

**Bookstore——网上购书功能网站后端（SQL）**

**课程：当代数据管理系统**

**姓名：唐小卉**

**学号：10215501437**

**2023年 12 月**

目录

[1、 项目运行 4](#_Toc15272)

[2、 数据库设计（schema） 4](#_Toc11030)

[2.1 new\_order 4](#_Toc1117)

[2.2 new\_order\_detail 4](#_Toc26888)

[2.3 new\_order\_paid 5](#_Toc29688)

[2.4 store 5](#_Toc18694)

[2.5 user 6](#_Toc17966)

[2.6 user\_store 6](#_Toc32746)

[2.7 ER图与索引 7](#_Toc3208)

[3、 功能实现 8](#_Toc4328)

[3.1 数据库连接与表格建立 8](#_Toc15884)

[3.1.1 数据库连接 8](#_Toc7429)

[3.1.2 表格建立 8](#_Toc19803)

[3.2 用户功能 10](#_Toc4937)

[3.2.1 token与jwt 10](#_Toc16543)

[3.2.2 register登录 11](#_Toc28527)

[3.2.3 check password检查密码 11](#_Toc25080)

[3.2.4 logout登出 12](#_Toc25376)

[3.2.5 unregister注销 12](#_Toc5199)

[3.2.6 change password 更改密码 13](#_Toc26109)

[3.3 卖家功能 14](#_Toc30643)

[3.3.1 add\_book 上架书本 14](#_Toc19203)

[3.3.2 add\_stock\_level 增加库存 14](#_Toc8619)

[3.3.3 create\_store 创建商店 15](#_Toc11795)

[3.4 买家功能 15](#_Toc4397)

[3.4.1 new order 新增订单 15](#_Toc25243)

[3.4.2 payment 付款 17](#_Toc780)

[3.4.3 add\_funds 余额充值 19](#_Toc5306)

[3.5 发货收货 19](#_Toc13086)

[3.5.1 初始准备 19](#_Toc6230)

[3.5.2 发货 21](#_Toc14197)

[3.5.3 收货 22](#_Toc32243)

[3.6 搜索图书 23](#_Toc20050)

[4、 测试 26](#_Toc859)

[4.1 原有pytest 26](#_Toc31934)

[4.2 关于搜索功能的pytest检验 26](#_Toc26188)

[4.3 关于发货收货功能的pytest检验 27](#_Toc2802)

[4.4 关于订单功能pytest检验 29](#_Toc4547)

[4.5 接口介绍 30](#_Toc5362)

[4.6 覆盖率与测试结果 33](#_Toc8880)

[4.6.1 覆盖率 33](#_Toc22789)

[4.6.2 测试结果 34](#_Toc26374)

[5、 项目说明 34](#_Toc31477)

# 项目运行

首先，在本地的MySQL数据库中建立bookstore数据库，具体操作命令如下：

*CREATE DATABASE bookstore;*

之后，将book\_lx.db数据库放置到fe/data文件夹中。

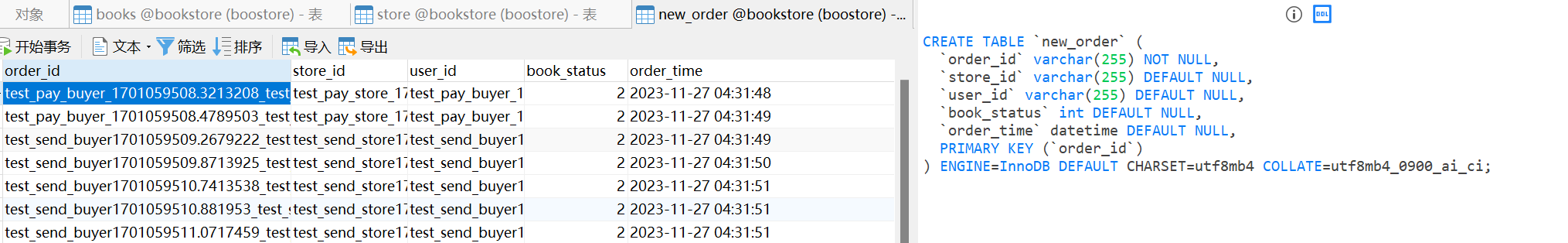


之后，在命令行中运行：bash script/test.sh，即可看到测试结果（此处需注意bookstore数据库必须为空，如果不为空请删除整个数据库重新建立）

# 数据库设计（Schema）

## 2.1 new\_order

New\_order表格中存储用户的新订单（未付款订单）。



其中:

Order\_id：string类型，表示订单编号。

Store\_id：string类型，表示商店的编号。

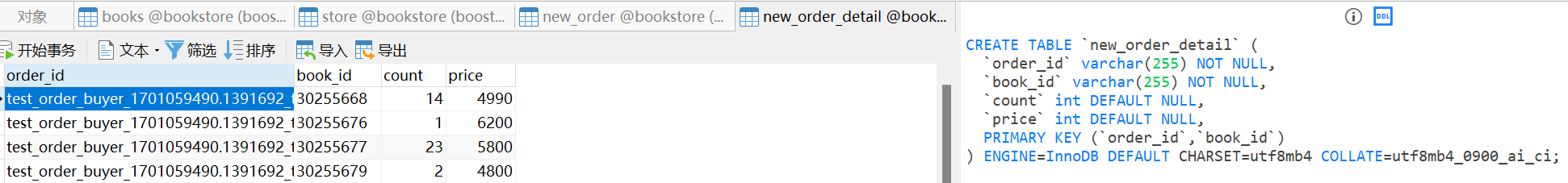
User\_id：string类型，表示用户编号。

Book\_status：int类型，表示图书状态。

Order\_time：string类型，表示下单时间。

## 2.2 new\_order\_detail

New\_order\_detail中存储订单的详细信息（未付款订单）。



其中：

Order\_id: string类型，表示订单的编号。

Book\_id: string类型，表示书本的编号。

count: int类型，表示书本的数量。

Price: int类型，表示价格。

## 2.3 new\_order\_paid

New\_order\_paid中存储已经付款的订单信息，未付款订单不会显示在该表中。



Order\_id：string类型，表示订单编号。

User\_id：string类型，表示用户编号。

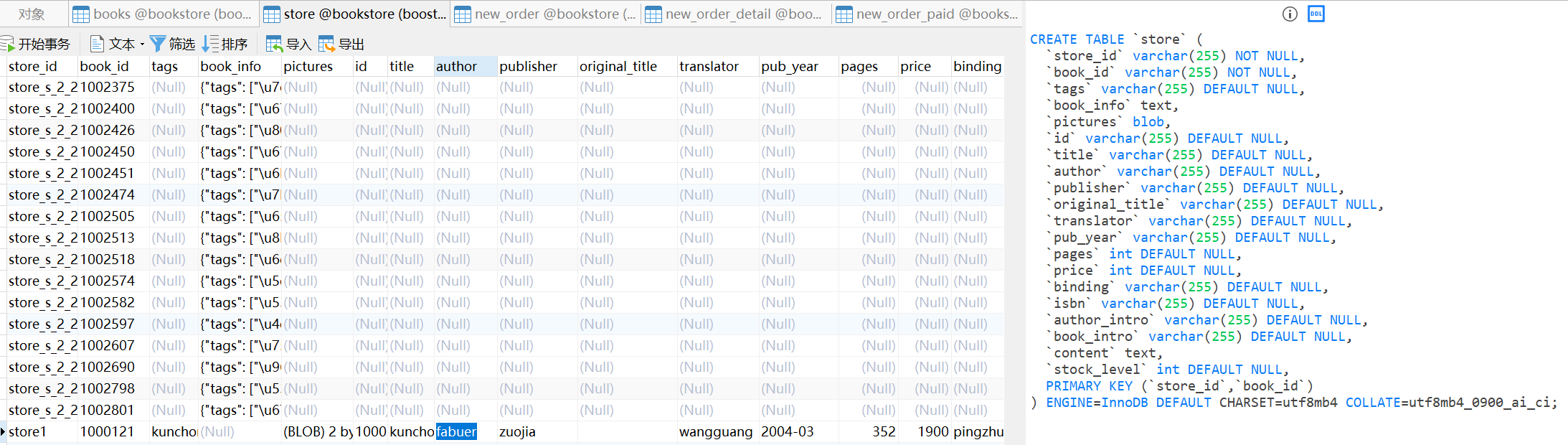
Store\_id：string类型，表示商店的编号。

Book\_status：int类型，表示图书状态。

Price：int类型，表示价格。

## 2.4 store

Store表中存储商铺的信息以及商铺中含有的书本。



Store\_id：string类型，表示商店编号。

book\_info：document，里面含有书本的详情。

book\_id：string类型，表示书本编号。

Stock\_level：int类型。

Id：string类型，表示书本的编号。

Title：string类型，表示书本的名字。

Author：string类型，表示作者的名字。

Publisher：string类型，表示出版社的名字。

Pub\_year：string类型，表示出版日期。

Pages：int类型，表示页数。

Price：int类型，表示价格。

Currency\_unit：string类型，表示货币。

Binding：string类型，表示封装类型。

Isbn：string类型。

Author\_intro：string类型，表示作者介绍。

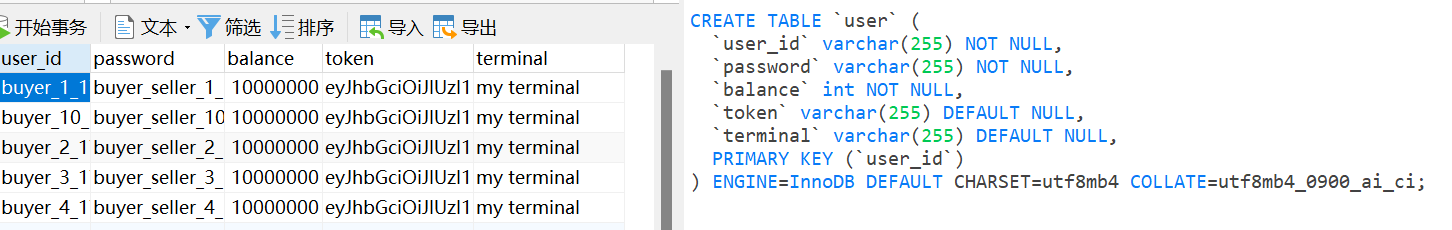
Book\_intro：string类型，表示书本介绍。

Content：string类型，表示内容。

Tags：string类型，表示标签。

## 2.5 user

User表中存储用户信息。



user\_id:string类型，表示用户编号。

Password：string类型，表示密码。

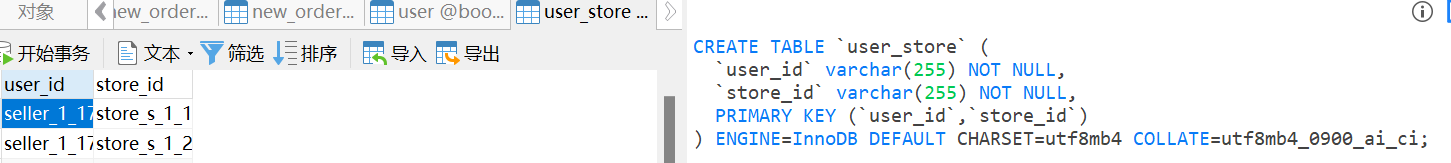
Balance：int类型，表示资产。

Token：string类型，表示用户令牌。

terminal：string类型，表示用户终端。

## 2.6 user\_store

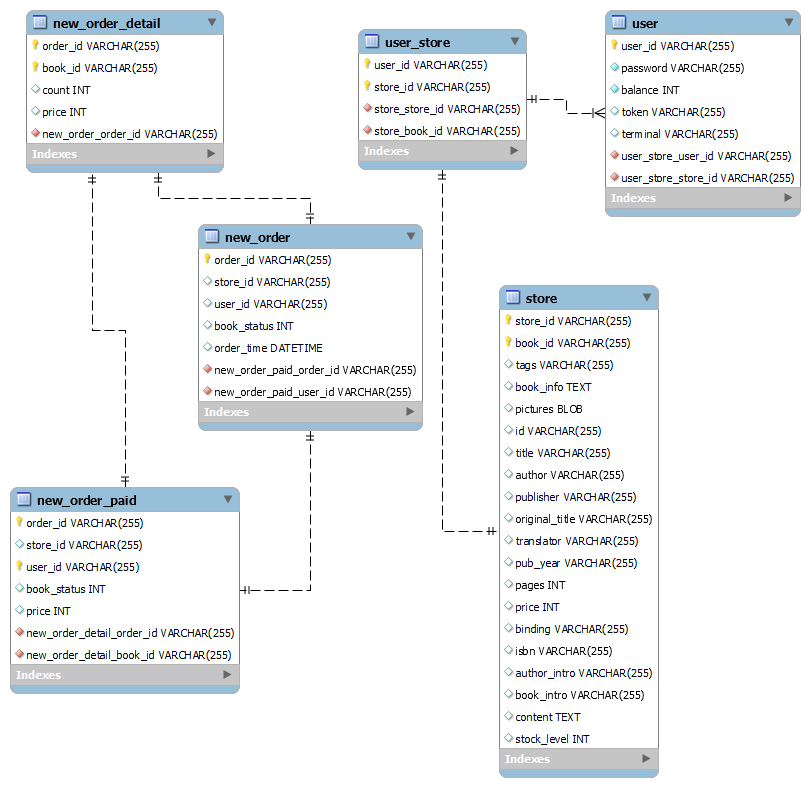
User\_store中存储每一个用户开张的店铺。

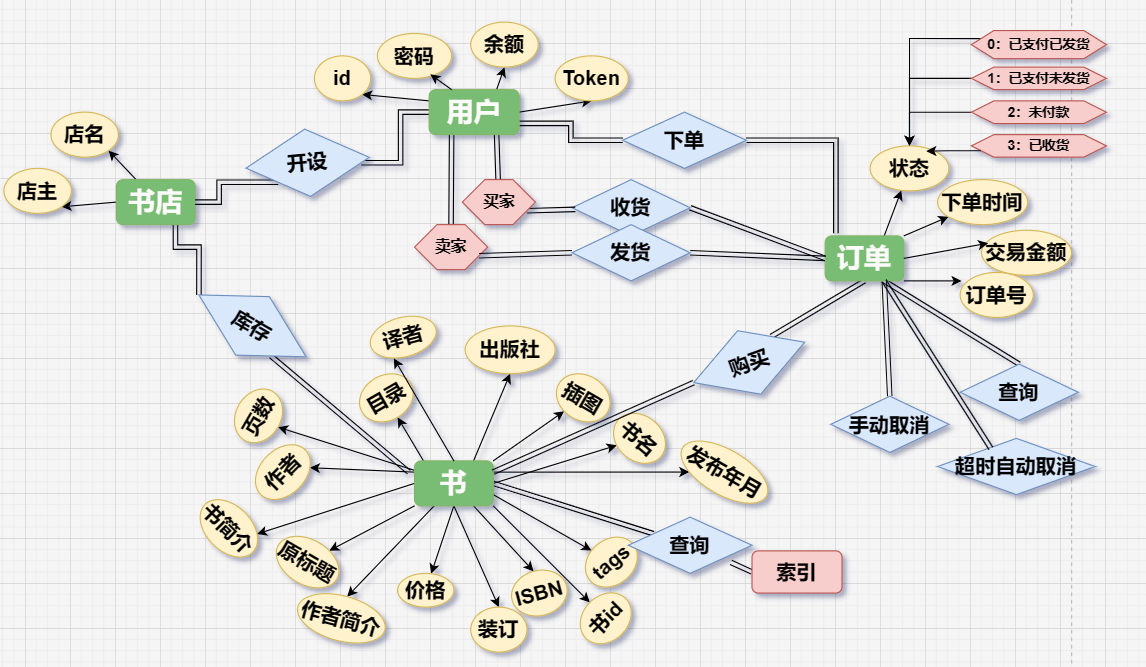


store\_id:string类型，表示店铺编号。

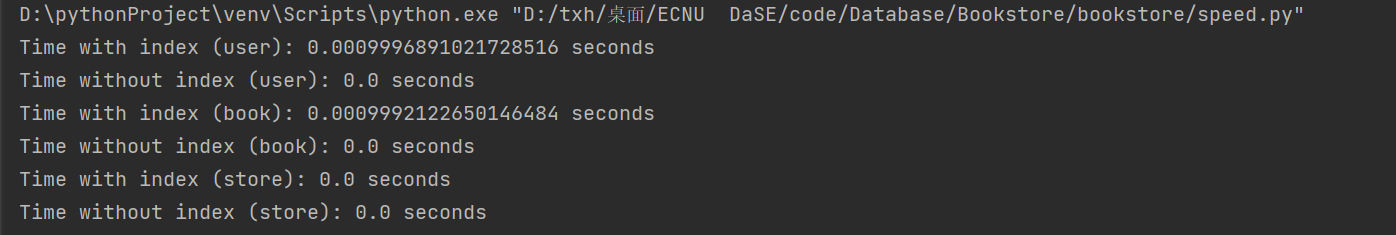
user\_id：string类型，表示用户编号。

## 2.7 ER图与索引



****

针对后期功能的使用频率和效果，我们设立了以下三个索引并按照升序排列：book\_id,store\_id,user\_id。由于这三个id是经常被访问的，所有为他们建立索引可以有效的提高访问速率。



虽然从结果来看提升效率不高，但是鉴于插入数据量比较少，而且数据复杂程度不够高，索引起到的优化作用可能比较微小，如果还能再丰富数据库存储的内容，相信索引起到的优化效果也会更好。

我也考虑过使用全文索引，但我的MySQL版本以及一些其他的版本原因导致全文索引的逻辑失败了，

# 功能实现

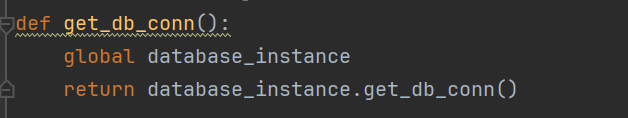
## 3.1 数据库连接与表格建立

### 3.1.1 数据库连接

在store.py中，我们连接到本地MySQL数据库：

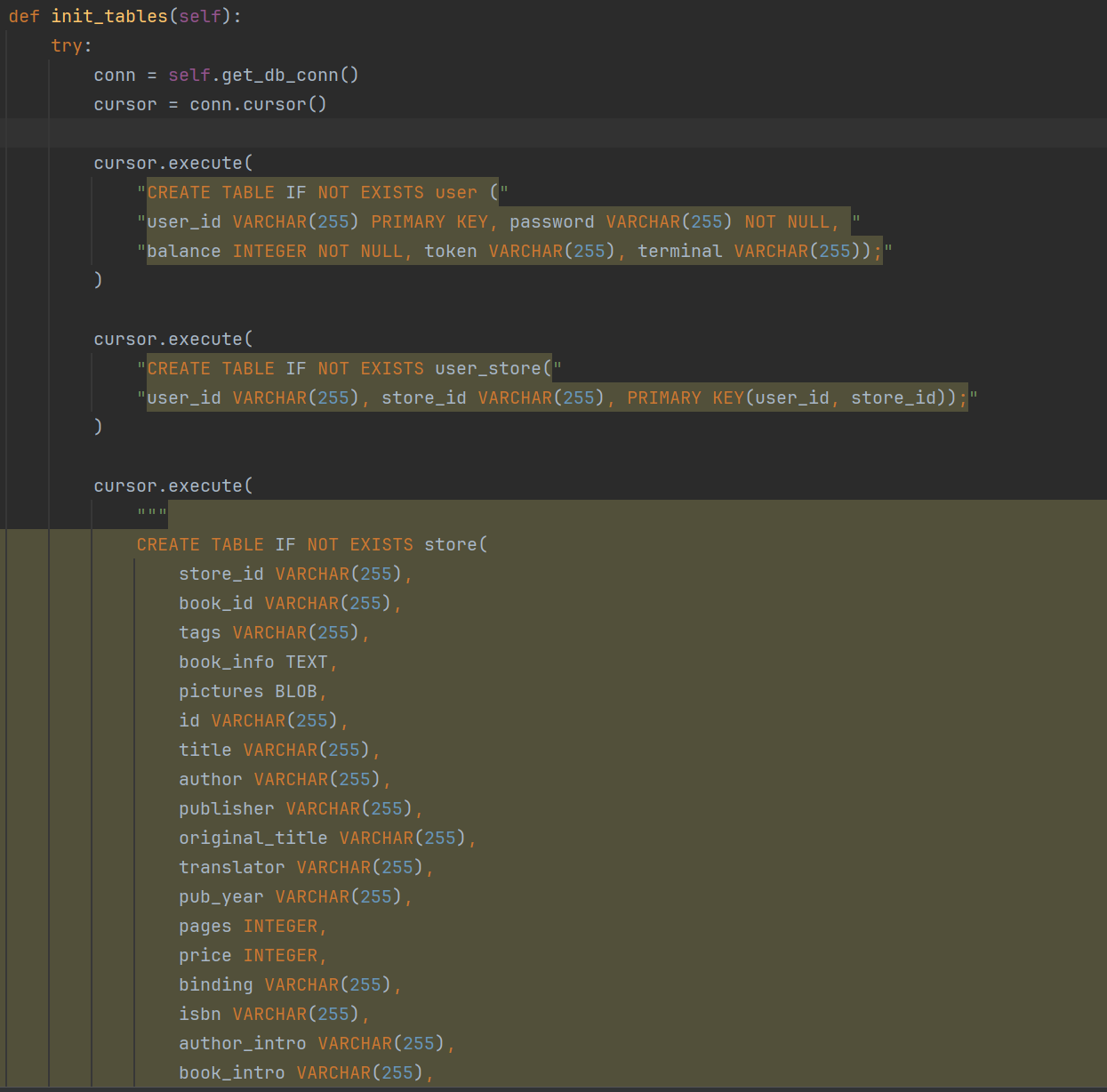


设置了全局变量database\_instance,在其他代码中可以调用get\_db\_conn()函数直接连接到MySQL数据库。



### 3.1.2 表格建立

同样在store.py中，init\_tables（）函数会建立上述需要的表格。





## 3.2 用户功能

用户功能的后端主要在user.py中实现。

### 3.2.1 token与jwt

可以看到在用户功能板块，利用了token和jwt来确保用户的账户安全。

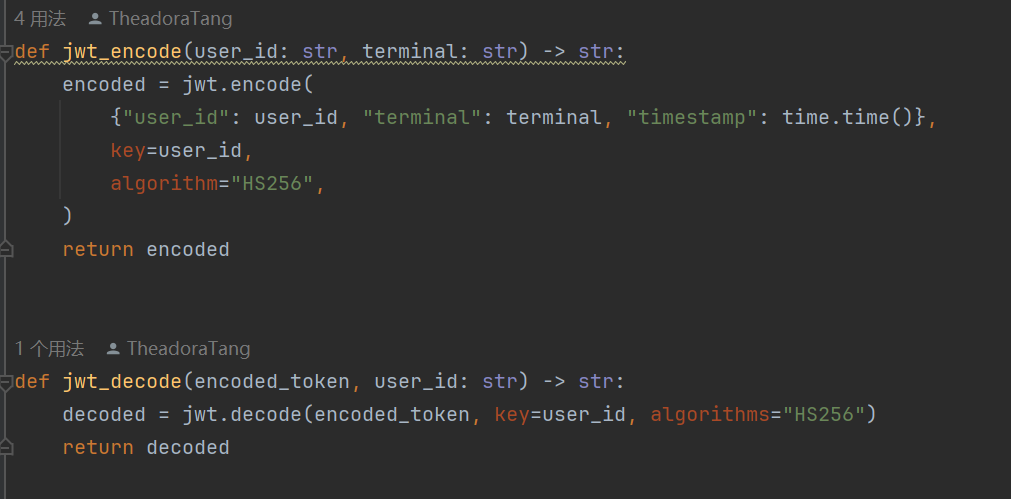
Token，即令牌，是一种用于验证用户身份和授权的技术。 token可以理解为一个字符串，里面包含了足够的信息，以便于服务器识别和验证用户。 token的基本原理是，服务器会为每个已验证的用户生成一个唯一的、有时效性的令牌。 当用户进行一些需要认证的操作时，他们需要携带这个token。而jwt也是经常作为一种安全的token使用。

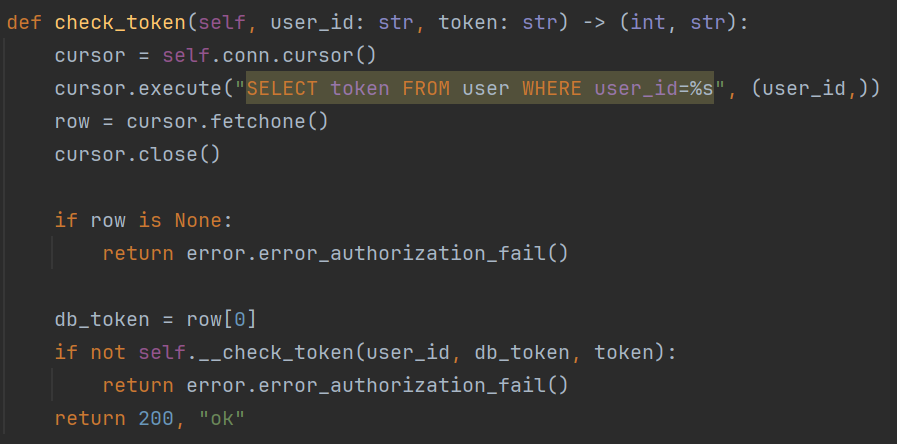
下面的函数分别用于：

Check\_token:用来检测用户的token，识别和验证用户信息。

Jwt\_encode:用来进行jwt加密。

Jwt\_decode:用来进行jwt解密。





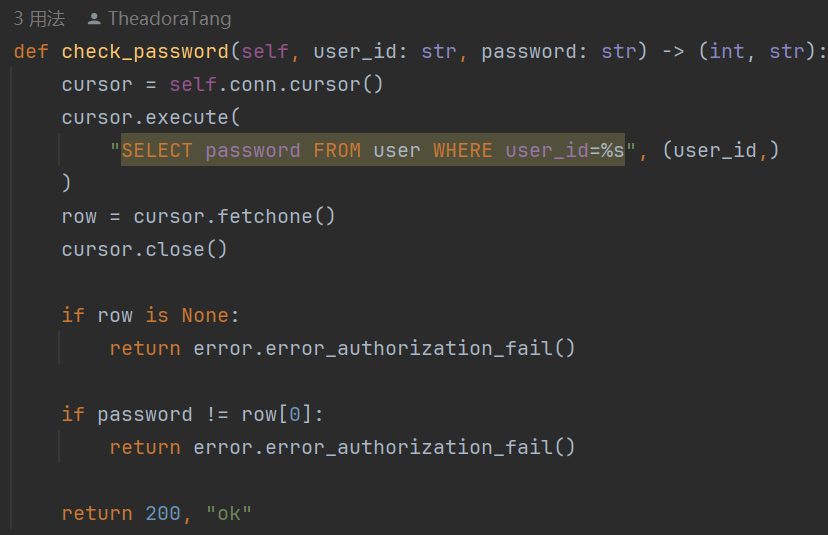
### 3.2.2 register登录

首先检测用户是否存在，如果已经存在，根据error中的定义返回错误的code和message，如果用户不存在，我们将用户的信息存储到bookstore数据库的use表格中，同时返回code和message。



### 3.2.3 check password检查密码

该函数是用来检测用户的密码是否正确，如果不正确返回对应error的code和message，如果正确的话，返回200。在login登录以及更改密码等功能中需要调用该函数检验用户的身份。



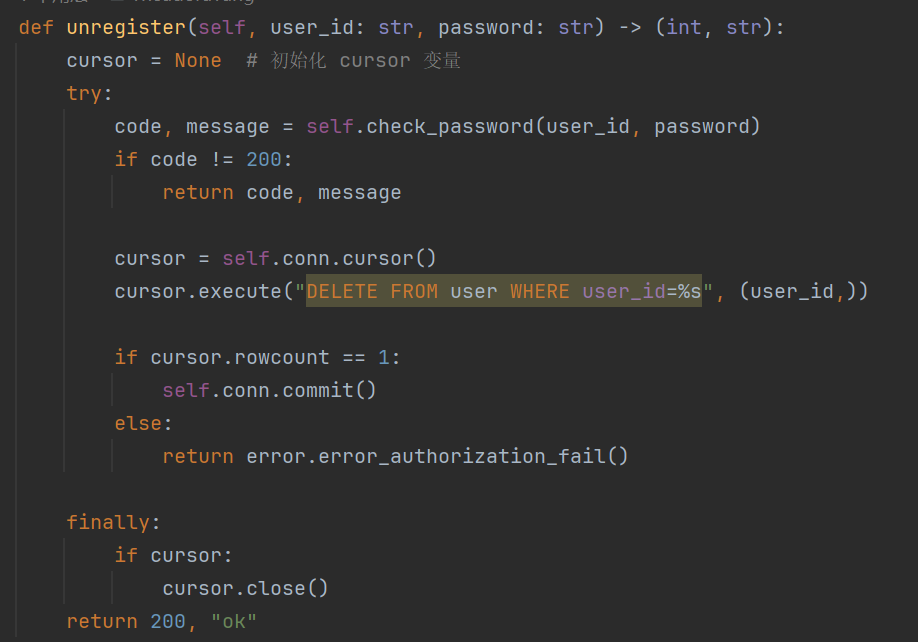
### 3.2.4 logout登出

首先利用chech token进行服务器识别和验证用户，然后更新terminal和token，填入到数据库中。



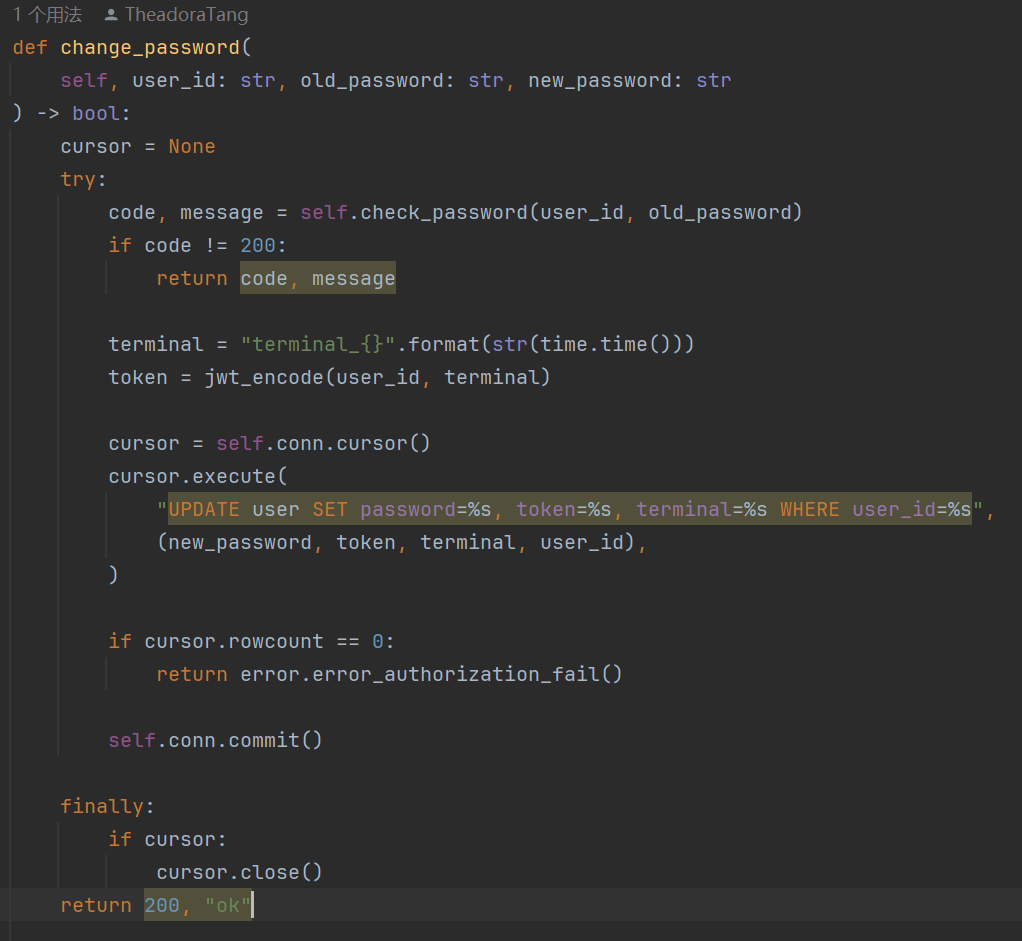
### 3.2.5 unregister注销

首先检测用户输入的密码是否正确，如果验证用户身份成功，则从users中删除有关该用户的所有信息。



### 3.2.6 change password 更改密码

首先检测用户输入的密码是否正确，如果验证用户身份成功，则在user的数据原有基础上对password进行更新（terminal和token也同时更新）。



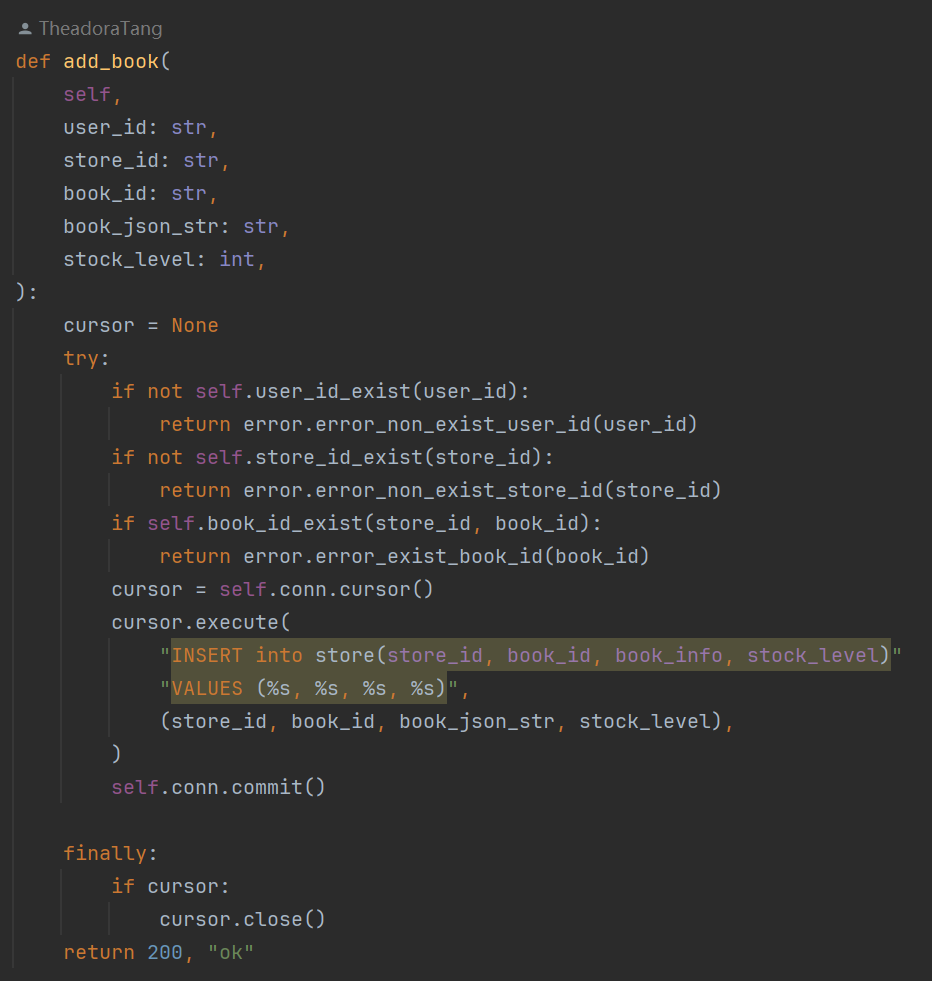
## 3.3 卖家功能

卖家功能主要在seller.py中实现。

### 3.3.1 add\_book 上架书本

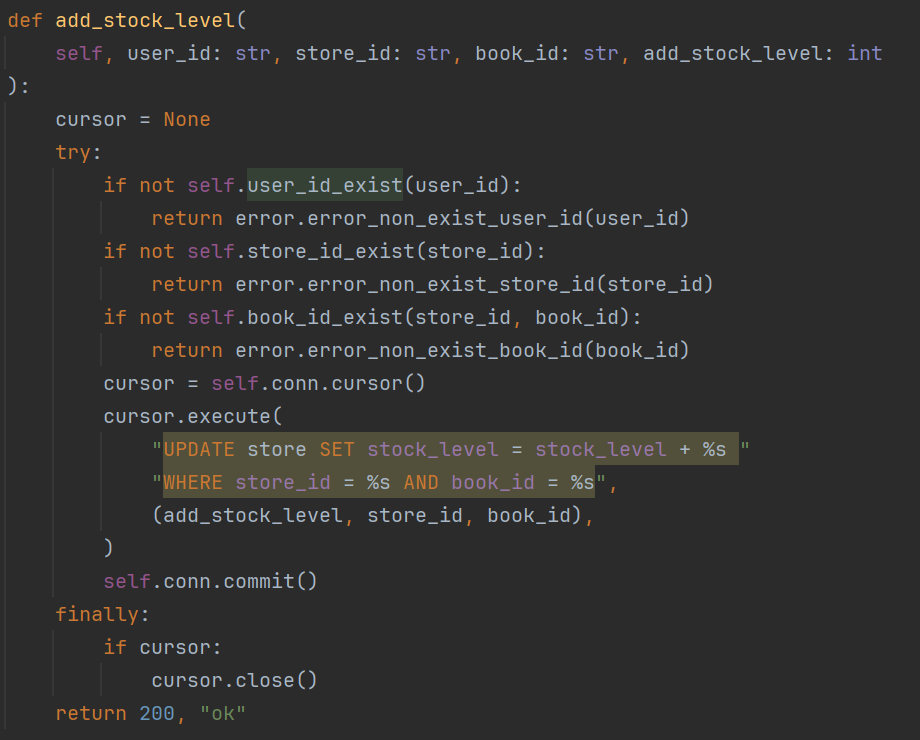
该方法负责将书籍添加到商店。它接受参数，如user\_id、store\_id、book\_id、book\_json\_str（可能是包含书籍信息的JSON字符串）和stock\_level。

首先，检查数据库中用户、商店和书籍ID的存在性，如果不存在则反对各自的code和message。如果没有错误则执行INSERT语句，将提供的信息添加到store表中。



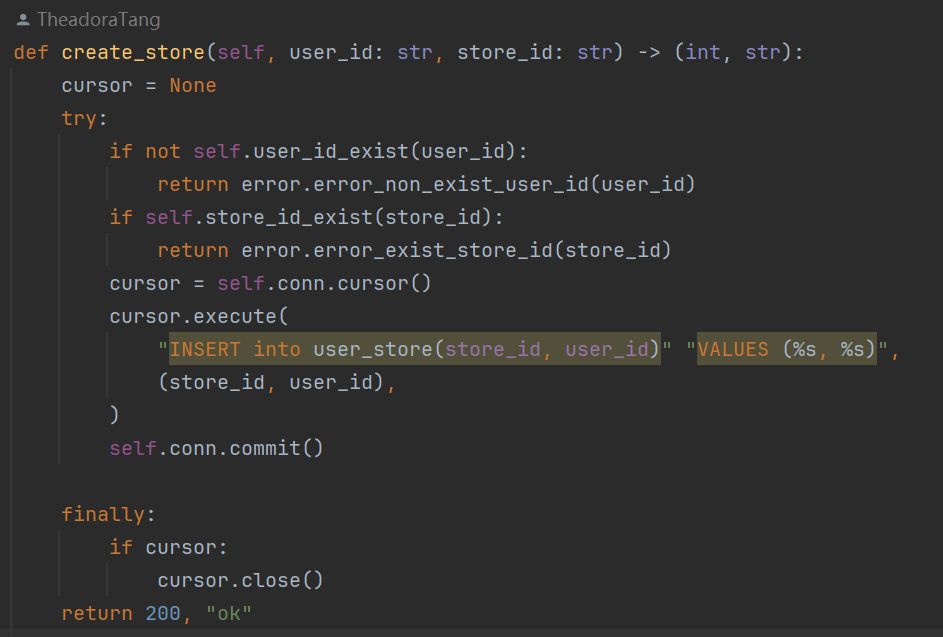
### 3.3.2 add\_stock\_level 增加库存

接受参数后，函数检查数据库中用户、商店和书籍ID的存在性，并返回对应的message和code，如果一切正常，则执行UPDATE语句，增加指定商店中指定书籍的库存水平。



### 3.3.3 create\_store 创建商店

接受参数后，函数检查数据库中用户ID以及商店ID的存在性，并返回对应的message和code，如果一切正常，则执行INSERT语句，在对应user\_id下创建一个store。

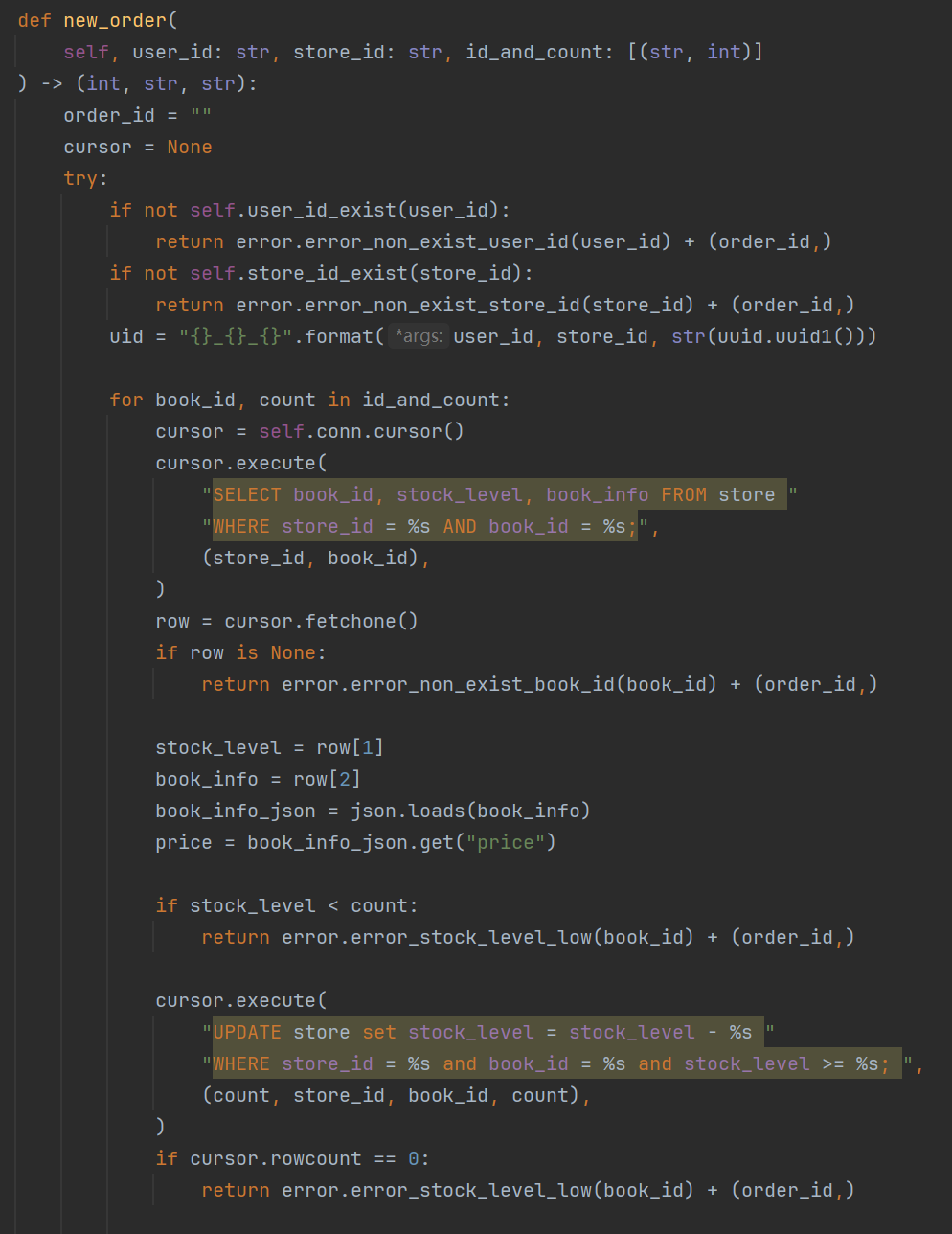


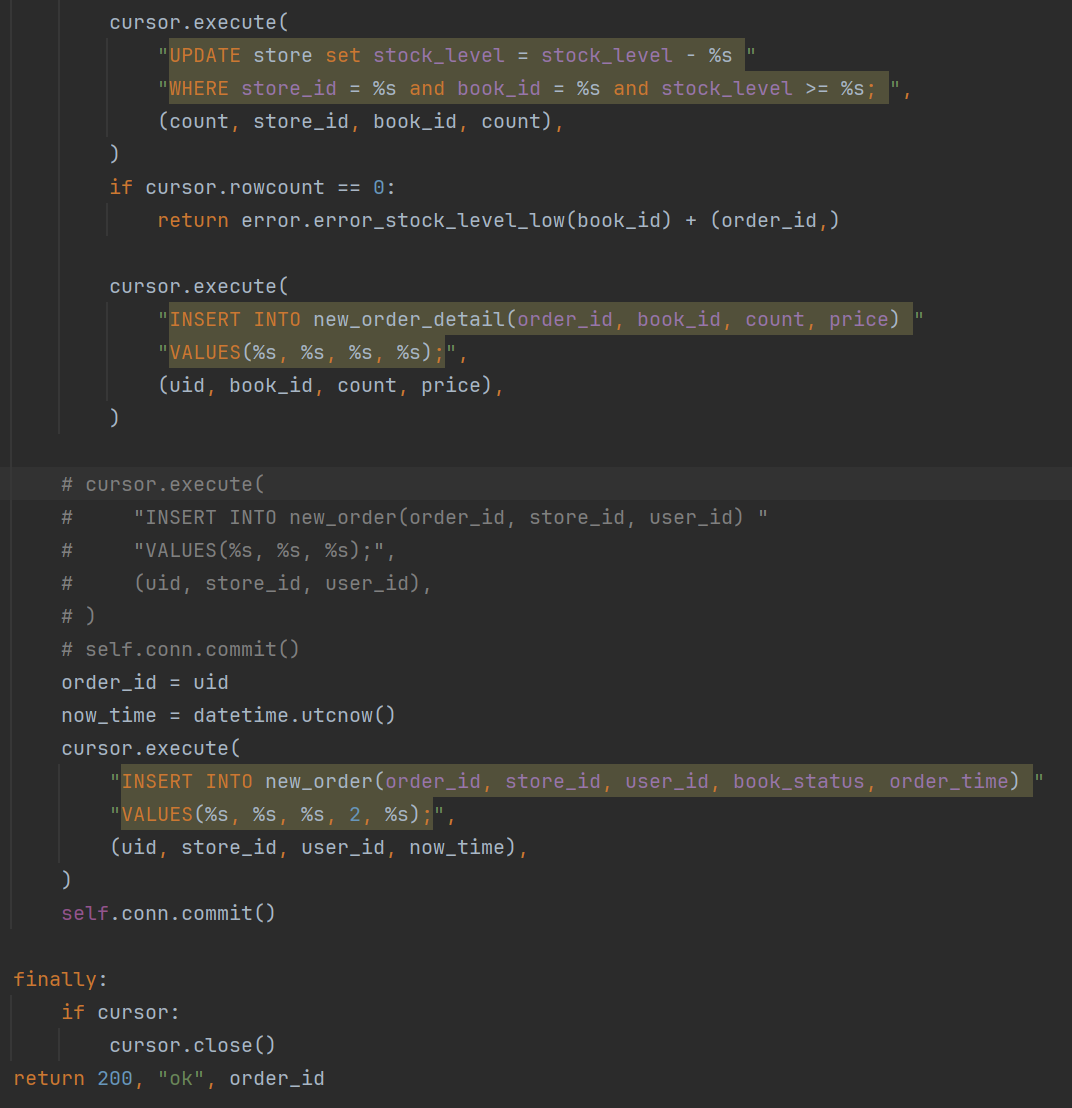
## 3.4 买家功能

### 3.4.1 new order 新增订单

该方法接受用户ID、商店ID和`id\_and\_count`，其中`id\_and\_count`是一个包含书籍ID和数量的列表。

使用`uuid.uuid1()`生成一个唯一的UUID，并将其与用户ID和商店ID组合以创建订单ID（`uid`）。遍历书籍ID和数量列表，对`id\_and\_count`中的每个书籍ID和数量进行迭代。之后检查用户、商店和书籍ID的存在性，如果不存在则返回相应的错误信息。之后查询书籍信息和库存，如果查询结果为空，表示书籍不存在；否则，检查库存是否足够以满足订单数量，如果不足则返回相应的错误信息，如果满足则更新库存，并将该订单的详情记录到表格中，包括订单ID、商店ID、用户ID、书籍状态（这里设为2）和订单时间。

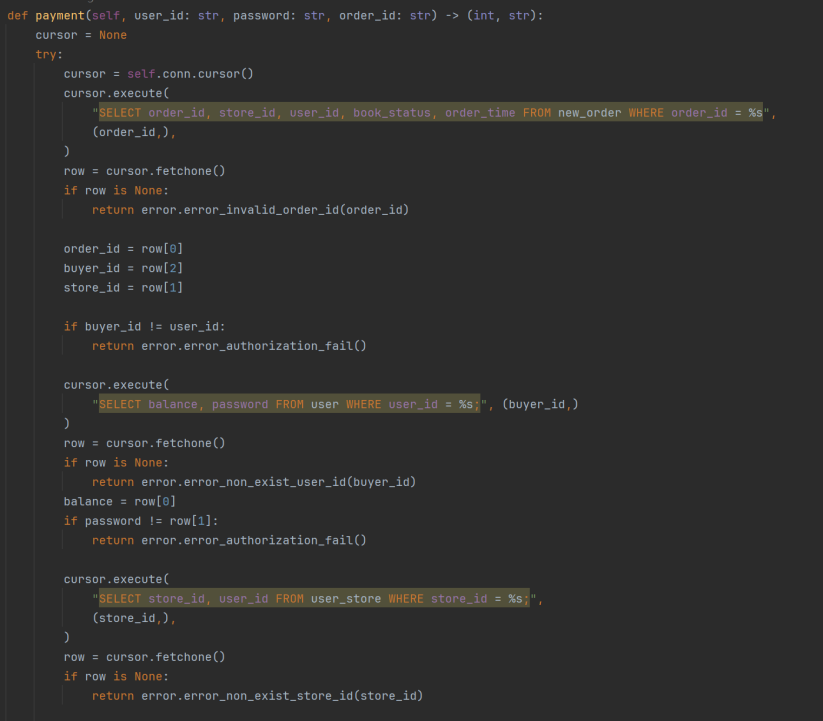


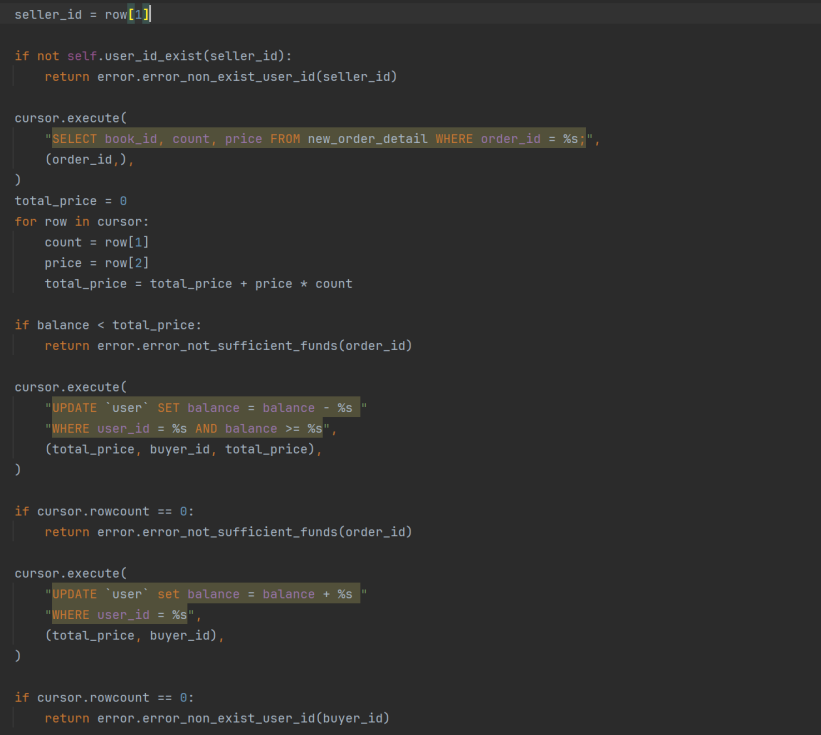


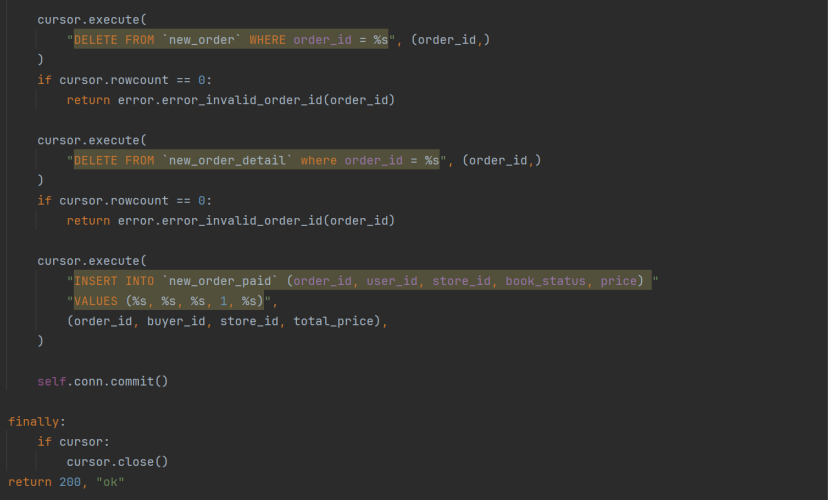
### 3.4.2 payment 付款

该方法接受用户ID、用户密码和订单ID，查询订单信息，包括订单ID、商店ID、用户ID、书籍状态和订单时间。如果查询结果为空，表示订单不存在，返回相应的错误信息。之后检查订单中的买家ID是否与传入的用户ID相符，如果不符则返回相应的权限错误。接下来查询用户表中的余额和密码信息，并验证用户的存在性以及密码是否正确，如果查询结果为空，表示用户不存在；否则，检查密码是否正确，如果不正确则返回相应的错误信息。

然后查询user\_store关联表，确保商店存在并检查商家ID是否存在，如果不存在则返回相应的错误信息。接下来查询订单详情表，计算订单的总价。如果用户余额小于订单总价，返回余额不足的错误信息。如果用户余额充足，则扣除用户余额，之后将已付款订单的信息插入到new\_order\_paid表格中，并将已付款订单的信息从new\_order和new\_order\_detail中删除，并更改book\_status的状态。



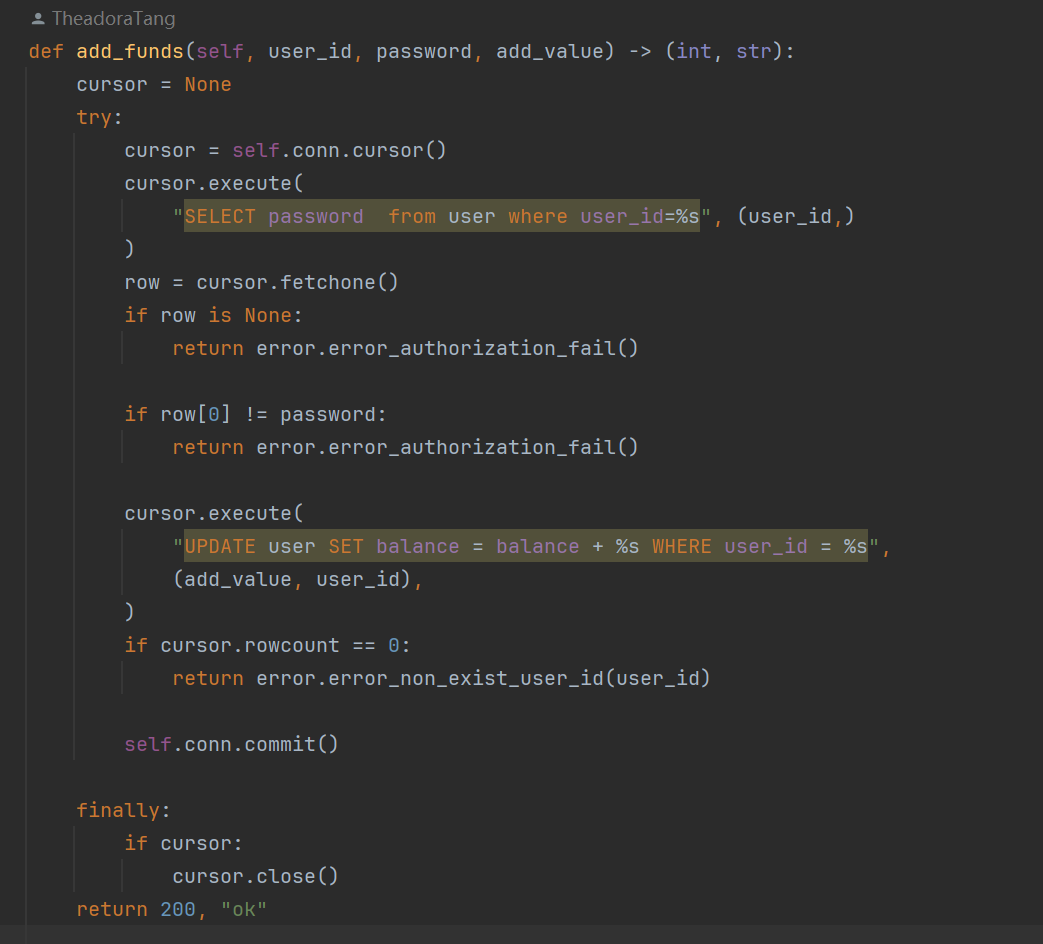




### 3.4.3 add\_funds 余额充值

该方法接受用户ID、用户密码和add\_value（要充值的金额）。

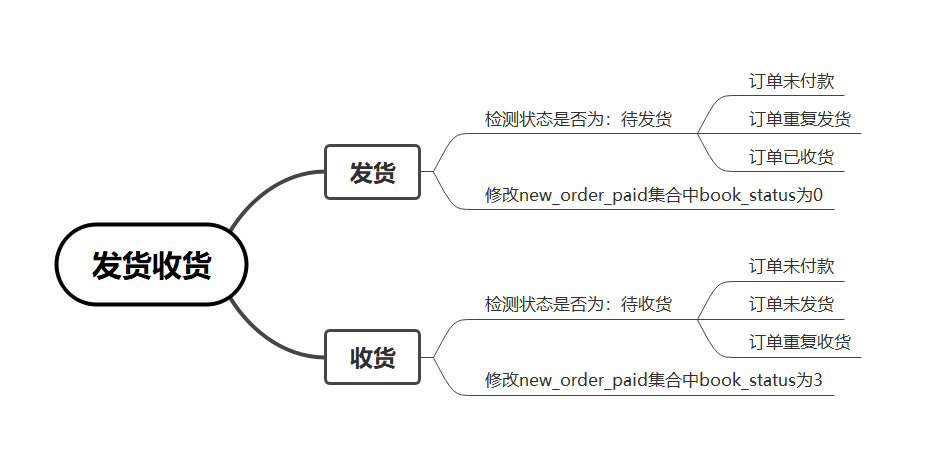
首先查询用户ID是否存在，然后检查用户的密码是否错误，之后利用UPDATE将对应用户的余额（balance）进行增加。



## 3.5 发货收货

### 3.5.1 初始准备

发货收货功能如下：



发货、收货功能基于上述的buyer和seller的实现，也就是在创建订单、充值、支付等功能实现成功的基础上进行开发。

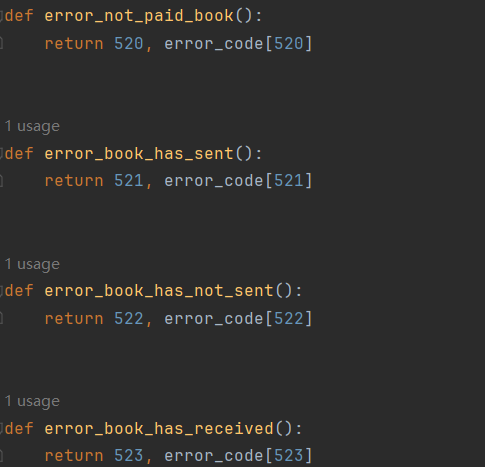
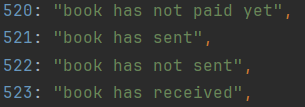
首先需要注意的是只有成功创建且完成支付的订单才可以进入到“待发货”，为了标记这个状态，我们在new\_order中增加书的状态“book\_status”

另外，因为payment的实现中，会把已经付款的订单从new\_order和new\_order\_details两个集合中删掉，而发货要基于这些这些订单，所以我们增加了一个新的集合new\_order\_paid来保存已经付款的订单，当发货的时候直接在这个订单进行搜索。

在这里定义“book\_status”的意义:

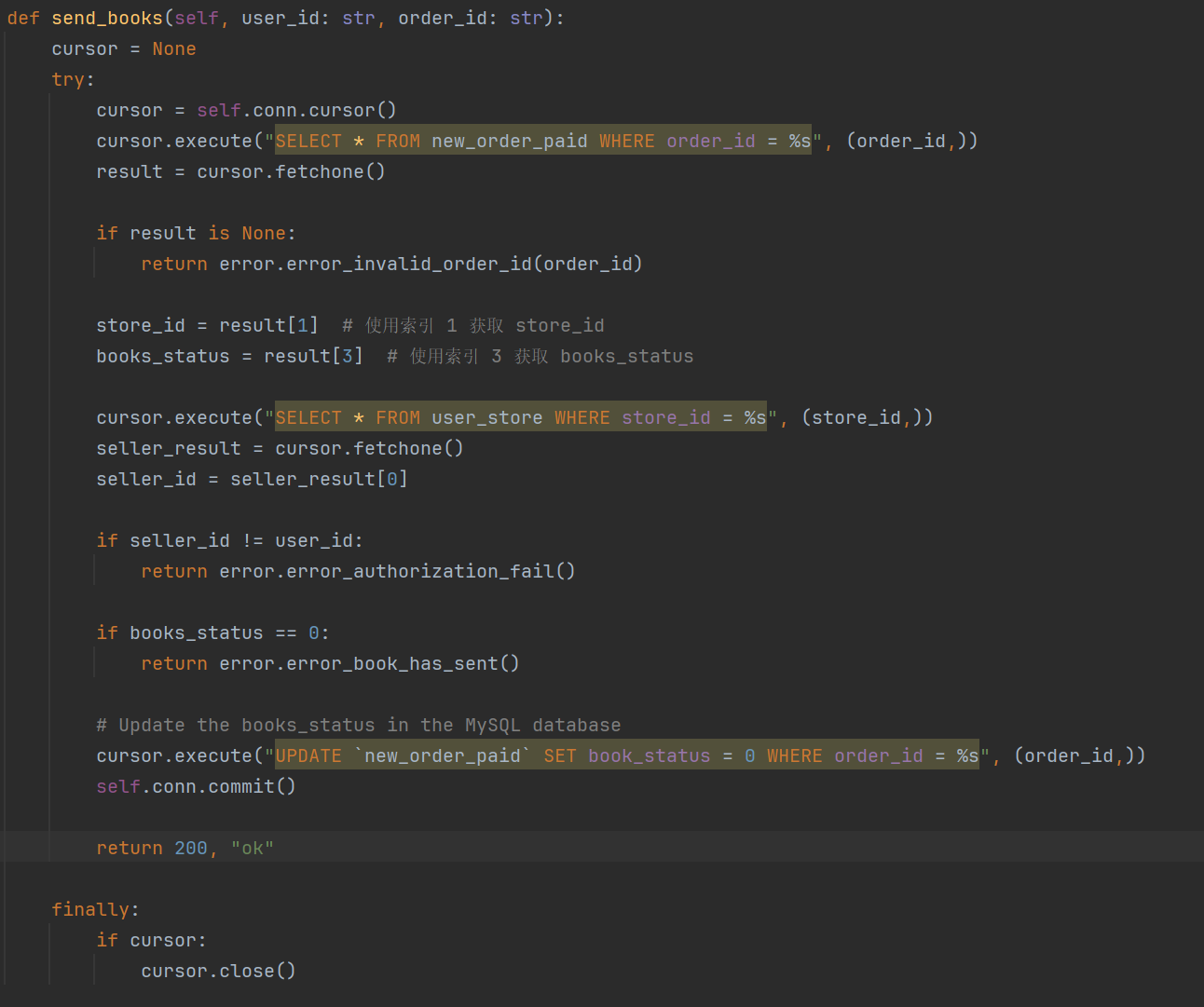


另外增加error.py中定义相应的对于书籍状态的报错：



### 3.5.2 发货

首先查询数据库，检查是否存在具有指定 order\_id 的订单，查询的表是 new\_order\_paid。如果未找到订单（result is None），则返回一个指示订单 ID 无效如果订单存在，代码从查询结果中提取了一些信息，如 `store\_id` 和 `books\_status`。接着，查询了 user\_store 表，获取了与订单相关联的商店信息，并提取了 `eller\_id。如果 seller\_id 不等于 user\_id，表示当前用户没有权限操作这个订单，返回一个表示授权失败的错误。然后检查书籍状态。如果以上检查都通过，它会更新 `new\_order\_paid` 表中的 `book\_status` 列为 0，表示书籍已发送，并提交事务。



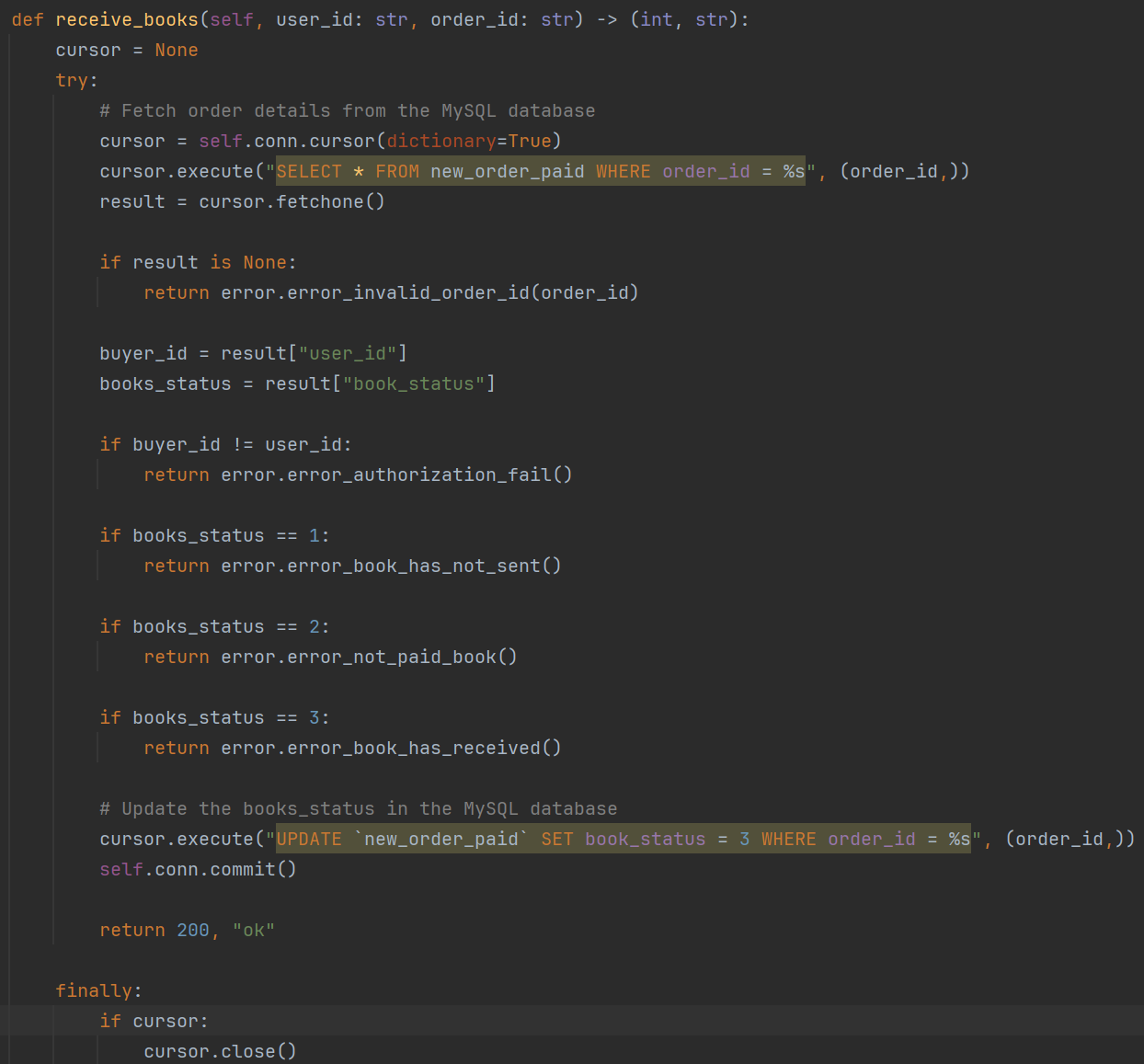
### 3.5.3 收货

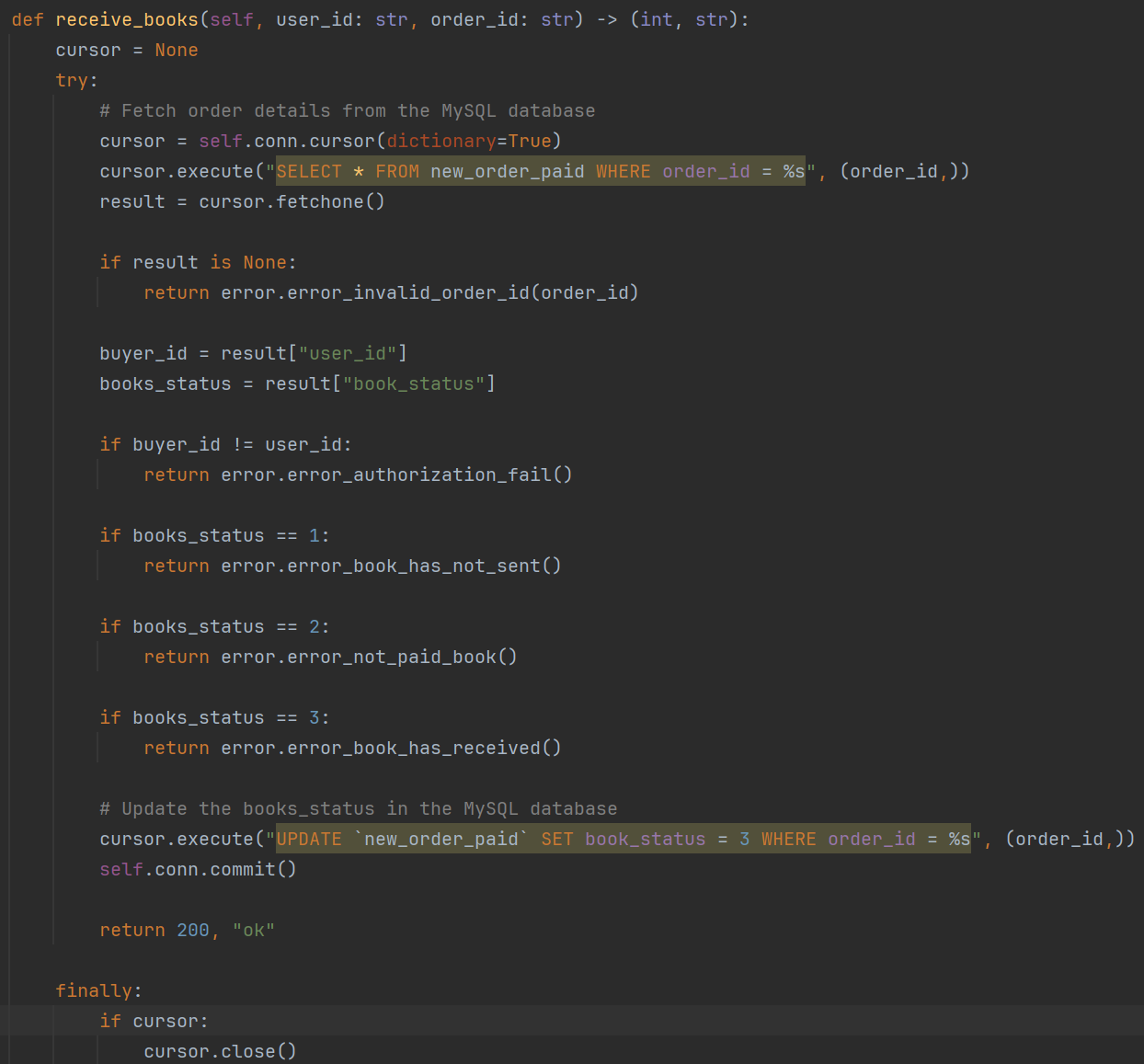
首先尝试从MySQL数据库中检索与给定订单ID相关的订单信息。使用SELECT查询从表`new\_order\_paid`中选择订单信息。如果找不到相关订单，将返回与无效订单ID相关的错误。获取到订单信息后，函数检查订单的各种状态。具体来说：

- 如果订单不存在（result为None），则返回与无效订单ID相关的错误。

- 如果订单属于不同的买家（buyer\_id与user\_id不匹配），则返回与授权失败相关的错误。

之后检查图书状态，如果图书状态不为0，则返回响应的错误code和message。如果通过了所有检查，函数将更新订单的图书状态为3（已接收）。这通过执行UPDATE查询来完成，并随后提交事务。

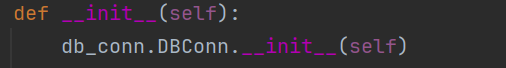




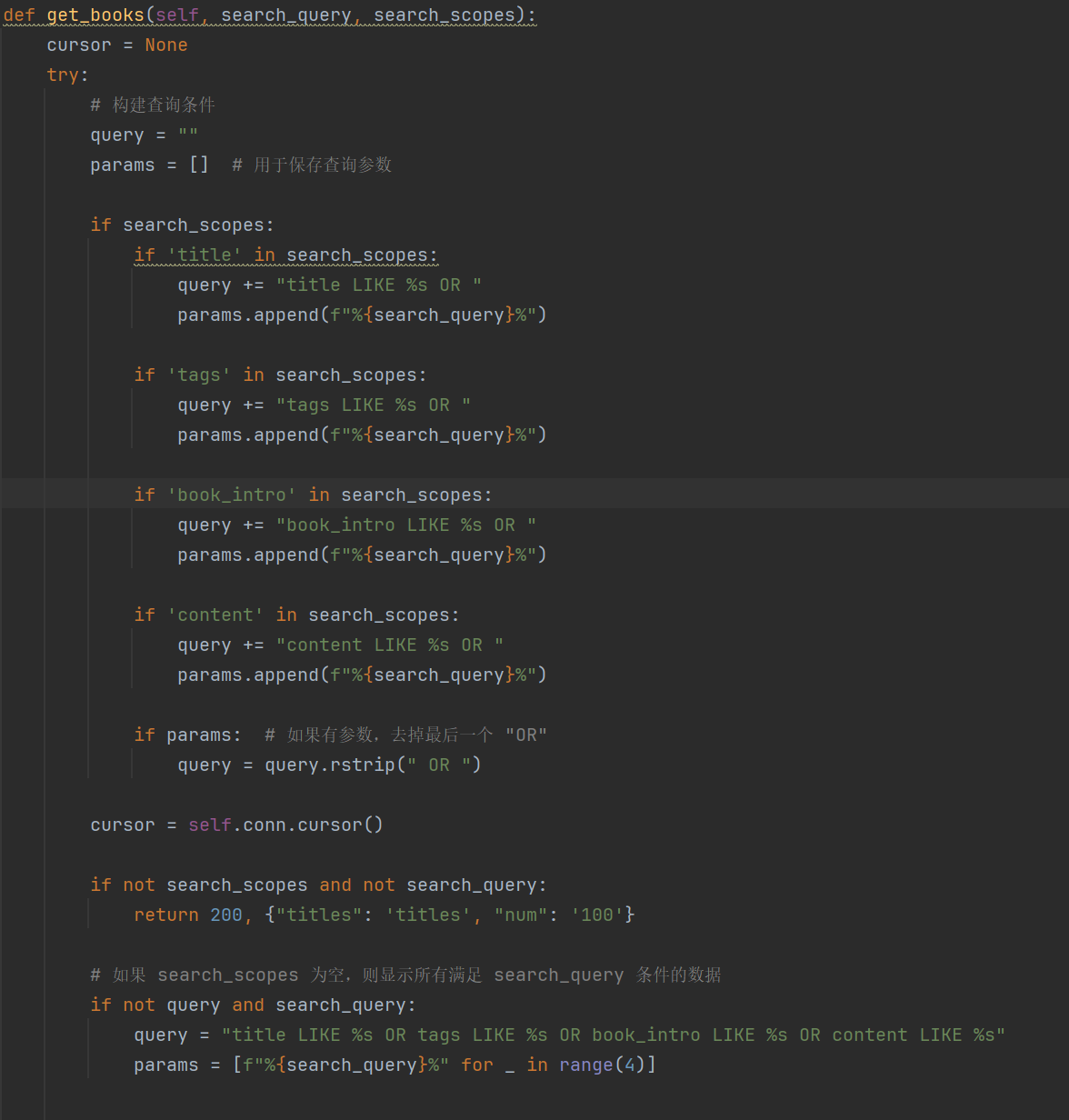
## 3.6 搜索图书

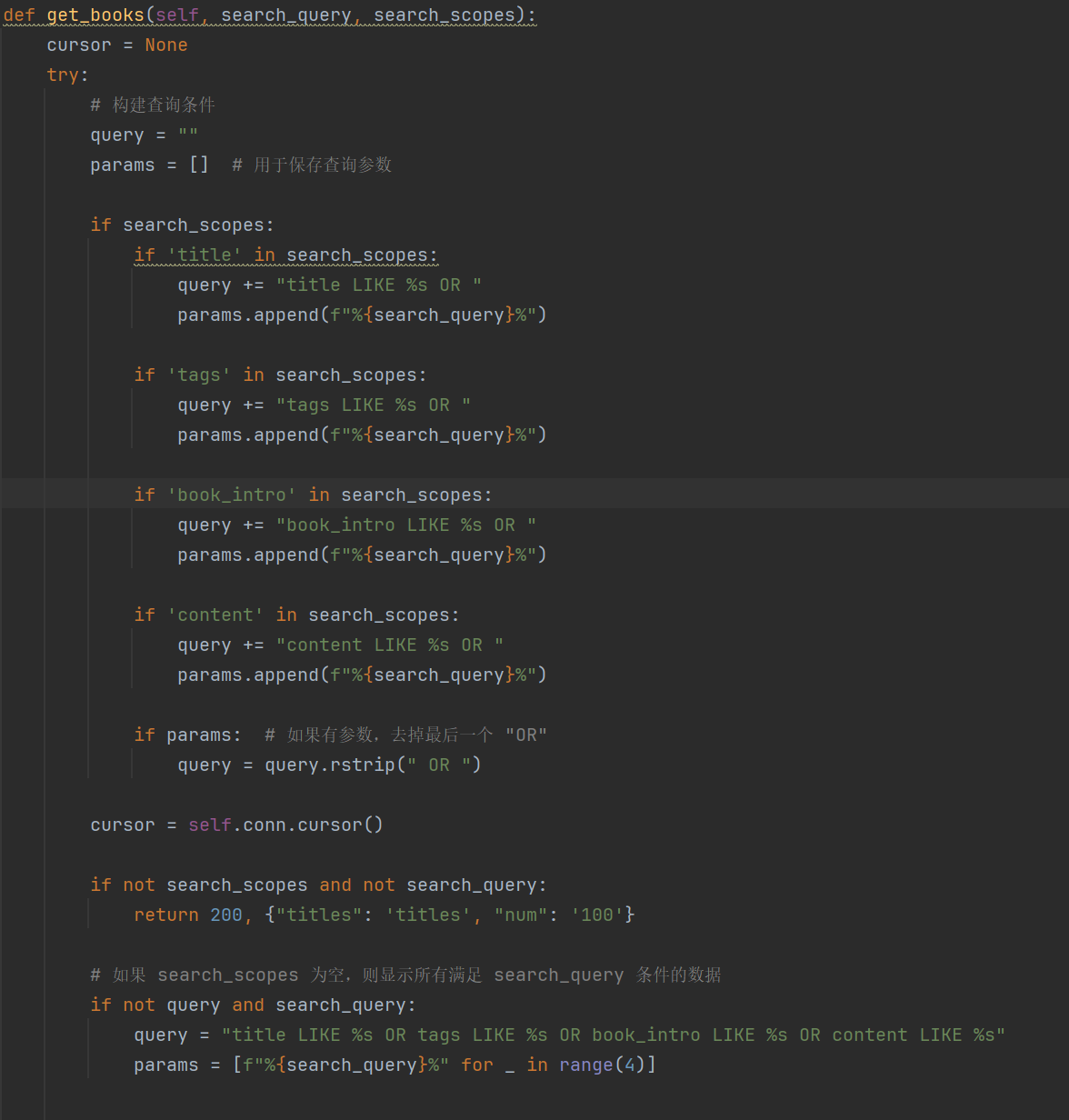
`SearchBooks`类：

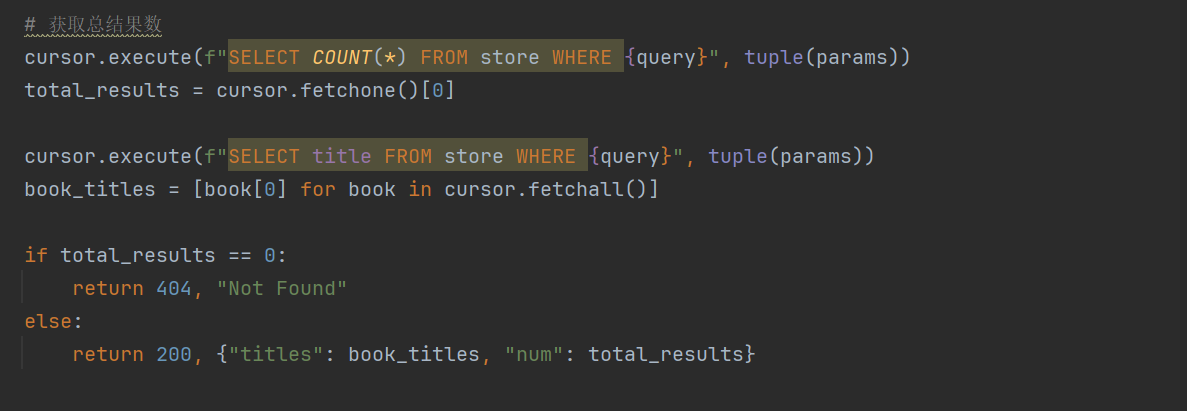
- `\_\_init\_\_` 方法：在初始化对象时，它创建了一个 MongoDB 客户端连接到本地的 MongoDB 数据库，并选择名为 'bookstore' 的数据库作为工作数据库。



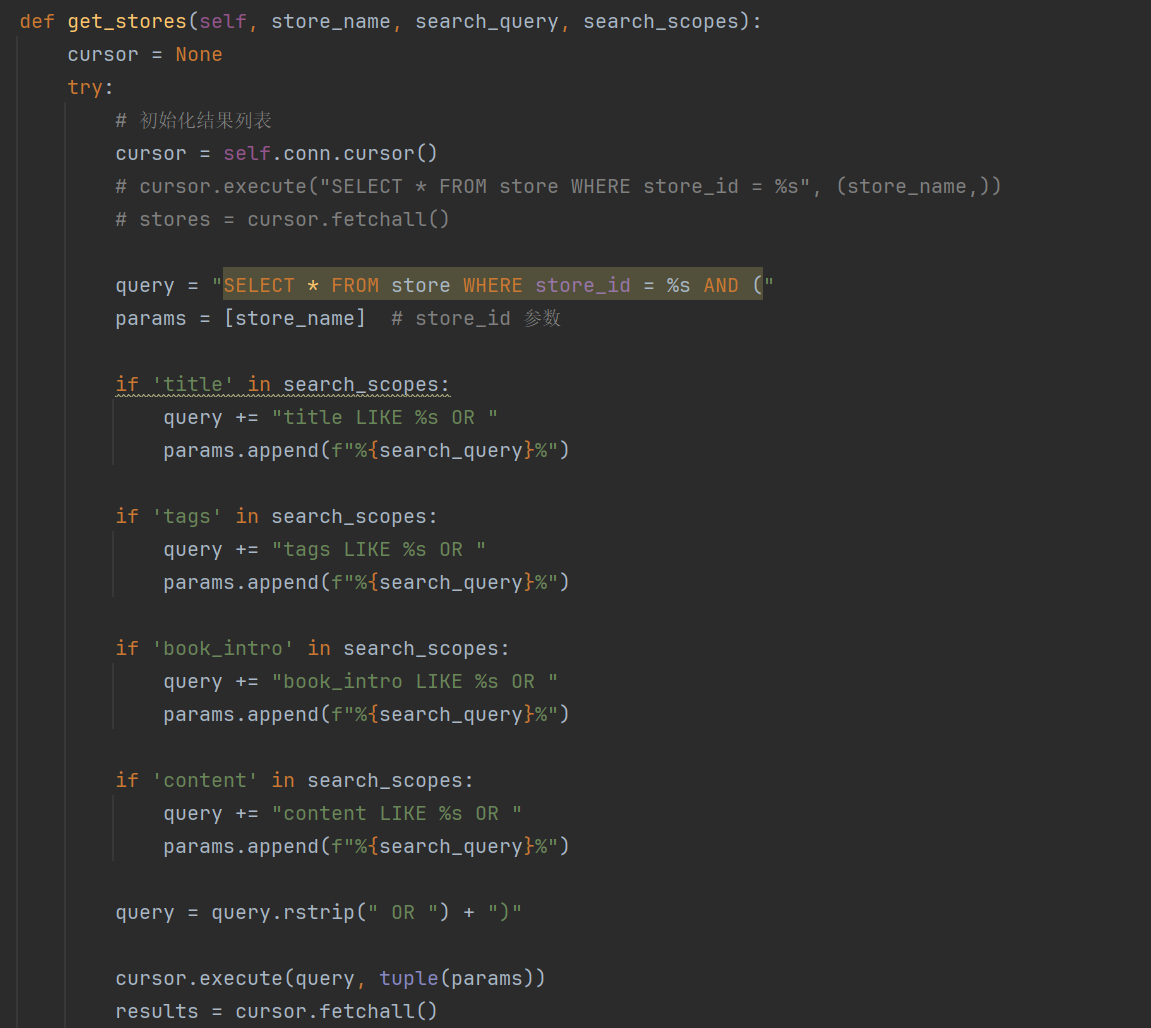
- `get\_books` 方法：用于搜索书籍。根据传入的搜索关键词 (`search\_query`) 和搜索范围 (`search\_scopes`) 构建 MySQL查询条件，然后查询数据库中的书籍信息。查询条件会根据搜索范围的不同包括书名、标签、书籍简介和内容，使用正则表达式进行模糊匹配。如果有匹配的书籍，它返回 HTTP 状态码 200 和包含书籍标题列表和总结果数的响应。如果没有匹配的书籍，返回 HTTP 状态码 404 和 "Not Found" 的响应。

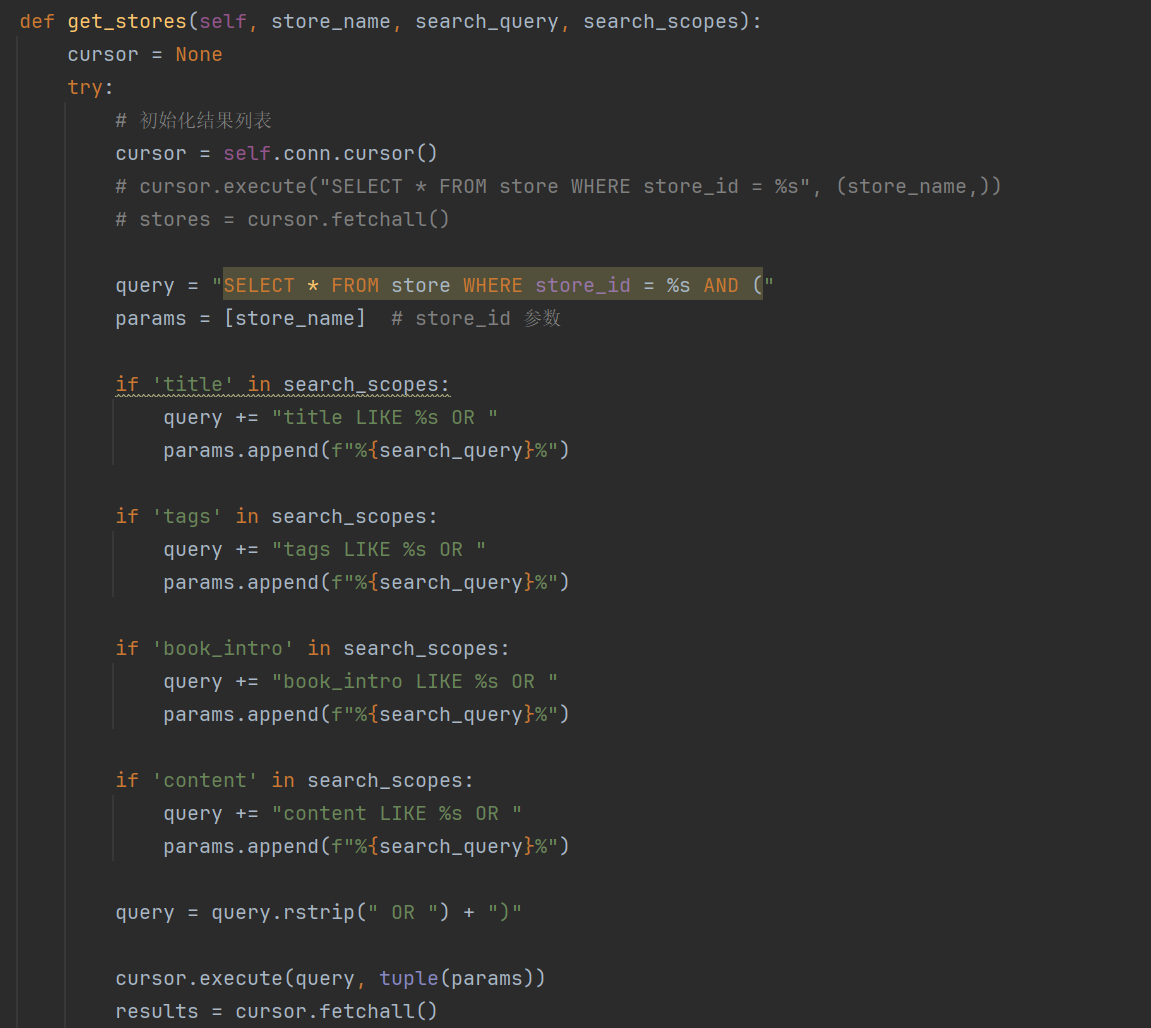


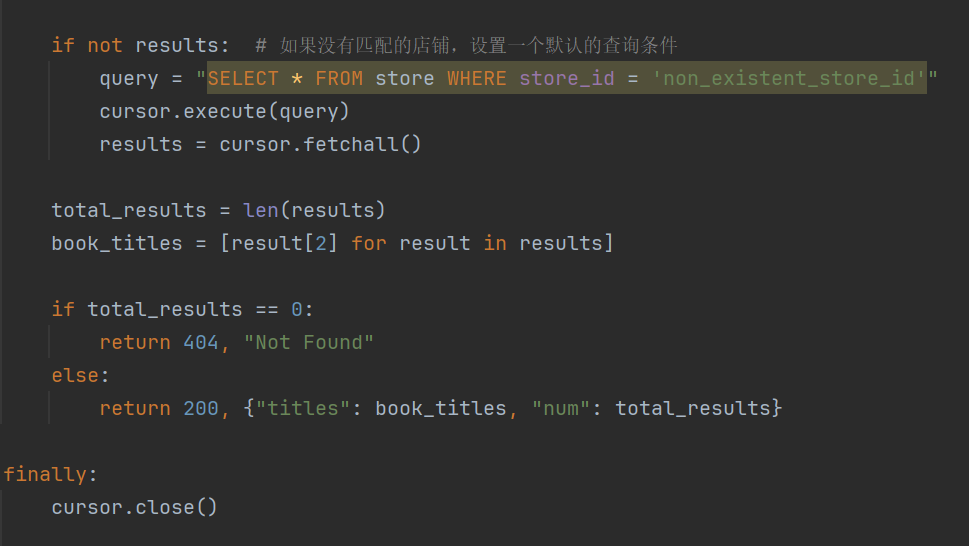




- `get\_stores` 方法：用于搜索店铺内的书籍。根据传入的店铺名称 (`store\_name`)、搜索关键词 (`search\_query`) 和搜索范围 (`search\_scopes`) 构建MySQL查询条件，然后查询数据库中的店铺信息。查询条件会根据搜索范围的不同包括书名、标签、书籍简介和内容，使用正则表达式进行模糊匹配。如果有匹配的书籍，它返回 HTTP 状态码 200 和包含书籍标题列表和总结果数的响应。如果没有匹配的书籍，返回 HTTP 状态码 404 和 "Not Found" 的响应。







总体来说，只要能根据条件找到图书，即可返回200，其他情况返回404。

# 测试

## 4.1 原有pytest

没有对原有pytest进行更改。

## 4.2 关于搜索功能的pytest检验

为了检验图书搜索功能是否正确，我自行编写了test\_search\_books.py代码。以下是这段代码的逻辑和功能介绍：

1. 定义测试类 `TestSearch`：

这个类包含了一系列测试用例，用于测试书店搜索功能。

2. `@pytest.fixture(autouse=True)` 装饰器：

-`pre\_run\_initialization` 方法被标记为自动使用的 pytest fixture，意味着它会在每个测试方法之前自动执行。

在这个 fixture 方法中，进行了一些初始化工作，包括创建用于检测的书本和店铺名称，以及创建了一个 `search\_books.Search` 实例、设置了搜索关键词（`search\_query`）、搜索范围（`search\_scopes`）和一个店铺名称（`store\_name`）。

3. 测试用例：

`test\_search\_books` 方法测试书店搜索功能。它使用预定义的搜索关键词和搜索范围，向书店发送搜索请求，并检查返回的状态码是否为 200（成功）。

`test\_search\_books\_wrong` 方法测试错误的搜索情况。它使用一个无效的搜索关键词（'txh'）发送搜索请求，然后检查返回的状态码是否为 404（未找到）。

`test\_search\_stores` 方法测试店铺搜索功能。它使用预定义的店铺名称、搜索关键词和搜索范围，向书店发送店铺搜索请求，并检查返回的状态码是否为 200（成功）。

`test\_search\_stores\_wrong` 方法测试错误的店铺搜索情况。它使用一个不存在的店铺名称（'store111'）发送店铺搜索请求，然后检查返回的状态码是否为 404（未找到）。

`test\_search\_books\_no\_scope`方法测试在没有指定搜索范围的情况下执行图书搜索的行为。并检查返回的状态码是否为 200（成功）。（在没有指定搜索范围的情况下默认进行全局搜索）

`test\_search\_books\_no\_query\_and\_scope`方法测试了在没有指定搜索查询和搜索范

围的情况下执行图书搜索的行为。并检查返回的状态码是否为 200（成功）。

`test\_search\_books\_single\_scope\_no\_match`方法测试了在指定了搜索范围但搜索不

到匹配项的情况下执行图书搜索的行为。然后检查返回的状态码是否为 404（未找到）。

`test\_search\_books\_multiple\_scopes\_no\_match`方法测试了在指定了多个搜索范围

但搜索不到匹配项的情况下执行图书搜索的行为。然后检查返回的状态码是否为 404（未找到）。

`test\_search\_stores\_single\_store\_no\_match`:方法测试了在指定了商店名称和搜索范

围但搜索不到匹配项的情况下执行商店搜索的行为。然后检查返回的状态码是否为 404（未找到）。

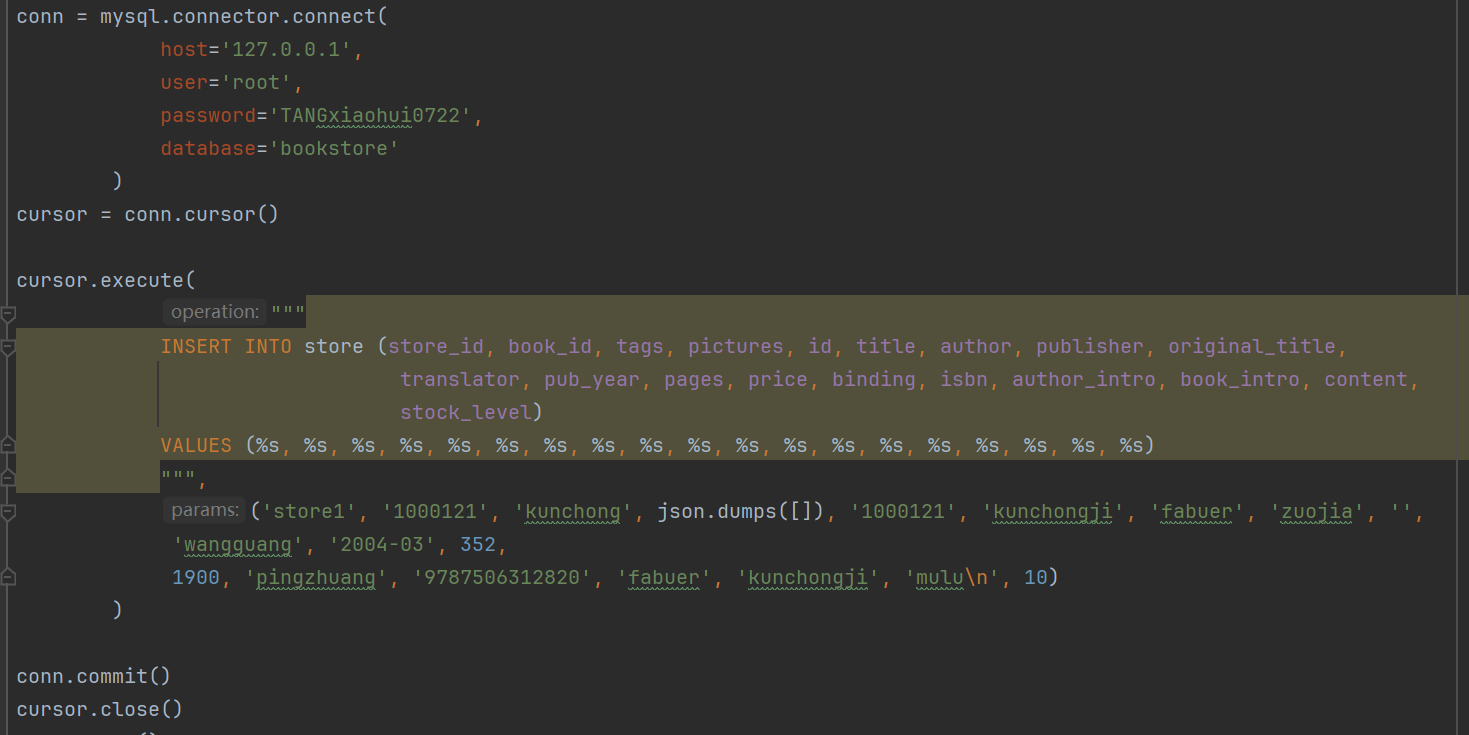
`test\_search\_stores\_multiple\_stores\_no\_match`:这个测试用例测试了在指定了多个

商店名称和搜索范围但搜索不到匹配项的情况下执行商店搜索的行为。然后检查返回的状态码是否为 404（未找到）。

## 4.3 关于发货收货功能的pytest检验

1. 数据库初始化：

使用`mysql.connector`建立数据库连接，并插入一条记录到`store`表中，以便后续测试使用。



2.测试函数：

`test\_search\_books`: 测试图书搜索功能是否正常工作，期望返回状态码200。

`test\_search\_books\_wrong`: 测试使用错误的查询关键字是否能够正确返回状态码404。

`test\_search\_stores`: 测试店铺搜索功能是否正常工作，期望返回状态码200。

`test\_search\_stores\_wrong`: 测试使用错误的店铺名称是否能够正确返回状态码404。

`test\_search\_books\_no\_scope`: 测试在没有搜索范围的情况下是否能够正常返回状态码200。

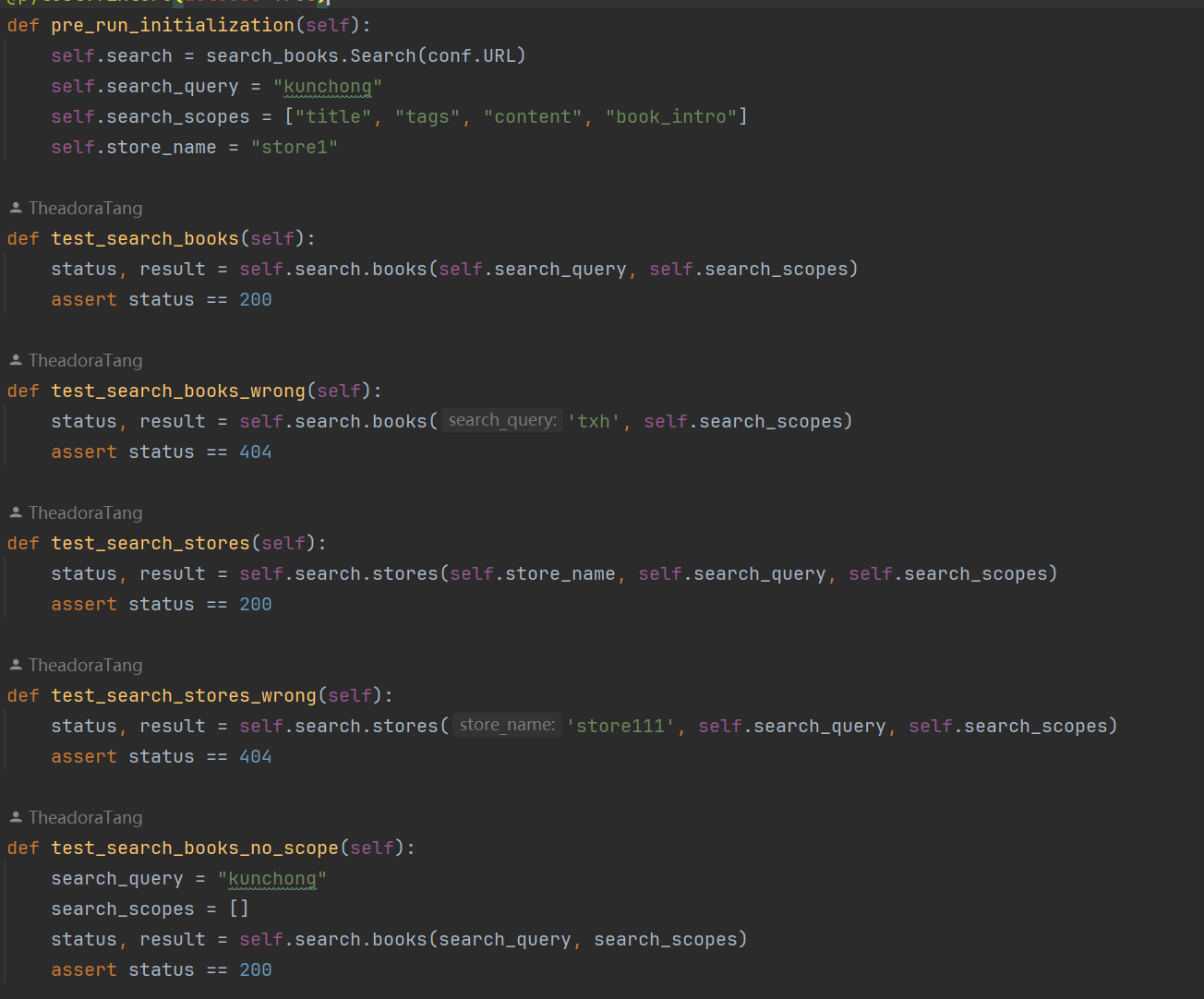
`test\_search\_books\_no\_query\_and\_scope`: 测试在没有查询关键字和搜索范围的情况下是否能够正常返回状态码200。

`test\_search\_books\_single\_scope\_no\_match`: 测试在单一搜索范围下没有匹配结果的情况下是否能够正常返回状态码404。

`test\_search\_books\_multiple\_scopes\_no\_match`: 测试在多个搜索范围下没有匹配结果的情况下是否能够正常返回状态码404。

`test\_search\_stores\_single\_store\_no\_match`: 测试在单一店铺下没有匹配结果的情况下是否能够正常返回状态码404。

`test\_search\_stores\_multiple\_stores\_no\_match`: 测试在多个店铺下没有匹配结果的情况下是否能够正常返回状态码404。



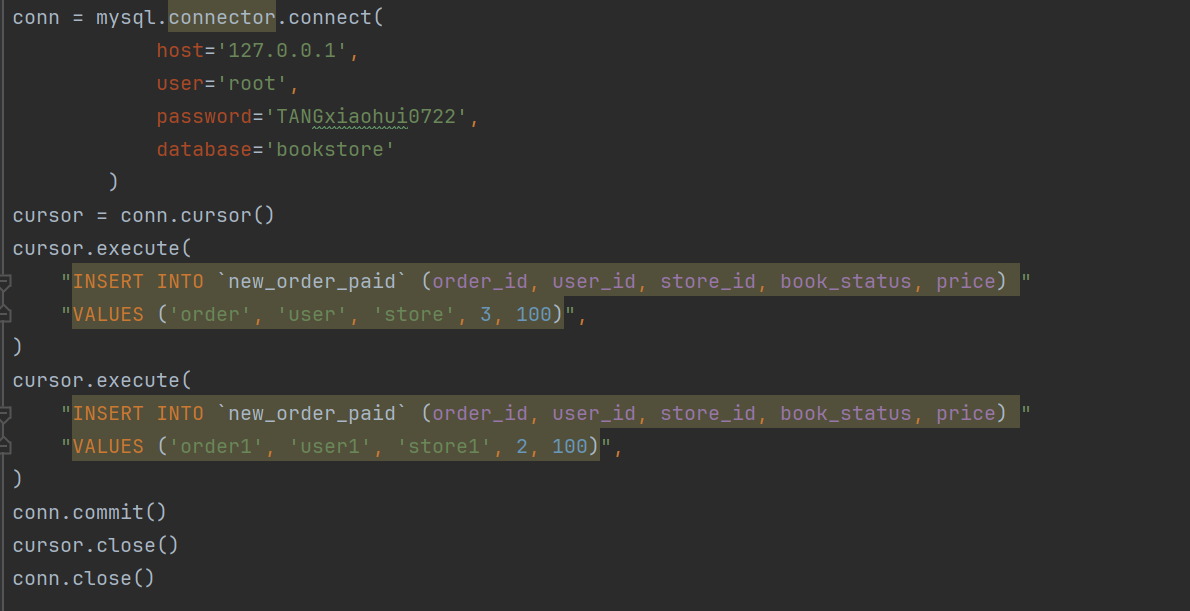


## 4.4 关于订单功能pytest检验

1. 数据库初始化：

- 使用`mysql.connector`建立数据库连接。

- 插入两条测试订单数据到`new\_order\_paid`表中，用于后续测试。



2. 测试类 `TestSendAndReceive`：

- `@pytest.fixture(autouse=True)` 注解用于实现测试前的初始化操作，包括生成卖家、店铺、买家、图书等测试数据。

3. \*\*测试函数：\*\*

- `test\_ok\_send`: 测试正常发货的情况，期望返回状态码200。

- `test\_invalid\_order\_id\_send`: 测试使用不存在的订单号发货是否能够正确返回非200状态码。

- `test\_authorization\_error\_send`: 测试权限错误情况，即卖家ID与订单中的卖家ID不匹配，期望返回非200状态码。

- `test\_books\_duplicate\_send\_send`: 测试重复发货的情况，期望第一次发货返回200，第二次发货返回非200。

- `test\_order\_not\_paid\_send`: 测试未付款的订单发货情况，期望返回518状态码。

- `test\_ok\_receive`: 测试正常收货的情况，期望返回状态码200。

- `test\_authorization\_error\_receive`: 测试权限错误情况，即买家ID与订单中的买家ID不匹配，期望返回非200状态码。

- `test\_invalid\_order\_id\_receive`: 测试使用不存在的订单号收货是否能够正确返回非200状态码。

- `test\_books\_not\_send\_receive`: 测试未发货的订单收货情况，期望返回非200状态码。

- `test\_books\_not\_paid\_receive`: 测试未付款的订单收货情况，期望返回518状态码。

- `test\_books\_already\_received`: 测试已经收到的订单再次收货情况，期望返回523状态码。

- `test\_books\_not\_paid`: 测试未付款的订单收货情况，期望返回520状态码。

这些测试函数涵盖了发货和收货的不同情况，确保系统在各种条件下都能够正确处理。

## 4.5 接口介绍

部分接口请见doc文件夹下原有的md文件。

后40%的接口介绍如下：

**1.全局搜索图书：**

URL：POST http://127.0.0.1/search/search\_books

Request：

{

“search\_query”: “search\_query”

“search\_scopes“: “search\_scopes“

}

**属性说明：**

|  |  |
| --- | --- |
| **Key** | **描述** |
| Search\_query | 书籍的名字 |
| Search\_scopes | 搜索的范围（tags，content，title等） |

**Status Code：**

|  |  |
| --- | --- |
| **Code** | **描述** |
| 200 | 找到图书 |
| 404 | 无法找到符合条件的图书 |

**2.在店铺中搜索图书：**

URL：POST http://127.0.0.1/search/search\_stores

Request：

{

“search\_query”: “search\_query”

“search\_scopes“: “search\_scopes“  
“store\_name”：“store\_name”

}

**属性说明**：

|  |  |
| --- | --- |
| **Key** | **描述** |
| Search\_query | 书籍的名字 |
| Search\_scopes | 搜索的范围（tags，content，title等） |
| Store\_name | 店铺的名字 |

**Status Code：**

|  |  |
| --- | --- |
| **Code** | **描述** |
| 200 | 找到图书 |
| 404 | 无法找到符合条件的图书 |

**3发货：**

URL：POST http://127.0.0.1/send\_receive/send\_books

Request：

{

"user\_id": seller\_id,

"order\_id": order\_id

}

**属性说明：**

|  |  |
| --- | --- |
| **Key** | **描述** |
| user\_id | 卖家 |
| order\_id | 订单号 |

**Status Code：**

|  |  |
| --- | --- |
| **Code** | **描述** |
| 200 | 找到订单并且成功发货 |
| 518 | 订单不存在 |
| 511 | 卖家不存在 |
| 520 | 订单未付款 |
| 521 | 订单已发货 |
| 523 | 订单已收货 |

1. **收货：**

URL：POST http://127.0.0.1/send\_receive/receive\_books

Request：

{

"user\_id": seller\_id,

"order\_id": order\_id

}

**属性说明：**

|  |  |
| --- | --- |
| **Key** | **描述** |
| user\_id | 买家 |
| order\_id | 订单号 |

**Status Code：**

|  |  |
| --- | --- |
| **Code** | **描述** |
| 200 | 找到订单并且成功收货 |
| 518 | 订单不存在 |
| 511 | 买家不存在 |
| 520 | 订单未付款 |
| 521 | 订单未发货 |
| 523 | 订单已收货 |

1. **订单取消**

URL：POST http://127.0.0.1/order/new\_order\_cancel

Request：

{

"user\_id": seller\_id,

"order\_id": order\_id

}

**属性说明：**

|  |  |
| --- | --- |
| **Key** | **描述** |
| user\_id | 买家 |
| order\_id | 订单号 |

**Status Code：**

|  |  |
| --- | --- |
| **Code** | **描述** |
| 200 | 找到订单并且成功取消 |
| 518 | 订单不存在（或已经被取消） |
| 511 | 买家不存在 |

1. **订单查询**

URL：POST http://127.0.0.1/order/check\_order

Request：

{

"user\_id": seller\_id,

}

**属性说明：**

|  |  |
| --- | --- |
| **Key** | **描述** |
| user\_id | 买家 |

**Status Code：**

|  |  |
| --- | --- |
| **Code** | **描述** |
| 200 | 找到订单并获取其详细信息 |
| 518 | 订单不存在 |
| 511 | 买家不存在 |

1. **超时自动取消订单**

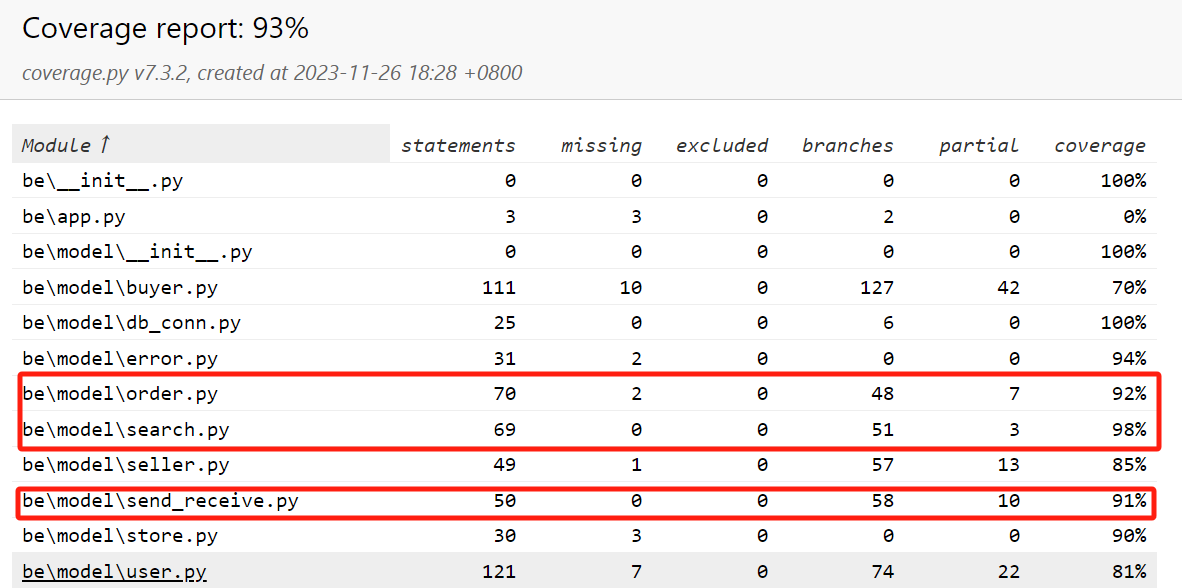
超时自动取消订单在这里我们设置成定时任务，自动执行，所以只需要在一定睡眠时间（超过超时自动取消的临界值）后检查订单是否存在（是否被自动删除）即可。

## 4.6 覆盖率与测试结果

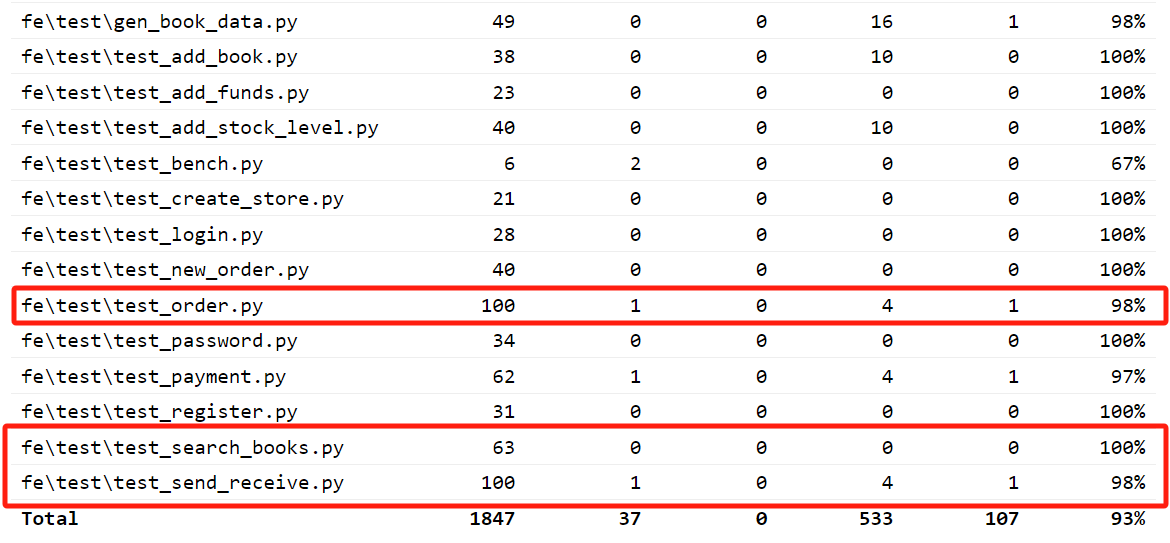
## 4.6.1 覆盖率

经过pytest检验后，所有代码的整体覆盖率为93%。下面两张图分别是be模块和fe模块的覆盖率，红色圈出的为后40%的功能。

Be：

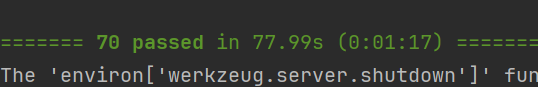


Fe:



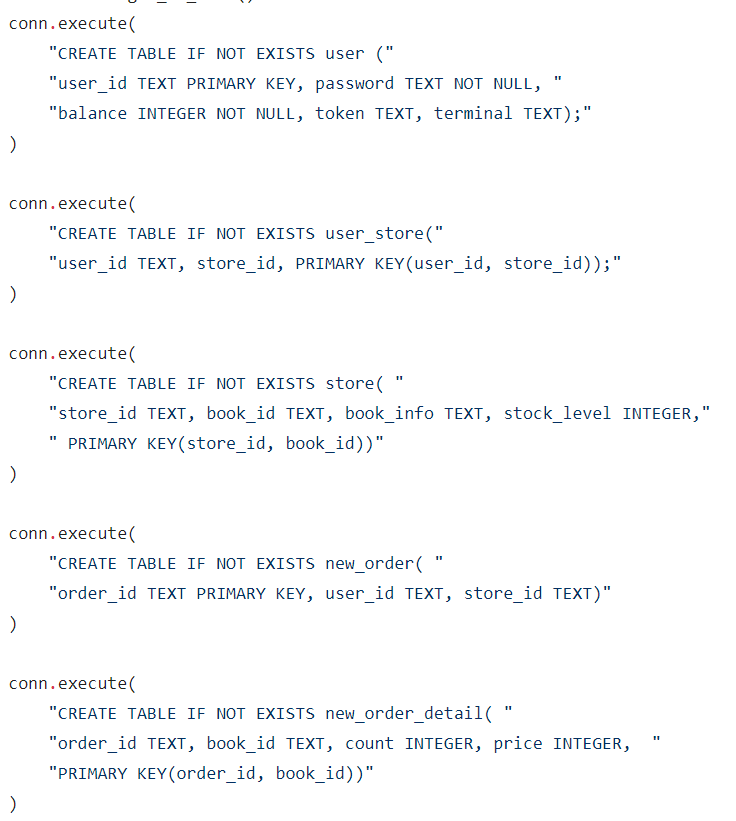
## 4.6.2 测试结果

原有pytest与后40%内容的原创pytest的检验均已通过。

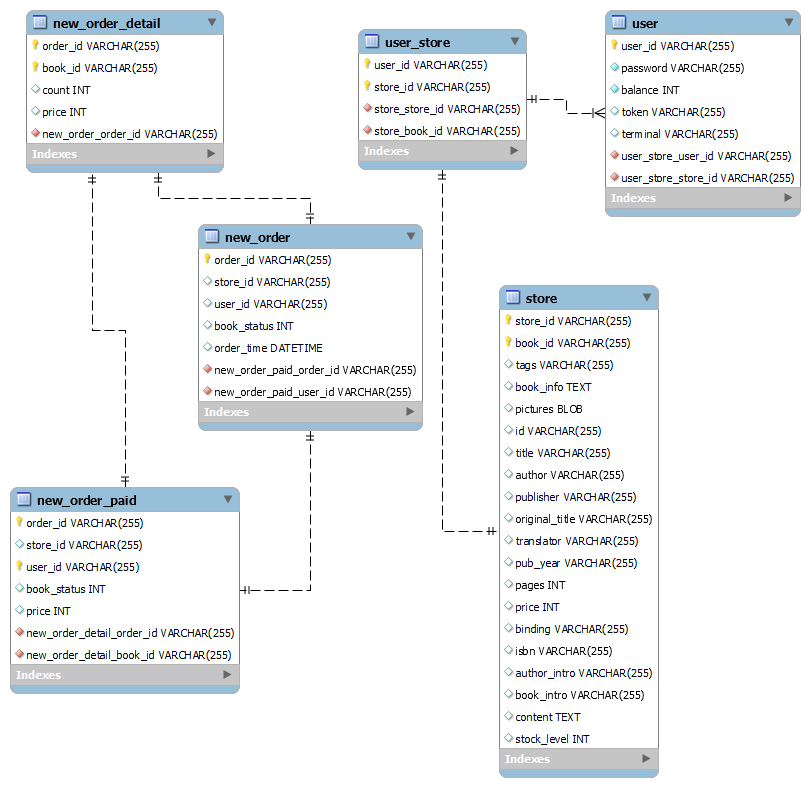


# 项目说明

1. 本次实验的后端代码已经在原有基础上进行了较大的改动，但是整体仓库的逻辑设计没有更改，只是增加了一些接口和后端代码，并不是直接运用了原有的后端设计。
2. 关于数据库schema的更改：本次实验的原有table如下：



目前的schema为：



主要更改为：

* 增加new\_order\_paid表格，用于记录已经付款过的订单，将已付款和未付款的订单分割开来。
* new\_order表格的存储结构发生改变，增加了下单时间、下单的书籍状态。不能将new\_order与new\_order合并为一个表格的原因是每一个order中可能存在n个不同的书籍，这些不同的书籍又对应不同的价格和量，所以最好的方法是另建新的表格存储new\_order的详细信息，否则会造成键值错误。
* store表格的存储结构发生改变，store中存储的书籍信息会详细展开，可以清楚的看到store中所有的书籍信息。这样可以方便进行搜索。
* user\_store表格不与user合并的原因是每一个user可能拥有多个store，会造成键值错误。

3. 将数据库从文档型数据库（如MongoDB）转换为关系型数据库（如MySQL）涉及一些重要的改动。

1. 数据模型的改变：

MongoDB是一个文档型数据库，使用 BSON（Binary JSON）格式来存储数据。在转换

关系型数据库时，需要重新设计数据模型，将文档结构拆分为适当的表格和关系。（见上文）

1. 数据关系的建立：

在关系型数据库中，数据之间的关系通常通过外键来建立。数据库的范式化（Normalization）也是关系型数据库的一个特点，需要对数据表进行拆分以减少冗余。（见上文）

1. 性能考虑：

关系型数据库在处理复杂查询和大型数据集时可能更有效率。书店需要频繁地执行复杂查询，关系型数据库可能会提供更好的性能。

4.本次实验利用了Github进行版本管理控制，地址为https://github.com/TheadoraTang/Bookstore-SQL