当代数据管理系统项目报告

课程名称：当代数据管理系统	**年级：20级**
指导教师：周煊	**组员：贾柏寒 10205501423 杨一帆 10205501455 薛沁晨 10205501413**
项目名称: 书店bookstore	**项目时间: 2022.11-2022.12**

一.相关功能实验要求

实现一个提供网上购书功能的网站后端。
网站支持书商在上面开商店，购买者可以通过网站购买。
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。

1.实现对应接口的功能，见项目的doc文件夹下面的.md文件描述 （60%）

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

通过对应的功能测试，所有test case都pass

2.为项目添加其它功能： （40%）

1)实现后续的流程
发货 -> 收货

2)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。
如果显示结果较大，需要分页
(使用全文索引优化查找)

3)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。
取消定单可由买家主动地取消定单，或者买家下单后，经过一段时间超时仍未付款，定单也会自动取消。

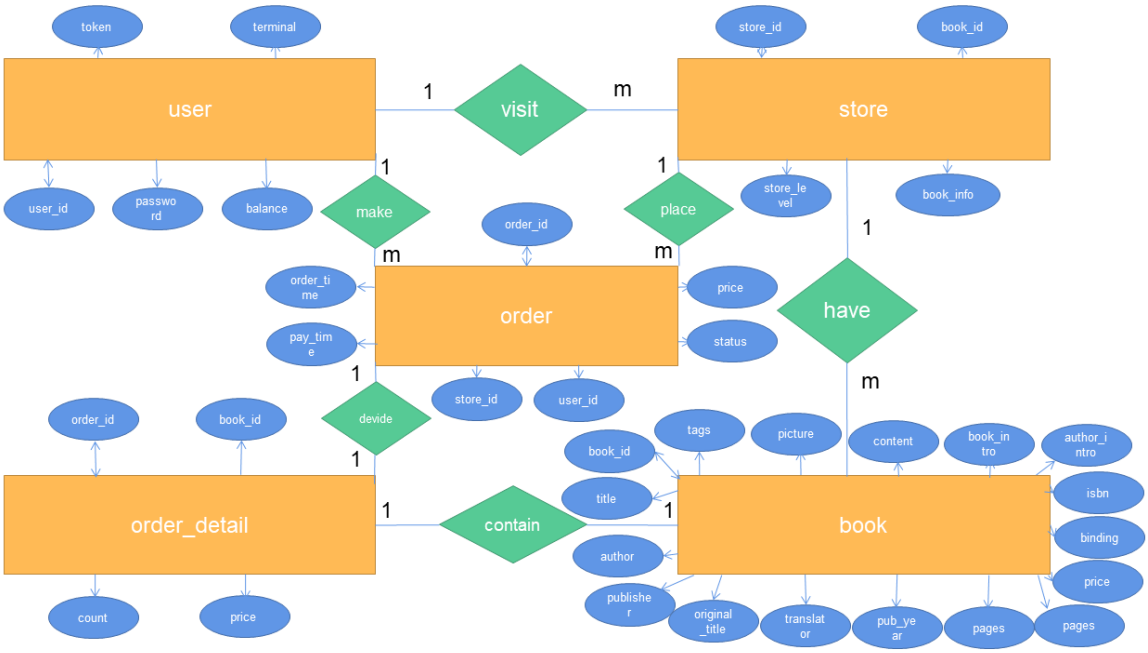
二. 关系数据库设计及初始化

使用: postgresql数据库+sqlalchemy ORM

1.概念设计

根据实验要求，我们需要实现一个提供网上购书功能的网站。用户在注册了自己的账号以后，可以开任意个网上书店售卖书籍，也可以在自己的账户上充值以购买图书；可以通过关键字来搜索图书获取信息，也可以查询订单状态和历史记录。

2.ER图



3.初始化

① 利用SQLalchemy+ORM连接本地Postgresql数据库

```
def init_database():
    # 建立连接('postgresql+psycopg2://postgres:密码@localhost/数据库名称')
    engine =
create_engine('postgresql+psycopg2://postgres:12345678@localhost/bookstore',
encoding='utf-8', echo=True)
    # 建立会话
    DbSession = sessionmaker(bind=engine)
    # 创建对象的基类
    Base = declarative_base()
```

② 建立表格

书籍表

```
# 定义原始的图书表(按照markdown文件中的Schema)
class book(Base):
    __tablename__ = 'book'
```

```
    id = Column(Integer, primary_key=True, autoincrement=True)
    title = Column(Text, nullable=False)
    author = Column(Text, nullable=True)
    publisher = Column(Text, nullable=True)
    original_title = Column(Text, nullable=True)
    translator = Column(Text, nullable=True)
    pub_year = Column(Text, nullable=True)
    pages = Column(Integer, nullable=True)
    price = Column(Integer, nullable=True)
    binding = Column(Text, nullable=True)
    isbn = Column(Text, nullable=True)
    author_intro = Column(Text, nullable=True)
    book_intro = Column(Text, nullable=True)
    content = Column(Text, nullable=True)
    tags = Column(Text, nullable=True)
    # LargeBinary类型可以存储Blob类型文件
    picture = Column(LargeBinary, nullable=True)
```

字段解释:

将表命名为'book'

（按从上至下的顺序各行分别为）

书本的id属性

书本的名称

书本的作者

书本的出版商

书本的原始名称（未经翻译）

书本的翻译者

书本的出版年份

书本的页数

书本的价格

书本的封装

书本的国际标准图书编号（International Standard Book Number)

书本的作者简介

书本的简介

书本的内容

书本的标签

书本的图片

用户表

```
# 用户表
class user(Base):
    __tablename__ = 'user'
    user_id = Column(Text, primary_key=True, unique=True, index=True)
    password = Column(Text, nullable=False)
    balance = Column(Integer, nullable=False)
    token = Column(Text, nullable=False)
    terminal = Column(Text, nullable=False)
```

字段解释：

将表命名为'user'

（按从上至下的顺序各行分别为）

用户的id

用户的密码

用户的账户余额

用户的代金券

用户的终端

用户商店关系表

```
# 用户商店关系表
class user_store(Base):
    __tablename__ = 'user_store'
    user_id = Column(Text, ForeignKey('user.user_id'), primary_key=True,
nullable=False, index=True)
    store_id = Column(Text, primary_key=True, nullable=False, unique=True,
index=True)
```

字段解释：

将表命名为'user_store'

（按从上至下的顺序各行分别为）

用户的id（设置外键，连接到user表中的user_id属性）

商店的id

商店表

```
# 商店表
class store(Base):
    __tablename__ = 'store'
    store_id = Column(Text, ForeignKey('user_store.store_id'), primary_key=True,
nullable=False, index=True)
    book_id = Column(Integer, ForeignKey('book.id'), primary_key=True,
nullable=False)
    book_info = Column(Text, nullable=True)
    stock_level = Column(Integer, nullable=True)
```

字段解释:

将表命名为'store'

（按从上至下的顺序各行分别为）

商店的id（设置外键，连接到use_store表中的store_id属性）

书本的id（设置外键，连接到book表中的id属性）

书本信息

库存量

订单表

```
# 订单表
class new_order(Base):
    __tablename__ = 'new_order'
    order_id = Column(Text, primary_key=True, index=True)
    user_id = Column(Text, nullable=False)
    store_id = Column(Text, nullable=False)
    price = Column(Integer, nullable=False)  # 取消订单后返还金额
    status = Column(Text, nullable=False)
    order_time = Column(DateTime, nullable=False)
    pay_time = Column(DateTime, nullable=True)
```

字段解释:

将表命名为'new_order'

（按从上至下的顺序各行分别为）

订单的id

用户的id

商店的id

价格

状态（有如下几种：ordered:已下单未付款 paid: 已付款未发货 delivered:已发货未收货 received:已收货 canceled:已取消）

下单时间

支付时间

订单细节表

```
class new_order_detail(Base):
    __tablename__ = 'new_order_detail'
    order_id = Column(Text, primary_key=True, nullable=False, index=True)
    book_id = Column(Integer, primary_key=True, nullable=False)
    count = Column(Integer, nullable=False)
    price = Column(Integer, nullable=False)
```

字段解释：

将表命名为'new_order_detail'

（按从上至下的顺序各行分别为）

订单的id

书本的id

数量

价格

注：订单表和订单细节表的区别是：订单表存储的是一个订单的整体信息，订单细节表存储的是一个订单里面每一条购买书籍的信息。因此用户购买一次书籍，订单表只有一条记录，而订单细节表有多条记录。

(下面的表格均为书籍信息分词存储表格)

搜索标题表

```
class Search_title(Base):
    __tablename__ = 'search_title'
    search_id = Column(Integer, nullable=False)
    title = Column(Text, nullable=False)
    id = Column(Integer, ForeignKey('book.id'), nullable=False)

    __table_args__ = (
        PrimaryKeyConstraint('search_id', 'title'),
        {},
    )
```

字段解释：

将表命名为'search_title'

（按从上至下的顺序各行分别为）

分词结果id

标题分词

原先的Book_id(设置外键，连接到book表中的id属性)

设置主键:

search_id和title的联合主键

搜索作者表

```
# 搜索作者表
class Search_author(Base):
    __tablename__ = 'search_author'
    search_id = Column(Integer, nullable=False)
    author = Column(Text, nullable=False)
    id = Column(Integer, ForeignKey('book.id'), nullable=False)

    __table_args__ = (
        PrimaryKeyConstraint('search_id', 'author'),
        {},
    )
```

字段解释：

将表命名为'search_author'

（按从上至下的顺序各行分别为）

分词结果id

作者分词

原先的Book_id(设置外键，连接到book表中的id属性)

设置主键:

search_id和author的联合主键

搜索书本内容表

```
# 搜索书本内容表
class Search_book_intro(Base):
    __tablename__ = 'search_book_intro'
    search_id = Column(Integer, nullable=False)
    book_intro = Column(Text, nullable=False)
    id = Column(Integer, ForeignKey('book.id'), nullable=False)

    __table_args__ = (
        PrimaryKeyConstraint('search_id', 'book_intro'),
        {},
    )
```

字段解释：

将表命名为'search_book_intro'

（按从上至下的顺序各行分别为）

分词结果id

书本内容分词

原先的Book_id(设置外键，连接到book表中的id属性)

设置主键:

search_id和book_intro的联合主键

③ 连接原先的sqlite数据库并将数据存储到postgresql数据库中

```
# 插入数据(将sqlite数据库的数据转存到postgresql中)
def insertData():
    # 建立一个会话
    session = DbSession()
    # 连接sqlite数据库book.db
    conn = sqlite.connect("../../fe/data/book.db")
    # 在book.db中提取数据
    cursor = conn.execute(
        "SELECT id, title, author, "
        "publisher, original_title, "
        "translator, pub_year, pages, "
        "price, currency_unit, binding, "
        "isbn, author_intro, book_intro, "
        "content, tags FROM book"
    )

    # 将数据存储到一个新对象中，并将对象插入postgresql数据库的Book表中
    for i in cursor:
        new_book = book(id=i[0], title=i[1], author=i[2], publisher=i[3],
original_title=i[4], translator=i[5],
                        pub_year=i[6], pages=i[7], price=i[8], binding=i[9],
isbn=i[10], author_intro=i[11],
                        book_intro=i[12], content=i[13], tags=i[14])
        # 添加到session
        session.add(new_book)
        # 提交即保存到数据库
        session.commit()

    # 关闭session
    session.close()
```

④ 将book表中的数据分词插入分词表

思路:

1.从book表中读取数据

2.预处理: 去掉停用词, 如果去掉后存在空数据则跳过空数据

3.分词, 使用jieba库的搜索引擎模式和textrank进行分词

4.将对应的分词结果插入分词表中

4.可视化

使用Navicat对数据库进行可视化, 这里以book表和搜索标题表进行展示:

book表

id	title	author	publisher	original_title	translator	pub_year	pages	price	binding	isbn	author_intro	book_intro	content
1000067	美丽心灵	[美] 西尔维娅	上海科技教育出	A Beautiful Mind	王尔山	2002-1	594	3879	元	平装	9787542823823	西尔维娅·娜萨，在这本感人	
1000134	三毛流浪记全	张乐平	少年儿童出版社	(Null)	(Null)	2001-8-1	261	3000	元	平装(无盘)	9787532446674	张乐平，是中国当	《三毛流浪
1000135	黑色棉花田	(美) 蜜尔德	中国少年儿童出	(Null)	吴桢祥	1998-08	322	1380	元	平装	9787500739197	蜜尔德瑞·泰勒(M	在民主国家
1000167	中国历代年号	李崇智	中华书局	(Null)	(Null)	2001-01	307	2000	元	平装	9787101025125		《中國歷代
1000280	袁氏当国	[美] 唐德刚	广西师范大学出	(Null)	(Null)	2004-11	209	1700		平装	9787563350032	唐德刚 1920年出	《袁氏当国
1000965	早年毛泽东	李锐	辽宁人民出版社	(Null)	(Null)	1997-01	575	2600		平装	9787205019648		
1000317	一个狗娘养的	(美)艾伦.纽哈	东方出版社	Confessions of an	李斯	2004-03	291	2500		平装	9787506018586	艾伦·纽哈斯：	如果你出身
1000364	新唐书（全二	宋祁	中华书局	(Null)	(Null)	1975-2-1	6472	25000		平装	9787101003208		《新唐书(套
1000531	第二次世界大	温斯顿·丘吉尔	南方出版社	(Null)	吴万沈等	2003-4-1	2975	29800		精装	9787806608050		《第二次世
1000687	学习的革命	(Null)	上海三联书店	(Null)	顾瑞荣	1998-12-1	524	2800		平装	9787542612045	珍妮特·沃斯（Ja	它对于商业
1000773	人类本性与社	(美)查尔斯.霍	华夏出版社	(Null)	包凡一/等	1999-01	305	1880		平装	9787508016450	查尔斯·霍顿·库利	"现代西方思
1000786	垃圾之歌	(Null)	中国社会科学出	(Null)	(Null)	1999-07	307	1800	元	平装	9787500424581		从纽约市斯
1000972	一生的忠告	[英] 查斯特菲	海潮出版社	(Null)	(Null)	2001-9	301	2000	元	平装	9787801514301	查斯特菲尔德勋爵	《一生的忠
1000993	女生日记-中:	杨红樱	作家出版社	(Null)	前五	2000-08	367	1600		平装	9787506317023	杨红樱，中国作家	杨红樱，中
1001136	彼得·潘	(Null)	哈尔滨出版社	Peter Pan	艾柯	2003-7	199	1380	元	平装	9787806398159	詹姆斯·巴里（Jam	彼得·潘，满
1001165	宋书（全八册	沈约	中华书局	(Null)	(Null)	1974-10	2471	9500		平装	9787101003093		宋书（1-8）
1001193	再见了，可鲁	摄影 [日] 秋元	南海出版公司	(Null)	[日] 猿渡静子 译	2003-1	155	2500	元	精装	9787544223447	石黑谦吾，1961年	可鲁是一条
1002898	父与子全集	[德] 埃·奥·卜	中国工人出版社	(Null)	杨莹	2003-4	204	2000	元	平装	9787500830146	埃·奥·卜劳恩（e.o	内容简介
1001204	山居笔记	余秋雨	文汇出版社	(Null)	(Null)	2002-1	324	2800		平装	9787806761120		《山居笔记
1001309	苦儿流浪记	[法] 埃克多·马	人民文学出版社	(Null)	(Null)	1998-5	706	2300	元	平装	9787020027644	埃克多·马洛（Hec	雷米被养父
1001605	呐喊	绘者 王伟君	漓江出版社	(Null)	(Null)	1999-12	198	2800	元	平装	9787540725235	鲁迅，本名周树人	《呐喊》收
1001885	带一本书去旅	林达	生活·读书·新知三	(Null)	(Null)	2002-5	287	3500	元	平装	9787108017079	林达为两名作者各	作者在浓厚
1001895	靠自己去成功	[美] 刘墉	长江文艺出版社	(Null)	(Null)	2003-10	238	1600	元	平装	9787535426253		《靠自己去
1001924	三国演义	罗贯中	岳麓书社	(Null)	(Null)	2001-9	596	1450	元	精装	9787806651094	罗贯中（约1330	《三国演义
1001953	张居正大传	朱东润	百花文艺出版社	(Null)	(Null)	2000-10-1	453	2200	元	平装	9787530630150	朱东润（1896 -	本书的内容
1002201	马桥词典	韩少功	上海文艺出版社	(Null)	(Null)	1997-3	401	2000	元	平装	9787532114016	韩少功，湖南长沙	《马桥词典
1002299	笑傲江湖（全	金庸	生活·读书·新知三	(Null)	(Null)	1994-5	1599	7680	元	平装	9787108006639	金庸本名查良镛，《	《笑傲江湖
1003000	悟空传	今何在	光明日报出版社	(Null)	(Null)	2001-4	300	1480	元	平装	9787801453648	今何在，网络文学	猪八戒和孙
1002400	城的灯	李佩甫	长江文艺出版社	(Null)	(Null)	2003-3-1	417	2000		平装	9787535424778	李佩甫李佩甫，1	作家通过历

搜索标题表

search_id	title	id
0	美丽	1000067
0	心灵	1000067
0	美丽心灵	1000067
0	三毛	1000134
0	流浪	1000134
0	流浪记	1000134
0	全集	1000134
0	三毛流浪记全集	1000134
0	黑色	1000135
0	棉花	1000135
0	棉花田	1000135
0	黑色棉花田	1000135
0	中国	1000167
0	历代	1000167
0	年	1000167
0	号	1000167
0	考	1000167
0	中国历代年号考	1000167
0	袁氏	1000280
0	当国	1000280
0	袁氏当国	1000280
0	早年	1000965
0	毛泽东	1000965
0	早年毛泽东	1000965
0	一个	1000317
0	狗娘养	1000317
0	的	1000317
0	自白	1000317
0	一个狗娘养的自白	1000317
0	唐书	1000364

可以看出搜索标题表中存储了标题分词结果

三.功能实现

1.用户权限接口相关功能

① 注册

实现思路:

1.检查用户id是否存在

2.生成一个终端

3.对标志位编码

4.生成一个新用户（包含信息：用户id、密码、账户余额、代金券、终端等）

5.将数据写入db

② 检查密码

1.将位设成第一个满足以user_id为条件的筛选的结果

2.检查位是否存在

3.检查密码是否正确

③ 登陆

1.检查密码是否正确

2.对标准位编码

3.检查用户表中该user_id的数据是否存在

4.更新标志位和终端的值

5.提交数据

④ 登出

1.检查标志位判断登录状态

2.终端输出时间

3.对新的标志位编码

4.检查用户表中该user_id的数据是否存在

5.更新标志位和终端的值

6.提交数据

⑤ 注销

1.检查密码

2.删除user表中该用户id的数据

3.检查是否已删除完毕，若已删完则提交数据

⑥ 修改密码

1.检查原来未修改的密码

2.终端输出时间

3.对新的标志位编码

4.检查用户表中该user_id的数据是否存在

5.更新标志位和终端的值

6.提交数据

⑦ 搜索图书

原先想法: 使用like模糊匹配实现

功能实现

1. 首先判断买家id是否存在
2. 设置查找的类型: 题目, 标签, 目录, 内容

3. 判断是全站搜索还是当前店铺搜索
4. 使用like进行搜索匹配
5. 搜索后判断是否搜索成功并添加搜索信息
6. 如果是当前店铺搜索则需要判断店铺id是否存在

性能分析

book表全站搜索一次查询，根据店铺book表搜索两次查询

分页的实现

当搜索匹配的书籍量过大时书籍信息较多, 一页展示不下, 则需要分页, 分页的实现如下:

```
# 分页
all_book = self.session.query(book).filter(
                    type[int(search_type)].like("%" + search_content +
"%")).all()
count = all_book.count()
pagesize = 5
if count >= 5:
    for i in count / 5:
        all_book = self.session.query(book).filter(
                type[int(search_type)].like("%" + search_content +
"%")).offset(pagesize * i).limit(pagesize).all()
```

实现过程

1. 统计查找结果的总条数
2. 设置一页最大展示条数为5, 如果查找结果超过了最大展示条数, 则需要分页
3. 对每一页, 用offset过滤掉之前展示过的结果, limits限制分页条数
4. 之后的流程按照不分页的情况即可

优化后的想法:使用分词进行全文索引搜索

思路:

1.分别创建几个函数对应不同的搜索模式

2.以全站搜索作者为例: 使用join将book和Search_author两表按照id连接, 查找给定作者关键词

(加入分页, 方法与上面的分页查找方法类似)

3.判断是否搜索无结果

4.取出相关书籍信息

性能分析:

book表和Search_author表各一次查询

其余搜索类型函数均类似

2.买家用户接口相关功能

① 下单

功能实现

1. 检查用户id和商家id是否存在
2. 生成每个用户id和商家id的标识符uid
3. 对于每一件要购买的书: 先查找商家和书籍信息,在商家表中查找商家id和书籍id, 判断书籍是否存在
4. 获取库存量, 书籍信息, 书籍价格, 判断库存是否充足
5. 更新商家图书信息: 先找到待更新商家图书, 判断是否库存不足, 然后更新库存
6. 更新订单细节表new_order_detail
7. 全部书籍查找结束后更新订单表new_order, 记录下单时间并设置状态status为已下单

性能分析

store表k次根据主键store_id查询，k次更新，new_order表k次插入，k为购买图书的次数

② 支付

功能实现

1. 根据订单id查找订单, 判断订单是否存在
2. 根据查找到订单的买家id, 判断买家用户信息是否一致
3. 信息一致后查找买家用户并判断是否存在该买家
4. 验证买家的密码是否正确
5. 查找user_store表判断店铺是否存在
6. 查找买家id判断是否存在
7. 查找订单细节表, 计算订单总价判断用户是否有足够余额支付
8. 查找订单表并更新订单总价, 设置订单状态为已支付, 记录订单支付时间
9. 更新买家余额(注意卖家余额不需要更新, 因为用户支付后卖家不会立刻收到付款, 而是到确认订单后)

性能分析

new_order表一次根据主键order_id查询，一次更新，user表两次根据主键user_id查询，两次更新余额（其中一次买家、一次卖家）。

③ 充值

功能实现

1. 判断用户是否存在,密码是否输入正确
2. 更新用户余额

性能分析

user表一次根据主键user_id查询，一次更新。

④ 收货

功能实现

1. 查找用户id判断该用户是否存在
2. 查找订单表判断订单是否在配送中(若待支付或已支付尚未配送则无法收货)
3. 判断用户id和买家id是否一致

4. 通过订单表取出卖家id, 查找卖家id判断卖家是否存在
5. 通过订单表取出商铺id, 查找商铺id判断商铺是否存在
6. 更新卖家余额

性能分析

new_order表一次根据主键order_id查询，一次更新，user_store表一次查找，user表一次根据主键user_id查询，一次更新余额（卖家）。

⑤ 查看历史订单

功能实现

1. 检查用户是否存在
2. 分类查询: 可以查询所有订单, 已下单未支付的, 已支付未发货的, 已发货未收货的和取消的订单
3. 对于每种查询, 两层循环, 第一层遍历所有订单取出订单id
4. 第二层用订单id去订单细节表中查找每个订单的子订单信息并记录

性能分析

查所有订单：new_order表一次查询，对应new_order_detail表k次根据主键查询（k为用户订单记录数）

查询待付款订单：new_order表一次查询，对应new_order_detail表k次根据主键查询（k为new_order.status=“ordered”的用户订单待付记录数）

查询已付款待发货订单：new_order表一次查询，对应new_order_detail表k次根据主键查询（k为new_order.status==“delivered”的该用户待发货记录数）

查询已发货待收货订单：new_order表一次查询，对应new_order_detail表k次根据主键查询（k为new_order.status==“ received”的该用户待发货记录数）

查询已取消订单：new_order表一次查询，对应new_order_detail表k次根据主键查询（kk为new_order.status==“ canceled”的该用户待发货记录数）

⑥ 取消订单

功能实现

1. 判断用户是否存在
2. 查找订单判断是否属于已下单未支付订单
3. 如果订单属于已支付订单, 判断订单是否发货(已发货订单无法取消)
4. 若订单属于已支付未发货订单, 则订单可以取消, 修改订单状态为已取消, 并未买家退款
5. 若订单属于已下单未支付订单, 则直接修改订单状态为已取消

性能分析

new_order表一次查询，一次更新，store表一次查询，new_order_detail表一次查询，k次更新，k为购买图书数量

user表一次查询，一次更新

⑧ 订单超时自动取消

功能实现

1. 设置最大待支付时间(这里为了方便测试设置为5s)
2. 查找待支付订单(订单若已经支付后则无法取消)并获取下单时间
3. 判断下单时间和当前时间差是否超过了最大支付时间, 若超过则取消订单, 并设置status为canceled

性能分析:

new_order表一次查询

3.卖家用户接口相关功能

① 添加书籍

功能实现

1. 判断用户id,店铺id,书籍id是否存在
2. 创建商铺对象, 包含书籍id,信息,库存
3. 将商铺对象插入store表中

性能分析

store表一次查询，一次更新

② 增加库存

功能实现

1. 判断用户id,店铺id,书籍id是否存在
2. 根据store_id和book_id查找相关店铺
3. 更新相关书籍库存

性能分析

store表根据主键一次查找，一次更新

③ 发货

功能实现

1. 判断该用户(买家)是否存在
2. 查找待发货订单并判断是否存在该订单
3. 根据store_id判断买家身份是否一致
4. 验证买家身份后修改订单状态为已发货

性能分析

new_order表一次查询，一次更新，user_store表一次查询

④ 创建店铺

功能实现

1. 判断用户id,店铺id是否存在
2. 按照用户id,店铺id创建用户店铺关系对象加入user_store表

性能分析

user_store一次查询，一次更新

四.功能测试

注: 1.原先的测试接口实现不变, 这里仅展示新加的测试接口实现

2.fe/access的调用接口和be/view的路由需要注意url的一致, 其余实现非常类似暂不展示

① test_cancel_order:

初始化:(初始化的过程每个测试文件基本相同,后续不再提及)

1.生成买家id, 卖家id, 店铺id

2.生成密码, 书籍id,书籍信息

3.注册买家

4.生成订单id

5.计算订单书籍总价格并给买家充值

测试:

1.测试已支付未发货后是否能正常取消

2.改变买家id判断是否会出错

3.判断确认收货后是否能取消

② test_deliver_order
初始化

测试:

1.测试已支付后订单是否能正常发货

2.测试修改卖家id后是否会出错

3.测试修改订单id后是否会出错

③ test_history_order

初始化

测试:

1.分类测试: 对于全部订单, 已下单未支付, 已支付未发货, 已发货未收货, 已取消订单分别测试

2.测试修改买家id是否会出错

④ test_receive_order

初始化

测试:
1.测试是否可以正常收货

2.测试修改买家id后是否会出错

3.测试修改订单id后是否会出错

⑤ test_timeout_cancel

初始化

测试:
1.先测试未超时的订单, 再time.sleep一段时间使其超时判断是否会取消

2.测试已支付的订单是否可以取消

⑥ test_search

初始化:

该部分初始化不需要买家和订单的生成, 故简化为创建不同的搜索内容

测试:

对不同搜索范围的接口分别搜索

测试结果

最后测试结果是44个通过

```
========================================================================== 44 passed in 1006.67s (0:16:46) ==========================================================================
```

五.优化

1.索引

创建索引以优化查询

① book表

id, title, book_intro, content, tags建立索引

```
id = Column(Integer, primary_key=True, autoincrement=True,index=True)
title = Column(Text, nullable=False,index=True)
book_intro = Column(Text, nullable=True,index=True)
content = Column(Text, nullable=True,index=True)
tags = Column(Text, nullable=True,index=True)
```

创建表结果:

```
INFO sqlalchemy.engine.Engine CREATE INDEX ix_book_content ON book (content)
INFO sqlalchemy.engine.Engine [no key 0.00012s] {}
INFO sqlalchemy.engine.Engine CREATE INDEX ix_book_id ON book (id)
INFO sqlalchemy.engine.Engine [no key 0.00011s] {}
INFO sqlalchemy.engine.Engine CREATE INDEX ix_book_title ON book (title)
INFO sqlalchemy.engine.Engine [no key 0.00008s] {}
INFO sqlalchemy.engine.Engine CREATE INDEX ix_book_tags ON book (tags)
INFO sqlalchemy.engine.Engine [no key 0.00008s] {}
INFO sqlalchemy.engine.Engine CREATE INDEX ix_book_book_intro ON book (book_intro)
```

② usr表

user_id创建索引

```
user_id = Column(Text, primary_key=True, unique=True,index=True)
```

创建表结果:

```
INFO sqlalchemy.engine.Engine CREATE UNIQUE INDEX ix_usr_user_id ON usr (user_id)
```

③ user_store表

user_id, store_id创建索引

```
user_id = Column(Text, ForeignKey('user.user_id'), primary_key=True,
nullable=False, index=True)
store_id = Column(Text, primary_key=True, nullable=False,
unique=True,index=True)
```

创建表结果:

```
INFO sqlalchemy.engine.Engine CREATE UNIQUE INDEX ix_user_store_store_id ON user_store (store_id)
INFO sqlalchemy.engine.Engine [no key 0.00010s] {}
INFO sqlalchemy.engine.Engine CREATE INDEX ix_user_store_user_id ON user_store (user_id)
```

④ store表

store_id创建索引

```
store_id = Column(Text, ForeignKey('user_store.store_id'), primary_key=True,
nullable=False, index=True)
```

创建表结果:

```
INFO sqlalchemy.engine.Engine CREATE INDEX ix_store_store_id ON store (store_id)
```

⑤ new_order表

order_id创建索引

```
order_id = Column(Text, primary_key=True,index=True)
```

创建表结果:

```
INFO sqlalchemy.engine.Engine CREATE INDEX ix_new_order_order_id ON new_order (order_id)
```

⑥ new_order_detail表

order_id创建索引

```
order_id = Column(Text, primary_key=True, nullable=False, index=True)
```

创建表结果:

```
INFO sqlalchemy.engine.Engine CREATE INDEX ix_new_order_detail_order_id ON new_order_detail (order_id)
```

⑦全文索引:

使用方法见分词部分

六.协作与版本管理

使用Github与Git进行协作与版本控制

1.协作

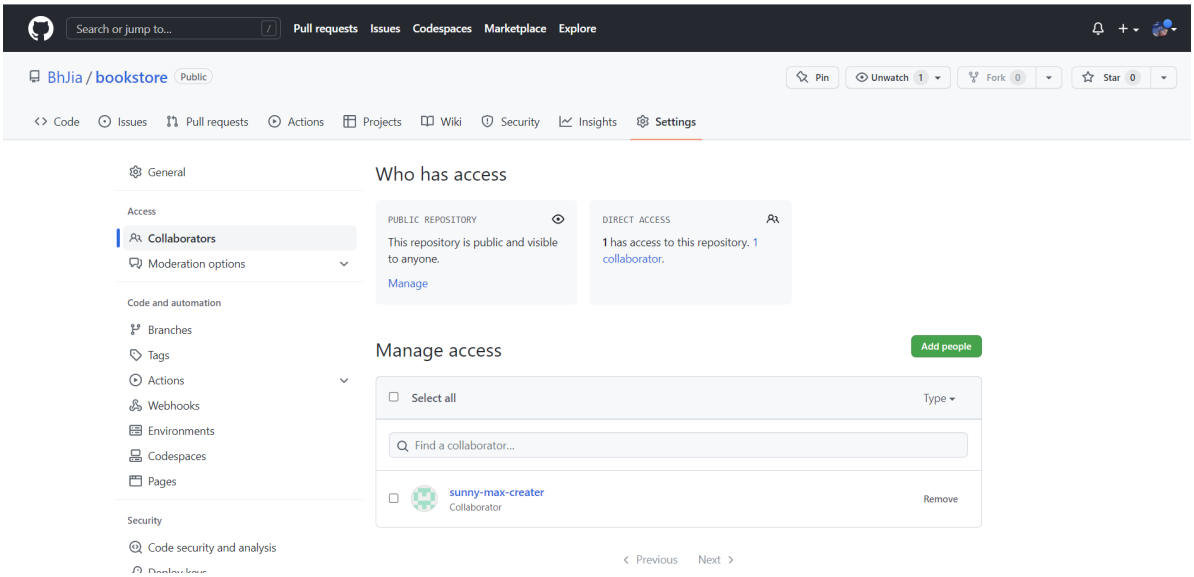
我们使用了creator和collaborator的模式进行协作, 创建者和协作者都可以直接修改提交项目并同步更新项目

创建者

① Github创建仓库

② 为仓库添加collaborator

协作者收到邮件确认后即可加入项目开发



③ 创建者在本地clone仓库(使用SSH), 并设置用户名和邮箱

```
git config user.name "name"
git config user.email "email"
```

④ 创建分支

由于一般项目开发都是先使用开发版本, 最后再合并为最终版本, 因此在本地创建分支dev并push到远程, 这样本地和远程都有新分支dev, 之后的开发都是在dev上进行

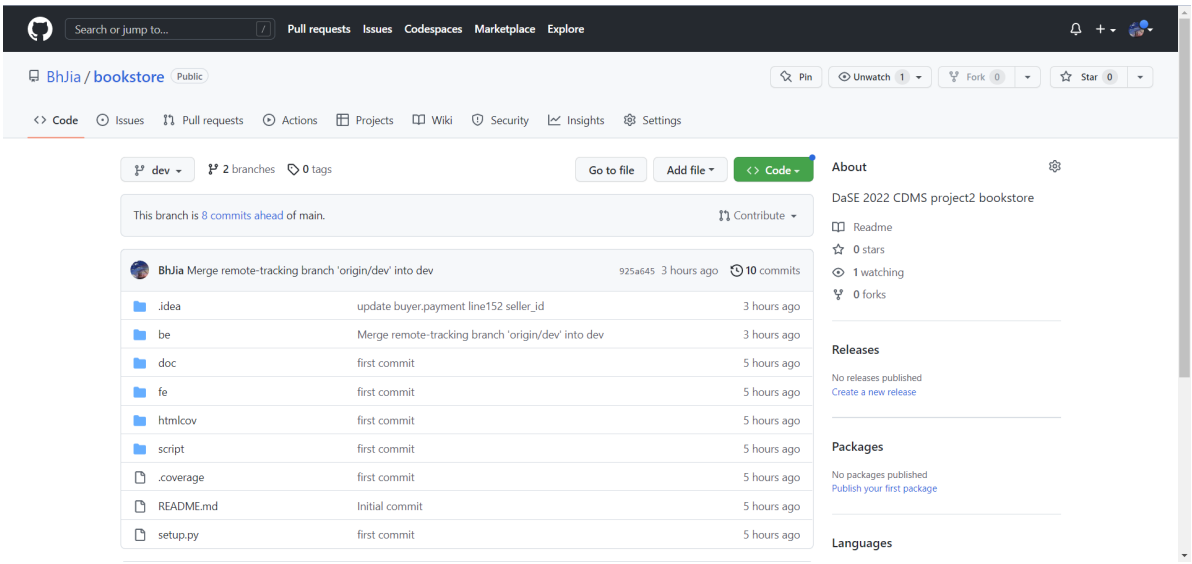
```
# 创建本地分支dev
git checkout -b dev

# push到远程
git push origin dev
```

⑤ 将文件提交到远程仓库

把已经写好的代码提交到dev分支

```
git add <filename>  # 或添加全部文件git add .
git status   # 查看状态信息
git commit -m "message"
git push <远程主机名> <本地分支名>:<远程分支名>
# 如果本地分支名与远程分支名相同，则可以省略冒号
git push <远程主机名> <本地分支名>  # git push -u origin dev
```



协作者

① 协作者在作为collaborator加入项目后, 同样先将远程仓库clone到本地, 设置用户名邮箱, 创建和创建者使用分支相同名称的分支

② 在提交更新前先要同步本地仓库

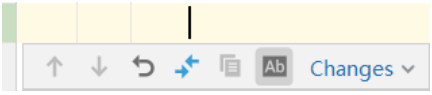
```
git pull <远程主机名> <远程分支名>:<本地分支名>
# 如果远程分支是与当前分支合并，则冒号后面的部分可以省略
git pull <远程主机名> <远程分支名> # git pull origin dev
```

③ 提交变更, 方法和创建者相同, 此时创建者可以看到变更, 创建者需要同步更新仓库

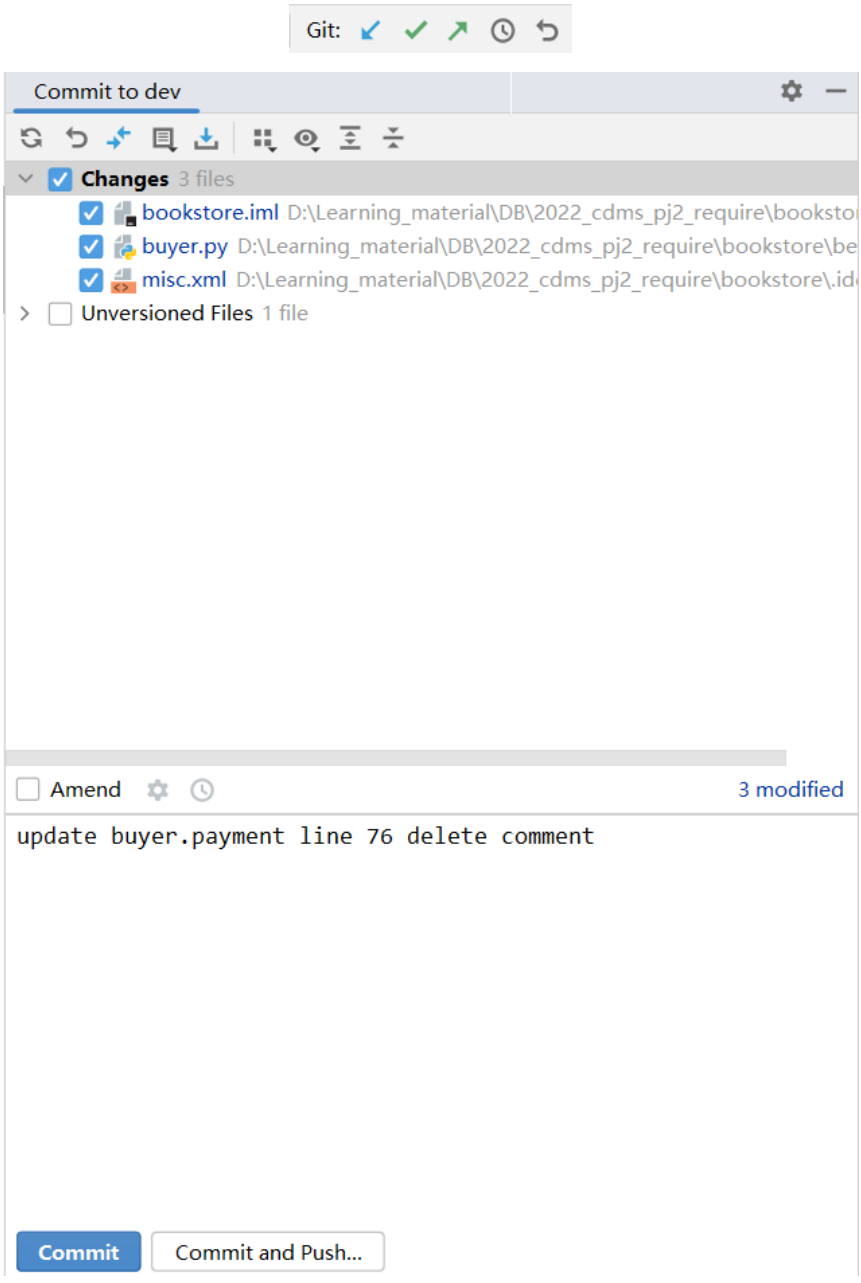
Pycharm中使用Git

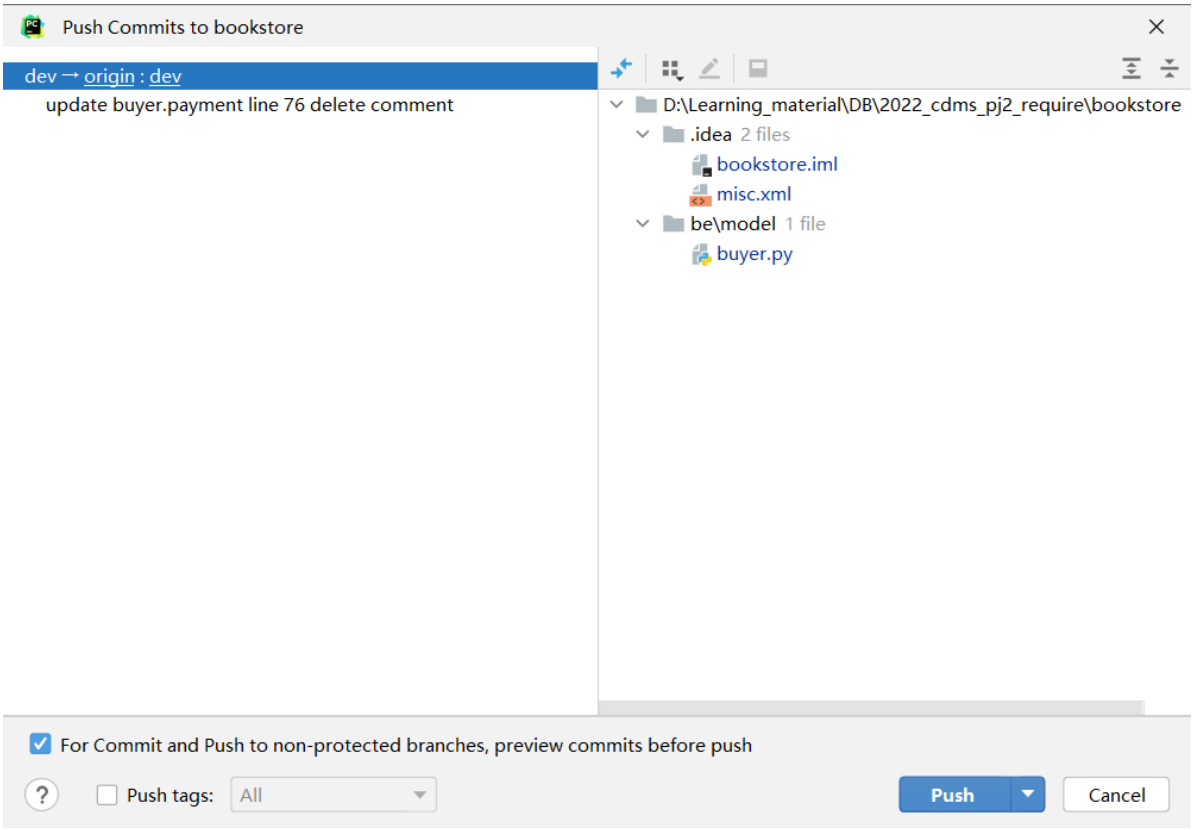
在settings-Version Control中配置Git路径, 之后打开clone的项目就可以使用Git

代码变动后左侧会有提示



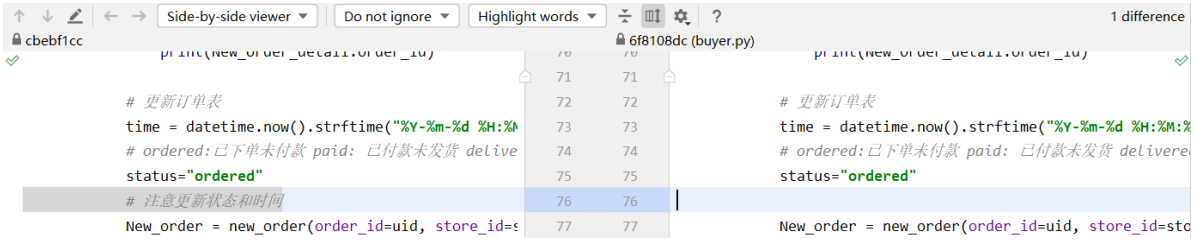
可以使用图形化界面进行update, commit,push等操作





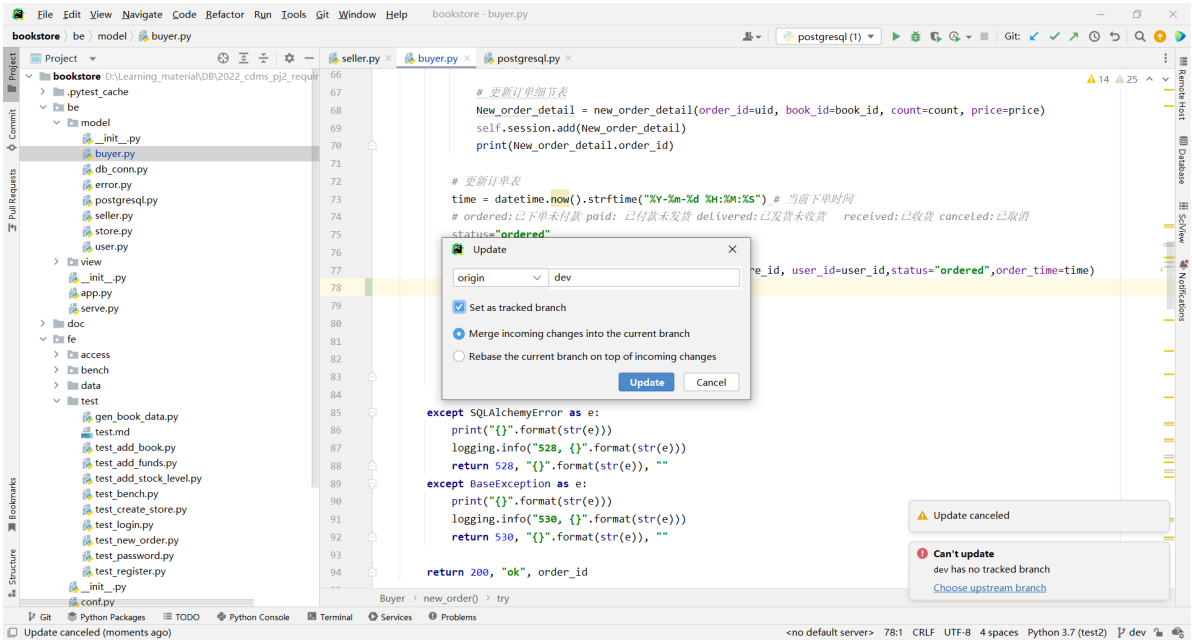
可以看到版本变更:

点开可以查看具体变更信息:



工程问题:

① 进行update若出现以下错误: dev has no tracked branch



分析: git有三种分支, 远程分支, 远程跟踪分支, 跟踪分支

远程分支(remote branch):即远程仓库的分支, 比如这里的main和dev

远程跟踪分支(remote-tracking branch): 本地仓库对远程仓库中的某个远程分支的状态的记录, 以**远程仓库名/远程分支名**来命名, 例如想查看最后一次与远程仓库origin通信时dev分支的情况, 那么就要查看远程跟踪分支origin/dev; 远程跟踪分支的作用是告诉用户其所跟踪的远程分支的状态（即指向哪一个commit），因而它在本地是只读的，用户是无法自行修改它的指向

跟踪分支(tracking branch): 从一个远程跟踪分支产生出的一个本地分支便是跟踪分支，该本地分支对应的远程跟踪分支称为上游分支(upstream branch), 跟踪分支为本地分支和远程分支之间建立了一种联系，方便了远程分支和本地分支的同步

当克隆一个仓库时，它通常会自动地创建一个跟踪 origin/main 的 main 分支(跟踪分支), 也可以checkout创建新分支

那么这样的报错就是由于没有设置远程跟踪分支导致的, 只要指定远程跟踪分支(上游分支即可)

方法1: 直接设置远程追踪分支

```
git branch --set-upstream-to origin/dev
```

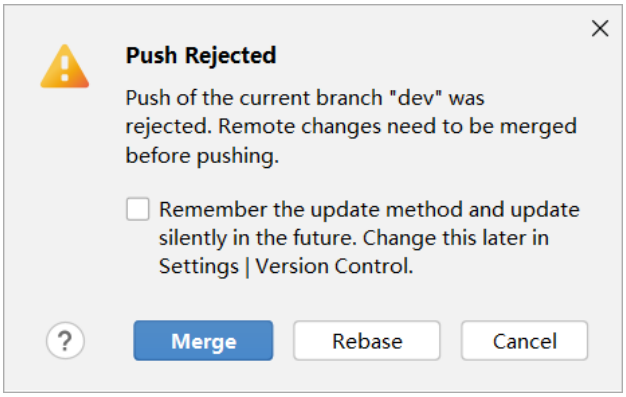
方法2:-u即记住这个远程追踪分支, 下次就不需要设置

```
git push -u origin master
```

②Push Rejected

原因: 本地仓库和远程仓库版本不一样

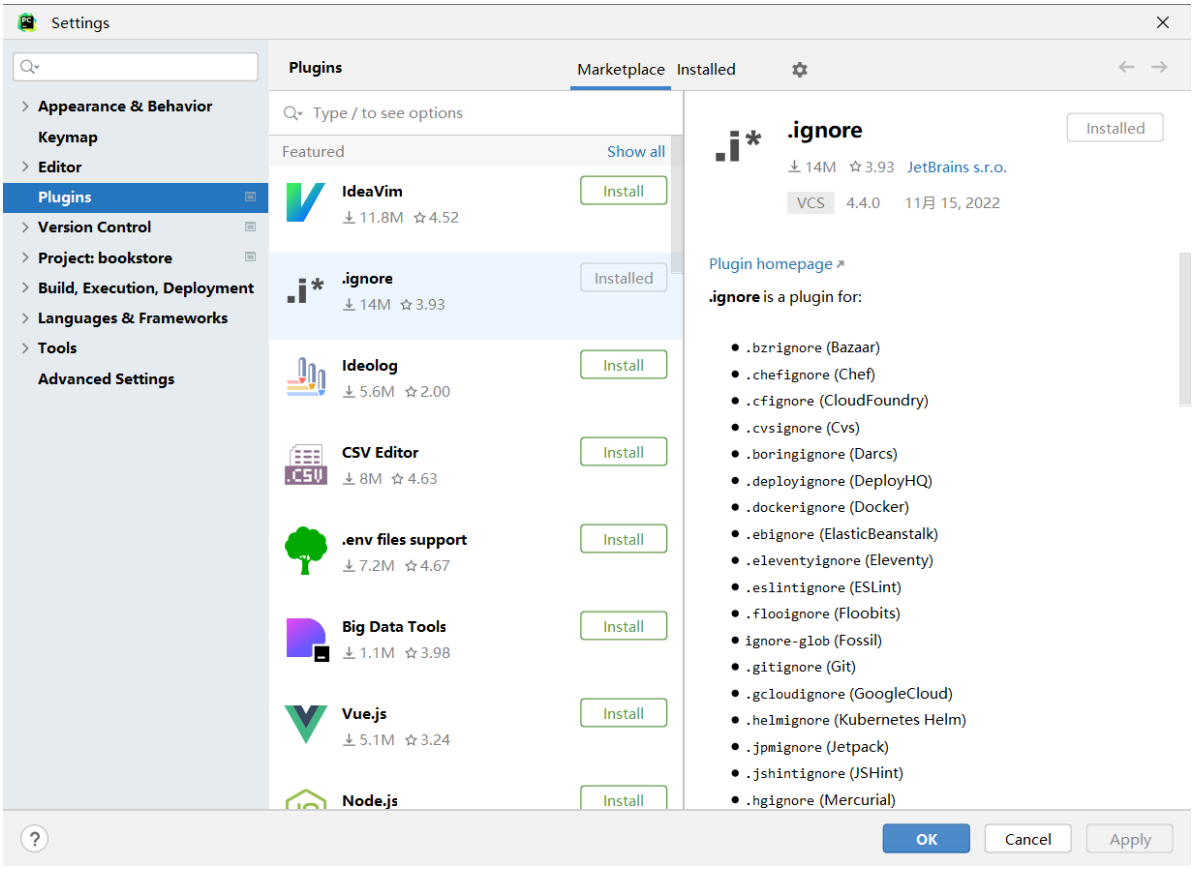
解决方案: ①先pull 更新本地仓库 ②本地创建新分支, 旧分支pull之后和新分支合并再push即可

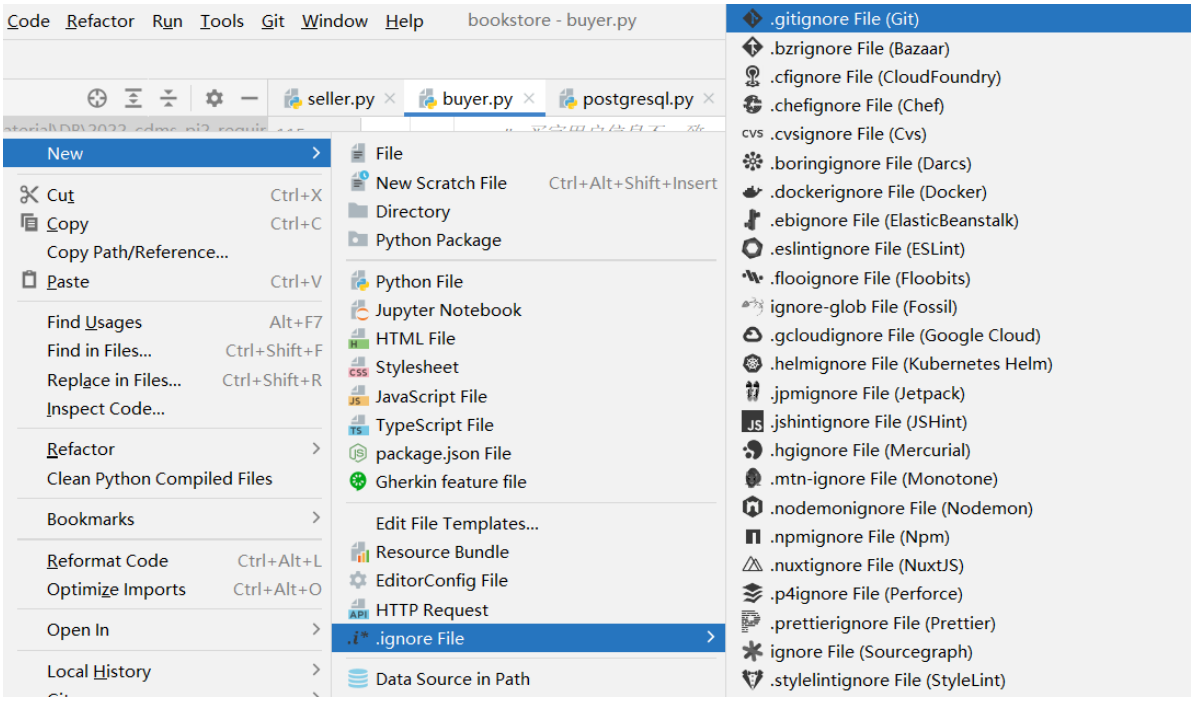


③The following untracked working tree files would be overwritten by merge

原因: 创建仓库的时候忘记加.gitignore文件, 导致协作者提交了很多idea的配置文件，然后创建者pull了之后会报错, 因为配置文件重复了

解决方法: Pycharm装插件.ignore, 然后新建一个.gitignore文件即可解决





注意: 一般情况下该报错信息的解决方案:

使用下面的命令删除没有被track的文件(没有add或commit)的

```
git clean -d -fx
```

④ ssh: connect to host github.com port 22: Connection timed out fatal: Could not read from remote repository. Please make sure you have the correct access rights and the repository exists.

ssh连接远程仓库失败

可能原因: 本地防火墙将ssh端口屏蔽了

解决方法:

1.在.ssh中加入config文件

```
Host github.com
Hostname ssh.github.com
Port 443
```

2.在git bash中输入:

```
ssh -T git@github.com
```

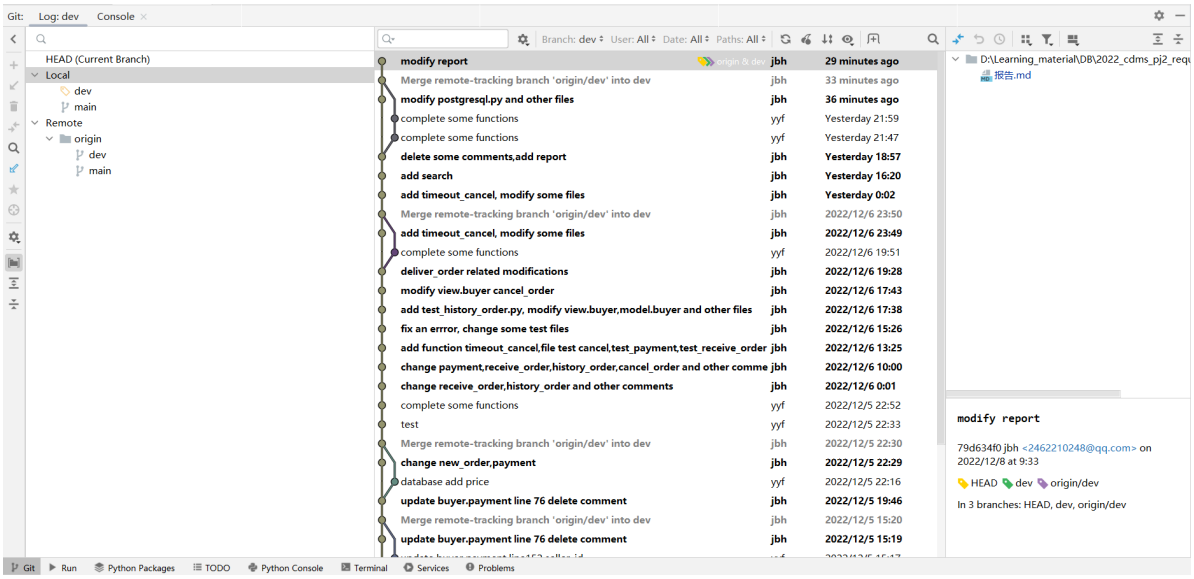
输入yes即可恢复ssh连接

⑤ 出现②的错误, 示要merge代码, merge出现conflict files无法merge

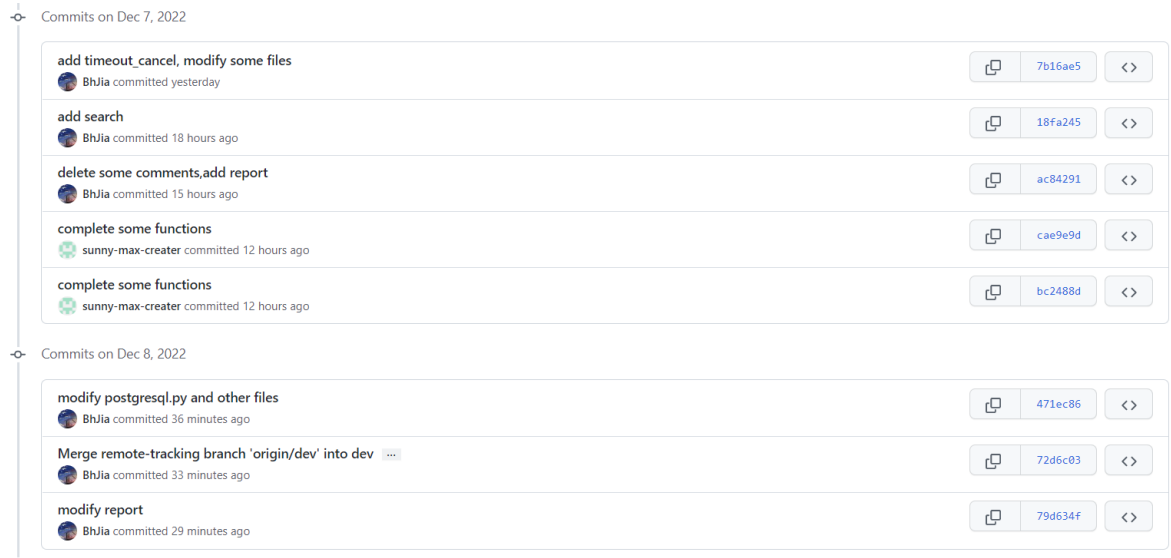
先update本地仓库保证版本一致再Push

2.版本管理

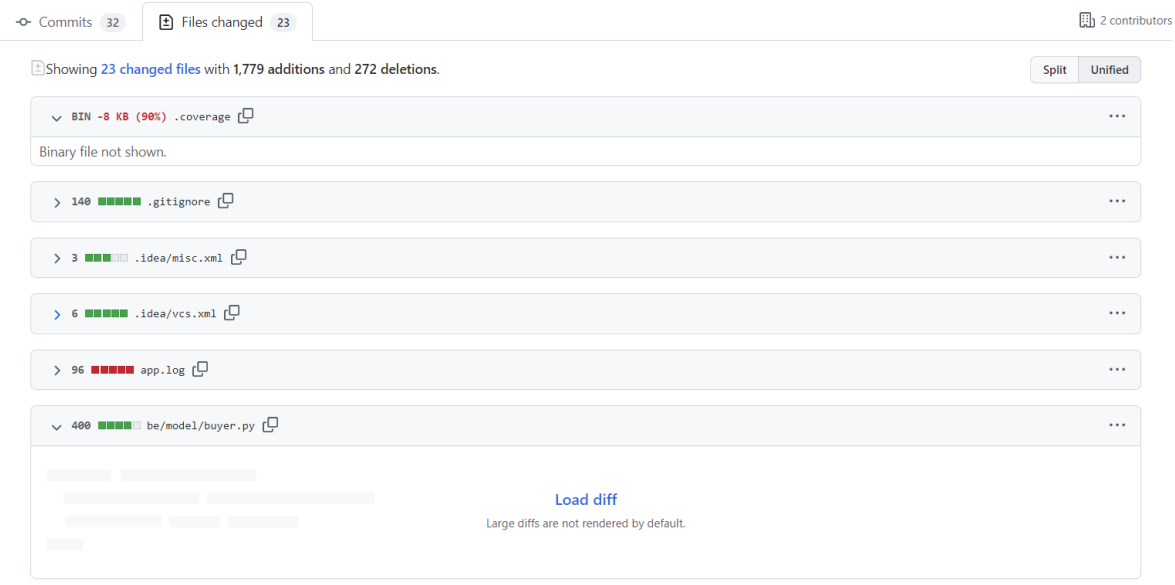
① pycharm中git可以直接看到版本的变更, 在上述协作部分已经展示, 可以进行Reset回退版本



② github上也可以看到版本变更, 并且可以回退



可以查看具体文件修改内容



```
23 be/model/error.py
@@ -1,4 +1,3 @@
-
 error_code = {
     401: "authorization fail.",
     511: "non exist user id {}",
@@ -10,8 +9,8 @@
     517: "stock level low, book id {}",
     518: "invalid order id {}",
     519: "not sufficient funds, order id {}",
-    520: "",
-    521: "",
+    520: "order already paid, order id{}",
+    521: "Cannot find books according to the search content, buyer id{}",
     522: "",
     523: "",
     524: "",
@@ -44,12 +43,12 @@ def error_exist_store_id(store_id):

 def error_non_exist_book_id(book_id):
     print(error_code[515].format(book_id))
-    return 515, error_code[515].format(book_id)
+    return 515, error_code[515].format(book_id)


 def error_exist_book_id(book_id):
     print(error_code[516].format(book_id))
-    return 516, error_code[516].format(book_id)
+    return 516, error_code[516].format(book_id)
```

七.总结

本次数据库大作业至此终于落下帷幕, 对于关系数据库的概念设计, 建立和操作, 索引的建立, ORM相关的操作等等都更加熟稔; 初次使用了github协作进行多人版本管理, 也遇到了很多的困难, 但好在有组员的积极讨论, 也感谢助教们的耐心指导,这次项目也充分展现了debug的重要性, debug对于项目的进展与维护具有不可替代的作用,debug的能力需要不断积累提升