# 图书管理系统的设计与实现——基于 openGauss 数据库

卢杨 PB23000043

## 目录

# 1 项目概述

本项目实现了一个功能完整的图书管理系统，旨在对图书馆日常运营中的核心业务进行数字化管理。系统基于 openGauss 数据库构建，涵盖图书采购、借阅、归还、损坏标记与淘汰等关键环节。

整个系统采用 Python 作为后端开发语言，结合 Flask 框架提供可视化操作界面，前端通过 HTML 模板实现交互，后端通过参数化 SQL 脚本与 openGauss 数据库高效通信，确保数据一致性、安全性与操作可靠性。本项目不仅满足实验功能要求，也体现了数据库设计、应用开发与业务逻辑集成的综合实践能力。

# 2 项目结构

项目目录结构如下所示：

```
library-system/
├── run.ps1 # PowerShell 脚本，用于设置环境变量并启动应用
├── run.sh # Bash 脚本，用于在 Linux 环境下启动应用（没有测试过，可能需要调整）
├── ui.py # Flask 主应用
├── library_ui.py # LibrarySQL 业务逻辑类
├── sql.py # openGauss 连接配置
├── library_start.sql # 数据库初始化脚本
├── library_end.sql # 数据库清理脚本
├── requirements.txt # Python 依赖库列表
└── templates/ # 前端模板
    ├── base.html # 基础模板
    ├── login.html # 登录页面
    ├── books.html # 图书列表页面/主页面
    ├── book_detail.html # 图书详情页面
    ├── search.html # 图书搜索页面
    ├── return.html # 图书归还页面
    ├── return_confirm.html # 图书归还确认页面
    ├── set_damaged.html # 图书损坏标记页面（管理员功能）
    ├── borrow_records.html # 借阅记录页面
    ├── add_book.html # 添加图书页面（管理员功能）
    ├── stats.html # 统计页面
    ├── 403.html # 权限不足页面
    └── 404.html # 页面未找到
```

# 3 功能说明

- **用户认证**：支持多用户登录，区分普通用户与管理员 (`is_admin` 字段控制权限)。
- **图书查看**：
  - 查看所有书籍及副本状态
- **借阅与归还** (具体可见6.2)：
  - 用户可借阅“可用且完好”的图书

  - 归还时可标记图书状态 (完好/损坏)
  - 损坏图书将被标记为 `fine = FALSE`，不可再借

- **管理员功能：**
  - 标记任意书籍实例为损坏 (`set_damaged`)(具体可见6.4)
  - 批量丢弃所有损坏图书 (`throw_away_damaged_books`)(具体可见6.4)
  - 查看所有用户的借阅记录 (具体可见6.3)
  - 按借阅人查询借阅记录 (具体可见6.3)
  - 添加新图书及其实例 (具体可见6.1)
  - 批量添加副本 (具体可见6.1)
  - 查看图书馆统计信息 (按区域、作者、年份、借阅人等维度统计)(具体可见6.6)
- **高级检索** (具体可见6.5)：
  - 支持按书名、作者、年份、价格、区域、状态、损坏情况等多条件组合查询
  - 支持三级排序 (如：按年份降序 → 书名升序 → 价格降序)
  - 查询结果以书籍实例 (`book_boxes`) 为单位展示，每条记录对应一本实体书

# 4 开发与运行环境

## 4.1 数据库环境配置

**Hint:** 本项目使用 openGauss 数据库，以下是环境配置与部署细节。具体部署请见附录8.1

- **操作系统：** Ubuntu 22.04.5
- **数据库：** openGauss 6.0.0
- **Docker：** Docker 28.5.1

  配置时使用 Docker 容器化部署 openGauss 数据库，确保环境一致性与易维护性。

## 4.2 主机环境配置

- **操作系统：** Windows 11
- **Python：** Python 3.10
- **依赖库：** Flask，psycopg2

  使用 Flask 框架开发后端应用，结合 psycopg2 库实现与 openGauss 数据库的连接与操作。

# 5 核心实现

## 关于前端实现的说明

本项目的核心目标是验证基于 openGauss 的图书管理数据库设计与业务逻辑实现，因此前端界面主要用于功能演示与交互验证。所有页面模板 (如登录、图书列表、借阅操作等) 均通过 Flask 渲染基础 HTML。具体页面逻辑 (如按钮跳转、表单提交) 由 Flask 路由控制，未引入复杂前端框架。

受限于个人能力以及提升开发效率，部分 HTML 模板结构使用 AI 辅助生成基础代码，但所有数据操作均严格通过后端 LibrarySQL 类调用参数化 SQL 语句完成，确保业务逻辑的完整性与安全性。因此，本报告聚焦于数据库设计、关键 SQL 操作与后端方法实现，前端细节不予展开。完整代码已随项目提交。

## 5.1 数据库设计

数据库设计遵循规范化原则，确保数据一致性与完整性。主要表结构如下：

- **library_sections**：存储图书馆各区域信息。
  - location_id：区域 ID，主键
  - section_name：区域名称

```sql
-- 图书馆区域表
CREATE TABLE library_sections (
    location_id INTEGER PRIMARY KEY,
    section_name VARCHAR(50) NOT NULL
);
```

- **books**：存储图书基本信息。
  - book_id：图书 ID，主键，使用 SERIAL 保证唯一性
  - title：书名
  - author：作者
  - year：出版年份
  - price：价格
  - num_books：库存数量
  - borrowed_count：已借出数量

```sql
-- 图书信息表
CREATE TABLE books (
    book_id SERIAL PRIMARY KEY,
    title VARCHAR(255) NOT NULL,
    author VARCHAR(255) NOT NULL,
    year INTEGER,
    price DECIMAL(10, 2),
    num_books INTEGER DEFAULT 0,
```

```
    borrowed_count INTEGER DEFAULT 0
);
```

- **book_boxes**：存储图书实例信息。
  - id：图书实例 ID，主键，使用 SERIAL 保证唯一性
  - book_id：关联图书 ID，外键 (和 books 表关联)
  - buy_date：购买日期
  - location：所在区域，外键 (和 library_sections 表关联)
  - be_borrowed：是否被借出
  - fine：是否完好，损坏则为 FALSE

**Hint:** books 的每个记录对应多本实体书 (比如，books 有个记录叫做《西游记》，而 book_boxes 对应的记录则是这本书的每一本实体书)。

```
-- 图书实例表
CREATE TABLE book_boxes (
    id SERIAL PRIMARY KEY,
    book_id INTEGER,
    buy_date DATE NOT NULL,
    location INTEGER NOT NULL,
    be_borrowed BOOLEAN DEFAULT FALSE,
    fine BOOLEAN DEFAULT TRUE,
    FOREIGN KEY (book_id) REFERENCES books(book_id),
    FOREIGN KEY (location) REFERENCES library_sections(location_id)
);
```

- **borrow_records**：存储借阅记录信息。
  - record_id：借阅记录 ID，主键，使用 SERIAL 保证唯一性
  - book_box_id：关联图书实例 ID，外键 (和 book_boxes 表关联)
  - borrower：借阅人用户名，外键 (和 users 表关联)
  - borrow_date：借阅日期
  - return_date：归还日期

```
-- 借阅记录表
CREATE TABLE borrow_records (
    record_id SERIAL PRIMARY KEY,
    book_box_id INTEGER NOT NULL,
    borrower VARCHAR(255) NOT NULL,
    borrow_date DATE NOT NULL,
    return_date DATE DEFAULT NULL,
    FOREIGN KEY (book_box_id) REFERENCES book_boxes(id),
    FOREIGN KEY (borrower) REFERENCES users(username)
);
```

- **users**：存储用户信息。
  - username：用户名，主键
  - password：密码
  - is_admin：是否为管理员

```sql
-- 用户表
CREATE TABLE users (
    username VARCHAR(255) NOT NULL PRIMARY KEY,
    password VARCHAR(255) NOT NULL,
    is_admin BOOLEAN DEFAULT FALSE
);
```

### 5.2 主体函数

- opengauss_run：openGauss 数据库连接与操作封装类，支持上下文管理器协议，确保连接的正确打开与关闭。其中，test 参数用于控制事务提交或回滚，方便测试与调试。
- 本项目中所有数据库操作均通过该类进行连接与执行，确保代码简洁与资源管理。
- **Hint:** 本项目使用环境变量配置数据库连接参数，方便不同环境下的部署与运行。保证安全性，避免硬编码敏感信息。

```python
host = os.getenv("DB_HOST", "localhost")
port = os.getenv("DB_PORT", 5432)
database = os.getenv("DB_NAME", "postgres")
user = os.getenv("DB_USER", "omm")
password = os.getenv("DB_PASSWORD", "password")

config = {
    "host": host,
    "port": port,
    "database": database,
    "user": user,
    "password": password
}

class opengauss_run(object):
    """openGauss 数据库连接与操作封装类"""
    def __init__(self, config, test=False):
        self.config = config
        self.conn = None
        self.cur = None
        self.test = test

    def __enter__(self):
```

```
        self.conn = psycopg2.connect(**self.config)
        self.cur = self.conn.cursor()
        return self

    def __exit__(self, exc_type, exc_val, exc_tb):
        if exc_type is not None or self.test:
            self.conn.rollback()
        else:
            self.conn.commit()
        self.cur.close()
        self.conn.close()
```

- LibrarySQL：图书管理系统数据库操作类，封装了所有与图书管理相关的数据库操作方法，包括图书列表查询、添加图书、副本管理、借阅与归还、损坏标记与淘汰、以及高级检索等功能。功能具体实现见第5.3节

```
from sql import opengauss_run

class LibrarySQL(object):
    """图书管理系统数据库操作类"""
    def __init__(self, config):
        self.config = config
        self.list_admin_users = []
        # 获取所有管理员用户列表
        with opengauss_run(self.config) as db:
            db.cur.execute("SELECT username FROM users WHERE is_admin = TRUE;")
            rows = db.cur.fetchall()
            self.list_admin_users = [row[0] for row in rows]

    # ========== Listing ==========
    def list_books(self):
        pass

    def list_book_boxes(self):
        pass

    # ========== Operations ==========
    def add_book(self, title: str, author: str, year: int, price: float, buy_date: str,
    location: int):
        pass

    def add_book_copies(self, book_id: int, count: int, buy_date: str, location: int):
        pass

    def borrow_book(self, id_: int, borrower: str, borrow_date: str):
        pass

```

```
    def return_book(self, id_: int, return_date: str, fine: bool = True):
        pass

    def set_damaged(self, id_: int):
        pass

    def throw_away_damaged_books(self):
        pass

    # ========== Advanced Querying ==========
    def query_books(self, title=None, author=None, book_id=None, year_min=None, year_max=None,
     price_min=None, price_max=None,
                    location=None, borrow=None, borrower=None, fine=None,
                    # Sorting with up to 3 levels
                    sort_by_1=None, sort_order_1='asc', sort_by_2=None, sort_order_2='asc',
    sort_by_3=None, sort_order_3='asc'):
        pass


    # ========== Borrow Records ==========
    def list_borrow_records(self, user: str = None):
        pass


    # ========= Overview Statistics ==========
    def get_overview_stats(self, group_by: str = None):
        pass

    def statistics_all(self):
        pass
```

## 5.3 LibrarySQL 方法具体呈现

### 5.3.1 list_books

- 功能：获取所有所有图书及其基本信息，包括图书 ID、书名、作者、年份、价格、库存数量和已借出数量。
  - 这里使用简单的 SQL 查询语句，从 books 表中获取所有图书记录，并将结果封装为字典列表返回。

```
def list_books(self):
    sql = """
        SELECT DISTINCT b.book_id, b.title, b.author, b.year, b.price, b.num_books, b.
    borrowed_count
        FROM books b
    """
    with opengauss_run(self.config) as db:
        db.cur.execute(sql)
```

```
        rows = db.cur.fetchall()
    result = []
    for row in rows:
        book_id, title, author, year, price, num_books, borrowed_count = row
        result.append({
            'book_id': book_id, 'title': title, 'author': author, 'year': year, 'price': price
    , 'num_books': num_books, 'borrowed_count': borrowed_count
        })
    return result
```

#### 5.3.2 list_book_boxes

- 功能：获取所有图书实例及其详细信息，包括实例 ID、关联图书 ID、购买日期、所在区域、是否被借出和是否完好。
  - 该方法通过 SQL 查询连接 book_boxes、books 和 library_sections 三个表，获取每本实体书的详细信息，并将结果封装为字典列表返回。

```
def list_book_boxes(self):
    sql = """
        SELECT b.book_id, b.title, b.author, bb.buy_date, ls.section_name, bb.be_borrowed,
                bb.fine, b.year, b.price, bb.id
        FROM book_boxes bb
        JOIN books b ON bb.book_id = b.book_id
        JOIN library_sections ls ON bb.location = ls.location_id
    """
    with opengauss_run(self.config) as db:
        db.cur.execute(sql)
        book_boxes = db.cur.fetchall()
    result = []
    for box in book_boxes:
        (book_id, title, author, buy_date, section_name, be_borrowed,
            fine, year, price, id_) = box
        status = "Borrowed" if be_borrowed else "Available"
        fine_status = "Yes" if fine else "No(wait to throw away)"
        result.append({
            'book_id': book_id, 'title': title, 'author': author, 'year': year, 'price': price
    , 'buy_date': buy_date, 'section': section_name, 'status': status, 'fine': fine_status, '
    fine_bool': fine, 'id': id_
        })
    return result
```

#### 5.3.3 add_book

- 功能：添加新图书及其实例。

– 该方法首先验证指定的存放位置是否存在，然后插入新图书记录到 books 表，并获取生成的 book_id。随后，插入对应的图书实例记录到 book_boxes 表，最终返回新图书的 ID。

```python
def add_book(self, title: str, author: str, year: int, price: float, buy_date: str, location: int):
    # make sure location exists
    with opengauss_run(self.config) as db:
        db.cur.execute("SELECT 1 FROM library_sections WHERE location_id = %s;", (location,))
        if not db.cur.fetchone():
            raise ValueError("Invalid location")

    with opengauss_run(self.config) as db:
        sql = "INSERT INTO books (title, author, year, price, num_books) VALUES (%s, %s, %s, %s, 1) RETURNING book_id;"
        db.cur.execute(sql, (title, author, year, price))
        book_id = db.cur.fetchone()[0]

        sql = "INSERT INTO book_boxes (book_id, buy_date, location) VALUES (%s, %s, %s);"
        db.cur.execute(sql, (book_id, buy_date, location))
    return book_id
```

#### 5.3.4 add_book_copies

- 功能：为已有图书添加多个副本。
  - 该方法首先验证指定的图书 ID 是否存在，然后循环插入指定数量的图书实例记录到 book_boxes 表，最后更新 books 表中的库存数量，并返回添加的副本信息。
  - 注意：该方法假设传入的 book_id 已存在于 books 表中。

```python
def add_book_copies(self, book_id: int, count: int, buy_date: str, location: int):
    with opengauss_run(self.config) as db:
        db.cur.execute("SELECT title, author FROM books WHERE book_id = %s;", (book_id,))
        result = db.cur.fetchone()
        if not result:
            raise ValueError("Book not found")
        title, author = result

    with opengauss_run(self.config) as db:
        values = [(book_id, buy_date, location)] * count
        db.cur.executemany("INSERT INTO book_boxes (book_id, buy_date, location) VALUES (%s, %s, %s)", values)
        db.cur.execute("UPDATE books SET num_books = num_books + %s WHERE book_id = %s;", (count, book_id))
    return {"title": title, "author": author, "added_copies": count}
```

### 5.3.5 borrow_book

- 功能：借阅图书实例。
  - 该方法首先插入一条新的借阅记录到 borrow_records 表，然后更新对应图书实例的借阅状态为已借出，并增加图书的借出计数，最后返回借阅成功的信息。
  - 注意：该方法假设传入的图书实例 ID 已存在且可借阅。这是在调用该方法前需要进行的验证。在本项目中，这些验证在前端逻辑中完成。

```
def borrow_book(self, id_: int, borrower: str, borrow_date: str):
    with opengauss_run(self.config) as db:
        sql = "INSERT INTO borrow_records (book_box_id, borrower, borrow_date) VALUES (%s, %s, %s);"
        db.cur.execute(sql, (id_, borrower, borrow_date))
        sql = "UPDATE book_boxes SET be_borrowed = TRUE WHERE id = %s;"
        db.cur.execute(sql, (id_,))
        db.cur.execute("""
            UPDATE books
            SET borrowed_count = borrowed_count + 1
            WHERE book_id = (SELECT book_id FROM book_boxes WHERE id = %s);
        """, (id_,))

    return {"success": True, "message": f"Book with ID {id_} borrowed by {borrower} on {borrow_date}."}
```

### 5.3.6 return_book

- 功能：归还图书实例。
  - 该方法首先检查图书实例是否处于借出状态，若未借出则返回错误信息。然后更新对应借阅记录的归还日期，修改图书实例的借阅状态为可用，并根据传入的 fine 参数更新图书的完好状态。最后减少图书的借出计数，并返回归还成功的信息。
  - 注意：该方法假设传入的图书实例 ID 已存在且已借出。这是在调用该方法前需要进行的验证。在本项目中，这些验证在前端逻辑中完成。

```
def return_book(self, id_: int, return_date: str, fine: bool = True):
    with opengauss_run(self.config) as db:
        db.cur.execute("SELECT be_borrowed FROM book_boxes WHERE id = %s;", (id_,))
        result = db.cur.fetchone()
        if not result or not result[0]:
            return {"success": False, "message": f"The book with ID {id_} is not currently borrowed."}

        sql = "UPDATE borrow_records SET return_date = %s WHERE book_box_id = %s AND return_date IS NULL;"
        db.cur.execute(sql, (return_date, id_))
```

```
        sql = "UPDATE book_boxes SET be_borrowed = FALSE, fine = %s WHERE id = %s;"
        db.cur.execute(sql, (fine, id_))
        sql = "UPDATE books SET borrowed_count = GREATEST(borrowed_count - 1, 0) WHERE book_id
    = (SELECT book_id FROM book_boxes WHERE id = %s);"
        db.cur.execute(sql, (id_,))
    return {"success": True, "message": f"Book with ID {id_} returned on {return_date}. Fine:
    {'Yes' if fine else 'No'}."}
```

#### 5.3.7 set_damaged

- 功能：标记图书实例为损坏。这是管理员专用功能。
  - 该方法将图书实例的状态标记为损坏

```
def set_damaged(self, id_: int):
    with opengauss_run(self.config) as db:
        db.cur.execute("SELECT fine FROM book_boxes WHERE id = %s and fine = TRUE and
    be_borrowed = FALSE;", (id_,))
        result = db.cur.fetchone()
        if not result:
            return {"success": False, "message": f"Book with ID {id_} not found."}
        if not result[0]:
            return {"success": False, "message": f"Book with ID {id_} already damaged."}

        db.cur.execute("UPDATE book_boxes SET fine = FALSE WHERE id = %s;", (id_,))
    return {"success": True, "message": f"Book with ID {id_} marked as damaged."}
```

#### 5.3.8 throw_away_damaged_books

- 功能：批量丢弃所有标记为损坏的图书实例。这是管理员专用功能。
  - 该方法首先查询所有标记为损坏的图书实例，然后删除对应的借阅记录和图书实例记录，并更新图书的库存数量。最后返回丢弃的图书信息。

```
def throw_away_damaged_books(self):
    with opengauss_run(self.config) as db:
        db.cur.execute("SELECT bb.id, b.title, b.author, b.book_id FROM book_boxes bb JOIN
    books b ON bb.book_id = b.book_id WHERE bb.fine = FALSE;")
        damaged = db.cur.fetchall()
        if not damaged:
            return {"success": True, "message": "No damaged books to throw away.", "thrown":
    []}

        db.cur.execute("""DELETE FROM borrow_records WHERE book_box_id IN (
            SELECT id FROM book_boxes WHERE fine = FALSE
        );""")

```

```
        db.cur.execute("DELETE FROM book_boxes WHERE fine = FALSE;")
        thrown = []
        for id_, title, author, book_id in damaged:
            thrown.append({'title': title, 'book_id': book_id})
            db.cur.execute("UPDATE books SET num_books = num_books - 1 WHERE book_id = %s;", (book_id,))

        db.cur.execute("DELETE FROM books WHERE num_books <= 0;")
    return {"success": True, "message": f"Threw away {len(thrown)} damaged books.", "thrown": thrown}
```

#### 5.3.9 query_books

- 功能：高级图书查询，支持多条件组合查询和三级排序。
  - 该方法根据传入的查询参数动态构建 SQL 查询语句，支持按书名、作者、年份、价格、区域、状态、借阅人和损坏情况等多条件过滤，并支持最多三级排序。查询结果以图书实例为单位返回，每条记录对应一本实体书。同时最后附带检索书籍数量。
  - 特别注意：由于 borrow_records 表仅记录借阅中的图书实例信息，因此在查询时使用左连接将其与 book_boxes 表关联。如果需要使用 borrower 进行查询 (实际设计时只有管理员可以具有这个功能)，加上 br.return_date IS NULL 以确保只获取当前借阅中的记录。

```
def query_books(self, title=None, author=None, book_id=None, year_min=None, year_max=None, price_min=None, price_max=None,
                location=None, borrow=None, borrower=None, fine=None,
                # Sorting with up to 3 levels
                sort_by_1=None, sort_order_1='asc', sort_by_2=None, sort_order_2='asc', sort_by_3=None, sort_order_3='asc'):
    sql = """
        SELECT b.title, b.author, b.book_id, bb.buy_date, ls.section_name, bb.be_borrowed,
                bb.fine, b.year, b.price, bb.id, br.borrow_date
        FROM book_boxes bb
        JOIN books b ON bb.book_id = b.book_id
        JOIN library_sections ls ON bb.location = ls.location_id
        LEFT JOIN borrow_records br ON bb.id = br.book_box_id AND br.return_date IS NULL
        WHERE 1=1
    """
    params = []
    if title:
        sql += " AND b.title = %s"
        params.append(title)
    if author:
        sql += " AND b.author = %s"
        params.append(author)
    if book_id:
```

```
        sql += " AND b.book_id = %s"
        params.append(book_id)
    if year_min is not None:
        sql += " AND b.year >= %s"
        params.append(year_min)
    if year_max is not None:
        sql += " AND b.year <= %s"
        params.append(year_max)
    if price_min is not None:
        sql += " AND b.price >= %s"
        params.append(price_min)
    if price_max is not None:
        sql += " AND b.price <= %s"
        params.append(price_max)
    if location:
        sql += " AND bb.location = %s"
        params.append(location)
    if borrow is not None:
        sql += " AND bb.be_borrowed = %s"
        params.append(borrow)
    if borrower:
        sql += " AND br.borrower = %s AND br.return_date IS NULL"
        params.append(borrower)
    if fine is not None:
        sql += " AND bb.fine = %s"
        params.append(fine)

    # Sorting
    if sort_by_1:
        sql += f" ORDER BY {sort_by_1} {sort_order_1.upper()}"
        if sort_by_2:
            sql += f", {sort_by_2} {sort_order_2.upper()}"
            if sort_by_3:
                sql += f", {sort_by_3} {sort_order_3.upper()}"

    with opengauss_run(self.config) as db:
        db.cur.execute(sql, params)
        rows = db.cur.fetchall()
    result = []
    for row in rows:
        (title, author, book_id, buy_date, section, be_borrowed,
            fine, year, price, id_, borrow_date) = row
        result.append({
            'title': title, 'author': author, 'book_id': book_id, 'year': year, 'price': price
, 'buy_date': buy_date, 'section': section, 'status': 'Borrowed' if be_borrowed else '
Available', 'fine': 'Yes' if fine else 'No(wait to throw away)', 'fine_bool': fine, 'id':
```

```
    id_, 'borrow_date': borrow_date
        })
    return result, len(result)
```

#### 5.3.10 list_borrow_records

- 功能：列出所有借阅记录，支持按借阅人过滤。
  - 该方法查询 borrow_records 表，并通过连接 book_boxes、books 和 library_sections 三个表，获取每条借阅记录的详细信息。如果传入了 user 参数，则只返回该用户的借阅记录；否则返回所有用户的借阅记录。结果按借阅日期和记录 ID 降序排序。

```
def list_borrow_records(self, user: str = None):
    if user is not None:
        sql = """
            SELECT br.record_id, br.borrower, br.borrow_date, br.return_date,
                bb.id AS box_id, b.title, b.author, ls.section_name, bb.fine
            FROM borrow_records br
            JOIN book_boxes bb ON br.book_box_id = bb.id
            JOIN books b ON bb.book_id = b.book_id
            JOIN library_sections ls ON bb.location = ls.location_id
            WHERE br.borrower = %s
            ORDER BY br.borrow_date DESC, br.record_id DESC;
        """
        with opengauss_run(self.config) as db:
            db.cur.execute(sql, (user,))
            rows = db.cur.fetchall()
    else:
        sql = """
            SELECT br.record_id, br.borrower, br.borrow_date, br.return_date,
                bb.id AS box_id, b.title, b.author, ls.section_name, bb.fine
            FROM borrow_records br
            JOIN book_boxes bb ON br.book_box_id = bb.id
            JOIN books b ON bb.book_id = b.book_id
            JOIN library_sections ls ON bb.location = ls.location_id
            ORDER BY br.borrow_date DESC, br.record_id DESC;
        """
        with opengauss_run(self.config) as db:
            db.cur.execute(sql)
            rows = db.cur.fetchall()

    result = []
    for row in rows:
        record_id, borrower, borrow_date, return_date, box_id, title, author, section_name,
    fine = row
        status = "Returned" if return_date else "Borrowed"
```

```
        result.append({
            'record_id': record_id, 'borrower': borrower, 'borrow_date': borrow_date, '
    return_date': return_date, 'box_id': box_id, 'title': title, 'author': author, 'section':
    section_name, 'status': status, 'fine': fine
        })
    return result
```

#### 5.3.11 get_overview_stats

- 功能：获取系统概览统计信息，支持按指定字段分组统计。
  - 该方法通过 SQL 查询计算图书的总种类数、平均价格、总价值和总副本数。如果传入了 group_by 参数，则按指定字段进行分组统计。支持的分组字段包括区域、作者、年份、状态、损坏情况、借阅人和购买日期。结果以字典列表形式返回，每条记录包含统计数据和分组键。

```
def get_overview_stats(self, group_by: str = None):
    """获取系统概览统计"""
    sql = """
        SELECT
            COUNT(DISTINCT b.book_id) AS total_titles,
            AVG(b.price) AS avg_price,
            SUM(b.price) AS total_value,
            COUNT(bb.id) AS total_copies
    """
    if group_by == 'location':
        sql += ", ls.section_name"
    elif group_by == 'author':
        sql += ", b.author"
    elif group_by == 'year':
        sql += ", b.year"
    elif group_by == 'status':
        sql += ", bb.be_borrowed"
    elif group_by == 'fine':
        sql += ", bb.fine"
    elif group_by == 'borrower':
        sql += ", br.borrower"
    elif group_by == "buy_date":
        sql += ", TO_CHAR(bb.buy_date, 'YYYY-MM-DD')"

    sql += """
        FROM books b
        JOIN book_boxes bb ON b.book_id = bb.book_id
        JOIN library_sections ls ON bb.location = ls.location_id
        LEFT JOIN borrow_records br ON bb.id = br.book_box_id AND br.return_date IS NULL
    """
    if group_by == 'location':
```

```
        sql += " GROUP BY ls.section_name"
    elif group_by == 'author':
        sql += " GROUP BY b.author"
    elif group_by == 'year':
        sql += " GROUP BY b.year"
    elif group_by == 'status':
        sql += " GROUP BY bb.be_borrowed"
    elif group_by == 'fine':
        sql += " GROUP BY bb.fine"
    elif group_by == 'borrower':
        sql += " GROUP BY br.borrower"
    elif group_by == "buy_date":
        sql += " GROUP BY bb.buy_date"
    sql += ";"
    with opengauss_run(self.config) as db:
        db.cur.execute(sql)
        rows = db.cur.fetchall()
    result = []
    for row in rows:
        (total_titles, avg_price, total_value, total_copies) = row[:4]
        result.append({
            'total_titles': total_titles, 'avg_price': avg_price, 'total_value': total_value,
    'total_copies': total_copies, 'group_key': row[4] if group_by else 'overall'
        })
    return result
```

#### 5.3.12 statistics_all

- 功能：获取所有概览统计信息。
  - 该方法调用 get_overview_stats 方法，依次获取整体统计和按各个字段分组的统计信息，并将结果汇总为一个字典返回。

```
def statistics_all(self):
    group_bys = [None, 'location', 'author', 'year', 'status', 'fine', 'borrower', 'buy_date']
    stats = {}
    for group in group_bys:
        stats[group if group else 'overall'] = self.get_overview_stats(group_by=group)

    return stats
```

## 6 项目演示

**Hint:** 为展现项目功能，下面每一个小节均配有相应功能展示的界面截图，展示系统的实际运行效果。由于图书列表 (主页) 和图书详细信息页面在各个功能中均有体现，故不作单独展示。

## 6.1 图书入库

- 以下为加入新书《数据库系统概念》的操作界面和主页截图对比：

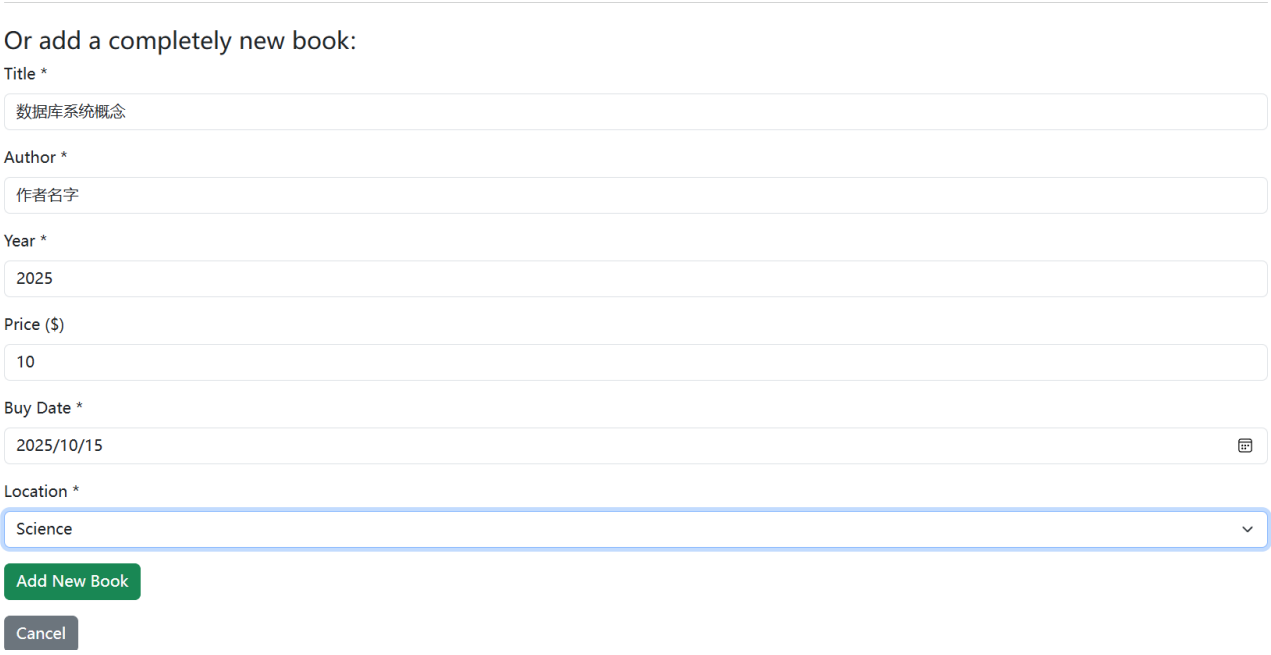
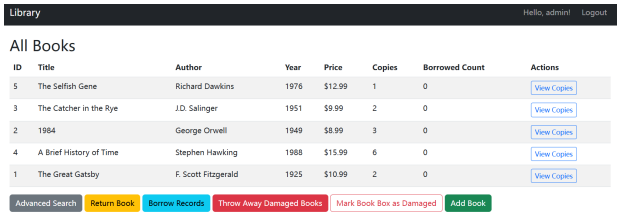
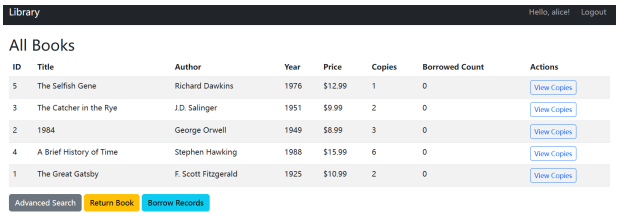


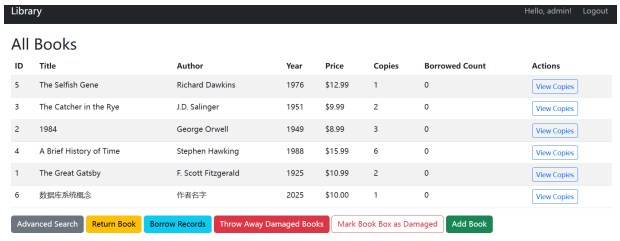

图 1: 图书入库操作



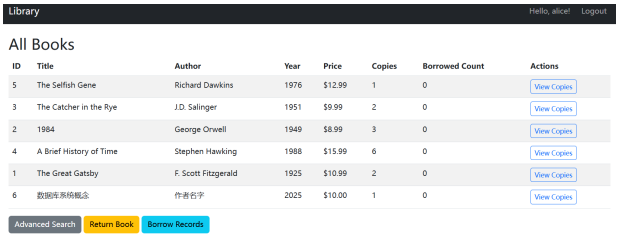

(a) 管理员主页 (操作前)




(b) 普通用户主页 (操作前)




(c) 管理员主页 (操作后)




(d) 普通用户主页 (操作后)

图 2: 新图书入库前后主页对比

- 以下为将《数据库系统概念》添加 10 个副本的操作界面和主页截图对比：

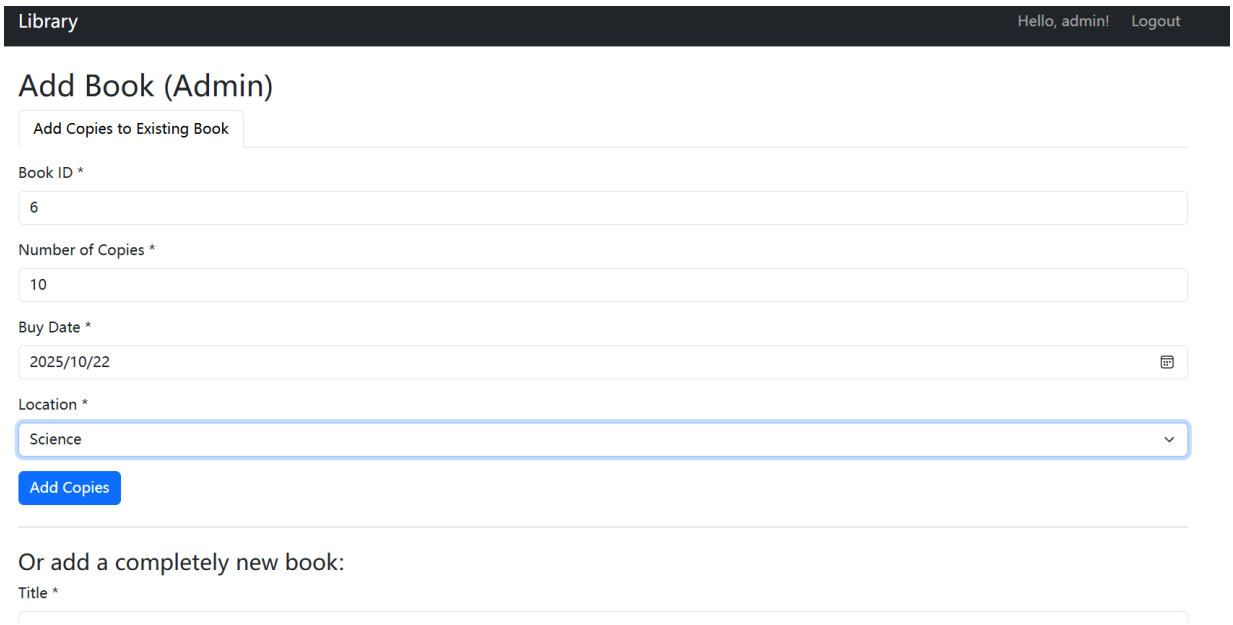
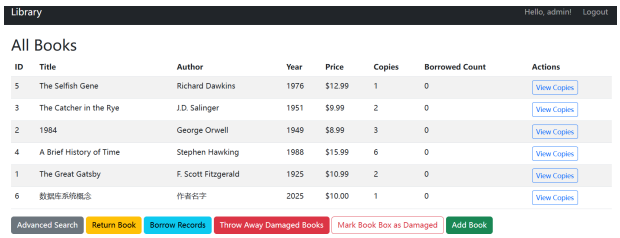
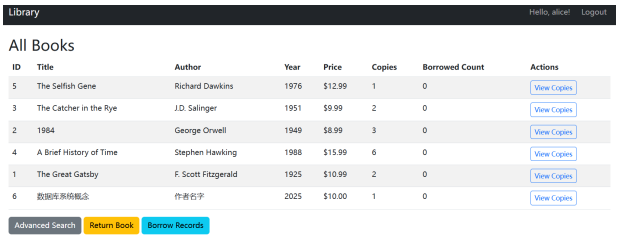


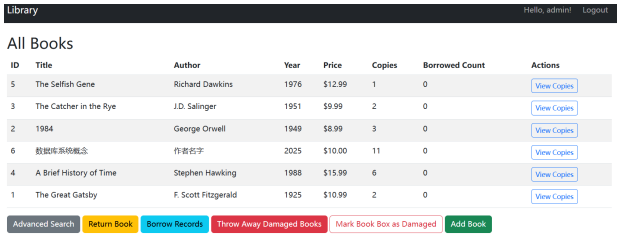

图 3: 图书入库操作



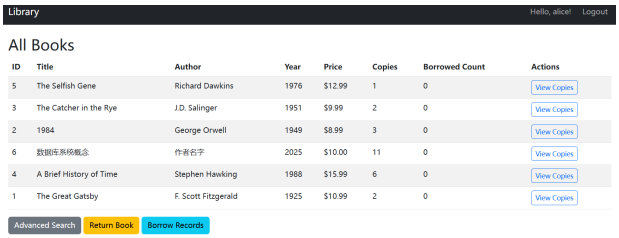

(a) 管理员主页 (操作前)




(b) 普通用户主页 (操作前)




(c) 管理员主页 (操作后)



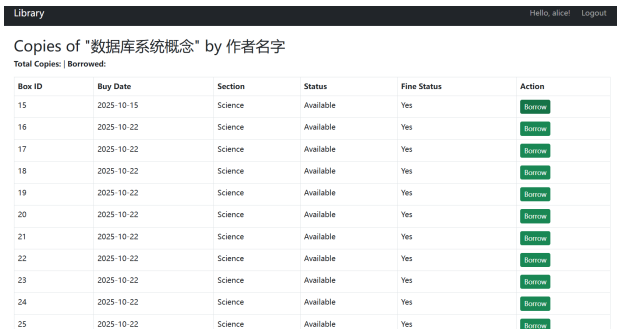

(d) 普通用户主页 (操作后)

图 4: 已有图书入库前后主页对比

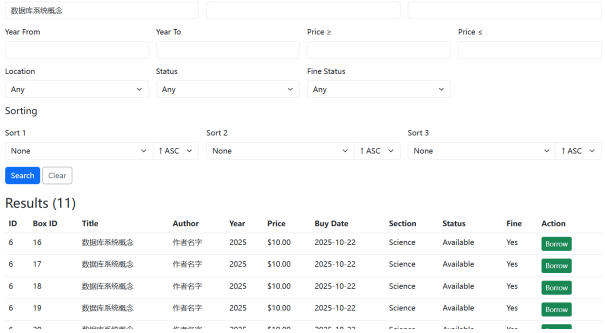
## 6.2 普通用户借阅与归还

借阅功能可以在图书详情页或高级检索页面完成，归还功能可以在主页点击归还按钮后进入归还页面完成。

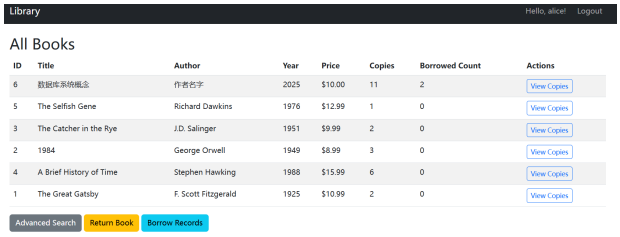
- 以下为普通用户借阅《数据库系统概念》的操作界面和部分截图 (为做演示，借两本书，一个在图书详情页，一个在高级检索页)：




(a) 从图书详情页借阅




(b) 从高级检索页借阅



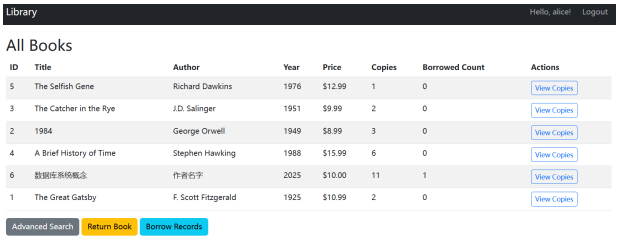

(c) 借书后的主页显示


(d) 借书后的详情页显示

图 5: 普通用户借阅图书操作

- 以下为普通用户归还《数据库系统概念》并标记为完好的操作界面和部分截图 (如果需要看归还记录，可见第6.3节)：


(a) 归还界面

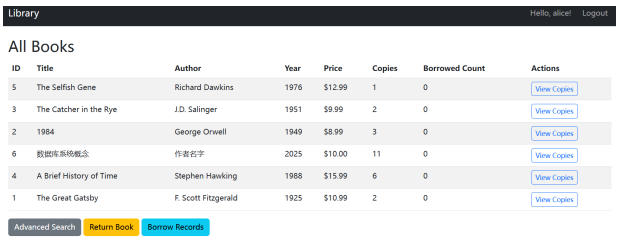



(b) 归还后的主页显示

图 6: 普通用户归还图书操作 (完好)

- 以下为普通用户归还《数据库系统概念》并标记为损坏的操作界面和部分截图 (如果需要看归还记录，可见第6.3节)：


(a) 归还界面




(b) 归还后的主页显示

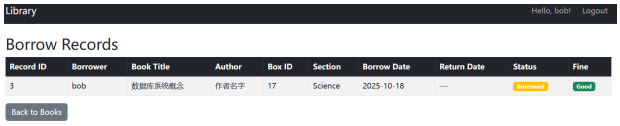
图 7: 普通用户归还图书操作 (损坏)

## 6.3 借阅记录查看

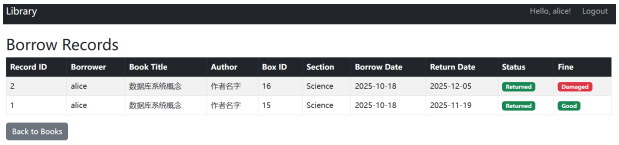
在第6.2节中，提到归还图书后可以在借阅记录中查看归还情况。借阅界面的每一个条目包含图书实例基本信息，是否归还，归还后是否完好的记录。下面展示普通用户和管理员查看借阅记录的界面截图。

**Hint:** 普通用户只能查看自己的借阅记录，而管理员可以查看所有用户的借阅记录，并可按用户名筛选。为验证功能，我们使用另一个普通用户进行借阅操作，以展示管理员查看所有记录的能力。

- 以下为用户 bob(普通用户) 和 alice(普通用户) 查看各自借阅记录的界面截图，其中 alice 在第6.2节中进行了借阅和归还操作，而 bob 则仅进行了借阅操作



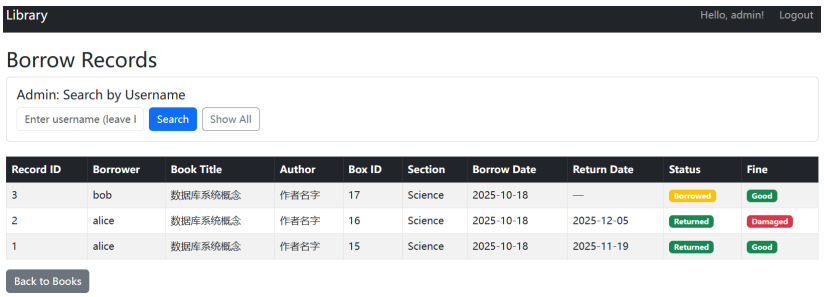

(a) 用户 bob 的借阅记录



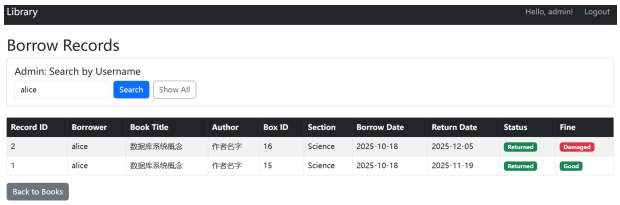

(b) 用户 alice 的借阅记录

图 8: 普通用户查看各自借阅记录

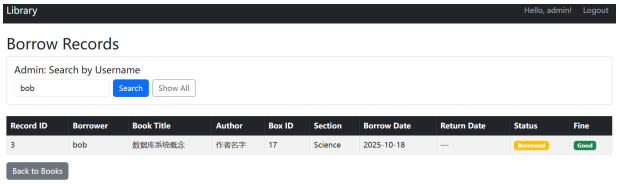
- 以下为管理员查看所有用户借阅记录的界面截图
- 管理员可以通过输入用户名筛选借阅记录，以查看特定用户的借阅情况




(a) 管理员查看所有用户的借阅记录




(b) 管理员筛选查看用户 alice 的借阅记录

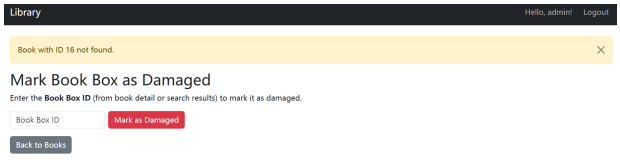



(c) 管理员筛选查看用户 bob 的借阅记录

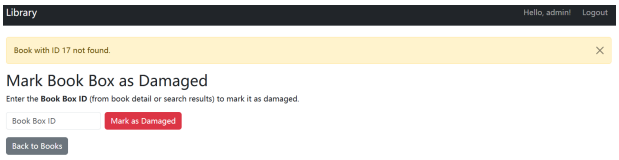
图 9: 管理员查看借阅记录

## 6.4 管理员标记损坏与批量淘汰

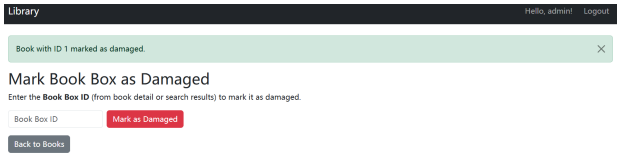
- 以下为管理员标记图书实例为损坏的操作界面和部分截图：
- 管理员只能标记未被借出且完好的图书实例为损坏，其余情况均会提示错误信息
- 注意：这里的 `id=16` 图书实例可以在第6.3节中看到是用户 alice 归还时标记为损坏的图书实例，用于验证管理员无法标记已损坏图书实例的功能。这里的 `id=17` 图书实例可以在第6.3节中看到是用户 bob 借阅但未归还的图书实例，用于验证管理员无法标记未归还图书实例的功能




(a) 标记已损坏的图书实例




(b) 标记未归还的图书实例

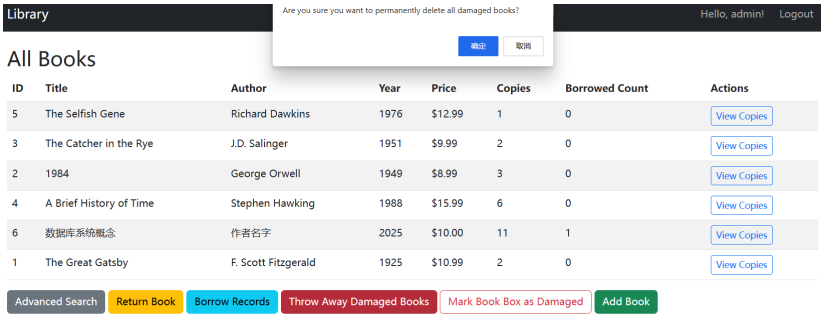



(c) 标记成功

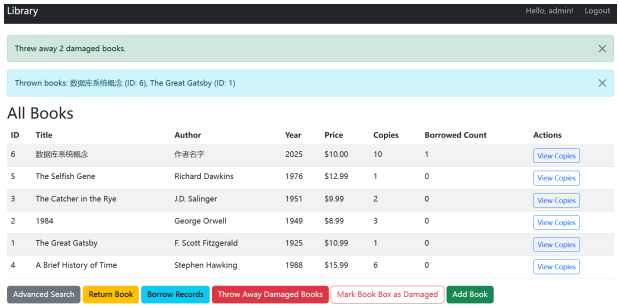

(d) 标记后的图书实例显示

图 10: 管理员标记图书实例为损坏操作

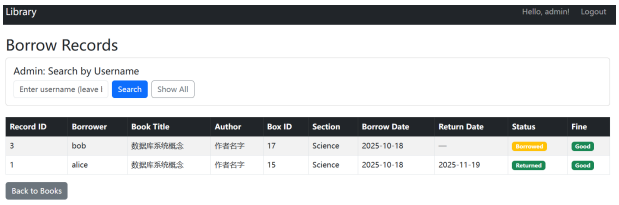
- 以下为管理员批量淘汰所有标记为损坏的图书实例的操作界面和淘汰前后截图对比：
- 注意：在前面的操作中，管理员标记了 `id=16` 和 `id=1` 图书实例为损坏，因此这两个图书实例会被批量淘汰
- 淘汰后，图书实例 `id=16` 和 `id=1` 不再出现在主页和借阅记录中




(a) 批量淘汰操作




(b) 淘汰后的主页显示

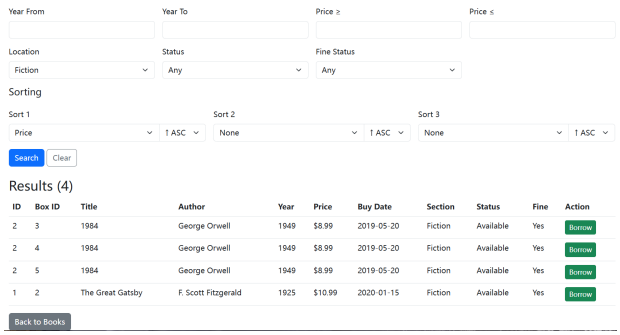



(c) 淘汰后的借阅记录显示

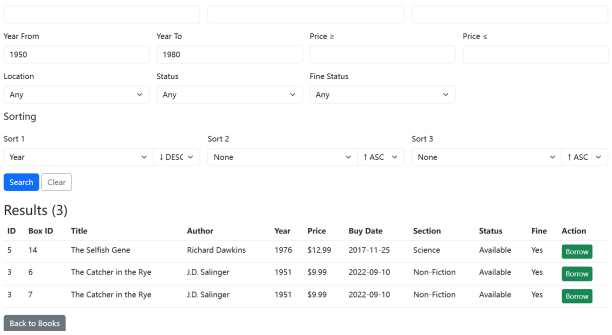
图 11: 管理员批量淘汰损坏图书实例操作

## 6.5 高级检索功能

- 以下为高级检索功能的操作界面和部分截图展示：
- 该功能支持多条件组合查询和三级排序，方便用户根据多种需求查找图书实例，在检索之后可以看到符合条件的图书实例列表以及总数量
- 由于数据量有限，较多的查询条件会导致无结果返回，因此这里只展示部分查询条件的效果作为示例



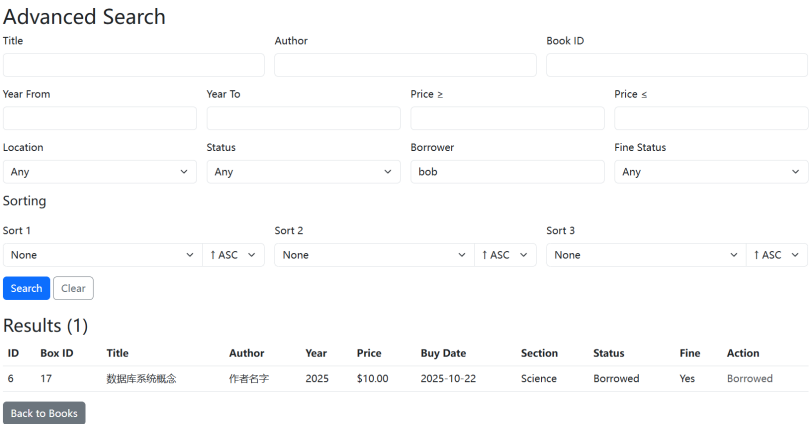

(a) 根据位置 location 进行筛选 + 根据价格从小到大排序




(b) 根据年份 year 进行筛选 + 根据年份从大到小排序

图 12: 高级检索功能展示 (普通用户)

- 管理员可以根据借阅人进行筛选，以查看某个用户当前借阅的图书实例




(a) 根据借阅人进行筛选

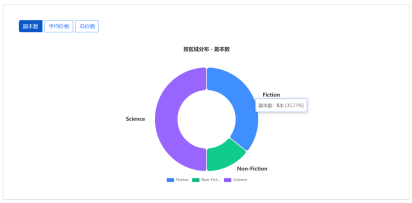
图 13: 高级检索功能展示 (管理员)

## 6.6 统计

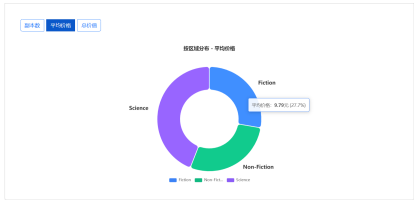
- 只有管理员可以查看统计信息，管理员通过点击界面 Library 跳转到统计信息页面，通过在 `stats` 页面不传入 `is_admin` 参数，再次点击 Library 会返回到主页。
- 为测试统计功能，重启 ui 服务，使用 alice 用户和 bob 用户分别借阅一本图书，并使用管理员用户设置一个书籍损坏
- 以下为管理员查看统计信息的操作界面和部分截图展示：


图 14: 整体统计信息

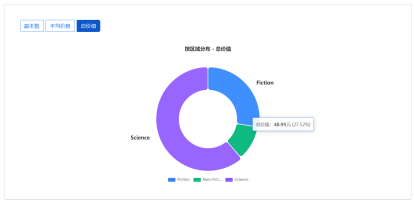
- 每一个统计方式都会对副本数量、平均价格和总价值进行统计。为节约篇幅，下面仅展示以 `location` 为例，展示所有统计目标的截图，其余只展示副本数量的统计图



(a) 对副本数量的统计图



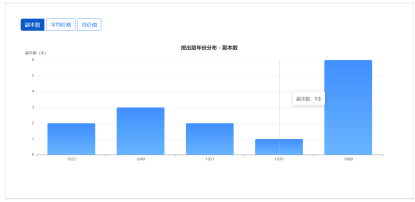
(b) 对平均价格的统计图



(c) 对总价值的统计图

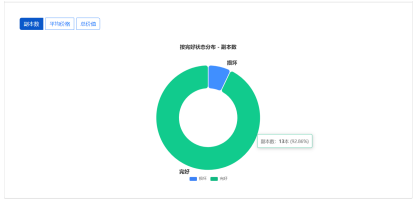
图 15: 按位置分组统计信息

(a) 按作者分组统计信息

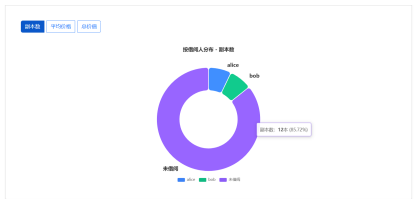


(b) 按年份分组统计信息

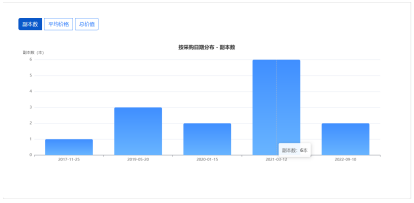
(c) 按是否借阅分组统计信息



(d) 按完好状态分组统计信息



(e) 按借阅人分组统计信息



(f) 按采购日期分组统计信息

图 16: 其他分组统计信息展示

# 7 总结

该项目实现了一个功能完整的图书管理系统，涵盖了图书的添加、借阅、归还、损坏标记与淘汰等核心功能。通过使用 openGauss 数据库和 Flask 框架，项目展示了数据库设计与应用开发的综合能力。

# 8 附录

## 8.1 数据库配置流程

- 使用 vmware 创建 Linux 虚拟机，使用 Ubuntu 22.04.5 LTS 版本
- 安装 docker

```
#安装前先卸载操作系统默认安装的docker,
sudo apt-get remove docker docker-engine docker.io containerd runc

#安装必要支持
sudo apt install apt-transport-https ca-certificates curl software-properties-common
    gnupg lsb-release

# 阿里源
curl -fsSL https://mirrors.aliyun.com/docker-ce/linux/ubuntu/gpg | sudo gpg --dearmor -o
    /usr/share/keyrings/docker-archive-keyring.gpg
echo "deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/keyrings/docker-archive
    -keyring.gpg] https://mirrors.aliyun.com/docker-ce/linux/ubuntu $(lsb_release -cs)
    stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null

#安装最新版本的Docker
sudo apt install -y docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-
    compose-plugin

# 配置 docker 国内镜像源
sudo mkdir -p /etc/docker
sudo tee /etc/docker/daemon.json <<EOF
{
    "registry-mirrors": [
        "https://docker.xuanyuan.me"
    ]
}
EOF
# 重启docker服务
sudo systemctl daemon-reload
sudo systemctl restart docker
```

- 安装 openGauss 数据库，设置端口号为 15432，设置密码为 Admin@123456，数据存储路径为 /data/opengauss/mysql

```
mkdir -p /data/opengauss/
docker run --name opengauss -d --restart always --privileged=true  -e GS_PASSWORD=
    Admin@123456  \
    -v /data/opengauss/mysql:/var/lib/opengauss  -u root -p 15432:5432 \
    enmotech/opengauss:latest
```

- 连接 openGauss 数据库

```
docker exec -it 28eface690e8 /bin/bash # 进入容器，容器ID根据实际情况填写
su - omm
gsql
```

- 创建数据库和用户，并设置 schema 为 mysql

```
-- 创建用户 luyang2008，密码为 Admin@123456
CREATE USER luyang2008 WITH PASSWORD 'Admin@123456';
-- 授予用户创建数据库的权限
ALTER USER luyang2008 createdb;

-- 进入数据库
gsql -d postgres -U luyang2008 -W 'Admin@123456' -r
-- 创建数据库 mydatabase，拥有者是 luyang2008
CREATE DATABASE mydatabase OWNER luyang2008;
-- 重进数据库
gsql -d mydatabase -U luyang2008 -W 'Admin@123456' -r
-- 创建 schema（模式），名字叫 mysql
CREATE SCHEMA mysql;

-- 设置默认 schema 为 mysql
ALTER USER luyang2008 SET search_path TO mysql;
```

- 此时，数据库配置完成，可以通过 psycopg2 库连接数据库，连接方式如下：

```
import psycopg2

conn = psycopg2.connect(
    host="192.168.88.140", # 根据实际情况填写虚拟机IP地址
    port="15432",
    database="mydatabase",
    user="luyang2008",
    password="Admin@123456"
)
```