

课程名称：当代数据管理系统	指导教师：周烜	上机实践名称： Bookstore
姓名：邓博昊	学号：10225501432	年级： 2022

## 1.实验要求

### 功能

- 1人完成下述内容：
  - 允许向接口中增加或修改参数，允许修改 HTTP 方法，允许增加新的测试接口，请尽量不要修改现有接口的 url 或删除现有接口，请根据设计合理的拓展接口（加分项+2 ~ 5分）。 测试程序如果有问题可以提bug （加分项，每提1个 bug +2, 提1个 pull request +5）。
  - 核心数据使用关系型数据库（PostgreSQL 或 MySQL 数据库）。 blob 数据（如图片和大段的文字描述）可以分离出来存其它 NoSQL 数据库或文件系统。
  - 对所有的接口都要写 test case，通过测试并计算代码覆盖率（有较高的覆盖率是加分项 +2~5）。
  - 尽量使用正确的软件工程方法及工具，如，版本控制，测试驱动开发 （利用版本控制是加分项 +2~5）
  - 后端使用技术，实现语言不限； **不要复制**这个项目上的后端代码（不是正确的实践， 减分项 -2~5）
  - 不需要实现页面
  - 最后评估分数时考虑以下要素：
    - 实现完整度，全部测试通过，效率合理
    - 正确地使用数据库和设计分析工具，ER图，从ER图导出关系模式，规范化，事务处理，索引等
    - 其它...

### bookstore目录结构

1	bookstore	
2	-- be	后端
3	-- model	后端逻辑代码
4	-- view	访问后端接口
5	-- ....	
6	-- doc	JSON API规范说明
7	-- fe	前端访问与测试代码
8	-- access	
9	-- bench	效率测试
10	-- data	
11	-- book.db	
12	-- scraper.py	从豆瓣爬取的图书信息数据的代码
13	-- test	功能性测试（包含对前60%功能的测试，不要修改已有的文件，可以提pull request或bug）
14	-- conf.py	测试参数，修改这个文件以适应自己的需要
15	-- conftest.py	pytest初始化配置，修改这个文件以适应自己的需要
16	-- ....	
17	-- ....	

## 2.MongoDB向Postgres迁移

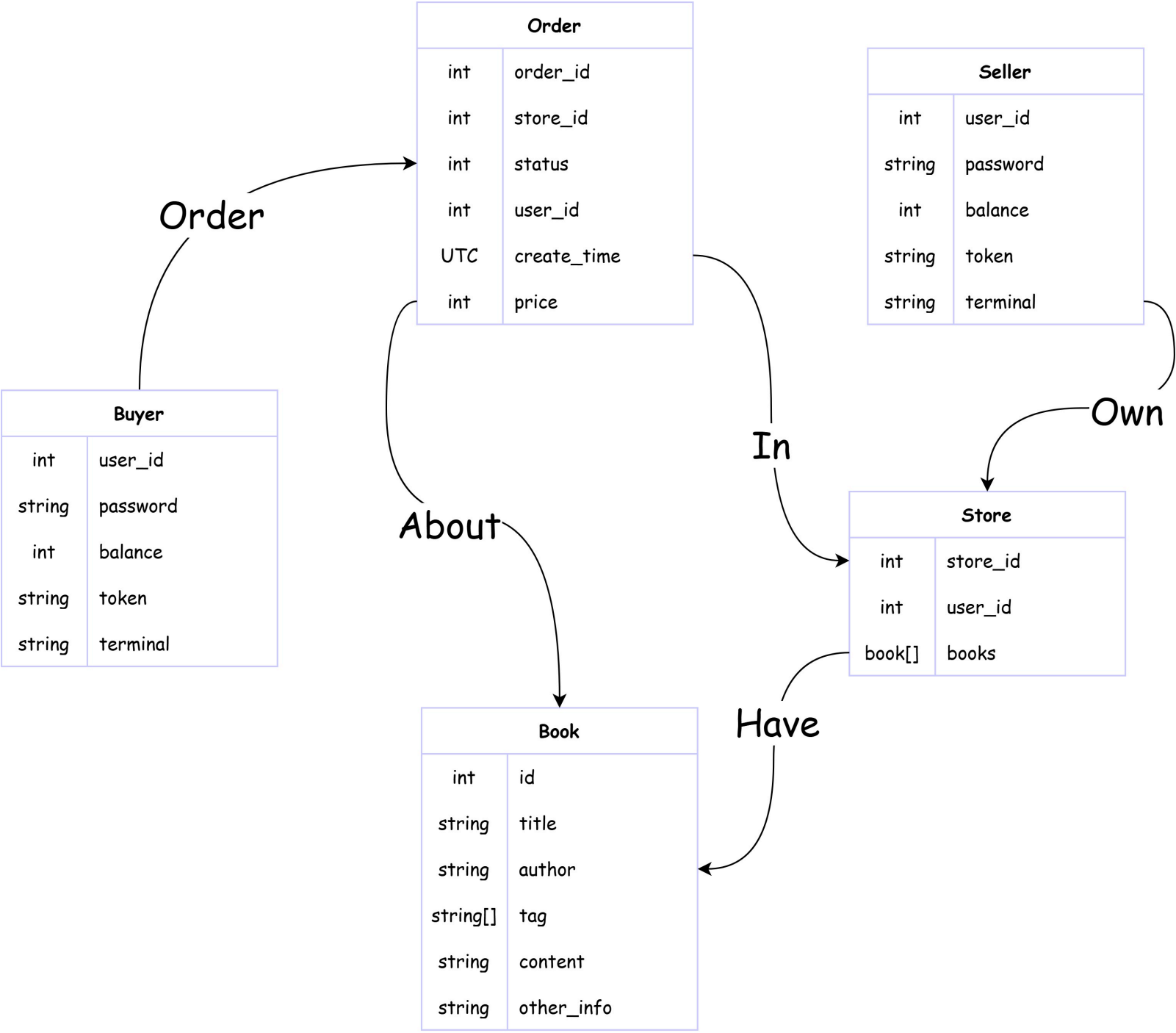
原项目为Homework1，其使用的核心数据库为 MongoDB ,现将其迁移到 Postgres 上

只需要改变后端 model 即可，其余可不变

## 3.前期分析与数据库设计与前期分析

### 3.1 数据库设计

#### 3.1.1 数据库逻辑设计



3.1.2 数据库结构设计

表：user

字段名	数据类型	约束	说明
user_id	TEXT	主键	用户ID
password	TEXT	非空	用户密码
balance	NUMERIC	非空	用户余额
token	TEXT		用户登录令牌
terminal	TEXT		用户终端信息

表：book

字段名	数据类型	约束	说明
id	SERIAL	主键	自增ID
book_id	TEXT	非空	书籍ID
title	TEXT	非空	书名
author	TEXT		作者
publisher	TEXT		出版社
content	TEXT		内容
original_title	TEXT		原书名
translator	TEXT		译者
pub_year	TEXT		出版年份
pages	INTEGER		页数
price	INTEGER		价格
currency_unit	TEXT		货币单位

字段名	数据类型	约束	说明
binding	TEXT		装帧
isbn	TEXT		ISBN号
author_intro	TEXT		作者简介
book_intro	TEXT		书籍简介
tags	JSONB		标签 (JSON数组)
pictures	JSONB		图片 (JSON数组)

表：store

字段名	数据类型	约束	说明
store_id	TEXT	主键	商店ID
user_id	TEXT	外键 (user 表)	用户ID
books	JSONB		书籍信息 (JSONB)

表：order

字段名	数据类型	约束	说明
id	SERIAL	主键	自增ID
order_id	TEXT		订单ID
store_id	TEXT		商店ID
user_id	TEXT		用户ID
create_time	TIMESTAMP	默认当前时间	订单创建时间
price	NUMERIC	非空	订单总价
status	TEXT	非空	订单状态

表：order\_detail

字段名	数据类型	约束	说明
id	SERIAL	主键	自增ID
order_id	TEXT		订单ID
book_id	TEXT		书籍ID
price	NUMERIC	非空	书籍单价
count	INTEGER	非空	书籍数

### 3.1.3索引优化

为了对数据库进行优化，可以对那些不怎么更改，但是需要经常查找的项建立索引，有了索引的存在就可以加快查找速度，所以设置了以下索引。

- **store 表中的 store\_id 上设置了唯一升序索引：**
  - store\_id 是商店的唯一标识符，经常用于查询商店信息。通过在该字段上创建唯一升序索引，可以快速定位到特定的商店记录。
- **user 表中的 user\_id 上设置了唯一升序索引：**
  - user\_id 是用户的唯一标识符，经常用于用户登录、查询用户信息等操作。通过在该字段上创建唯一升序索引，可以快速定位到特定用户记录，提升查询效率。
- **books 表中设置了多个索引：**
  - **title 字段上的普通索引：**title 是书籍的标题，经常用于根据书名进行搜索。通过在 title 上创建普通索引，可以加快基于书名的查询速度。
  - **tags 字段上的 GIN 索引：**tags 是一个 JSONB 类型的字段，存储书籍的标签信息。
  - **book\_intro 字段上的 GIN 索引：**book\_intro 是书籍的简介，通常包含较长的文本信息。

3.2 前期分析

3.2.1 文件树分析

```
1 (.venv) ~\Desktop\大三上\当代数据管理系统\Homework\大作业\CDMS.Xuan_ZHOU.2024Fall.DaSE\project1\bookstore git:
2 [dev-dbh]
3 tree /A
4 Folder PATH listing for volume win11Prow x64
5 Volume serial number is 16B4-98DD
6 C:.\
7 +---be # 后端文件夹，处理实际的业务逻辑
8 |   +---model # 数据模型，主要为实际运行时的内部逻辑与sqlite数据库读写，需要重构为操作Mongodb
9 |   |   \---__pycache__
10 |   +---view # flask的后端逻辑
11 |   |   \---__pycache__
12 |   \---__pycache__
13 +---doc
14 +---fe # 前端文件夹，主要存放测试和调用后端相关函数(模拟浏览器行为)
15 |   +---access # 规定了前端如何调用后端，基本上不需要改动了
16 |   |   \---__pycache__
17 |   +---bench # 测试打分相关
18 |   |   \---__pycache__
19 |   +---data # 存放的sqlite数据，需要后面转到Mongodb上
20 |   +---test # 用于pytest框架的测试函数
21 |   |   \---__pycache__
22 |   \---__pycache__
23 \---script # 存放测试脚本
```

3.2.2 业务逻辑分析

一次测试中标准的业务流程如下

- 1. 搜索测试文件
  - Flask 框架
    - 搜索当前文件树下的测试文件
      - 规则：
        - 以 test\_ 开头
        - 以 \_test 结尾
      - 包含特定目录中的测试：
        - /fe/bench
- 2. 测试文件执行逻辑
  - 测试文件调用
    - /fe/access 中的前端逻辑
    - 前端执行逻辑：
      - 向后端发送 HTTP 指令
- 3. 后端逻辑
  - 接收前端 HTTP 指令
    - 后端通过 /be/view 中的函数解析 HTTP 参数
    - 执行解析包含的指令
- 4. 后端调用
  - /be/model 中的对应函数

3.2.3 对比分析

PostgreSQL和MongoDB对比

MySQL 术语	MongoDB 术语	解释
Database (数据库)	Database (数据库)	两者中都使用相同术语，表示数据库的集合。
Table (表)	Collection (集合)	在MySQL中称为表，在MongoDB中称为集合，存储数据记录的容器。

MySQL 术语	MongoDB 术语	解释
Row (行)	Document (文档)	在MySQL中每行代表一条记录，而在MongoDB中每个文档是类似的概念，包含数据的键值对。
Column (列)	Field (字段)	MySQL的列对应MongoDB的字段，字段表示文档中键值对的键。
Primary Key (主键)	<code>_id</code> (唯一标识符)	MySQL中的主键在MongoDB中对应每个文档的 <code>_id</code> 字段，默认由MongoDB生成唯一ID。
Index (索引)	Index (索引)	两者都用索引来加速查询，但MongoDB的索引可以是嵌套的字段或数组。
Schema (模式)	Schema-less (无模式)	MySQL有固定模式，MongoDB是无模式的，即文档的结构不必一致。
JOIN (连接)	Embedded Documents (嵌入文档) 或 <code>\$lookup</code>	MongoDB没有直接的JOIN，但可以通过嵌入文档或 <code>\$lookup</code> 操作实现类似的功能。
SQL (查询语言)	MongoDB Query (MongoDB查询) 或 Aggregation Pipeline (聚合管道)	MySQL使用SQL语言，MongoDB有自己的查询语法和聚合框架。
Transaction (事务)	Transaction (事务)	MongoDB 4.0+版本支持多文档ACID事务，类似MySQL的事务。
Foreign Key (外键)	Reference (引用)	MongoDB没有明确的外键概念，但可以通过引用文档的方式实现类似的功能。
View (视图)	View (视图)	MongoDB 3.4+版本支持视图，与MySQL的视图类似。

3.2.4 初次运行

请将Flask降级到2.0.0

将 `~/script/test.sh` 改成 `test.bat`

运行提示，请在项目根目录下运行

```
1  @echo off
2  setlocal
3
4  echo 设置当前目录为 PYTHONPATH
5  set PYTHONPATH=%cd%
6
7  echo 运行 coverage，并指定路径和参数
8  coverage run --timid --branch --source=fe,be --concurrency=thread -m pytest -v --ignore=fe\data
9
10 echo 合并覆盖率数据
11 coverage combine
12
13 echo 生成覆盖率报告
14 coverage report
15
16 echo 生成 HTML 覆盖率报告
17 coverage html
18
19 endlocal
20
```

然后运行

可以看到 `./be/model/user.py` 中的 `jwt_encode` 出了问题

```
fe/test/test_add_book.py::TestAddBook::test_ok 2024-10-12 10:48:59.519 [Thread-2 (pr)] [ERROR] Exception on /auth/register [POST]
Traceback (most recent call last):
  File "C:\Users\Administrator\Desktop\大三上\当代数据管理系统\Homework\大作业\CDMS.Xuan_ZHOU.2024Fall.DaSE\project1\bookstore\venv\Lib\site-packages\flask\app.py", line 2051, in wsgi_app
    response = self.full_dispatch_request()
                ^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "C:\Users\Administrator\Desktop\大三上\当代数据管理系统\Homework\大作业\CDMS.Xuan_ZHOU.2024Fall.DaSE\project1\bookstore\venv\Lib\site-packages\flask\app.py", line 1501, in full_dispatch_request
    rv = self.handle_user_exception(e)
         ^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "C:\Users\Administrator\Desktop\大三上\当代数据管理系统\Homework\大作业\CDMS.Xuan_ZHOU.2024Fall.DaSE\project1\bookstore\venv\Lib\site-packages\flask\app.py", line 1499, in full_dispatch_request
    rv = self.dispatch_request()
         ^^^^^^^^^^^^^^^^^^^^^^
  File "C:\Users\Administrator\Desktop\大三上\当代数据管理系统\Homework\大作业\CDMS.Xuan_ZHOU.2024Fall.DaSE\project1\bookstore\venv\Lib\site-packages\flask\app.py", line 1485, in dispatch_request
    return self.ensure_sync(self.view_functions[rule.endpoint])(**req.view_args)
           ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "C:\Users\Administrator\Desktop\大三上\当代数据管理系统\Homework\大作业\CDMS.Xuan_ZHOU.2024Fall.DaSE\project1\bookstore\be\view\auth.py", line 35, in register
    code, message = u.register(user_id=user_id, password=password)
                    ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "C:\Users\Administrator\Desktop\大三上\当代数据管理系统\Homework\大作业\CDMS.Xuan_ZHOU.2024Fall.DaSE\project1\bookstore\be\model\user.py", line 59, in register
    token = jwt_encode(user_id, terminal)
            ^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "C:\Users\Administrator\Desktop\大三上\当代数据管理系统\Homework\大作业\CDMS.Xuan_ZHOU.2024Fall.DaSE\project1\bookstore\be\model\user.py", line 22, in jwt_encode
    return encoded.decode("utf-8")
           ^^^^^^^^^^^^^^
```

查看 `jwt_encode`

```
1 def jwt_encode(user_id: str, terminal: str) -> str:
2     encoded = jwt.encode(
3         {"user_id": user_id, "terminal": terminal, "timestamp": time.time()},
4         key=user_id,
5         algorithm="HS256",
6     )
7     return encoded.decode("utf-8")
```

查询网络后发现，在 pyjwt 的较早版本中，`jwt.encode` 返回的是字节对象，需要通过 `.decode("utf-8")` 将其转换为字符串。

从2.0版本开始，`jwt.encode` 直接返回字符串，因此不再需要进行解码。

只需要将 `jwt_encode` 函数中的 `.decode("utf-8")` 移除。修改后的函数如下：

```
1 def jwt_encode(user_id: str, terminal: str) -> str:
2     encoded = jwt.encode(
3         {"user_id": user_id, "terminal": terminal, "timestamp": time.time()},
4         key=user_id,
5         algorithm="HS256",
6     )
7     return encoded # 不再需要 decode
```

可以看到上述问题已解决

```
===== test session starts =====
platform win32 -- Python 3.12.6, pytest-8.3.3, pluggy-1.5.0 -- C:\Users\Administrator\Desktop\大三上\当代数据管理系统\Homework\大作业\CDMS.Xuan_ZHOU.2024Fall.DaSE\project1\bookstore\
...venv\Scripts\python.exe
cachedir: .pytest_cache
rootdir: C:\Users\Administrator\Desktop\大三上\当代数据管理系统\Homework\大作业\CDMS.Xuan_ZHOU.2024Fall.DaSE\project1\bookstore
collecting ... frontend begin test
* Serving Flask app 'be.serve' (lazy loading)
* Environment: production
  WARNING: This is a development server. Do not use it in a production deployment.
  Use a production WSGI server instead.
* Debug mode: off
2024-10-12 11:11:41,563 [Thread-1 (ru)] [INFO] * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
collected 33 items

fe/test/test_add_book.py::TestAddBook::test_ok 2024-10-12 11:11:41,745 [Thread-2 (pr)] [INFO] 127.0.0.1 - - [12/Oct/2024 11:11:41] "POST /auth/register HTTP/1.1" 200 -
2024-10-12 11:11:41,758 [Thread-3 (pr)] [INFO] 127.0.0.1 - - [12/Oct/2024 11:11:41] "POST /auth/login HTTP/1.1" 200 -
2024-10-12 11:11:41,770 [Thread-4 (pr)] [INFO] 127.0.0.1 - - [12/Oct/2024 11:11:41] "POST /seller/create_store HTTP/1.1" 200 -
ERROR [ 3%]
```

同时整体修改思路如下(仅基础功能)

- 前端 `fe` 不需要更改，其模拟的是浏览器行为，存放设计好的测试函数
- 后端 `be/view` 不需要更改，其为后端 `Flask` 处理HTTP请求并调用处理核心 `be/model` 部分
- 后端 `be/model/error.py` 不需要更改，预先设计好的错误代码
- 其余 `be/model/` 下的文件需要更改

## 4.基本功能实现

从上次的实验一代码，将MongoDB为核心改为以Postgres为核心

### 4.1 buyer部分实现

#### 4.1.1 new\_order

函数输入：

- `user_id`：字符串，表示用户的唯一标识。
- `store_id`：字符串，表示商店的唯一标识。
- `id_and_count`：列表，包含多个元组，每个元组由书本的 ID 和其对应的数量组成。

函数输出：

- 整数：状态码，200 表示操作成功，528 表示错误。
- 字符串：描述操作结果的消息，可能是成功或错误信息。
- 字符串：表示订单的 ID。

函数流程：

1. 初始化一个空字符串 `order_id`，用于后续存储订单 ID。
2. 检查用户和商店的存在性。如果任一不存在，函数将返回相应的错误消息和空的订单 ID。
3. 生成一个唯一的订单 ID `uid`，该 ID 由用户 ID、商店 ID 及一个基于当前时间的唯一标识符组成。
4. 遍历 `id_and_count` 列表中的每个书本 ID 及其数量，执行以下操作：
  - 查询商店的库存，确认书本的存在性。如果书本未找到，将返回相应的错误信息和空的订单 ID。
  - 检查库存量，若库存不足，返回库存不足的错误信息和空的订单 ID。
  - 如果库存充足，更新库存，减少相应书本的数量。
  - 将书本的订单详细信息记录到 `order_detail` 表中，并计算订单的总价。



- 5. 在计算完总价后，函数获取当前时间，将订单的详细信息插入到 `order` 表中，包括订单 ID、商店 ID、用户 ID、创建时间、总价格及订单状态。
- 6. 如果所有步骤均成功，函数返回状态码 200 表示成功以及生成的订单 ID。
- 7. 如果在执行过程中捕获到异常，会记录相关日志，并返回 528 表示错误，同时附带异常信息作为错误消息，并返回空的订单 ID。

4.1.2 payment

函数输入：

- `user_id`：用户标识。
- `password`：用户密码。
- `order_id`：订单唯一标识。

函数输出：

- 整数：状态码，200 表示成功，528 表示发生错误。
- 字符串：操作结果的描述。

函数流程：

- 1. 查询 `order` 表中与给定订单 ID 对应的订单信息，状态为 0（未付款）。如果未找到有效的订单，将返回一个错误消息，提示无效的订单 ID。
- 2. 如果找到订单信息，则提取买家的 ID、商店 ID 和订单的总价。
- 3. 验证用户 ID 是否与订单的买家 ID 一致，如果不匹配，返回授权失败的错误消息。
- 4. 查询 `user` 表中买家的信息，确认用户存在并核对密码。如果出现错误，返回授权失败的消息。
- 5. 查询 `store` 表中商店信息，确保商店存在。如果找不到，将返回相应的错误消息。
- 6. 从商店信息中提取卖家的 ID，检查卖家是否存在。如果未找到，返回错误消息。
- 7. 检查用户余额，如果余额不足，返回余额不足的错误消息。
- 8. 如果余额充足，从买家的账户中扣除订单总价，并将该金额添加到卖家的账户中。
- 9. 更新 `order` 表中订单状态为 1（已支付）。
- 10. 如果执行过程中发生任何异常，返回状态码 528。

4.1.3 add\_funds

函数输入：

- `user_id`：用户标识。
- `password`：用户密码。
- `add_value`：增加到账户余额的金额。

函数输出：

- 整数：状态码，200 表示成功，528 表示发生错误。
- 字符串：描述操作结果的消息。

函数流程：

- 1. 查询 `user` 表中与给定用户 ID 对应的用户信息。如果未找到，返回授权失败的错误消息。
- 2. 如果找到信息，验证输入的密码是否与数据库中的密码一致。如果不匹配，返回授权失败的错误提示。
- 3. 更新 `user` 表中用户的余额，将 `add_value` 添加到当前余额。
- 4. 如果更新失败，返回用户不存在的错误消息。
- 5. 如果在执行过程中发生任何异常，返回状态码 528。

4.1.4 改动的理由：

- 便于编写业务逻辑代码：
  - PostgreSQL 支持复杂的 SQL 查询和事务处理，能够简化业务逻辑代码的编写。
- 数据一致性：
  - PostgreSQL 的 ACID 事务能够确保数据的一致性，避免数据不一致的问题。
- 扩展性：
  - PostgreSQL 支持多种数据类型和扩展，能够满足复杂的业务需求。

## 4.2 seller部分实现

### 4.2.1 创建商铺

函数输入：

- `user_id`：卖家用户 ID。
- `store_id`：商铺 ID。

函数输出：

- 整数：状态码，200 表示成功，530 表示发生错误。
- 字符串：描述操作结果的消息。

函数流程：

1. 检查给定的 `user_id` 是否存在。如果不存在，返回一个错误，表示用户 ID 不存在。
2. 检查给定的 `store_id` 是否存在。如果存在，返回一个错误，表示商店 ID 已存在。
3. 使用 `INSERT INTO` 语句向 `store` 表中插入新商店。商店的基本信息包括：
  - `store_id`：新商店的唯一标识符。
  - `user_id`：关联的用户 ID。
  - `books`：初始化为空的 JSONB 数组，表示该商店当前没有任何书籍。
4. 如果在执行数据库操作时发生异常，返回错误代码 530 和错误信息。
5. 如果所有操作成功完成，返回状态码 200 和消息 `"ok"`，表示操作成功。

### 4.2.2 添加书籍信息

函数输入：

- `user_id`：卖家用户 ID。
- `store_id`：商铺 ID。
- `book_id`：书籍 ID。
- `book_json_str`：表示包含书籍信息的 JSON 字符串。
- `stock_level`：库存数量。

函数输出：

- 整数：状态码，200 表示成功，530 表示发生错误。
- 字符串：描述操作结果的消息。

函数流程：

1. 检查给定的 `user_id` 是否存在。如果不存在，返回一个错误，表示用户 ID 不存在。
2. 检查给定的 `store_id` 是否存在。如果不存在，返回一个错误，表示商店 ID 不存在。
3. 检查给定的 `book_id` 是否已存在于商店中。如果存在，返回一个错误，表示书籍 ID 已存在。
4. 解析书籍信息的 JSON 字符串，并将其插入到 `book` 表中。
5. 使用 `jsonb_insert` 函数将书籍 ID 和库存数量添加到商店的 `books` 字段中。
6. 如果在执行数据库操作时发生异常，返回错误代码 530 和错误信息。
7. 如果所有操作成功完成，返回状态码 200 和消息 `"ok"`，表示操作成功。

### 4.2.3 添加书籍库存

函数输入：

- `user_id`：卖家用户 ID。
- `store_id`：商铺 ID。
- `book_id`：书籍 ID。
- `add_stock_level`：增加的库存数量。

函数输出：

- 整数：状态码，200 表示成功，528 或 530 表示发生错误。
- 字符串：描述操作结果的消息。

函数流程：

1. 检查给定的 `user_id` 是否存在。如果不存在，返回一个错误，表示用户 ID 不存在。
2. 检查给定的 `store_id` 是否存在。如果不存在，返回一个错误，表示商店 ID 不存在。
3. 检查给定的 `book_id` 是否存在于商店中。如果不存在，返回一个错误，表示书籍 ID 不存在。
4. 使用 `jsonb_set` 函数更新商店中指定书籍的库存数量。



- 如果在执行数据库操作时发生 `psycopg2.Error` 异常，返回错误代码 528 和错误信息。如果发生其他类型的异常，返回错误代码 530 和错误信息。
- 如果所有操作成功完成，返回状态码 200 和消息 `"ok"`，表示操作成功。

## 4.3 user部分实现

### 4.3.1 注册

函数输入：

- `user_id`：用户 ID。
- `password`：用户密码。

函数输出：

- 整数：状态码，200 表示成功，528 表示发生错误。
- 字符串：描述操作结果的消息。

函数流程：

- 生成一个唯一的终端标识，格式为 `terminal_<当前时间戳>`。
- 使用 `jwt_encode` 函数生成一个 JWT Token。
- 将用户信息插入到 `user` 表中，包括 `user_id`、`password`、`balance`、`token` 和 `terminal`。
- 如果在执行数据库操作时发生异常，返回错误代码 528 和错误信息。
- 如果操作成功，返回状态码 200 和消息 `"ok"`。

### 4.3.2 登录

函数输入：

- `user_id`：用户 ID。
- `password`：用户密码。
- `terminal`：用户终端信息。

函数输出：

- 整数：状态码，200 表示成功，528 表示发生错误。
- 字符串：描述操作结果的消息。
- 字符串：生成的 JWT Token。

函数流程：

- 调用 `check_password` 函数验证用户提供的密码是否正确。
- 如果密码验证失败，返回相应的错误代码和消息。
- 使用 `jwt_encode` 函数生成一个新的 JWT Token。
- 更新 `user` 表中的 `token` 和 `terminal` 字段。
- 如果在执行数据库操作时发生异常，返回错误代码 528 和错误信息。
- 如果操作成功，返回状态码 200、消息 `"ok"` 以及生成的 Token。

### 4.3.3 登出

函数输入：

- `user_id`：用户 ID。
- `token`：用户提供的 Token。

函数输出：

- 整数：状态码，200 表示成功，528 表示发生错误。
- 字符串：描述操作结果的消息。

函数流程：

- 调用 `check_token` 函数验证用户提供的 Token 是否有效。
- 如果 Token 验证失败，返回相应的错误代码和消息。
- 生成一个新的虚拟终端标识和 Token。
- 更新 `user` 表中的 `token` 和 `terminal` 字段。
- 如果在执行数据库操作时发生异常，返回错误代码 528 和错误信息。
- 如果操作成功，返回状态码 200 和消息 `"ok"`。

4.3.4 注销

函数输入：

- `user_id`：用户 ID。
- `password`：用户密码。

函数输出：

- 整数：状态码，200 表示成功，530 表示发生错误。
- 字符串：描述操作结果的消息。

函数流程：

1. 调用 `check_password` 函数验证用户提供的密码是否正确。
2. 如果密码验证失败，返回相应的错误代码和消息。
3. 从 `user` 表中删除用户记录。
4. 如果在执行数据库操作时发生异常，返回错误代码 530 和错误信息。
5. 如果操作成功，返回状态码 200 和消息 `"ok"`。

4.3.5 改密

函数输入：

- `user_id`：用户 ID。
- `old_password`：旧密码。
- `new_password`：新密码。

函数输出：

- 整数：状态码，200 表示成功，528 表示发生错误。
- 字符串：描述操作结果的消息。

函数流程：

1. 调用 `check_password` 函数验证用户提供的旧密码是否正确。
2. 如果旧密码验证失败，返回相应的错误代码和消息。
3. 生成一个新的终端标识和 Token。
4. 更新 `user` 表中的 `password`、`token` 和 `terminal` 字段。
5. 如果在执行数据库操作时发生异常，返回错误代码 528 和错误信息。
6. 如果操作成功，返回状态码 200 和消息 `"ok"`。

修改后端 `model/store.py`

原有 `store.py` 关联的是SQLite数据库


被调用链

1. `be/model/` 中的类初始化时  
调用 `db_conn.DBConn` 的初始化函数

```
class User(db_conn.DBConn):
    token_lifetime: int = 3600 # 3600 second

    👤 zty08111220
    def __init__(self):
        db_conn.DBConn.__init__(self)
```

2. `be/model/db_conn.DBConn` 初始化时  
调用 `store.get_db_conn()` 函数
3. `store.get_db_conn()` 调用 `database_instance.get_db_conn()` 函数  
`database_instance` 为 `Store` 类的一个实例

1 个用法  zty08111220

```
def get_db_conn(self) -> sqlite.Connection:
    return sqlite.connect(self.database)
```

修改方向

- 1. 全部改为self.xxx，作为类的属性，方便调用
- 2. 删除database: str，因为不再是SQLite,显然返回的类型应该为 Map 类型
- 3. get\_db\_conn(self) 返回类自己即可
- 4. 删除 db\_path
- 5. init\_database 在 serve.py 里被调用，原来参数为父文件夹目录，现在无参数输入，数据库初始化在 store() 类里完成

```
def get_db_conn():
    """
    获取数据库连接对象。如果数据库实例未初始化，则先初始化。

    :return: 数据库连接对象
    """
    global database_instance
    if database_instance is None:
        init_database()
    return database_instance.get_db_conn()
```

- 6. 创建关于图书标题，标签，简洁，内容的索引

```
1  import logging
2  import psycopg2
3  import psycopg2.extras
4  import threading
5  import os
6
7  logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(levelname)s - %(message)s')
8
9  class Store:
10     def __init__(self):
11         """
12         初始化 Store 类，连接 PostgreSQL 数据库并初始化数据表。
13         """
14         # 初始化数据库连接
15         self.conn = psycopg2.connect(
16             host="127.0.0.1",
17             port="5432",
18             user="postgres",
19             password=os.getenv("DB_PASSWORD", "Aqua7296"),
20             database="bookstore"
21         )
22         # 初始化数据表
23         self.init_tables()
24
25     def init_tables(self):
26         """
27         初始化数据库表结构，包括 user、book、store、order 和 order_detail 表。
28         如果表已存在，则先删除再重新创建。
29         """
30         try:
31             with self.conn.cursor() as cursor:
32                 # 删除并重新创建 user 表
33                 cursor.execute('DROP TABLE IF EXISTS "user";')
34                 cursor.execute("""
35                     CREATE TABLE "user" (
36                         user_id TEXT PRIMARY KEY, -- 用户ID为主键
37                         password TEXT NOT NULL, -- 用户密码
38                         balance NUMERIC NOT NULL, -- 用户余额
39                         token TEXT, -- 用户登录令牌
40                         terminal TEXT -- 用户终端信息
41                     );
42                 """)
43
44                 # 在 user_id 上创建唯一升序索引（主键已自动创建索引）
45                 cursor.execute('CREATE UNIQUE INDEX idx_user_id ON "user" (user_id);')
```

```
46
47 # 删除并重新创建 book 表
48 cursor.execute('DROP TABLE IF EXISTS "book";')
49 cursor.execute("""
50     CREATE TABLE "book" (
51         id SERIAL PRIMARY KEY,
52         book_id TEXT NOT NULL,      -- 书籍ID
53         title TEXT NOT NULL,        -- 书名
54         author TEXT,                -- 作者
55         publisher TEXT,             -- 出版社
56         content TEXT,               -- 内容
57         original_title TEXT,        -- 原书名
58         translator TEXT,            -- 译者
59         pub_year TEXT,              -- 出版年份
60         pages INTEGER,              -- 页数
61         price INTEGER,              -- 价格
62         currency_unit TEXT,         -- 货币单位
63         binding TEXT,               -- 装帧
64         isbn TEXT,                  -- ISBN号
65         author_intro TEXT,          -- 作者简介
66         book_intro TEXT,            -- 书籍简介
67         tags JSONB,                 -- 标签（JSON数组）
68         pictures JSONB              -- 图片链接（JSON数组）
69     );
70 """)
71
72 # 在 title 上创建普通索引
73 cursor.execute('CREATE INDEX idx_title ON "book" (title);')
74
75 # 在 tags 上创建 GIN 索引
76 cursor.execute('CREATE INDEX idx_tags ON "book" USING GIN (tags);')
77
78 # 在 book_intro 上创建 GIN 索引，支持全文搜索
79 cursor.execute("""
80     CREATE INDEX idx_book_intro ON "book" USING GIN (to_tsvector('english',
book_intro));
81 """)
82
83 # 在 content 上创建 GIN 索引，支持全文搜索
84 cursor.execute("""
85     CREATE INDEX idx_content ON "book" USING GIN (to_tsvector('english', content));
86 """)
87
88 # 删除并重新创建 store 表
89 cursor.execute('DROP TABLE IF EXISTS "store";')
90 cursor.execute("""
91     CREATE TABLE "store" (
92         store_id TEXT PRIMARY KEY, -- 商店ID为主键
93         user_id TEXT REFERENCES "user"(user_id), -- 用户ID，外键
94         books JSONB                -- 书籍信息，使用 JSONB 类型存储
95     );
96 """)
97
98 # 在 store_id 上创建唯一升序索引（主键已自动创建索引）
99 cursor.execute('CREATE UNIQUE INDEX idx_store_id ON "store" (store_id);')
100
101 # 删除并重新创建 order 表
102 cursor.execute('DROP TABLE IF EXISTS "order";')
103 cursor.execute("""
104     CREATE TABLE "order" (
105         id SERIAL PRIMARY KEY,
106         order_id TEXT,
107         store_id TEXT,
108         user_id TEXT,
109         create_time TIMESTAMP DEFAULT CURRENT_TIMESTAMP, -- 订单创建时间
110         price NUMERIC NOT NULL,      -- 订单总价
111         status TEXT NOT NULL         -- 订单状态
112     );
113 """)
114
115 # 删除并重新创建 order_detail 表
116 cursor.execute('DROP TABLE IF EXISTS "order_detail";')
117 cursor.execute("""
118     CREATE TABLE "order_detail" (
119         id SERIAL PRIMARY KEY,
120         order_id TEXT,
121         book_id TEXT,
122         price NUMERIC NOT NULL,      -- 书籍单价
123         count INTEGER NOT NULL       -- 书籍数量
124     );
```

```
125         """  
126  
127         # 提交事务  
128         self.conn.commit()  
129     except psycopg2.Error as e:  
130         # 捕获数据库错误并记录日志  
131         logging.error(e)  
132         # 回滚事务  
133         self.conn.rollback()  
134  
135     def get_db_conn(self):  
136         """  
137         获取数据库连接对象。  
138  
139         :return: 数据库连接对象  
140         """  
141         return self.conn  
142  
143     def close(self):  
144         """  
145         关闭数据库连接。  
146         """  
147         if self.conn:  
148             self.conn.close()  
149  
150  
151     # 全局变量，用于存储数据库实例  
152     database_instance: Store = None  
153     # 全局变量，用于数据库初始化的同步  
154     init_completed_event = threading.Event()  
155  
156  
157     def init_database():  
158         """  
159         初始化数据库实例。  
160         """  
161         global database_instance  
162         if database_instance is None:  
163             # 初始化数据库实例  
164             database_instance = Store()  
165  
166  
167     def get_db_conn():  
168         """  
169         获取数据库连接对象。如果数据库实例未初始化，则先初始化。  
170  
171         :return: 数据库连接对象  
172         """  
173         global database_instance  
174         if database_instance is None:  
175             init_database()  
176         return database_instance.get_db_conn()
```

## 再次运行

注意将 fe/conf.py 里的 Use\_Large\_DB = True 改为False

因为是本地测试，还未用助教所提的超大数据库

运行结果如下



Name	Stmts	Miss	Branch	BrPart	Cover
-----					
be\__init__.py	0	0	0	0	100%
be\app.py	3	3	2	0	0%
be\model\buyer.py	94	21	38	9	77%
be\model\db_conn.py	19	0	6	0	100%
be\model\error.py	23	1	0	0	96%
be\model\seller.py	55	13	16	1	80%
be\model\store.py	36	2	0	0	94%
be\model\user.py	103	15	30	6	84%
be\serve.py	36	1	2	1	95%
be\view\auth.py	42	0	0	0	100%
be\view\buyer.py	34	0	2	0	100%
be\view\seller.py	38	5	0	0	87%
fe\__init__.py	0	0	0	0	100%
fe\access\__init__.py	0	0	0	0	100%
fe\access\auth.py	31	0	0	0	100%
fe\access\book.py	72	2	12	2	95%
fe\access\buyer.py	36	0	2	0	100%
fe\access\new_buyer.py	8	0	0	0	100%
fe\access\new_seller.py	8	0	0	0	100%
fe\access\seller.py	31	0	0	0	100%
fe\bench\__init__.py	0	0	0	0	100%
fe\bench\run.py	13	0	6	0	100%
fe\bench\session.py	47	0	12	1	98%
fe\bench\workload.py	125	1	20	2	98%
fe\conf.py	11	0	0	0	100%
fe\conftest.py	19	0	0	0	100%
fe\test\gen_book_data.py	49	0	16	0	100%
fe\test\test_add_book.py	37	0	10	0	100%
fe\test\test_add_funds.py	23	0	0	0	100%
fe\test\test_add_stock_level.py	40	0	10	0	100%
fe\test\test_bench.py	6	2	0	0	67%
fe\test\test_create_store.py	20	0	0	0	100%
fe\test\test_login.py	28	0	0	0	100%
fe\test\test_new_order.py	40	0	0	0	100%
fe\test\test_password.py	33	0	0	0	100%
fe\test\test_payment.py	60	1	4	1	97%
fe\test\test_register.py	31	0	0	0	100%
-----					
TOTAL	1251	67	188	23	94%
Wrote HTML report to htmlcov\index.html					

## 5.附加功能

### 5.1收货发货

#### 5.1.1发货

函数输入：

user\_id：卖家用户ID

order\_id：订单ID

函数功能解释：

- 使用 `$or` 操作符查找具有指定 `order_id` 并且状态为 `1`、`2` 或 `3` 的订单。`find_one` 方法用于查找符合条件的第一条记录，并将结果存储在 `result` 变量中。
- 如果查询结果为 `None`，表示没有找到符合条件的订单，函数会返回一个错误，表示订单 ID 无效。
- 从查询结果中获取订单的状态。如果状态为 `2`（已交付）或 `3`（已完成），则返回一个错误，指示该订单已经被交付或完成，不能重复交付。
- 如果订单状态有效且可以交付，则使用 `update_one` 方法将订单的状态更新为 `2`（表示已交付）。



- 5.如果在执行数据库操作时发生 `sqlite.Error` 异常，返回错误代码 528 和错误信息。如果发生其他类型的异常，返回错误代码 530 和错误信息。
- 6.如果所有操作成功完成，返回状态码 200 和消息 "ok"，表示操作成功。

5.1.2收货

函数输入：

user\_id：卖家用户ID

store\_id：商铺ID

book\_id：书籍ID

add\_stock\_level：增加的库存量

函数功能解释：

- 1.使用 `$or` 操作符查找具有指定 `order_id` 并且状态为 1、2 或 3 的订单。`find_one` 方法用于查找符合条件的第一条记录，并将结果存储在 `result` 变量中。
- 2.如果查询结果为 `None`，表示没有找到符合条件的订单，函数会返回一个错误，表示示订单 ID 无效。
- 3.从查询结果中提取买家的用户 ID 和订单状态。
- 4.检查请求的用户 ID 是否与订单的买家 ID 匹配。如果不匹配，返回授权失败的错误信息。
- 5.如果订单状态为 1，表示书籍尚未发货，返回相应的错误信息。如果订单状态为 3，表示书籍已经被重复接收，返回相应的错误信息。
- 6.如果以上检查都通过，更新订单状态为 3（已接收）。
- 7.如果在执行数据库操作时发生 `sqlite.Error` 异常，返回错误代码 528 和错误信息。如果发生其他类型的异常，返回错误代码 530 和错误信息。
- 8.如果所有操作成功完成，返回状态码 200 和消息 "ok"，表示操作成功。

5.1.3发货测试

1.test\_ok 发货成功

操作：卖家发货。

预期结果：发货成功，返回码为200。

2.test\_order\_error 订单不存在

操作：使用不存在的订单ID尝试发货。

预期结果：发货失败，返回码不为200。

3.test\_books\_repeat\_deliver 重复发货

操作：卖家发货两次。

预期结果：第一次发货成功，第二次发货失败，返回码不为200。

5.1.4收货测试

1.test\_ok 收货成功

操作：卖家发货，买家收货。

预期结果：收货成功，返回码为200。

2.test\_order\_error 订单不存在

操作：卖家发货，使用不存在的订单ID尝试收货。

预期结果：收货失败，返回码不为200。

3.test\_authorization\_error 买家不存在

操作：卖家发货，使用不存在的买家ID尝试收货。

预期结果：收货失败，返回码不为200。

4.test\_books\_not\_deliver 未发货

操作：未发货的订单尝试收货。

预期结果：收货失败，返回码不为200。

5.test\_books\_repeat\_receive 重复收货

操作：卖家发货，买家收货两次。

预期结果：第一次收货成功，第二次收货失败，返回码不为200。

### 5.1.5前后端接口

be/view/seller.py

be/view/buyer.py

fe/access/seller.py

fe/access/buyer.py

这个几个文件都新增一个函数，就照着之前的写，改几个变量就好了。

### 5.1.6error

新增三个对应发货和收获的错误

```
1 520: "books not deliver.",
2 521: "books deliver repeatedly.",
3 522: "books receive repeatedly.",
```

## 5.2搜索图书

### 5.2.1 搜索功能概述

根据前文业务流程分析依次需要实现的

- 添加 fe/test/search.py
- 添加 fe/access/search.py
- 添加 be/view/search.py (记得在serve.py里注册蓝图)
- 添加 be/model/book.py

实现了以下搜索功能：

1. **搜索指定标题的图书**：根据书名搜索书籍，支持分页和指定商店。
2. **搜索指定标签的图书**：根据标签搜索书籍，支持分页和指定商店。
3. **搜索指定内容的图书**：根据书籍简介或内容进行全文搜索，支持分页和指定商店。
4. **搜索指定作者的图书**：根据作者搜索书籍，支持分页和指定商店。

这些功能通过 be/model/book.py 中的方法实现，并通过 be/view/search.py 提供 HTTP 接口。

### 5.2.2 搜索指定标题的图书

函数输入：

- title：要搜索的书名。
- store\_id：商店 ID，如果为空字符串，则搜索所有商店。
- page\_num：页码，从 1 开始。
- page\_size：每页显示的书籍数量。

函数输出：

- 整数：状态码，200 表示成功，501 表示未找到书籍，530 表示发生错误。
- 字符串：描述操作结果的消息。
- 列表：包含搜索到的书籍信息。

函数流程：

1. 使用 SELECT 语句查询 book 表中标题等于 title 的书籍，并按 id 排序，支持分页。
2. 如果指定了 store\_id，则进一步过滤出该商店中的书籍。
3. 如果未找到书籍，返回状态码 501 和错误信息。
4. 如果找到书籍，返回状态码 200、消息 "ok" 以及书籍列表。

代码实现：

```
1 def search_title_in_store(self, title: str, store_id: str, page_num: int, page_size: int):
2     try:
3         with self.conn.cursor() as cursor:
4             # 查询书籍信息
5             cursor.execute("""
6                 SELECT * FROM book
7                 WHERE title = %s
8                 ORDER BY id
9                 LIMIT %s OFFSET %s;
10            """, (title, page_size, (page_num - 1) * page_size))
11            result_list = cursor.fetchall()
```

```
12
13         # 如果指定了 store_id, 过滤出该商店中的书籍
14         if store_id:
15             books_in_store = []
16             for book in result_list:
17                 cursor.execute("""
18                     SELECT 1 FROM store
19                     WHERE store_id = %s AND books @> jsonb_build_array(jsonb_build_object('book_id',
%s));
20
21                     """, (store_id, book['book_id']))
22                 if cursor.fetchone():
23                     books_in_store.append(book)
24             result_list = books_in_store
25
26         if not result_list:
27             return 501, f"{title} book not exist", []
28         return 200, "ok", result_list
29
30     except Exception as e:
31         logging.error(str(e))
32         self.conn.rollback()
33         return 530, "{}".format(str(e)), []
34
35 def search_title(self, title: str, page_num: int, page_size: int):
36     return self.search_title_in_store(title, "", page_num, page_size)
```

### 5.2.3 搜索指定标签的图书

函数输入：

- tag：要搜索的标签。
- store\_id：商店 ID，如果为空字符串，则搜索所有商店。
- page\_num：页码，从 1 开始。
- page\_size：每页显示的书籍数量。

函数输出：

- 整数：状态码，200 表示成功，501 表示未找到书籍，530 表示发生错误。
- 字符串：描述操作结果的消息。
- 列表：包含搜索到的书籍信息。

函数流程：

1. 使用 SELECT 语句查询 book 表中 tags 字段包含指定标签的书籍，并按 id 排序，支持分页。
2. 如果指定了 store\_id，则进一步过滤出该商店中的书籍。
3. 如果未找到书籍，返回状态码 501 和错误信息。
4. 如果找到书籍，返回状态码 200、消息 "ok" 以及书籍列表。

代码实现：

```
1 def search_tag_in_store(self, tag: str, store_id: str, page_num: int, page_size: int):
2     try:
3         with self.conn.cursor() as cursor:
4             # 查询书籍信息
5             cursor.execute("""
6                 SELECT * FROM book
7                 WHERE tags @> %s
8                 ORDER BY id
9                 LIMIT %s OFFSET %s;
10            """, (json.dumps([tag]), page_size, (page_num - 1) * page_size))
11            result_list = cursor.fetchall()
12
13            # 如果指定了 store_id, 过滤出该商店中的书籍
14            if store_id:
15                books_in_store = []
16                for book in result_list:
17                    cursor.execute("""
18                        SELECT 1 FROM store
19                        WHERE store_id = %s AND books @> jsonb_build_array(jsonb_build_object('book_id',
%s));
20
21                        """, (store_id, book['book_id']))
22                    if cursor.fetchone():
23                        books_in_store.append(book)
24                result_list = books_in_store
25
26            if not result_list:
27                return 501, f"{tag} book not exist", []
```

```
27         return 200, "ok", result_list
28
29     except Exception as e:
30         logging.error(str(e))
31         self.conn.rollback()
32         return 530, "{}".format(str(e)), []
33
34 def search_tag(self, tag: str, page_num: int, page_size: int):
35     return self.search_tag_in_store(tag, "", page_num, page_size)
```

### 5.2.4 搜索指定内容的图书

函数输入：

- `content`：要搜索的内容。
- `store_id`：商店 ID，如果为空字符串，则搜索所有商店。
- `page_num`：页码，从 1 开始。
- `page_size`：每页显示的书籍数量。

函数输出：

- 整数：状态码，200 表示成功，501 表示未找到书籍，530 表示发生错误。
- 字符串：描述操作结果的消息。
- 列表：包含搜索到的书籍信息。

函数流程：

1. 使用 `to_tsvector` 和 `to_tsquery` 进行全文搜索，查询 `book` 表中 `book_intro` 或 `content` 字段包含指定内容的书籍，支持分页。
2. 如果指定了 `store_id`，则进一步过滤出该商店中的书籍。
3. 如果未找到书籍，返回状态码 501 和错误信息。
4. 如果找到书籍，返回状态码 200、消息 "ok" 以及书籍列表。

代码实现：

```
1  def search_content_in_store(self, content: str, store_id: str, page_num: int, page_size: int):
2      cursor = self.conn.cursor(cursor_factory=psycopg2.extras.DictCursor)
3
4      # 全文搜索查询（同时匹配 book_intro 和 content 字段）
5      query = """
6      SELECT * FROM book
7      WHERE to_tsvector('english', book_intro || ' ' || content) @@ to_tsquery('english', %s)
8      OFFSET %s LIMIT %s
9      """
10     content_query = " & ".join(content.split()) # 预处理 content
11     logging.info(
12         f"Executing full-text search query: {query} with params: {content_query}, {(page_num - 1) *
page_size}, {page_size}")
13     cursor.execute(query, (content_query, (page_num - 1) * page_size, page_size))
14     result_list = cursor.fetchall()
15     logging.info(f"Full-text search result: {result_list}")
16
17     if store_id:
18         # 查询指定商店中的书籍
19         store_query = """
20         SELECT store_id
21         FROM store
22         WHERE store_id = %s AND EXISTS (
23             SELECT 1
24             FROM jsonb_array_elements(books) AS book
25             WHERE book->>'book_id' = %s
26         )
27         """
28         books_in_store = []
29         for book in result_list:
30             logging.info(f"Checking if book {book['book_id']} exists in store {store_id}")
31             cursor.execute(store_query, (store_id, book["book_id"]))
32             store_result = cursor.fetchone()
33             logging.info(f"Store query result for book {book['book_id']}: {store_result}")
34             if store_result: # 判断查询结果是否为空
35                 books_in_store.append(book)
36         result_list = books_in_store
37
38     if len(result_list) == 0:
39         logging.warning(f"No books found for content: {content} in store: {store_id}")
40         return 501, f"{content} book not exist", []
41     return 200, "ok", result_list
```

```
42
43 def search_content(self, content: str, page_num: int, page_size: int):
44     return self.search_content_in_store(content, "", page_num, page_size)
```

### 5.2.5 搜索指定作者的图书

函数输入：

- `author`：要搜索的作者。
- `store_id`：商店 ID，如果为空字符串，则搜索所有商店。
- `page_num`：页码，从 1 开始。
- `page_size`：每页显示的书籍数量。

函数输出：

- 整数：状态码，200 表示成功，501 表示未找到书籍，530 表示发生错误。
- 字符串：描述操作结果的消息。
- 列表：包含搜索到的书籍信息。

函数流程：

1. 使用 `SELECT` 语句查询 `book` 表中作者等于 `author` 的书籍，并按 `id` 排序，支持分页。
2. 如果指定了 `store_id`，则进一步过滤出该商店中的书籍。
3. 如果未找到书籍，返回状态码 501 和错误信息。
4. 如果找到书籍，返回状态码 200、消息 `"ok"` 以及书籍列表。

代码实现：

```
1 def search_author_in_store(self, author: str, store_id: str, page_num: int, page_size: int):
2     try:
3         with self.conn.cursor() as cursor:
4             # 查询书籍信息
5             cursor.execute("""
6                 SELECT * FROM book
7                 WHERE author = %s
8                 ORDER BY id
9                 LIMIT %s OFFSET %s;
10            """, (author, page_size, (page_num - 1) * page_size))
11            result_list = cursor.fetchall()
12
13            # 如果指定了 store_id, 过滤出该商店中的书籍
14            if store_id:
15                books_in_store = []
16                for book in result_list:
17                    cursor.execute("""
18                        SELECT 1 FROM store
19                        WHERE store_id = %s AND books @> jsonb_build_array(jsonb_build_object('book_id',
20% s));
21
22                        """, (store_id, book['book_id']))
23                    if cursor.fetchone():
24                        books_in_store.append(book)
25                result_list = books_in_store
26
27            if not result_list:
28                return 501, f"{author} book not exist", []
29            return 200, "ok", result_list
30
31        except Exception as e:
32            logging.error(str(e))
33            self.conn.rollback()
34            return 530, "{}".format(str(e)), []
35
36 def search_author(self, author: str, page_num: int, page_size: int):
37     return self.search_author_in_store(author, "", page_num, page_size)
```

## 5.3 订单状态以及查询

### 5.3.1 check\_hist\_order

函数输入：

- `user_id`：用户标识

函数输出：

在内部创建一个包含历史订单信息的列表 `ans`，并在最后返回该列表

函数流程：

1. 检查用户是否存在。如果用户不存在，将返回一个错误消息，提示无效的用户ID
2. 如果存在，初始化一个空列表 `ans`，用于存储历史订单
3. 查询不同状态的订单，依次处理：
  - 查找未付款的订单，并将这些订单的信息添加到 `ans` 列表
  - 查找已支付但尚未发货、已支付但未收到和已收到的订单，并将相关信息也添加到 `ans` 列表
  - 查找已取消的订单，并将这些信息加入 `ans` 列表
4. 对于每种订单状态，执行以下操作：
  - 查询与用户ID和相应订单状态匹配的订单信息
  - 针对每个订单，获取与其相关的书籍详细信息
  - 为每个订单创建一个字典，包含状态、订单ID、买家ID、商店ID、总价及书籍详细信息，并将该字典添加到 `ans` 列表中
5. 在处理完所有状态后，检查 `ans`列表：
  - 如果列表为空，返回一条成功消息，指示没有找到任何历史订单
  - 如果列表非空，返回成功消息，并附上包含历史订单信息的 `ans` 列表
6. 如果在以上过程中发生任何异常，函数将捕获并返回状态码528

### 5.3.2 cancel\_order

函数输入：

- `user_id`：用户标识
- `order_id`：订单标识

函数输出：

- 一个整数：状态码，200表示成功，528表示发生错误
- 一个字符串：描述操作结果的消息

函数流程：

1. 函数在订单集合中查找状态为0的订单，使用给定的订单ID，如果找到匹配的订单，提取买家ID、商店ID和订单价格，然后从订单集合中删除该订单
2. 如果未能找到未付款的订单，函数将使用 `$or`操作符查找状态为1、2或3的订单。如果找到相应订单，提取买家ID、商店ID和订单价格，并执行以下操作：
  - 验证买家ID是否与输入的用户ID匹配，以确保用户有权取消订单。如果不匹配，返回授权失败的错误消息
  - 提取卖家ID，从卖家的账户中扣除订单价格，同时将相同金额返还到买家的账户
  - 删除匹配的订单记录
3. 如果在上述步骤中未找到任何适用的订单，返回无效订单ID的错误消息
4. 查询订单详细信息集合，获取与该订单相关的已购书籍信息。遍历书籍信息，恢复库存，增加相应的库存数量
5. 创建一个新订单，状态设置为4，包括订单ID、用户ID、商店ID和订单价格，并将其插入到订单集合中
6. 如果在执行过程中发生任何异常，函数会捕获并返回状态码528

### 5.3.3 auto\_cancel\_order

函数输入：

该函数不接受任何参数，

函数输出：

- 一个整数：状态码，200表示成功，528表示发生错误
- 一个字符串：描述操作结果的消息

函数流程：

1. 定义一个等待时间 `wait_time`，设置为20秒，用于判断未支付订单的自动取消时限
2. 获取当前的UTC时间 `now`，计算出一个时间点 `interval`，表示当前时间减去 `wait_time` 秒后的时间
3. 构建查询条件 `cursor`，用于查找未支付（状态为0）且创建时间早于 `interval` 的订单
4. 将待取消的订单信息存储在 `orders_to_cancel` 中
5. 如果找到待取消的订单，遍历这些订单并执行以下操作：
  - 提取每个订单的相关信息，包括订单ID、用户ID、商店ID和订单价格
  - 从订单集合中删除该订单，以实现订单取消
  - 查询已取消订单的书籍详细信息，并遍历这些书籍
  - 对每本书籍，恢复库存，增加相应数量
  - 如果库存恢复失败，返回库存不足的错误消息
6. 对于每个成功取消的订单，函数创建一个新的订单文档 `cancel_order`，将状态设置已取消，然后将其插入到订单集合中
7. 如果在执行过程中发生任何异常，函数将捕获异常并返回状态码528





```
(.venv) ~\Desktop\大三上\当代数据管理系统\Homework\大作业2\bookstore-main git:[master]
coverage report
Name                               Stmts  Miss Branch BrPart  Cover
-----
be\__init__.py                     0      0      0      0    100%
be\app.py                           3      3      2      0      0%
be\model\book.py                   97     16     32      4     84%
be\model\buyer.py                 365    102    106     23     73%
be\model\db_conn.py                32      0      6      0    100%
be\model\error.py                  33      2      0      0     94%
be\model\seller.py                 86     24     22      2     76%
be\model\store.py                  47      5      6      1     85%
be\model\user.py                  121     21     26      4     83%
be\serve.py                        37      1      2      1     95%
be\view\auth.py                    42      0      0      0    100%
be\view\buyer.py                   74      4      2      0     95%
be\view\search.py                  86      4     32     12     86%
be\view\seller.py                  38      0      0      0    100%
fe\__init__.py                     0      0      0      0    100%
fe\access\__init__.py              0      0      0      0    100%
fe\access\auth.py                  31      0      0      0    100%
fe\access\book.py                  72      2     12      2     95%
fe\access\buyer.py                 73      6      4      1     91%
fe\access\new_buyer.py              8      0      0      0    100%
fe\access\new_seller.py            8      0      0      0    100%
fe\access\search.py                53      0      0      0    100%
fe\access\seller.py                37      0      0      0    100%
fe\bench\__init__.py               0      0      0      0    100%
fe\bench\run.py                    13      0      6      0    100%
fe\bench\session.py                47      0     12      1     98%
fe\bench\workload.py              125      1     20      2     98%
fe\conf.py                         11      0      0      0    100%
fe\conftest.py                     19      0      0      0    100%
fe\test\gen_book_data.py           49      1     16      1     97%
fe\test\test_add_book.py           37      0     10      0    100%
fe\test\test_add_funds.py          23      0      0      0    100%
fe\test\test_add_stock_level.py    40      0     10      0    100%
fe\test\test_bench.py               6      2      0      0     67%
fe\test\test_cancel_auto.py        54      1      4      1     97%
fe\test\test_cancel_order.py       82      1      4      1     98%
fe\test\test_create_store.py       20      0      0      0    100%
fe\test\test_deliver.py            44      1      4      1     96%
fe\test\test_history_order.py      65      3     14      2     94%
fe\test\test_login.py              28      0      0      0    100%
fe\test\test_new_order.py          40      0      0      0    100%
fe\test\test_password.py           33      0      0      0    100%
fe\test\test_payment.py            60      1      4      1     97%
fe\test\test_receive.py            58      1      4      1     97%
fe\test\test_register.py           31      0      0      0    100%
fe\test\test_search.py            100      0      0      0    100%
-----
TOTAL                             2328    202    360     61     90%
```

## 7.2 大数据库 book1x.db

最终HTML 覆盖率报告储存在 results/book\_1x\_db 中

详细终端输出将 results/console.log

```
(.venv) ~\Desktop\大三上\当代数据管理系统\Homework\大作业2\bookstore-main git:[master]
coverage combine
coverage report
coverage html
Name                               Stmts  Miss Branch BrPart  Cover
-----
be\__init__.py                      0      0      0      0   100%
be\app.py                           3      3      2      0     0%
be\model\book.py                   97     16     32      4    84%
be\model\buyer.py                  365    102    106     23    73%
be\model\db_conn.py                32      0      6      0   100%
be\model\error.py                  33      2      0      0    94%
be\model\seller.py                  86     24     22      2    76%
be\model\store.py                   47     18      6      1    60%
be\model\user.py                   121     21     26      4    83%
be\serve.py                        37      1      2      1    95%
be\view\auth.py                    42      0      0      0   100%
be\view\buyer.py                   74      4      2      0    95%
be\view\search.py                  86      4     32     12    86%
be\view\seller.py                  38      0      0      0   100%
fe\__init__.py                      0      0      0      0   100%
fe\access\__init__.py              0      0      0      0   100%
fe\access\auth.py                  31      0      0      0   100%
fe\access\book.py                  72      0     12      1    99%
fe\access\buyer.py                 73      6      4      1    91%
fe\access\new_buyer.py              8      0      0      0   100%
fe\access\new_seller.py            8      0      0      0   100%
fe\access\search.py                53      0      0      0   100%
fe\access\seller.py                37      0      0      0   100%
fe\bench\__init__.py               0      0      0      0   100%
fe\bench\run.py                    13      0      6      0   100%
fe\bench\session.py                47      0     12      1    98%
fe\bench\workload.py               125      2     20      2    97%
fe\conf.py                         11      0      0      0   100%
fe\conftest.py                     19      0      0      0   100%
fe\test\gen_book_data.py           49      0     16      1    98%
fe\test\test_add_book.py           37      0     10      0   100%
fe\test\test_add_funds.py          23      0      0      0   100%
fe\test\test_add_stock_level.py    40      0     10      0   100%
fe\test\test_bench.py               6      2      0      0    67%
fe\test\test_cancel_auto.py        54      1      4      1    97%
fe\test\test_cancel_order.py       82      1      4      1    98%
fe\test\test_create_store.py       20      0      0      0   100%
fe\test\test_deliver.py            44      1      4      1    96%
fe\test\test_history_order.py      65      6     14      3    89%
fe\test\test_login.py              28      0      0      0   100%
fe\test\test_new_order.py          40      0      0      0   100%
fe\test\test_password.py           33      0      0      0   100%
fe\test\test_payment.py            60      1      4      1    97%
fe\test\test_receive.py            58      1      4      1    97%
fe\test\test_register.py           31      0      0      0   100%
fe\test\test_search.py            100      0      0      0   100%
-----
TOTAL                             2328    216    360     61    89%
```

## 8. 总结

本次实验通过将 MongoDB 迁移到 PostgreSQL，实现了更高效的数据管理和查询。通过合理设计数据库表结构和索引，优化了查询性能。同时，通过测试驱动开发，确保了代码的健壮性和高覆盖率。最终，项目实现了基础功能和附加功能，并通过了全面的测试