\_books  => ", result);

246.

247.         const transformedData = result.map(item => ({

248.                 title: item.书名,

249.                 author: item.作者,

250.                 SaleQuantity: item.销售数量,

251.                 SaleAmount: item.销售金额,

252.                 SaleDate: item.销售日期

253.             }));

254.             console.log("transformedData => ", transformedData);

255.

256.             if (result.length > 0) {

257.                 res.send(transformedData)

258.             } else {

259.                 res.send({

260.                     msg: "获取数据失败"

261.                 })

262.             }

263.         })

264.     } catch (err) {

265.         console.log("err => ", err);

266.

267.         res.status(500).send('Error fetching articles');

268.     }

269. });

270.

271. router.post('/get\_books\_sale\_month', async (req, res) => {

272.     console.log("/get\_books\_sale\_month req.body => ", req.body);

273.     console.log("typeof req.body => ", typeof req.body);

274.

275.     let {yearMonth} = req.body

276.     // console.log("year", typeof year);

277.

278.     // year = parseInt(year, 10)

279.     // month = parseInt(month, 10)

280.

281.     // console.log("year", typeof year);

282.     try {

283.         // let sql = "SELECT 书名, 作者, SUM(销售数量) AS 总销售数量, SUM(销售金额) AS 总销售金额 FROM 图书销售 WHERE 销售日期 LIKE ? GROUP BY 书名, 作者;";

284.

285.         let sql = "CALL get\_book\_sale\_month(?);"

286.

287.         console.log("before sql => ", sql);

288.

289.         // 准备 SQL 查询的参数

290.         // const yearMonth = `${year}-${String(month).padStart(2, '0')}`; // 确保月份是两位数

291.

292.         console.log("sql => ", sql);

293.         console.log("yearMonth => ", yearMonth);

294.

295.

296.

297.         connectMySQL(sql, [yearMonth], function(result){

298.         console.log("get\_books\_sale\_month result  => ", result);

299.         const transformedData = result[0].map(item => ({

300.                 title: item.书名,

301.                 author: item.作者,

302.                 totalSaleQuantity: item.总销售数量,

303.                 totalSaleAmount: item.总销售金额

304.             }));

305.             console.log("transformedData => ", transformedData);

306.

307.             if (result.length > 0) {

308.                 res.status(200).send(transformedData)

309.             } else {

310.                 res.status(500).send('Error fetching articles');

311.             }

312.         })

313.     } catch (err) {

314.         console.log("err => ", err);

315.

316.         res.status(500).send('Error fetching articles');

317.     }

318. });

319.

320. module.exports = router

321.

### 5.2.3 index.js

是 Express 应用的入口文件，它负责设置和启动整个服务器，设置了 Express 应用的基本配置，包括中间件、路由和服务器监听，使得应用能够接收和响应 HTTP 请求。

1. Index.js

2. const express = require('express');

3. const app = express();

4. const router = require('./router.js')

5. const bodyParser = require('body-parser')

6. const cors = require('cors');

7.

8. app.use(express.json());

9. app.use(cors());

10. app.use(bodyParser.urlencoded({

11.     extended: true

12. }))

13. app.use("/api", router)

14.

15. app.listen(3001, () => {

16.   console.log('http://localhost:3001/');

17. });

18.

# 六、课程设计成果及展示

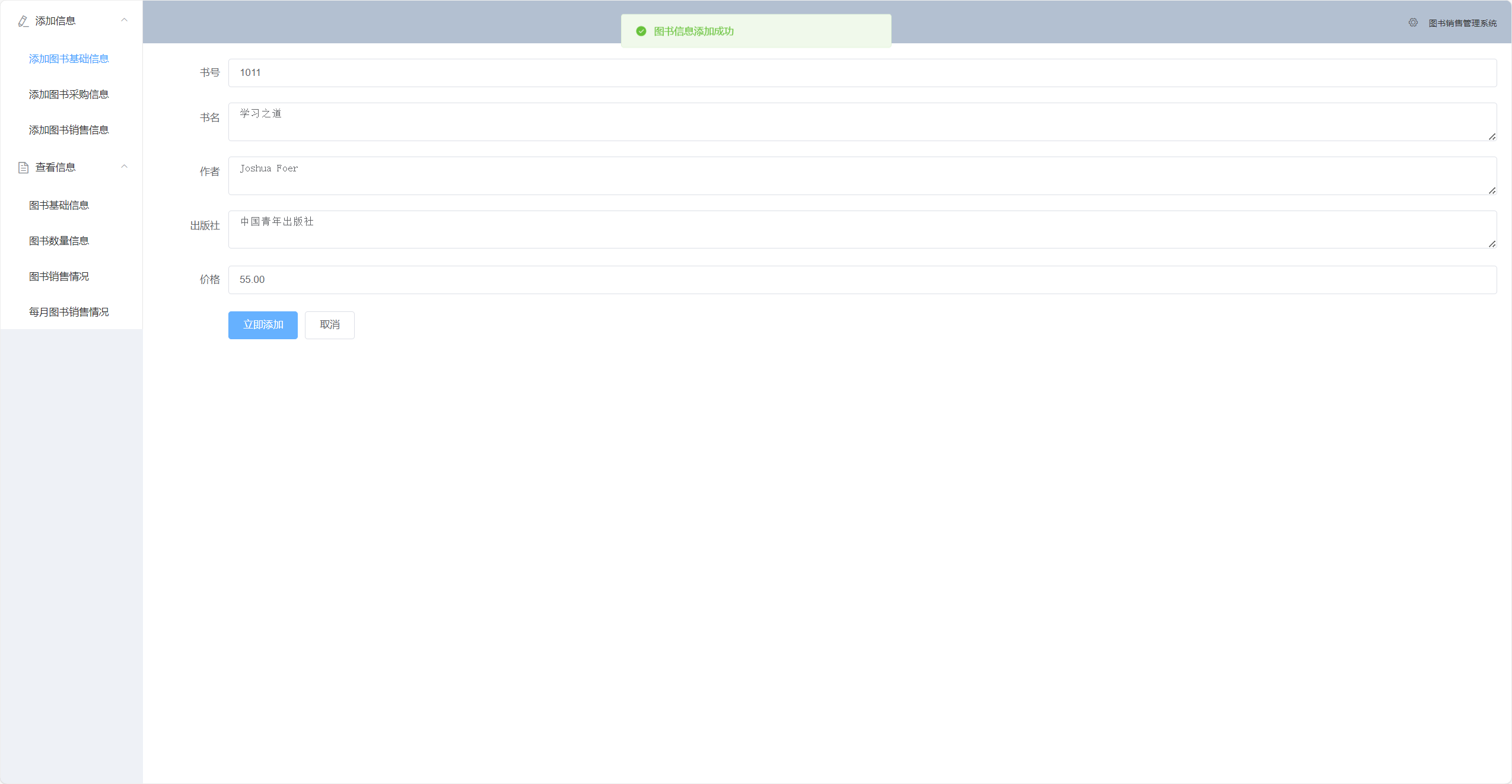


图3 添加图书基础信息

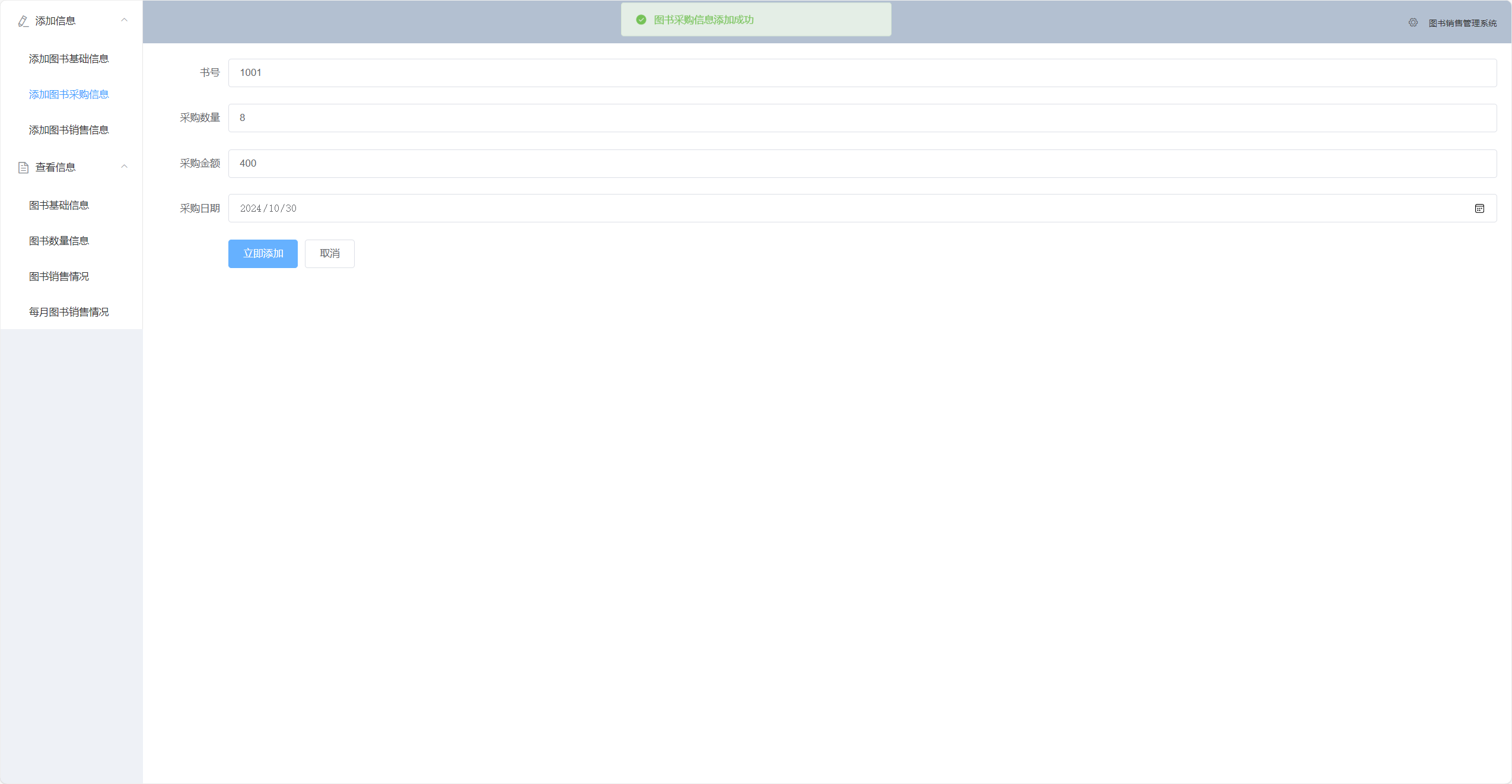


图4 添加图书采购信息

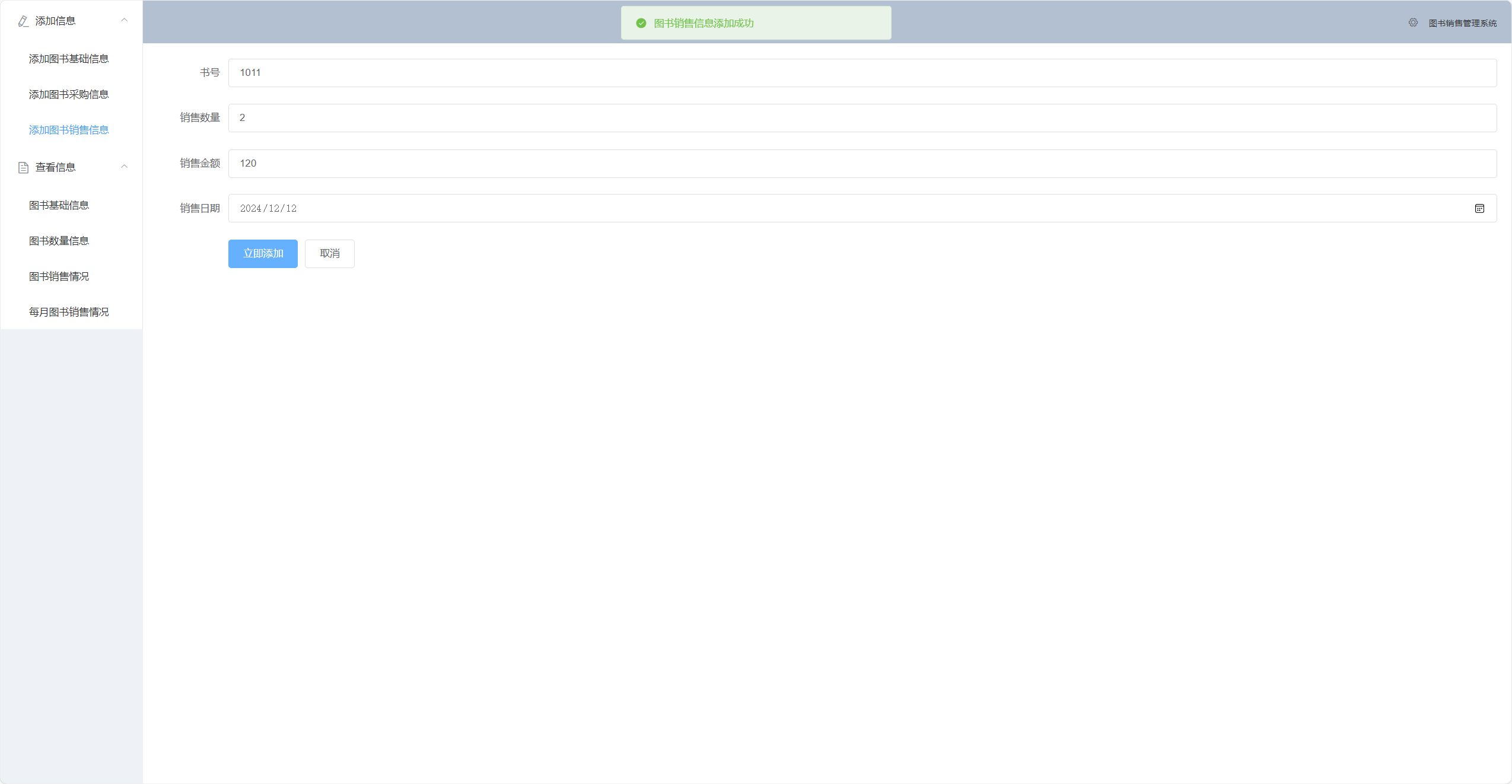


图5 添加图书销售信息

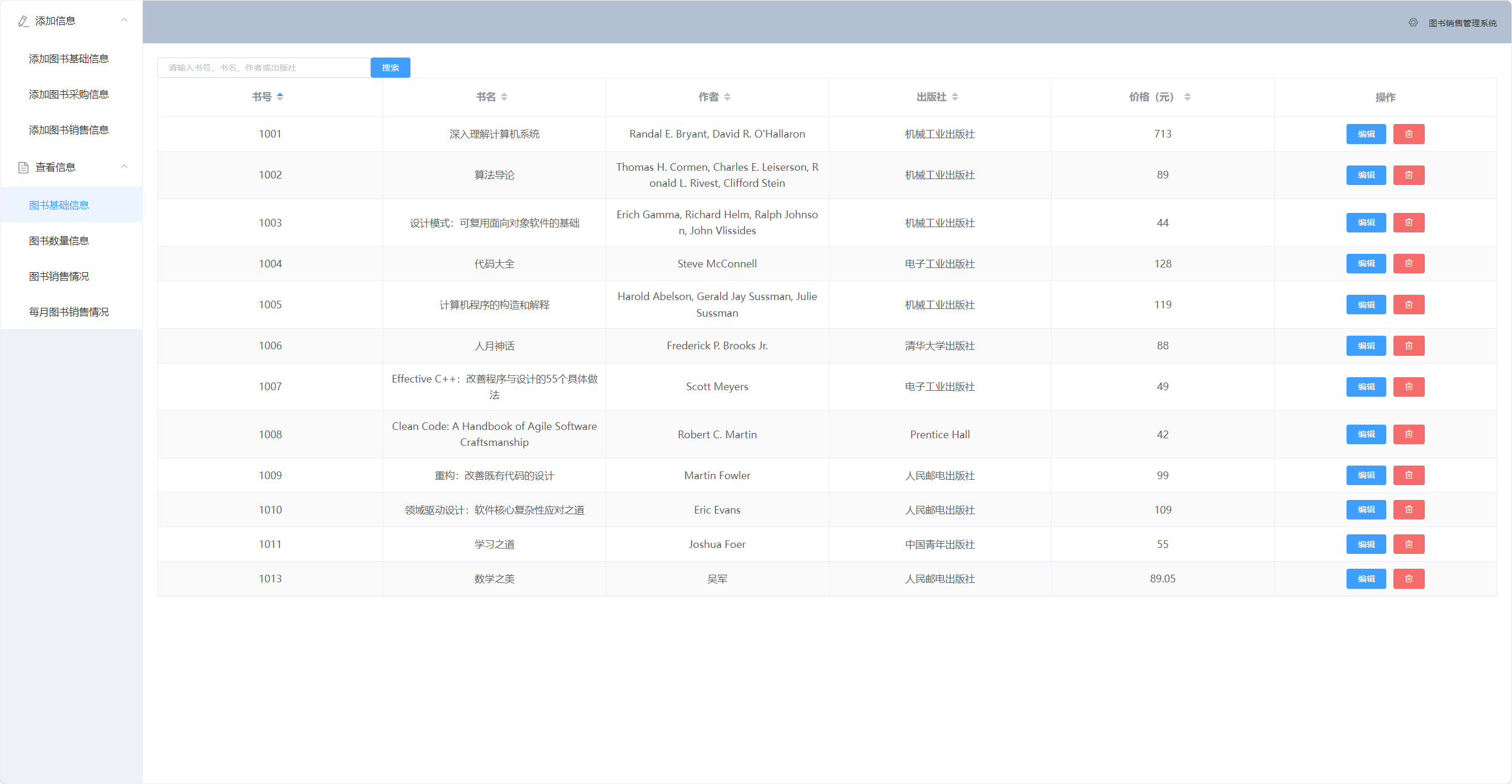


图6 查看图书基础信息



图7 查看图书数量信息

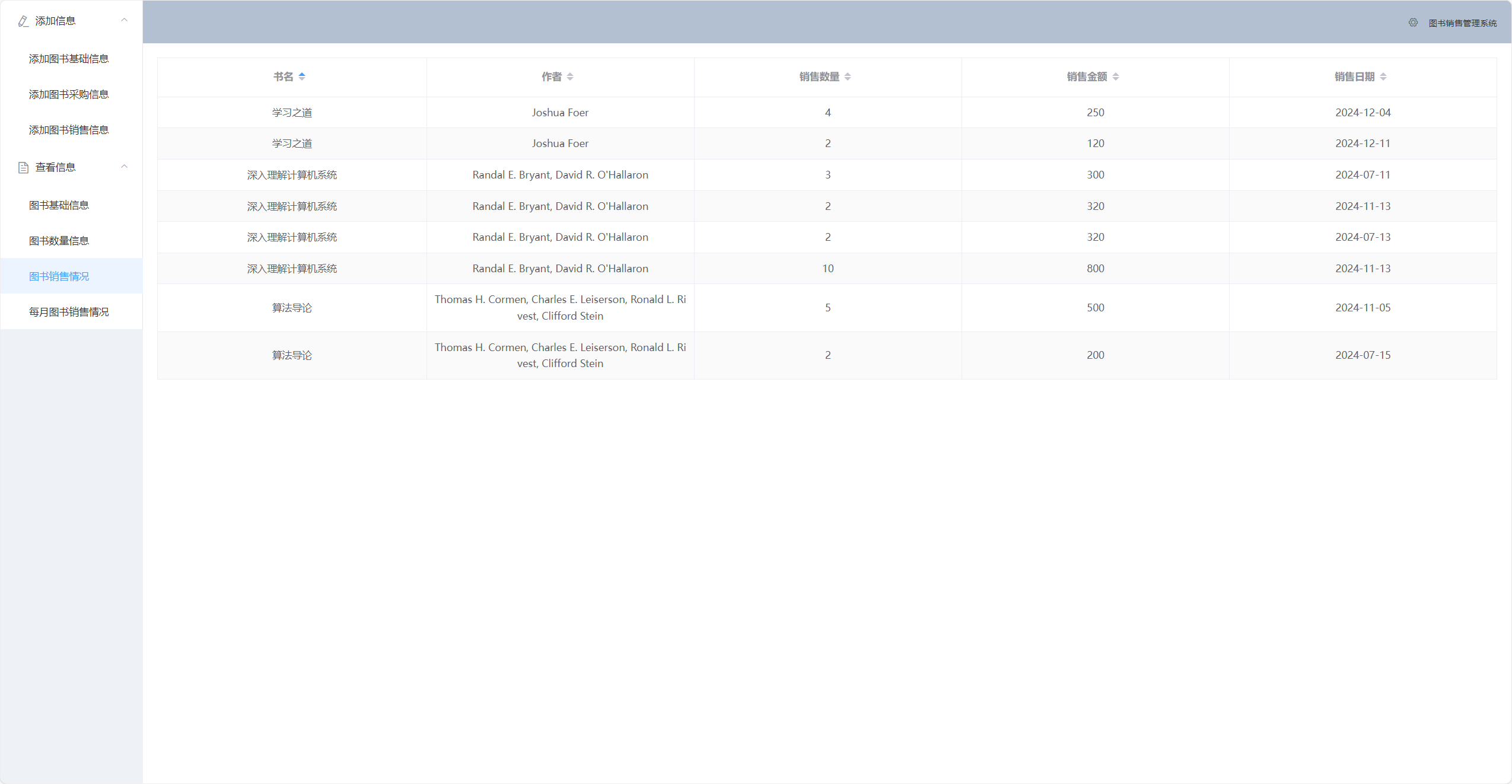


图8 查看图书销售情况

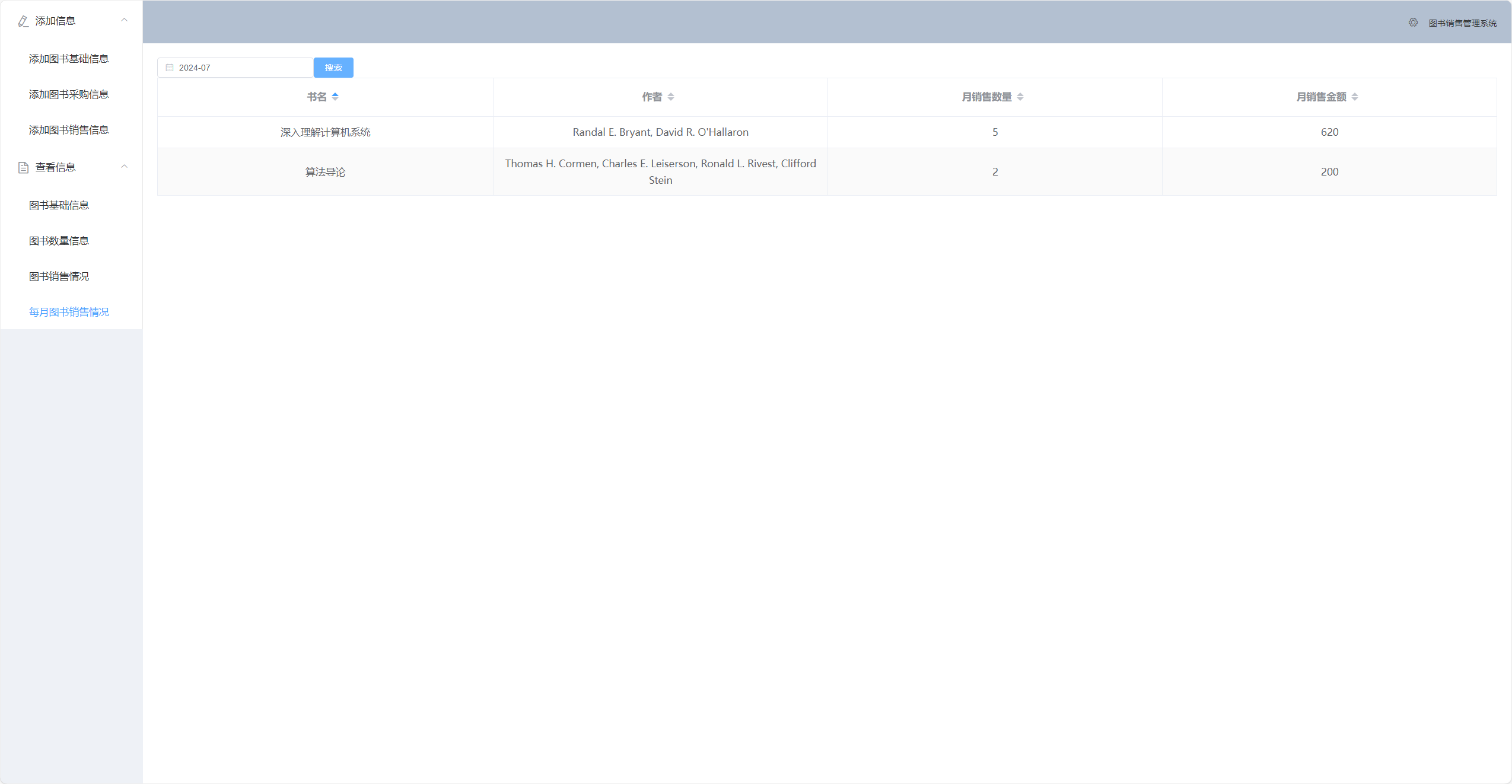


图9 查看每月图书销售情况

# 七、课程设计总结

在为期一周的课程设计中，我加深了对数据库的理解，还提高了对数据库的相关操作的能力。在这一周内，我查阅了很多的资料，也看了很多的关于数据库设计的设计和使用的视频，这对我帮助很大，对于数据库的设计和使用有了总体的概念。查阅资料后，解决了多个问题，总的来说这次课程设计对我的学习帮助很大，解决了很多以前不了解甚至完全没接触过的问题。