

华东师范大学数据科学与工程学院上机实践报告

课程名称：当代数据管理系统 年级：21 级 上机实践名称：网上书店
指导教师：周烜 姓名：姚修齐
上机实践编号：No.1 学号：10215501419 上机截止日期：12.23

1 实验目的

- 1) 实现一个提供网上购书功能的网站后端。
- 2) 网站支持书商在上面开商店，购买者可以通过网站购买。
- 3) 买家和卖家都可以注册自己的账号。
- 4) 一个卖家可以开一个或多个网上商店。
- 5) 买家可以为自己的账户充值，在任意商店购买图书。
- 6) 支持“下单→付款→发货→收货”的购买全流程。

2 实验任务

2.1 实现对应接口的功能，并通过对应的功能测试，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。

2.2 为项目添加其它功能，包括：

- 1) 实现后续的流程，例如发货与收货；
- 2) 图书搜索功能，例如关键字搜索、参数化的搜索方式、范围搜索、搜索结果分页、全文索引优化查找等；
- 3) 订单状态，订单查询和取消定单，定单可由买家主动取消或超时仍未付款自动取消。

3 实验环境

本次实验基于 Python 3.7 环境，且安装了 Flask (2.0.0), Werkzeug (2.0.0), simplejson, uuid, pymysql, datetime, json, logging, sqlite3, schedule, threading, os, PyJWT, random, base64 等模块以保证实验的正常进行。

本次实验在 **Github** 的个人账号上建立了本项目的 repository 以存储项目代码，并采用多分支存储实现**版本控制**，确保代码的维护、回退和查看正常进行。

本次实验中，我们使用 **Git Bash**, **Github Desktop** 和 **Visual Studio Code** 同步本地内容与线上仓库。代码测试过程中，我们使用 **Git Bash** 作为测试的终端。

4 实验过程

4.1 文档数据库结构

4.1.1 需求分析

本次实验中，我们需要为一个线上书店设计后端架构。该网站应实现以下功能：

允许任何人于本网站注册账号，该账号可以同时以买家和卖家的身份活动；

作为卖家时，用户可以开设商店，个数不限；

作为买家时，用户可以为自己的账户充值，并使用余额在任意商店购买图书；

网站功能应覆盖从下单到付款再到发货和收货的购买全流程。

用户端，我们允许用户注册账号并登录网站。登录后，用户可以浏览各家店铺。浏览结束后，用户可以登出账号。若用户不希望继续使用本平台产品，我们允许用户注销其账号。

卖家端，用户可以创建和管理自己的店铺，包括增加店铺中挂出的书籍，补充书籍库存等。在收到新订单后，卖家需要确认订单并发货。

买家端，用户可以通过搜索功能检索自己想要的书籍，并购买自己看中的书籍。用户也可以为自己的账户充值，所有购买行为都依赖于账户余额。一笔订单可以以“完成”或“取消”两种方式结束。买家确认收货后，订单视为完成。除买家自愿取消外，购书订单下单后一定时间内买家未支付购书款，订单也会取消。

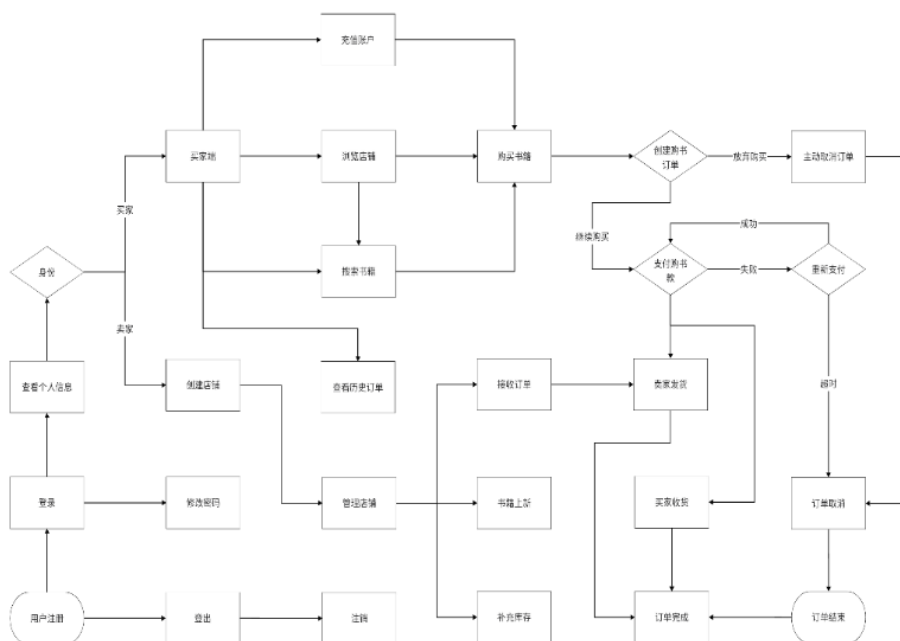


图 1 数据库需求流程图

4.1.2 概念结构设计

分析题设需求，我们计划对书籍（book），订单（order），店铺（store）和用户（user）四个对象进行设计。分别包含以下信息：

书籍：书籍 ID、书籍题目、作者、出版社、原书题目、译者、出版年月、页数、价格、装帧方式、ISBN 号、作者简介、书籍简介、样章试读、标签和照片；

订单：订单 ID、买家 ID、店铺 ID、书籍 ID、购买数量、价格、订单状态和截止日期；

店铺：卖家 ID、店铺 ID、书籍（包括书籍 ID、书籍信息（见上述书籍信息）和库存）；

用户：用户 ID、密码、余额、登录令牌、终端、店铺（见上述店铺信息）和订单（见上述订单信息）。

上述信息中已经包含了一部分对象间的存储关系，我们再来梳理一下：

书籍与订单：一笔订单可以包含多本不同的书籍，同一书籍也可以在不同的订单中出现；

书籍与店铺：同理，一家店铺可以出售多本不同的书籍，同一的书籍也可以在不同的店铺出售；

订单与店铺：一笔订单只能发送至一家店铺，但一家店铺可以拥有多笔订单；

订单与用户：同理，一笔订单只能由一个用户创建，但一个用户可以创建多笔订单；

店铺与用户：同理，一家店铺只能由一个用户创建，但一个用户可以创建多家店铺。

具体的概念结构如下所示：

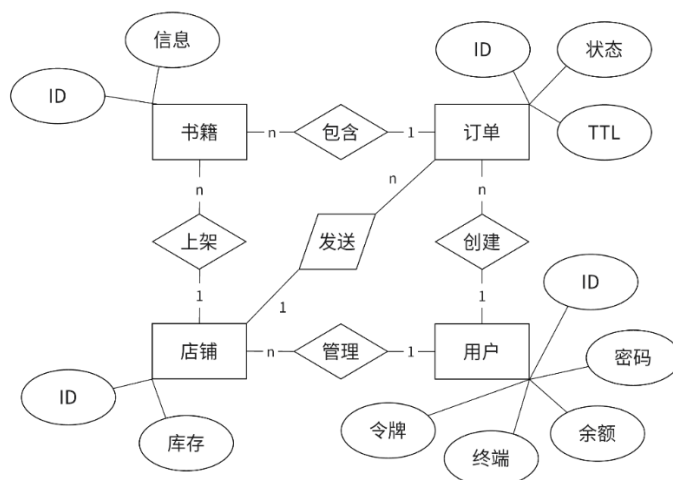
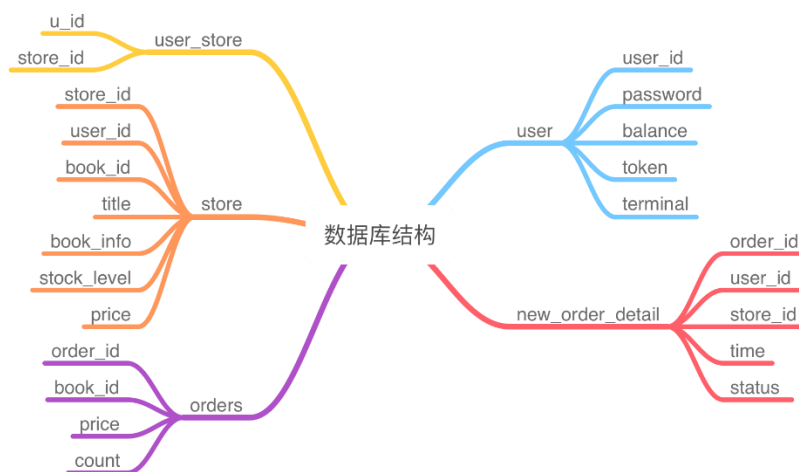


图 2 数据库概念结构 ER 图

4.1.3 逻辑结构设计

上一次的项目中，我们使用 MongoDB 实现了该线上书店的功能。MongoDB 作为一种文档数据库，其嵌套结构能够显著提升一部分查询的效率。其结构如下：



文档数据库通常以文档形式存储数据，这些文档可以是 JSON 或 BSON 格式。然而，这种非结构化的存储方式导致了数据的不同部分可能有不同的结构，难以对数据进行强制的结构化验证，可能导致数据一致性和完整性的问题。随着应用程序的复杂性增加，可能需要执行复杂的查询，而文档数据库对于这类查询的支持可能相对较弱。复杂的关联和聚合操作

可能会变得冗长且难以维护。如果数据之间存在复杂的关系，例如多对多关系，文档数据库的表达能力可能较弱，而关系数据库在处理关系型数据方面更为适用。

相对应地，关系数据库使用表格结构，每个表格有预定义的列和数据类型，通过模式（Schema）对数据的结构进行强制性的定义，确保了数据的一致性和完整性。关系数据库具有强大的查询语言（SQL），支持复杂的查询操作，包括联结、子查询和聚合等。此外，关系数据库提供事务处理机制，确保了对数据库的多步骤操作是原子的、一致的、隔离的、持久的（ACID 特性）。关系数据库模型适用于处理多表之间的关系，能够有效地表示多对多、一对多和一对一等复杂的关系型数据模型。关系数据库是业界最为成熟和广泛应用的数据库模型之一，拥有丰富的工具、技术和社区支持，便于开发者进行使用、维护和优化。

本次实验中，原先提供的 SQLite 是一种嵌入式数据库，适用于单用户或轻量级应用，但在处理高并发和大规模数据时性能相对较差。由于其单一文件存储结构，SQLite 在需要处理大量并发连接或进行水平扩展时表现不佳。虽然支持基本的数据库操作，但相对而言，SQLite 的功能相对较少，缺少一些高级特性，如存储过程和触发器。

而 MySQL 是一种独立的服务器型数据库，适用于高并发和大规模数据处理，具有更优越的性能，还能提供更好的可扩展性，支持复杂的并发操作和水平扩展。它还拥有丰富的数据库特性，适合处理复杂的业务逻辑。并且，MySQL 支持复杂的事务处理，提供完备的事务控制和隔离级别以及更大的存储容量支持，适用于需要存储大量数据的应用。

基于以上需求，我们在转换语法的同时对原有的数据库逻辑结构进行了调整，增加了索引以进一步提升检索效率，并为未来的开发提供方便。新的 MySQL 逻辑结构如下：

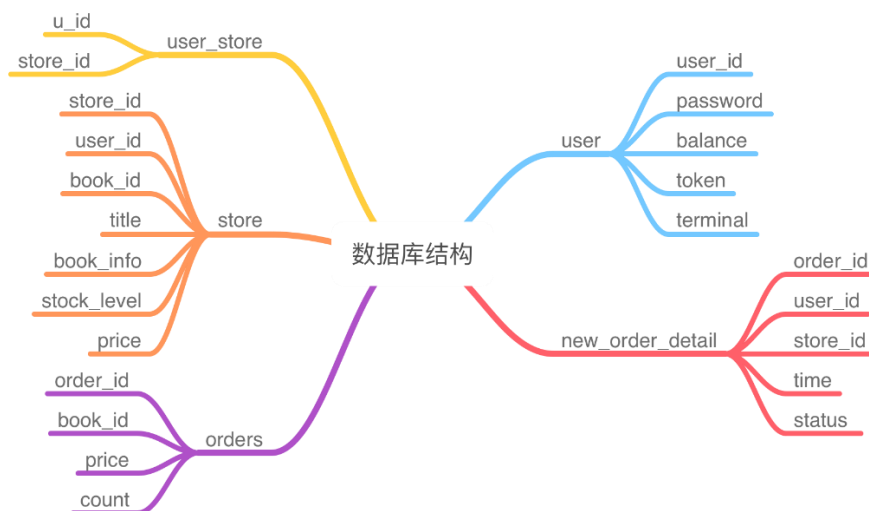


图 4 关系数据库结构导图

4.2 代码实现

4.2.1 理解代码

正式开始工作前，我们应对本项目所包含的文件有一个初步的了解。

- 1) **fe**: 前端。其中 **test** 负责测试代码正确性，**bench** 负责测试数据库吞吐量，**access** 负责前后端交互。
- 2) **be**: 后端。其中 **view** 负责解析前端发送的请求，**model** 负责与 MySQL 数据库进行交互。本次实验主要对 **model** 文件夹中的 **buyer**, **db_conn**, **error**, **seller**, **store** 和 **user** 文件作出修改。

4.2.2 基础功能

实现基础功能，首先我们需要将原有的 SQLite 语法转换为 MySQL 语法。具体而言，包括：

- 1) 修改占位符，从 “?” 修改为 “%s” ；
- 2) 增加游标提取操作，从 “self.conn” 修改为 “self.conn.cursor()”
- 3) 重新创建表格，按照新的数据库逻辑结构增加键与索引；

以上三步我们以 **store.py** 中创建表格的代码为例进行对比：

```
cursor = self.conn.execute(
    "SELECT book_id, stock_level, book_info FROM store "
    "WHERE store_id = ? AND book_id = ?;",
    (store_id, book_id),
)

conn = self.get_db_conn()
cursor = conn.cursor()
cursor.execute("USE DBProj2;")
cursor.execute(
    "CREATE TABLE IF NOT EXISTS user ("
    "user_id VARCHAR(300) PRIMARY KEY, password VARCHAR(300) NOT NULL, "
    "balance INTEGER NOT NULL, token VARCHAR(500), terminal "
    "VARCHAR(500), "
    "INDEX index_user (user_id));"
)
```

- 4) 修改报错，从 “sqlite.Error” 修改为 “pymysql.Error” ；

以 **store.py** 中报错的代码为例进行对比：

```
except sqlite.Error as e:
    logging.error(e)
```

```
conn.rollback()

except pymysql.Error as e:
    logging.error(e)
    conn.rollback()
```

经历了这些修改之后，我们再逐步修改每一个文件。

be/model/store.py

这个文件实现了初始化数据库的功能，并连接数据库。根据前文所述，我们重新对数据库的逻辑结构进行了设计。我们在这里为每个表格设置了主键、外键和索引。

在 MySQL 中，主键确保表中的条目在某个属性上不重复，作为区分不同条目的标志。有些表格中，我们为特定属性添加了主键以确保互异性。外键在 MySQL 中用于确定数据一致性，建立表之间的联系。通过添加外键约束，我们可以避免存在没有实际意义的孤立数据。因此，在一些表格中，我们引入了外键约束，以维护数据关联性。索引的使用有助于提升数据检索效率，使得检索不再依赖于简单的遍历数据库。因此，我们在一些表格的特定属性上添加了索引，以加速数据检索过程。

在`user`表的创建中，我们为`user_id`添加了主键，确保`user`表中不允许存在重复的`user_id`。同时，为了提高检索效率，我们为`user_id`添加了索引，优化了对`user`表的检索操作。

在创建`user_store`表时，我们为`store_id`创建了主键，杜绝了`user_store`表中存在重复的`store_id`条目。此外，为了维护数据一致性，我们为`user_store`表中的`user_id`添加了外键约束，确保每个`user_id`必须对应于`user`表中的某个`user_id`的值。为了提升检索效率，我们还为`store_id`添加了索引，优化了对`user_store`表的检索。

在`store`表的创建中，我们为`(store_id, book_id)`创建了主键，允许店铺 ID 和书本 ID 分别重复，但不允许它们同时重复。同时，为了保持数据关联性，我们为`store_id`属性添加了外键约束，要求每个`store_id`必须对应于`user_store`表中的某个`store_id`值。为了提高检索效率，我们为`(store_id, book_id)`创建了复合索引，并为`title`、`tags`、`author`和`book_intro`分别添加了全文索引，以在检索 ID 和买家进行图书检索时提升效率。

在`new_order`表的创建中，我们为`order_id`创建了主键，防止表中存在相同的`order_id`。为了维护数据一致性，我们为`user_id`和`store_id`添加了外键，确保每个`user_id`对应于`user`表中的某个`user_id`，每个`store_id`对应于`user_store`表中的某个`store_id`。为了提升检索效率，我们还为`order_id`添加了索引，提高了对`new_order`表的检索效率。

在`orders`表的创建中，我们为`(order_id, book_id)`创建了主键，禁止同时重复的条目。此外，为了维护数据关联性，我们为`order_id`添加了外键约束，确保每个`order_id`对应于`new_order`表中的某个`order_id`。为了提高检索效率，我们为`(order_id, book_id)`创建了复合索引，以提升对`orders`表的检索效率。

```
def __init__(self, db_path):
    self.database = os.path.join(db_path, "be.db")
    self.init_tables()

    def init_tables(self):
        try:
            conn = self.get_db_conn()
            cursor = conn.cursor()
            cursor.execute("USE DBProj2;")
            cursor.execute(
                "CREATE TABLE IF NOT EXISTS user ("
                "user_id VARCHAR(300) PRIMARY KEY, password VARCHAR(300) NOT NULL, "
                "balance INTEGER NOT NULL, token VARCHAR(500), terminal"
                "VARCHAR(500), "
                "INDEX index_user (user_id));"
            )

            cursor.execute(
                "CREATE TABLE IF NOT EXISTS user_store ("
                "user_id VARCHAR(300), store_id VARCHAR(300) PRIMARY KEY,"
                "FOREIGN KEY (user_id) REFERENCES user(user_id),"
                "INDEX index_store (store_id))"
            )

            cursor.execute(
                "CREATE TABLE IF NOT EXISTS store ("
                "store_id VARCHAR(300), book_id VARCHAR(300), title VARCHAR(100),"
                "price INTEGER, "
                "tags VARCHAR(100), author VARCHAR(100),"
                "book_intro VARCHAR(2000), stock_level INTEGER,"
                "PRIMARY KEY (store_id, book_id),"
                "FOREIGN KEY (store_id) REFERENCES user_store(store_id),"
                "INDEX index_store_book (store_id, book_id),"
                "FULLTEXT INDEX index_title(title),"
                "FULLTEXT INDEX index_tags(tags),"
                "FULLTEXT INDEX index_author(author),"
                "FULLTEXT INDEX index_book_intro(book_intro))"
            )

            cursor.execute(
                "CREATE TABLE IF NOT EXISTS new_order ("
```



```

        "order_id VARCHAR(300) PRIMARY KEY , user_id VARCHAR(300), store_id
VARCHAR(300), "
        "time TIMESTAMP, status INTEGER,"
        "FOREIGN KEY (user_id) REFERENCES user(user_id), "
        "FOREIGN KEY (store_id) REFERENCES user_store(store_id),"
        "INDEX index_order (order_id))"
    )

    cursor.execute(
        "CREATE TABLE IF NOT EXISTS orders ("
        "order_id VARCHAR(300), book_id VARCHAR(300), count INTEGER, price
INTEGER,"
        "FOREIGN KEY (order_id) REFERENCES new_order(order_id),"
        "PRIMARY KEY (order_id, book_id), "
        "INDEX index_order_book (order_id, book_id))"
    )

    conn.commit()

    def update_data():
        cursor.execute(
            "SELECT * from new_order"
            "WHERE status = 0"
        )
        row = cursor.fetchall()
        for each in row:
            if (datetime.now() - each[3]).total_seconds() > 20:
                cursor.execute(
                    "UPDATE new_order SET status = -1"
                    "WHERE order_id = %s;",
                    (each[0], )
                )
        conn.commit()

    schedule.every(1).second.do(update_data)

    def run_schedule():
        while time.time() - start_time < 10:
            schedule.run_pending()
            time.sleep(1)

    schedule_thread = threading.Thread(target=run_schedule)
    schedule_thread.start()

    except pymysql.Error as e:
        logging.error(e)
        conn.rollback()

    def get_db_conn(self):

```

```

return pymysql.connect(
    host="127.0.0.1",
    port=3306,
    user="root",
    password="021103",
    database="DBProj2"
)

```

我们在创建表时采用`CREATE TABLE IF NOT EXISTS`语句。这确保了在数据库中表不存在时创建表，存在时不执行创建操作，避免重复创建同一表带来的混乱。在创建表时，我们为关键属性添加了主键和索引。例如，在创建 user 表时，我们为 user_id 添加了主键，以防止存在重复的用户 ID。同时，为 user_id 添加索引可以提高在检索`user`表时的效率。

在 get_db_conn() 中连接 MySQL 数据库，返回数据库对象。

```

def get_db_conn():
    global database_instance
    return database_instance.get_db_conn()

```

be/model/db_conn.py

该文件通过调用 store.py 实现了连接数据库的功能，并有三个方法分别可以判断 user、book、store 的 id 是否存在。三种方法实现一样，这里以 store_id_exist() 为例，查找特定 store_id 的用户，若找不到则返回 False。

```

def store_id_exist(self, store_id):
    self.cursor = self.conn.cursor()
    self.cursor.execute(
        "SELECT store_id FROM user_store"
        "WHERE store_id = %s;",
        (store_id,)
    )
    row = self.cursor.fetchone()
    if row is None:
        return False
    else:
        return True

```

be/model/user.py

该文件实现了 user 的功能：注册、注销、登录、登出、修改密码。此处以 register() 为例，初始化用户的信息，将其插入进 user 表中。

```

def register(self, user_id: str, password: str):
    try:

```

```

terminal = "terminal_{}".format(str(time.time()))
token = jwt_encode(user_id, terminal)
self.cursor = self.conn.cursor()
self.cursor.execute(
    "INSERT into user(user_id, password, balance, token, terminal) "
    "VALUES (%s, %s, %s, %s, %s);",
    (user_id, password, 0, token, terminal)
)
self.conn.commit()
except pymysql.Error:
    return error.error_exist_user_id(user_id)
return 200, "ok"

```

be/model/buyer.py

该文件实现了 buyer 的功能：下单、付款、充值。以下单为例，订单信息确认无误后将商店的库存更新，并将该订单插入 order 表中。

```

def new_order(
    self, user_id: str, store_id: str, id_and_count: [(str, int)]
) -> (int, str, str):
    order_id = ""
    try:
        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()))

        self.cursor = self.conn.cursor()

        self.cursor.execute(
            "INSERT INTO new_order(order_id, user_id, store_id, time, status) "
            "VALUES(%s, %s, %s, %s, %s);",
            (uid, user_id, store_id, datetime.now(), 0)
        )
        for book_id, count in id_and_count:
            self.cursor.execute(
                "SELECT book_id, stock_level, price FROM store "
                "WHERE store_id = %s AND book_id = %s;",
                (store_id, book_id)
            )
            row = self.cursor.fetchone()
            if row is None:
                return error.error_non_exist_book_id(book_id) + (order_id,)

            stock_level = row[1]

```

```

        price = row[2]

        if stock_level < count:
            return error.error_stock_level_low(book_id) + (order_id,)

        self.cursor.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 self.cursor.rowcount == 0:
            return error.error_stock_level_low(book_id) + (order_id,)

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

        self.conn.commit()
        order_id = uid

    except pymysql.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

```

be/model/seller.py

该文件实现了 seller 的功能：创建商铺、添加书籍、添加库存。以 add_book() 为例，先将 book 信息组织好，再将其插入 store 表中。

```

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)

    book_info_json = json.loads(book_json_str)
    title = book_info_json.get("title")
    tags = book_info_json.get("tags")

    if tags is not None:
        tags = ",".join(tags)
    author = book_info_json.get("author")
    book_intro = book_info_json.get("book_intro")
    price = book_info_json.get("price")
    self.cursor = self.conn.cursor()

    self.cursor.execute(
        "INSERT into store(store_id, book_id, title, price, tags, author,
book_intro, stock_level)"
        "VALUES (%s, %s, %s, %s, %s, %s, %s, %s);",
        (store_id, book_id, title, price, tags, author, book_intro,
stock_level)
    )

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

```

4.2.3 拓展功能

后 40% 部分由于代码水平和时间原因没能很好地编写测试接口。然而，我们仍然在这里将已写好的代码给出。

I. 卖家发货

编写函数使用 SELECT 语句查找订单，判断是否可以发货，更新订单。若订单存在且已付款，则函数执行。

```

def deliver_order(self, order_id: str) -> (int, str):
    try:
        self.cursor = self.conn.cursor()
        self.cursor.execute(

```

```

        "SELECT status FROM new_order"
        "WHERE order_id = %s and status < 3;",
        (order_id, )
    )
    row = self.cursor.fetchone()

    if row is None:
        return error.error_invalid_order_id(order_id)

    status = row[0]

    if status == -1:
        return error.error_invalid_order_id(order_id)
    elif status == 0:
        return error.error_order_not_paid(order_id)
    elif status == 2:
        return error.error_order_delivered(order_id)

    self.cursor.execute(
        "UPDATE new_order set status = %s"
        "WHERE order_id = %s;",
        (2, order_id)
    )

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

```

II. 买家收货

与发货类似，编写函数使用 SELECT 语句查找用户，判断订单状态并使用 UPDATE 语句更新订单状态。

```

def receive_order(self, user_id, password, order_id) -> (int, str):
    try:
        self.cursor = self.conn.cursor()
        self.cursor.execute(
            "SELECT password from user"
            "WHERE user_id=%s;", (user_id,)
        )
        row = self.cursor.fetchone()
        if row is None:
            return error.error_authorization_fail()

```

```

        if row[0] != password:
            return error.error_authorization_fail()

        self.cursor.execute(
            "SELECT user_id, status from new_order"
            "WHERE order_id = %s;", (order_id, )
        )
        row = self.cursor.fetchone()
        if row is None:
            return error.error_invalid_order_id(order_id)

        status = row[1]

        if status == -1:
            return error.error_invalid_order_id(order_id)
        elif status == 0:
            return error.error_order_not_paid(order_id)
        elif status == 1:
            return error.error_order_not_delivered(order_id)

        self.cursor.execute("UPDATE new_order SET status = %s where order_id
= %s;",
                            (3, order_id))
        self.conn.commit()
    except pymysql.Error as e:
        return 528, "{}".format(str(e))
    except BaseException as e:
        return 530, "{}".format(str(e))

    return 200, "ok"

```

本操作中，我们在 error.py 中添加错误代码 522 来返回订单状态非已发货状态的情况。

```
522: "order {} undelivered",
```

III. 取消订单

设计函数使用 SELECT 语句查找待取消订单，之后使用 UPDATE 语句回滚书本库存、卖家余额和买家余额，最后修改订单状态。

```

def cancel_order(self, user_id, password, order_id) -> (int, str):
    try:
        self.cursor = self.conn.cursor()
        self.cursor.execute(
            "SELECT password FROM user"

```

```
        "WHERE user_id = %s;",
        (user_id, )
    )
    row = self.cursor.fetchone()

    if row is None:
        return error.error_non_exist_user_id(user_id)

    if not password == row[0]:
        return error.error_authorization_fail()

    self.cursor.execute(
        "SELECT user_id, status, store_id FROM new_order"
        "WHERE order_id = %s;",
        (order_id, )
    )
    row = self.cursor.fetchone()
    if row is None:
        return error.error_invalid_order_id(order_id)

    status = row[1]
    store_id = row[2]

    if status == -1:
        return error.error_invalid_order_id(order_id)
    elif status == 2:
        return error.error_order_delivered(order_id)
    elif status == 3:
        return error.error_order_was_received(order_id)

    self.cursor.execute(
        "SELECT book_id, count, price FROM orders"
        "WHERE order_id = %s;",
        (order_id, )
    )
    book_info = []
    row = self.cursor.fetchall()
    for each in row:
        temp = {
            "book_id": each[0],
            "count": each[1],
            "price": each[2]
        }
        book_info.append(temp)

    self.cursor.execute(
        "SELECT user_id FROM user_store"
        "WHERE store_id = %s;",
```



```

        (store_id, )
    )
    row = self.cursor.fetchone()
    seller_id = row[0]

    self.cursor.execute(
        "UPDATE new_order SET status = %s"
        "WHERE order_id = %s;",
        (-1, order_id)
    )

    total_price = 0
    for each in book_info:
        self.cursor.execute(
            "UPDATE store SET stock_level = stock_level + %s"
            "WHERE store_id = %s AND book_id = %s",
            (each["count"], store_id, each["book_id"])
        )
        total_price = total_price + each["count"] * each["price"]

    self.cursor.execute(
        "UPDATE user SET balance = balance + %s"
        "WHERE user_id = %s;",
        (total_price, user_id)
    )
    self.cursor.execute(
        "UPDATE user SET balance = balance + %s"
        "WHERE user_id = %s;",
        (-total_price, seller_id)
    )

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

return 200, "ok"

```

本操作中，我们在 `error.py` 中添加错误代码 523 来返回订单已发货无法取消的情况。

```

def error_order_was_received(order_id):
    return 523, error_code[523].format(order_id)

```

IV. 查询历史订单

使用 `SELECT` 语句查找特定用户的历史订单，使用 `LEFT JOIN` 连接 `new_order` 与 `orders` 以获取完整信息。

```
def search_order(self, user_id: str, password: str) -> (int, str):
    try:
        self.cursor = self.conn.cursor()
        self.cursor.execute(
            "SELECT password FROM user"
            "WHERE user_id = %s;",
            (user_id, )
        )
        row = self.cursor.fetchone()

        if row is None:
            return error.error_non_exist_user_id(user_id)

        if not password == row[0]:
            return error.error_authorization_fail()

        self.cursor.execute(
            "SELECT * FROM new_order LEFT JOIN orders ON new_order.order_id = "
            "orders.order_id "
            "WHERE new_order.user_id = %s;",
            (user_id, )
        )
    except pymysql.Error as e:
        return 528, "{}".format(str(e))
    except BaseException as e:
        return 530, "{}".format(str(e))

    return 200, "ok"
```

V. 搜索图书

使用 SELECT 语句查询数据库中内容，查询关键字和查询范围可选。使用 LIMIT 和 OFFSET 分页，每页 25 条。

```
def search_order(self, user_id: str, password: str) -> (int, str):
    try:
        self.cursor = self.conn.cursor()
        self.cursor.execute(
            "SELECT password FROM user"
            "WHERE user_id = %s;",
            (user_id, )
        )
        row = self.cursor.fetchone()

        if row is None:
            return error.error_non_exist_user_id(user_id)
```

```

        if not password == row[0]:
            return error.error_authorization_fail()

        self.cursor.execute(
            "SELECT * FROM new_order LEFT JOIN orders ON new_order.order_id = "
            "orders.order_id "
            "WHERE new_order.user_id = %s;",
            (user_id, )
        )
    except pymysql.Error as e:
        return 528, "{}".format(str(e))
    except BaseException as e:
        return 530, "{}".format(str(e))

    return 200, "ok"

```

4.2.4 pytest 测试结果

接下来展示完整的代码测试结果。测试结果如下图所示，所有测试案例均通过，覆盖率达到 91%。由于代码中存在 try-except 结构，所以部分文件覆盖率低（例如 test_bench 的覆盖率仅有 67%）是正常情况，不影响实际测试覆盖率。

Name	Stmts	Miss	Branch	BrPart	Cover
be__init__.py	0	0	0	0	100%
be\app.py	3	3	2	0	0%
be\model__init__.py	0	0	0	0	100%
be\model\buyer.py	108	22	44	8	76%
be\model\db_conn.py	25	0	6	0	100%
be\model\error.py	33	6	0	0	82%
be\model\seller.py	62	13	24	2	76%
be\model\store.py	49	5	6	4	84%
be\model\user.py	125	23	38	6	77%
be\serve.py	34	1	2	1	94%
be\view__init__.py	0	0	0	0	100%
be\view\auth.py	37	0	0	0	100%
be\view\buyer.py	31	0	2	0	100%
be\view\seller.py	28	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	68	0	10	1	99%
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	2	22	2	97%
fe\conf.py	11	0	0	0	100%
fe\conf\test.py	17	0	0	0	100%
fe\test\gen_book_data.py	49	1	16	2	95%
fe\test\test_add_book.py	36	0	10	0	100%
fe\test\test_add_funds.py	22	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_create_store.py	19	0	0	0	100%
fe\test\test_login.py	27	0	0	0	100%
fe\test\test_new_order.py	39	0	0	0	100%
fe\test\test_password.py	32	0	0	0	100%
fe\test\test_payment.py	59	1	4	1	97%
fe\test\test_register.py	30	0	0	0	100%
TOTAL	1288	79	216	28	91%

图 5 测试结果

4.3 设计亮点

- 实现额外功能

实现了包括发货、收货、查询书籍、查询订单、取消订单在内的额外功能。

- 基于 Github 进行版本管理

本项目存在多个代码版本，为了实现更好的回滚和修改我们基于 Github 实现版本管理。

具体请参见 [AntaresXY/DBMSProj2: 数据库第二次大作业 \(github.com\)](https://github.com/AntaresXY/DBMSProj2)

- 绘制 ER 图并基于 ER 图导出关系模式、键与索引

参见前文。