

华东师范大学数据科学与工程学院实验报告

课程名称：当代数据管理系统	年级：2018	上机实践成绩：
指导教师：周烜	姓名：俞泽恺、陈郅睿、曾国龙	学号：10185501408、10185501410、10174507108
上机实践名称：在线书店 bookstore		上机实践日期：2021/1
上机实践编号：	组号：	上机实践时间：

一、实验目的

实现一个提供网上购书功能的网站后端。

二、实验任务

网站支持书商在上面开商店，购买者可能通过网站购买。

买家和卖家都可以注册自己的账号。

一个卖家可以开一个或多个网上商店， 买家可以为自己的账户充值，在任意商店购买图书。

支持下单->付款->发货->收货，流程。

其中包括：

1)用户权限接口，如注册、登录、登出、注销

2)买家用户接口，如充值、下单、付款

3)卖家用户接口，如创建店铺、填加书籍信息及描述、增加库存

4)发货 -> 收货

5)搜索图书。用户可以通过关键字搜索，参数化的搜索方式； 如搜索范围包括，题目，标签，目录，内容；全站搜索或是当前店铺搜索

6)订单状态，订单查询和取消定单

用户可以查自己的历史订单，用户也可以取消订单。

任务分工：

基础功能：陈郅睿、曾国龙、俞泽恺

搜索功能、报告、演讲：俞泽恺

收货发货、取消订单：陈郅睿

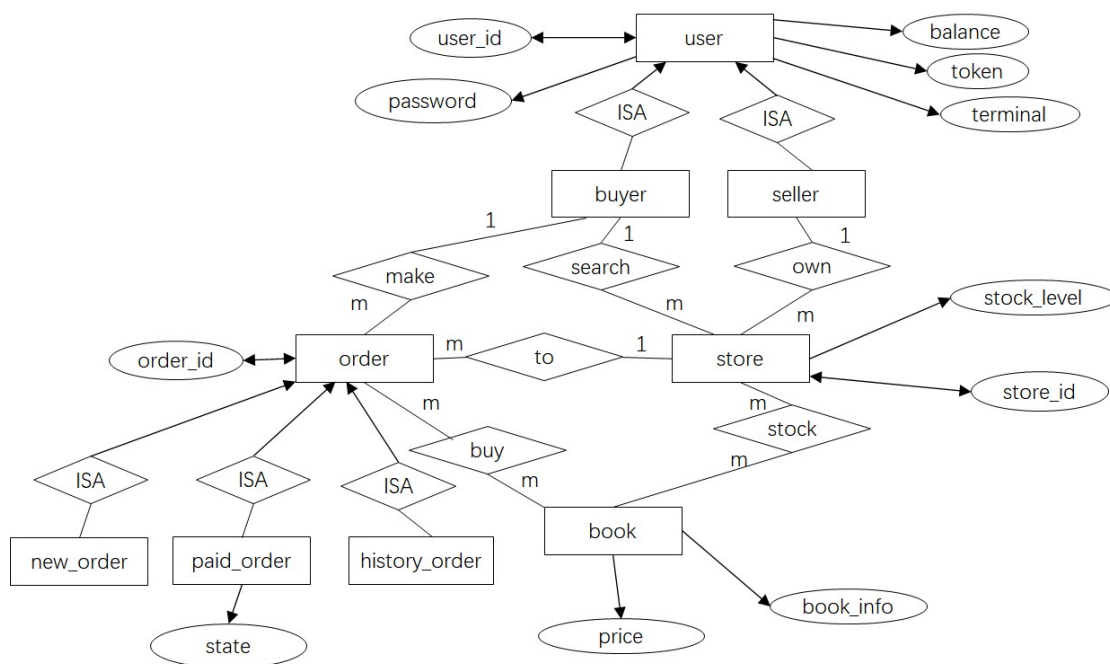
测试：曾国龙

三、使用环境

Python3

四、实验过程

ER 图：



ER 图的实体类主要有：用户 订单 商店 书本

用户包括买家和卖家，拥有自己的 id password 等信息

卖家拥有自己的商店，有自己的库存

商店中存放书本，书本有自己的 price 等信息

买家可以下订单，为了实现所需功能，可以将订单分为三类（新订单、已付款订单和历史订单），其中，新订单是用户下单后生成的，已付款订单是付款后生成的，记录商品的状态（待发货和发货中），而历史订单是由新订单或者已付款订单生成的

对应关系：一个用户可以下多个订单，一个订单只能由一个用户生成；一个订单只能访问一家商店，一家商店可以有多个订单；一个订单可以买多本书，一本书也可以出现在不同订单中（同一本书可以有很多库存）；一家商店可以存多本书，一本书也可以出现在不同商店；一个商家可以拥有多个商店，一个商店只能被一个人拥有

数据库设计（共 9 张表）：

users

user_id [PK] text	password text	balance integer	token text	terminal text
-----------------------------	-------------------------	---------------------------	----------------------	-------------------------

user_store

user_id [PK] text	store_id [PK] text
-----------------------------	------------------------------

store

store_id [PK] text	book_id [PK] text	book_info text	stock_level integer
------------------------------	-----------------------------	--------------------------	-------------------------------

new_order/history_order (schema 相同)

order_id [PK] text	user_id text	store_id text
------------------------------	------------------------	-------------------------

paid_order

order_id [PK] text	user_id text	store_id text	state text
------------------------------	------------------------	-------------------------	----------------------

new_order_detail/paid_order_detail/history_order_detail (schema 相同)

order_id [PK] text	book_id [PK] text	count integer	price integer
------------------------------	-----------------------------	-------------------------	-------------------------

介绍分表原因：

买家对于看中的书可以进行下单，订单存在 new_order/new_order_detail 中，此时还未进行付款

付款后，订单信息会转存到 paid_order/paid_order_detail 中（new_order 中信息删除），此时多了一个 state 位，用于记录发货信息，初始化为“未发货”。商家在买家付款后便可对其进行发货处理

发货后，买家收到货物，订单信息会转存到 history_order/history_order_detail（paid_order 中信息删除）中，以供用户查询历史订单

索引：对每个表中搜索时最常用到的键（id 等）建立了 B 树索引。

基础功能部分：

注册功能：

传入用户名和密码，插入数据库。若用户名已存在则无法插入。

```
def register(self, user_id: str, password: str):
    try:
        terminal = "terminal_{}".format(str(time.time()))
        token = jwt_encode(user_id, terminal)
        self.conn.execute(
            "INSERT into users(user_id, password, balance, token, terminal) "
            "VALUES (%s, %s, %s, %s, %s);",
            (user_id, password, 0, token, terminal), )
        #self.conn.commit()
    except pg.Error:
        return error.error_exist_user_id(user_id)
    return 200, "ok"
```

检查用户 token：

根据用户名取出并比对。

```
def check_token(self, user_id: str, token: str) -> (int, str):
    cursor = self.conn.execute("SELECT token from users where user_id=%s", (user_id,))
    row = self.conn.fetchone()
    if row is None:
        return error.error_authorization_fail()
    db_token = row['token']
    if not self.__check_token(user_id, db_token, token):
        return error.error_authorization_fail()
    return 200, "ok"
```

检查密码：

根据用户名取出并比对。

```
def check_password(self, user_id: str, password: str) -> (int, str):
    self.conn.execute("SELECT password from users where user_id=%s", (user_id,))
    row = self.conn.fetchone()
    if row is None:
        return error.error_authorization_fail()

    if password != row['password']:
        return error.error_authorization_fail()

    return 200, "ok"
```

登陆功能：

检查用户名和密码是否匹配，更新用户 token，返回 token。

```

def login(self, user_id: str, password: str, terminal: str) -> (int, str, str):
    token = ""
    try:
        code, message = self.check_password(user_id, password)
        if code != 200:
            return code, message, ""

        token = jwt_encode(user_id, terminal)
        cursor = self.conn.execute(
            "UPDATE users set token= %s , terminal = %s where user_id = %s",
            (token, terminal, user_id), )
        """if cursor.rowcount == 0:
            return error.error_authorization_fail() + ("", )"""
        # self.conn.commit()
    except pg.Error as e:
        print(e)
        return 528, "{}".format(str(e)), ""
    except BaseException as e:
        print(e)
        return 530, "{}".format(str(e)), ""
    return 200, "ok", token

```

登出功能:

检查用户名和 token 是否匹配, 更新 token。

```

def logout(self, user_id: str, token: str) -> bool:
    try:
        code, message = self.check_token(user_id, token)
        if code != 200:
            return code, message

        terminal = "terminal_{}".format(str(time.time()))
        dummy_token = jwt_encode(user_id, terminal)

        cursor = self.conn.execute(
            "UPDATE users SET token = %s, terminal = %s WHERE user_id=%s",
            (dummy_token, terminal, user_id), )
        """if cursor.rowcount == 0:
            return error.error_authorization_fail()

        self.conn.commit()"""
    except pg.Error as e:
        return 528, "{}".format(str(e))
    except BaseException as e:
        return 530, "{}".format(str(e))
    return 200, "ok"

```

注销用户功能:

检查用户名和密码是否匹配。删除用户信息。

```
def unregister(self, user_id: str, password: str) -> (int, str):
    try:
        code, message = self.check_password(user_id, password)
        if code != 200:
            return code, message

        cursor = self.conn.execute("DELETE from users where user_id=%s", (user_id,))
        """if self.conn.rowcount == 1:
            self.conn.commit()
        else:
            return error.error_authorization_fail()"""
    except pg.Error as e:
        return 528, "{}".format(str(e))
    except BaseException as e:
        return 530, "{}".format(str(e))
    return 200, "ok"
```

修改密码：

检查用户名和密码是否匹配。更新用户信息。

```
def change_password(self, user_id: str, old_password: str, new_password: str) -> bool:
    try:
        code, message = self.check_password(user_id, old_password)
        if code != 200:
            return code, message

        terminal = "terminal_{}".format(str(time.time()))
        token = jwt_encode(user_id, terminal)
        cursor = self.conn.execute(
            "UPDATE users set password = %s, token= %s , terminal = %s where user_id = %s",
            (new_password, token, terminal, user_id), )
        """if cursor.rowcount == 0:
            return error.error_authorization_fail()

        self.conn.commit()"""
    except pg.Error as e:
        return 528, "{}".format(str(e))
    except BaseException as e:
        return 530, "{}".format(str(e))
    return 200, "ok"
```

创建新订单：

检查参数是否合法，修改库存，创建订单。

```
def new_order(self, user_id: str, store_id: str, id_and_count: [(str, int)]) -> (int, str, str):
    order_id = ""
    try:
        #检查 id 是否存在
```

```
if not self.user_id_exist(user_id):
    return error.error_non_exist_user_id(user_id) + (order_id, )
if not self.store_id_exist(store_id):
    return error.error_non_exist_store_id(store_id) + (order_id, )
uid = "{}_{}_{}".format(user_id, store_id, str(uuid.uuid1()))

for book_id, count in id_and_count:
    cursor = self.conn.execute(
        "SELECT book_id, stock_level, book_info FROM store "
        "WHERE store_id = %s AND book_id = %s;",
        (store_id, book_id))
    row = self.conn.fetchone()
    #检查商店和书是否存在
    if row is None:
        return error.error_non_exist_book_id(book_id) + (order_id, )

    stock_level = row['stock_level']
    book_info = row['book_info']
    book_info_json = json.loads(book_info)
    price = book_info_json.get("price")
    #检查库存是否充足
    if stock_level < count:
        return error.error_stock_level_low(book_id) + (order_id, )
    #更改库存数量
    cursor = self.conn.execute(
        "UPDATE store set stock_level = stock_level - %s "
        "WHERE store_id = %s and book_id = %s and stock_level >= %s; ",
        (count, store_id, book_id, count))
    """if cursor.rowcount == 0:
        return error.error_stock_level_low(book_id) + (order_id, )"""

    self.conn.execute(
        "INSERT INTO new_order_detail(order_id, book_id, count, price) "
        "VALUES(%s, %s, %s, %s);",
        (uid, book_id, count, price))

    self.conn.execute(
        "INSERT INTO new_order(order_id, store_id, user_id) "
        "VALUES(%s, %s, %s);",
        (uid, store_id, user_id))
    #self.conn.commit()
    order_id = uid
except pg.Error as e:
    logging.info("528, {}".format(str(e)))
    return 528, "{}".format(str(e)), ""
except BaseException as e:
    logging.info("530, {}".format(str(e)))
```

```
return 530, "{}".format(str(e)), ""  
  
return 200, "ok", order_id
```

支付某个订单的钱款：

检查参数是否合法，修改买方卖方信息，修改订单状态信息。

```
def payment(self, user_id: str, password: str, order_id: str) -> (int, str):  
    conn = self.conn  
    try:  
        #检查订单是否存在  
        cursor = conn.execute("SELECT order_id, user_id, store_id FROM new_order WHERE  
order_id = %s", (order_id,))  
        row = conn.fetchone()  
        if row is None:  
            return error.error_invalid_order_id(order_id)  
  
        order_id = row['order_id']  
        buyer_id = row['user_id']  
        store_id = row['store_id']  
        #检查买家信息  
        if buyer_id != user_id:  
            return error.error_authorization_fail()  
        #核对账户密码  
        cursor = conn.execute("SELECT balance, password FROM users WHERE user_id = %s",  
(buyer_id,))  
        row = conn.fetchone()  
        if row is None:  
            return error.error_non_exist_user_id(buyer_id)  
        balance = row['balance']  
        if password != row['password']:  
            return error.error_authorization_fail()  
  
        #核对店铺信息  
        cursor = conn.execute("SELECT store_id, user_id FROM user_store WHERE store_id  
= %s", (store_id,))  
        row = conn.fetchone()  
        if row is None:  
            return error.error_non_exist_store_id(store_id)  
  
        seller_id = row['user_id']  
  
        if not self.user_id_exist(seller_id):  
            return error.error_non_exist_user_id(seller_id)  
  
        conn.execute("SELECT book_id, count, price FROM new_order_detail WHERE order_id  
= %s", (order_id,))  
        cursor=conn.fetchall()
```



```
total_price = 0
bookids=[]
counts=[]
prices=[]
for row in cursor:
    bookids.append(row['book_id'])
    counts.append(row['count'])
    prices.append(row['price'])
    count = row['count']
    price = row['price']
    total_price = total_price + price * count

if balance < total_price:
    return error.error_not_sufficient_funds(order_id)
#买家扣除相应钱款
cursor = conn.execute("UPDATE users set balance = balance - %s"
                        "WHERE user_id = %s AND balance >= %s",
                        (total_price, buyer_id, total_price))

"""if cursor.rowcount == 0:
    return error.error_not_sufficient_funds(order_id)"""
#卖家增加相应钱款
cursor = conn.execute("UPDATE users set balance = balance + %s"
                        "WHERE user_id = %s",
                        (total_price, seller_id))

"""if cursor.rowcount == 0:
    return error.error_non_exist_user_id(buyer_id)"""
#删除订单信息
cursor = conn.execute("DELETE FROM new_order WHERE order_id = %s", (order_id, ))
"""if cursor.rowcount == 0:
    return error.error_invalid_order_id(order_id)"""

cursor = conn.execute("DELETE FROM new_order_detail where order_id = %s",
(order_id, ))
"""if cursor.rowcount == 0:
    return error.error_invalid_order_id(order_id)"""

#增加状态信息
cursor = conn.execute(
    "INSERT INTO paid_order(order_id,user_id,store_id,state) "
    "VALUES(%s, %s, %s, '待发货');",
    (order_id, user_id, store_id))

for i in range(len(bookids)):
    cursor=conn.execute(
        "INSERT INTO paid_order_detail(order_id,book_id,count,price) "
        "VALUES(%s, %s, %s, %s);",
```

```

        (order_id,bookids[i],counts[i],prices[i]))
    #conn.commit()

except pg.Error as e:
    return 528, "{}".format(str(e))

except BaseException as e:
    return 530, "{}".format(str(e))

return 200, "ok"

```

给买方增加资金：

```

def add_funds(self, user_id, password, add_value) -> (int, str):
    try:

        cursor = self.conn.execute("SELECT password  from users where user_id=%s",
(user_id,))
        row = self.conn.fetchone()
        if row is None:
            return error.error_authorization_fail()
        #核对密码信息
        if row['password'] != password:
            return error.error_authorization_fail()
        #充值
        cursor = self.conn.execute(
            "UPDATE users SET balance = balance + %s WHERE user_id = %s",
            (add_value, user_id))
        """if cursor.rowcount == 0:
            return error.error_non_exist_user_id(user_id)

        self.conn.commit()"""
    except pg.Error as e:
        return 528, "{}".format(str(e))
    except BaseException as e:
        return 530, "{}".format(str(e))

    return 200, "ok"

```

卖方添加书：

#添加书的信息。从 user_store\store\users 中查询 id 是否存在。书的信息不能已经存在。
 #向 store 中插入书的信息，包括卖家 id、书 id、书信息（结构体）、库存数量。

```

def add_book(self, user_id: str, store_id: str, book_id: str, book_json_str: str, stock_level: int):
    try:
        if not self.user_id_exist(user_id):
            return error.error_non_exist_user_id(user_id)
        if not self.store_id_exist(store_id):

```

```

        return error.error_non_exist_store_id(store_id)
    if self.book_id_exist(store_id, book_id):
        return error.error_exist_book_id(book_id)

    self.conn.execute("INSERT into store(store_id, book_id, book_info, stock_level)"
                      "VALUES (%s, %s, %s, %s)", (store_id, book_id, book_json_str,
stock_level))
    #self.conn.commit()
    except pg.Error as e:
        return 528, "{}".format(str(e))
    except BaseException as e:
        return 530, "{}".format(str(e))
    return 200, "ok"

```

增加库存:

#增加书的库存水平。用户、店铺、书必须存在。将指定店铺中指定书籍的库存容量增加 add_stock_level 个。

```

def add_stock_level(self, user_id: str, store_id: str, book_id: str, add_stock_level: int):
    try:
        if not self.user_id_exist(user_id):
            return error.error_non_exist_user_id(user_id)
        if not self.store_id_exist(store_id):
            return error.error_non_exist_store_id(store_id)
        if not self.book_id_exist(store_id, book_id):
            return error.error_non_exist_book_id(book_id)

        self.conn.execute("UPDATE store SET stock_level = stock_level + %s "
                          "WHERE store_id = %s AND book_id = %s", (add_stock_level,
store_id, book_id))
        #self.conn.commit()
    except pg.Error as e:
        return 528, "{}".format(str(e))
    except BaseException as e:
        return 530, "{}".format(str(e))
    return 200, "ok"

```

创建店铺:

#创建店铺。用户 id 不能不存在，店铺 id 不能已经存在。向 user_store 中插入（店铺，用户）对

```

def create_store(self, user_id: str, store_id: str) -> (int, str):
    try:
        if not self.user_id_exist(user_id):
            return error.error_non_exist_user_id(user_id)
        if self.store_id_exist(store_id):
            return error.error_exist_store_id(store_id)
        self.conn.execute("INSERT into user_store(store_id, user_id)"
                          "VALUES (%s, %s)", (store_id, user_id))

```

```

        #self.conn.commit()
    except pg.Error as e:
        return 528, "{}".format(str(e))
    except BaseException as e:
        return 530, "{}".format(str(e))
    return 200, "ok"

```

收货发货功能实现：

为卖家提供一个发货接口，对于已经付款的订单（存在 paid_order 中），确认信息之后进行发货，改变 paid_id 状态位（“待发货”——“发货中”）

函数实现：

```

def send_item(self, user_id: str, order_id: str) -> (int, str):
    try:
        if not self.user_id_exist(user_id):
            return error.error_non_exist_user_id(user_id)
        #检查发货人信息
        cursor = self.conn.execute("SELECT store_id,state FROM paid_order WHERE order_id = %s", (order_id,))
        row = self.conn.fetchone()
        if row is None:
            return error.error_invalid_order_id(order_id)
        store_id = row['store_id']
        state=row['state']
        cursor = self.conn.execute("SELECT user_id FROM user_store WHERE store_id = %s", (store_id,))
        row = self.conn.fetchone()
        seller_id = row['user_id']
        if seller_id != user_id:
            return error.error_authorization_fail()
        if state=='发货中':
            return error.error_repeatsend()
        self.conn.execute("UPDATE paid_order SET state = %s "
                           "WHERE order_id = %s ", ('发货中',order_id))

    except pg.Error as e:
        return 528, "{}".format(str(e))
    except BaseException as e:
        return 530, "{}".format(str(e))
    return 200, "ok"

```

检查完用户和订单信息之后，还要检查 state 位，如果该货物已经在发货中则报错

为买家提供一个收货接口，收货之后，该订单变为历史订单，存入 history_order/history_order_detail（paid_order 中信息删除）中
函数实现：

```
def receive_item(self, user_id: str, order_id: str) -> (int, str):
    try:
        if not self.user_id_exist(user_id):
            return error.error_non_exist_user_id(user_id)

        # 检查收货人信息
        cursor = self.conn.execute("SELECT user_id,store_id,state FROM paid_order WHERE
order_id = %s", (order_id,))
        row = self.conn.fetchone()
        if row is None:
            return error.error_invalid_order_id(order_id)

        buyer_id = row['user_id']
        state = row['state']
        store_id = row['store_id']
        if buyer_id != user_id:
            return error.error_authorization_fail()

        if state == '待发货':
            return error.error_receive_fail()

        # 收货完成，该订单记录为历史订单
        cursor = self.conn.execute(
            "INSERT INTO history_order(order_id,user_id,store_id) "
            "VALUES(%s, %s, %s);",
            (order_id, user_id, store_id))

        self.conn.execute("SELECT book_id, count, price FROM paid_order_detail WHERE
order_id = %s;", (order_id,))
        cursor = self.conn.fetchall()
        total_price = 0
        bookids = []
        counts = []
        prices = []
        for row in cursor:
            bookids.append(row['book_id'])
            counts.append(row['count'])
            prices.append(row['price'])

        for i in range(len(bookids)):
            cursor = self.conn.execute(
```

```

        "INSERT INTO history_order_detail(order_id,book_id,count,price) "
        "VALUES(%s, %s, %s, %s);",
        (order_id, bookids[i], counts[i], prices[i]))

    # 删除订单信息
    cursor = self.conn.execute("DELETE FROM paid_order WHERE order_id = %s", (order_id,))
    cursor = self.conn.execute("DELETE FROM paid_order_detail where order_id = %s",
    (order_id,))

    # self.conn.commit()
except pg.Error as e:
    return 528, "{}".format(str(e))
except BaseException as e:
    return 530, "{}".format(str(e))
return 200, "ok"

```

订单查询实现：

由于订单可能出现在三个表中（new_order、paid_order、history_order）所以首先执行三个sql 语句查询订单当前进度（位于哪个表中），并从对应的 xxx_order_detail 中提取订单详细信息，最后加上一个状态位返回给用户（未付款、待发货、发货中、已完成四种）

函数实现：

查询历史订单

```

def query(self, user_id: str, order_id: str) -> (int, str, str):
    conn = self.conn
    try:
        # 检查订单进度（未付款、已付款、已交付）
        cursor = conn.execute("SELECT user_id, store_id FROM new_order WHERE order_id
= %s",
                                (order_id,))
        row_new = conn.fetchone()
        cursor = conn.execute("SELECT user_id, store_id,state FROM paid_order WHERE order_id
= %s",
                                (order_id,))
        row_paid = conn.fetchone()
        cursor = conn.execute("SELECT user_id, store_id FROM history_order WHERE order_id
= %s",
                                (order_id,))
        row_history = conn.fetchone()

        if row_new is None and row_paid is None and row_history is None:
            return error.error_invalid_order_id(order_id)
        elif row_new is not None:
            print('new')
            buyer_id = row_new['user_id']
            store_id = row_new['store_id']
            if buyer_id != user_id:

```

```
        return error.error_authorization_fail()

    conn.execute("SELECT book_id, count FROM new_order_detail WHERE order_id
= %s;", (order_id,))
    cursor = conn.fetchall()

    books = []
    for row in cursor:
        dict = {}
        dict["id"] = row['book_id']
        dict["count"] = row['count']
        books.append(dict)
    json_text = {}
    json_text["user_id"] = buyer_id
    json_text["store_id"] = store_id
    json_text["books"] = books
    json_text["order_state"] = "unpaid"
    return 200, "ok", str(json_text)
elif row_paid is not None:
    print('paid')
    buyer_id = row_paid['user_id']
    store_id = row_paid['store_id']
    state = row_paid['state']
    if buyer_id != user_id:
        return error.error_authorization_fail()

    conn.execute("SELECT book_id, count FROM paid_order_detail WHERE order_id
= %s;", (order_id,))
    cursor = conn.fetchall()

    books = []
    for row in cursor:
        dict = {}
        dict["id"] = row['book_id']
        dict["count"] = row['count']
        books.append(dict)
    json_text = {}
    json_text["user_id"] = buyer_id
    json_text["store_id"] = store_id
    json_text["books"] = books
    json_text["order_state"] = state
    return 200, "ok", str(json_text)
elif row_history is not None:
    print('his')
    buyer_id = row_history['user_id']
    store_id = row_history['store_id']
    if buyer_id != user_id:
```

```

        return error.error_authorization_fail()

    conn.execute("SELECT book_id, count FROM history_order_detail WHERE order_id
= %s;", (order_id,))
    cursor = conn.fetchall()

    books = []
    for row in cursor:
        dict = {}
        dict["id"] = row['book_id']
        dict["count"] = row['count']
        print(dict)
        books.append(dict)
    json_text = {}
    json_text["user_id"] = buyer_id
    json_text["store_id"] = store_id
    json_text["books"] = books
    json_text["order_state"] = "finished"
    print(json_text)
    return 200, "ok", str(json_text)

except pg.Error as e:
    return 528, "{}".format(str(e))

except BaseException as e:
    return 530, "{}".format(str(e))

```

取消订单实现：

做出以下规定——只有未付款或者是已付款但未发货的用户才能取消订单，付款后已发货的用户无法取消订单。如果已经付款，则会退回相应钱款数目。取消后的订单存放在 history_order 中供用户查询。

```

def cancel(self, user_id: str, order_id: str) -> (int, str):
    conn = self.conn
    try:

        cursor = conn.execute("SELECT user_id, store_id FROM new_order WHERE order_id
= %s",
                                (order_id,))
        row_new = conn.fetchone()
        cursor = conn.execute("SELECT user_id, store_id, state FROM paid_order WHERE order_id
= %s",
                                (order_id,))
        row_paid = conn.fetchone()
        if row_new is None and row_paid is None:
            return error.error_invalid_order_id(order_id)
        elif row_new is not None:
            buyer_id = row_new['user_id']

```



```
store_id = row_new['store_id']
cursor = conn.execute("SELECT store_id, user_id FROM user_store WHERE store_id
= %s;", (store_id,))
row = conn.fetchone()
seller_id = row['user_id']

if not self.user_id_exist(seller_id):
    return error.error_non_exist_user_id(seller_id)
if buyer_id != user_id:
    return error.error_authorization_fail()

# 订单未付款，接受取消操作，该订单写入历史订单
cursor = self.conn.execute(
    "INSERT INTO history_order(order_id,user_id,store_id) "
    "VALUES(%s, %s, %s);",
    (order_id, user_id, store_id))

self.conn.execute("SELECT book_id, count, price FROM new_order_detail WHERE
order_id = %s;",
                    (order_id,))
cursor = self.conn.fetchall()

bookids = []
counts = []
prices = []

for row in cursor:
    bookids.append(row['book_id'])
    counts.append(row['count'])
    prices.append(row['price'])

for i in range(len(bookids)):
    cursor = self.conn.execute(
        "INSERT INTO history_order_detail(order_id,book_id,count,price) "
        "VALUES(%s, %s, %s, %s);",
        (order_id, bookids[i], counts[i], prices[i]))

# 删除订单信息
cursor = self.conn.execute("DELETE FROM new_order WHERE order_id = %s",
(order_id,))
cursor = self.conn.execute("DELETE FROM new_order_detail where order_id = %s",
(order_id,))

return 200, "ok"
elif row_paid is not None:
```



```

# 卖家减少相应钱款
cursor = conn.execute("UPDATE users set balance = balance - %s"
                        "WHERE user_id = %s",
                        (total_price, seller_id))

# 删除订单信息
cursor = self.conn.execute("DELETE FROM paid_order WHERE order_id = %s",
(order_id,))
cursor = self.conn.execute("DELETE FROM paid_order_detail where order_id = %s",
(order_id,))

return 200, "ok"

except pg.Error as e:
    return 528, "{}".format(str(e))

except BaseException as e:
    return 530, "{}".format(str(e))

```

搜索功能：

参数化的搜索。可搜索的参数和 store 及 book_info 中参数一致，每个检索条件以列表形式传入（也可不传入），返回匹配满足条件的所有结果。对 tags，要求结果满足包含传入参数的所有元素。对其他参数，要求结果满足包含传入参数中的至少一个。

例如，传入参数

target_store_id=['store1','store2'],target_title=['book1','book2'],target_tags=['tags1','tags2','tags3']
,则查询的对象为 store1 和 store2 中书标题为 book1 或 book2、且书的 tags 包含全部三者的
条目。

```

def
search_by_arguments(self,target_store_id:list=None,target_id:list=None,target_title:list=None,tar
get_tags:list=None,

target_author:list=None,target_publisher:list=None,target_original_title:list=None,target_translat
or:list=None,

target_pub_year:list=None,target_pages:list=None,target_price:list=None,target_binding:list=Non
e,

target_isbn:list=None,target_author_intro:list=None,target_book_intro:list=None,target_content:li
st=None,

                        target_stock_level:int=None
):
    if target_store_id == None or target_store_id==[]:#全站搜索
        self.conn.execute("""
            select * from store
            """)

```

```

else:#指定店铺
    if len(target_store_id)==1:
        self.conn.execute("""
            select * from store where store_id=%s
            """, (target_store_id[0],))
    else:
        cond='store_id='+\'\"'+target_store_id[0]+'\'\"
        for x in target_store_id[1:]:
            cond+=' or store_id='+\'\"'+x+'\'\"
        print(cond)
        self.conn.execute("""
            select * from store where {
            """.format(cond,))

ans = self.conn.fetchall()
ret = []
for entry in ans:
    flag = 1
    store_id = entry['store_id']
    stock_level = entry['stock_level']
    book_info = entry['book_info']
    book_info = json.loads(book_info)
    tags = book_info['tags']
    pictures = book_info['pictures']
    id = book_info['id']
    title = book_info['title']
    author = book_info['author']
    publisher = book_info['publisher']
    original_title = book_info['original_title']
    translator = book_info['translator']
    pub_year = book_info['pub_year']
    pages = book_info['pages']
    price = book_info['price']
    binding = book_info['binding']
    isbn = book_info['isbn']
    author_intro = book_info['author_intro']
    book_intro = book_info['book_intro']
    content = book_info['content']
    if target_id!=None and target_id!=[] and not any(item in id for item in target_id):#id 任意
        满足

        continue
    if target_tags != None and target_tags !=[] and not all(item in tags for item in
target_tags):#tag 要全部满足
        flag = 0
        continue
    if target_title!=None and target_title!=[] and not any(item in title for item in target_title):

```

```
        continue
        if target_author!=None and target_author!=[] and not any(i in author for i in
target_author):

            continue
            if target_publisher!=None and target_publisher!=[] and not any(i in publisher for i in
target_publisher):

                continue
                if target_original_title!=None and target_original_title!=[] and not any(i in original_title
for i in target_original_title):

                    continue
                    if target_translator!=None and target_translator!=[] and not any(i in translator for i in
target_translator):

                        continue
                        if target_pub_year!=None and target_pub_year!=[] and not any(i in pub_year for i in
target_pub_year):

                            continue
                            if target_pages!=None and target_pages!=[] and not any(i in pages for i in target_pages):

                                continue
                                if target_price!=None and target_price!=[] and not any(i in price for i in target_price):

                                    continue
                                    if target_binding!=None and target_binding!=[] and not any(i in binding for i in
target_binding):

                                        continue
                                        if target_isbn!=None and target_isbn!=[] and not any(i in isbn for i in target_isbn):

                                            continue
                                            if target_author_intro!=None and target_author_intro!=[] and not any(i in author_intro
for i in target_author_intro):

                                                continue
                                                if target_book_intro!=None and target_book_intro!=[] and not any(i in book_intro for i in
target_book_intro):

                                                    continue
                                                    if target_content!=None and target_content!=[] and not any(i in content for i in
target_content):

                                                        continue
                                                        if target_stock_level!=None and target_stock_level!=[] and target_stock_level >
```

stock_level:

```

        continue
    if flag==1:
        ret.append(dict(entry))

    return 200, ret

```

Pytest:

The screenshot shows the PyCharm IDE interface with the following details:

- Project:** bookstore-master 2
- File:** store.py
- Terminal:** Local
- Test Results:**
 - fe/test/test_new_order.py::TestNewOrder::test_ok PASSED [49%]
 - fe/test/test_new_order.py::TestNewOrder::test_non_exist_user_id PASSED [50%]
 - fe/test/test_new_order.py::TestNewOrder::test_non_exist_store_id PASSED [52%]
 - fe/test/test_password.py::TestPassword::test_ok PASSED [54%]
 - fe/test/test_password.py::TestPassword::test_error_password PASSED [56%]
 - fe/test/test_password.py::TestPassword::test_error_user_id PASSED [58%]
 - fe/test/test_payment.py::TestPayment::test_ok PASSED [60%]
 - fe/test/test_payment.py::TestPayment::test_authorization_error PASSED [62%]
 - fe/test/test_payment.py::TestPayment::test_not_suff_funds PASSED [64%]
 - fe/test/test_payment.py::TestPayment::test_repeat_pay PASSED [66%]
 - fe/test/test_query.py::TestQuery::test_ok PASSED [67%]
 - fe/test/test_query.py::TestQuery::test_error_authorization_fail PASSED [69%]
 - fe/test/test_query.py::TestQuery::test_error_invalid_order_id PASSED [71%]
 - fe/test/test_receive_item.py::TestSendItem::test_receive_item_ok PASSED [73%]
 - fe/test/test_receive_item.py::TestSendItem::test_error_authorization_fail PASSED [75%]
 - fe/test/test_receive_item.py::TestSendItem::test_error_receive_fail PASSED [77%]
 - fe/test/test_receive_item.py::TestSendItem::test_error_repeatreceive PASSED [79%]
 - fe/test/test_register.py::TestRegister::test_register_ok PASSED [81%]
 - fe/test/test_register.py::TestRegister::test_unregister_ok PASSED [83%]
 - fe/test/test_register.py::TestRegister::test_unregister_error_authorization PASSED [84%]
 - fe/test/test_register.py::TestRegister::test_register_error_exist_user_id PASSED [86%]
 - fe/test/test_send_item.py::TestSendItem::test_send_item_ok PASSED [88%]
 - fe/test/test_send_item.py::TestSendItem::test_error_repeat_send PASSED [90%]
 - fe/test/test_send_item.py::TestSendItem::test_error_send_fail PASSED [92%]
 - fe/test/test_send_item.py::TestSendItem::test_error_authorization_fail PASSED [94%]
 - fe/test/test_send_item.py::TestSendItem::test_error_invalid_order_id PASSED [96%]
 - fe/test/test_serch_by_args.py::TestSearchByArgs::test_ok1 PASSED [98%]
 - fe/test/test_serch_by_args.py::TestSearchByArgs::test_ok2 PASSED [100%]
- Summary:** ===== 53 passed in 143.90s (0:02:23) =====

Test Case	Time (ms)	Success	Fail	Error	Latency (ms)
fe/access/book.py	71	1	12	2	96%
fe/access/buyer.py	54	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	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	22	2	98%
fe/conf.py	11	0	0	0	100%
fe/conftest.py	17	0	0	0	100%
fe/test/__init__.py	0	0	0	0	100%
fe/test/gen_book_data.py	48	1	16	1	97%
fe/test/test_add_book.py	36	0	10	0	100%
fe/test/test_add_funds.py	23	0	0	0	100%
fe/test/test_add_stock_level.py	39	0	10	0	100%
fe/test/test_bench.py	6	2	0	0	67%
fe/test/test_cancel.py	74	1	4	1	97%
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_query.py	95	1	4	1	98%
fe/test/test_receive_item.py	80	1	4	1	98%
fe/test/test_register.py	31	0	0	0	100%
fe/test/test_send_item.py	89	1	4	1	98%
fe/test/test_serch_by_args.py	123	0	6	0	100%
TOTAL	2118	111	369	69	92%

覆盖率 92%

性能测试 吞吐量、延时等:

Thread ID	Timestamp	Latency (ms)
INFO:root:TPS_C=53436, NO=0K:465465 Thread_num:969	TOTAL:465465	LATENCY:0.011964194470601932
INFO:root:TPS_C=53490, NO=0K:466435 Thread_num:970	TOTAL:466435	LATENCY:0.011964366460718774
INFO:root:TPS_C=53545, NO=0K:467406 Thread_num:971	TOTAL:467406	LATENCY:0.01196453520545338
INFO:root:TPS_C=53600, NO=0K:468378 Thread_num:972	TOTAL:468378	LATENCY:0.011964696314179828
INFO:root:TPS_C=53654, NO=0K:469351 Thread_num:973	TOTAL:469351	LATENCY:0.011964855722771868
INFO:root:TPS_C=53709, NO=0K:470325 Thread_num:974	TOTAL:470325	LATENCY:0.01196500945381461
INFO:root:TPS_C=53763, NO=0K:471300 Thread_num:975	TOTAL:471300	LATENCY:0.011965147895517558
INFO:root:TPS_C=53818, NO=0K:472276 Thread_num:976	TOTAL:472276	LATENCY:0.011965272131159152
INFO:root:TPS_C=53873, NO=0K:473253 Thread_num:977	TOTAL:473253	LATENCY:0.011965387788490669
INFO:root:TPS_C=53928, NO=0K:474231 Thread_num:978	TOTAL:474231	LATENCY:0.011965514302233863
INFO:root:TPS_C=53982, NO=0K:475210 Thread_num:979	TOTAL:475210	LATENCY:0.011965648357481023
INFO:root:TPS_C=54037, NO=0K:476190 Thread_num:980	TOTAL:476190	LATENCY:0.011965785935483888
INFO:root:TPS_C=54092, NO=0K:477171 Thread_num:981	TOTAL:477171	LATENCY:0.011965909114704068
INFO:root:TPS_C=54146, NO=0K:478153 Thread_num:982	TOTAL:478153	LATENCY:0.011966037730000005
INFO:root:TPS_C=54201, NO=0K:479136 Thread_num:983	TOTAL:479136	LATENCY:0.011966182359142733
INFO:root:TPS_C=54256, NO=0K:480120 Thread_num:984	TOTAL:480120	LATENCY:0.01196635175394851
INFO:root:TPS_C=54310, NO=0K:481105 Thread_num:985	TOTAL:481105	LATENCY:0.011966506550832161
INFO:root:TPS_C=54365, NO=0K:482091 Thread_num:986	TOTAL:482091	LATENCY:0.011966659663077551
INFO:root:TPS_C=54419, NO=0K:483078 Thread_num:987	TOTAL:483078	LATENCY:0.011966804169880866
INFO:root:TPS_C=54474, NO=0K:484066 Thread_num:988	TOTAL:484066	LATENCY:0.01196695393011352
INFO:root:TPS_C=54528, NO=0K:485055 Thread_num:989	TOTAL:485055	LATENCY:0.011967261837251105
INFO:root:TPS_C=54583, NO=0K:486045 Thread_num:990	TOTAL:486045	LATENCY:0.0119674237322676
INFO:root:TPS_C=54637, NO=0K:487036 Thread_num:991	TOTAL:487036	LATENCY:0.011967673024085122
INFO:root:TPS_C=54692, NO=0K:488028 Thread_num:992	TOTAL:488028	LATENCY:0.011967861401365174
INFO:root:TPS_C=54746, NO=0K:489021 Thread_num:993	TOTAL:489021	LATENCY:0.011968053267422616
INFO:root:TPS_C=54801, NO=0K:490015 Thread_num:994	TOTAL:490015	LATENCY:0.01196824496334142
INFO:root:TPS_C=54855, NO=0K:491010 Thread_num:995	TOTAL:491010	LATENCY:0.01196843979909841
INFO:root:TPS_C=54910, NO=0K:492006 Thread_num:996	TOTAL:492006	LATENCY:0.01196863377843873
INFO:root:TPS_C=54965, NO=0K:493003 Thread_num:997	TOTAL:493003	LATENCY:0.011968813546759311
INFO:root:TPS_C=55019, NO=0K:494001 Thread_num:998	TOTAL:494001	LATENCY:0.01196902304877034
INFO:root:TPS_C=55074, NO=0K:495000 Thread_num:999	TOTAL:495000	LATENCY:0.011969249016829211
INFO:root:TPS_C=55128, NO=0K:496000 Thread_num:1000	TOTAL:496000	LATENCY:0.011969458729028703

其他:

创建了 github 组织并提交了项目代码

The screenshot shows a GitHub repository interface. At the top, the repository name is 'DBMS2020YZKCZRZGL / dbms2020' with a 'Private' label. Below this is a navigation bar with links for Code, Issues, Pull requests, Actions, Projects, Security, Insights, and Settings. The main content area shows the repository's file structure. It includes a header for the 'master' branch with 1 branch and 0 tags. Below this is a table listing the files and folders in the repository, all committed 24 seconds ago by user 'YuZekai'.

File/Folder	Commit Hash	Commit Time	Commit Count
be	dbms2020 YZKCZRZGL	24 seconds ago	1
doc	dbms2020 YZKCZRZGL	24 seconds ago	1
fe	dbms2020 YZKCZRZGL	24 seconds ago	1
script	dbms2020 YZKCZRZGL	24 seconds ago	1
.DS_Store	dbms2020 YZKCZRZGL	24 seconds ago	1
.gitignore	dbms2020 YZKCZRZGL	24 seconds ago	1
.pre-commit-config.yaml	dbms2020 YZKCZRZGL	24 seconds ago	1
.travis.yml	dbms2020 YZKCZRZGL	24 seconds ago	1
1.txt	dbms2020 YZKCZRZGL	24 seconds ago	1
README.md	dbms2020 YZKCZRZGL	24 seconds ago	1
README2.md	dbms2020 YZKCZRZGL	24 seconds ago	1
requirements.txt	dbms2020 YZKCZRZGL	24 seconds ago	1
setup.py	dbms2020 YZKCZRZGL	24 seconds ago	1

五、总结

本次项目使我们更好地了解了关系型数据库的设计和使用，也进一步熟悉了 flask 架构和 pytest 代码测试等。由于时间和水平有限，项目还有很多可以改进的空间，例如使用 ORM、优化表的设计、实现更多功能等等，如有机会将继续改进。