

# 数据管理系统:Bookstore

孙秋实 郑佳辰 汤琼

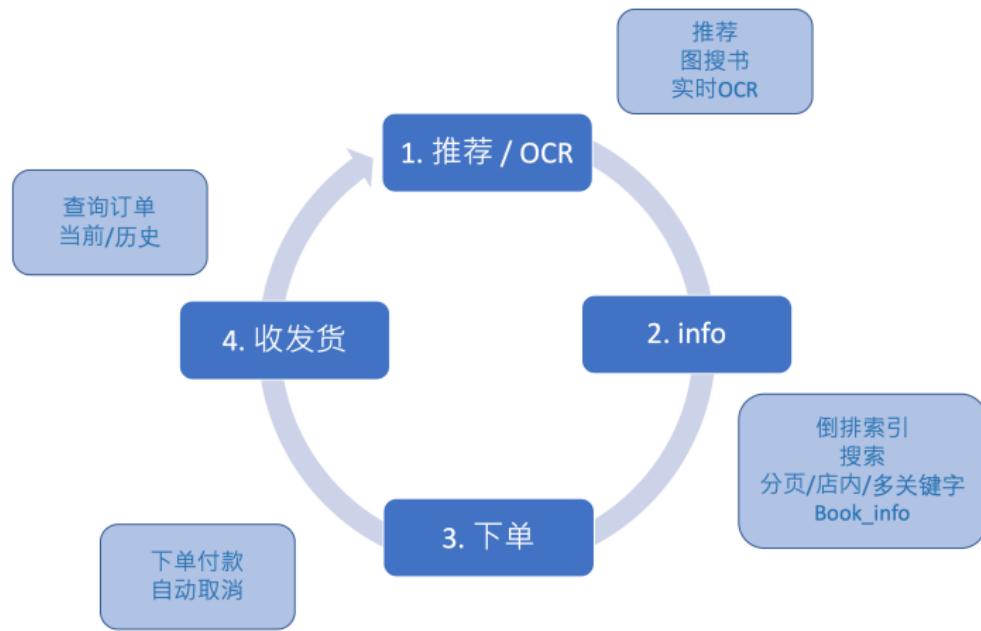
数据科学与工程学院 2018 级

January 13, 2021

- 1 数据库设计
- 2 基本功能设计与实现
- 3 进阶功能设计与实现
- 4 接口测试与性能分析
- 5 小组协作与分工
- 6 演示与提问

# 项目简介

## A Brief Introduction



## 1 数据库设计

- ER 图
- 表设计与属性
- 索引设计

## 2 基本功能设计与实现

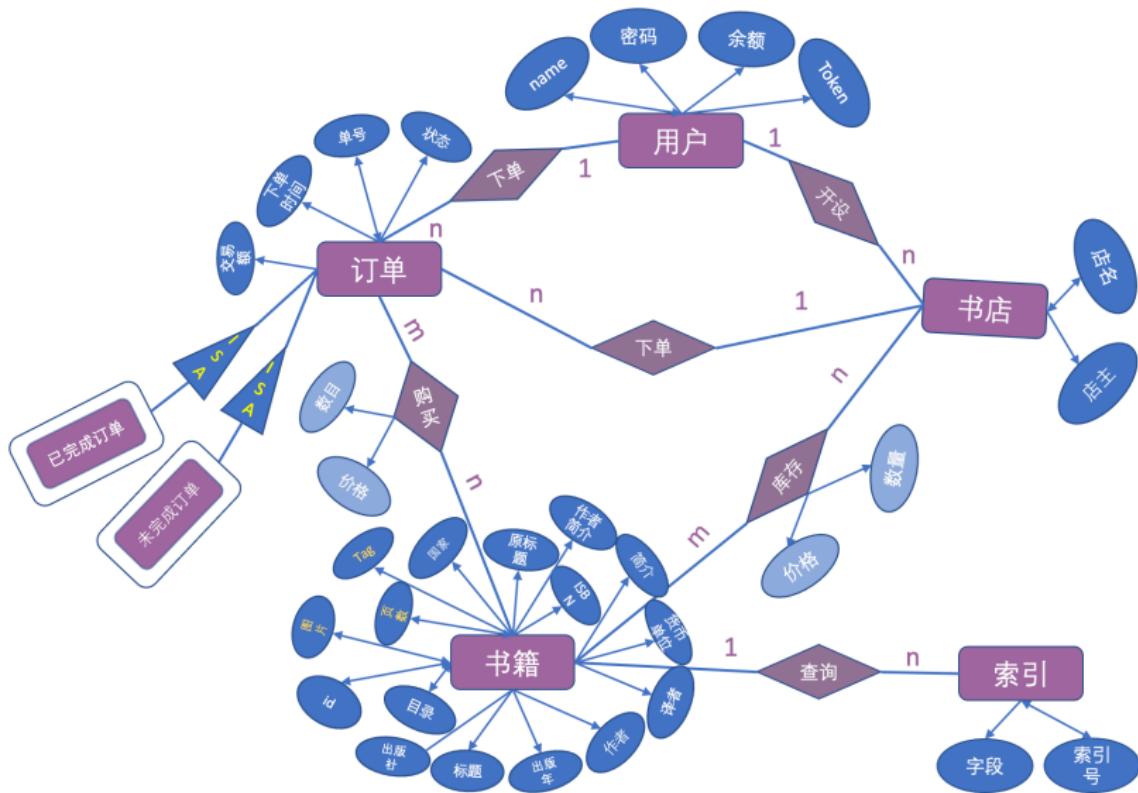
## 3 进阶功能设计与实现

## 4 接口测试与性能分析

## 5 小组协作与分工

## 6 演示与提问

# ER 图



# 重要表设计: users

The screenshot shows the OceanBase database interface with the following details:

- Left Sidebar:** Shows the database structure under 'bookstore2021'. It includes sections for 'oceanbase', 'qishi', 'bookstore2021', 'public', 'book', 'ha\_health\_check', 'invert\_index', 'new\_order', 'new\_order\_detail', 'store', 'user\_store', and 'users'. There are also 'Views', 'Materialized Views', 'Functions', 'Queries', and 'Backups' sections.
- Top Navigation:** Includes tabs for 'Objects', 'dex@public (bo...)', 'new\_order@public {book...', 'new\_order\_detail@public...', 'user\_store@public {book...', and 'users@public {bookstore...}'.
- Table View:** The 'users' table is displayed with the following columns:
  - user\_id
  - password
  - balance
  - token
- Data Preview:** The table contains approximately 100 rows of sample data, starting with:
  - test\_add\_books\_seller\_id\_b84fca6e-54eb-11eb-a449-a1999d059e5380e13e2f90cf
  - test\_add\_books\_seller\_id\_bfa763a-54eb-11eb-9eb-a-bd5d225f143945fbc25e07
  - test\_add\_books\_seller\_id\_c22f030b-54eb-11eb-b256-a08da5425021144db4f5cb7d
  - test\_add\_book\_stock\_level\_user\_c88a0e46-54eb-11e-b7d05a1f4654c6ffbb05f82f
  - test\_add\_funds\_c3057f62-54eb-11eb-8a6d-acde48001
  - test\_add\_funds\_d3475fb4-54eb-11eb-8a6d-acde48001
  - test\_add\_funds\_e3475fb4-54eb-11eb-8a6d-acde48001
  - test\_add\_funds\_f3475fb4-54eb-11eb-8a6d-acde48001
  - test\_add\_book\_stock\_level\_user\_c469a468-54eb-11e-88de289a5133972ecc444cf
  - test\_add\_book\_stock\_level\_user\_cbe1781be-54eb-11eb-b06c25060640719a85df9128
  - test\_add\_book\_stock\_level\_user\_ce1781be-54eb-11eb-b06c25060640719a85df9128
  - test\_add\_book\_stock\_level\_user\_cfaf444c-54eb-11eb-bcd7d9f23ba7016e79a6b309a
  - seller\_1\_created/b0c-54eb-11eb-8a6d-acde48001122
  - seller\_1\_created/b0c-54eb-11eb-8a6d-acde48001122
  - buyer\_1\_created/b0c-54eb-11eb-8a6d-acde48001122
  - buyer\_1\_created/b0c-54eb-11eb-8a6d-acde48001122
  - buyer\_2\_created/b0c-54eb-11eb-8a6d-acde48001122
  - buyer\_2\_created/b0c-54eb-11eb-8a6d-acde48001122
  - buyer\_4\_created/b0c-54eb-11eb-8a6d-acde48001122
  - buyer\_4\_created/b0c-54eb-11eb-8a6d-acde48001122
  - buyer\_5\_created/b0c-54eb-11eb-8a6d-acde48001122
  - buyer\_5\_created/b0c-54eb-11eb-8a6d-acde48001122
  - buyer\_7\_created/b0c-54eb-11eb-8a6d-acde48001122
  - buyer\_7\_created/b0c-54eb-11eb-8a6d-acde48001122
  - buyer\_8\_created/b0c-54eb-11eb-8a6d-acde48001122
  - buyer\_9\_created/b0c-54eb-11eb-8a6d-acde48001122
  - test\_add\_funds\_af6610b2-54ec-11eb-b127-acde48001
  - test\_add\_funds\_af6610b2-54ec-11eb-b127-acde48001
  - test\_add\_funds\_af6610b2-54ec-11eb-b5e2-acde48001
  - test\_add\_funds\_af6610b2-54ec-11eb-b5e2-acde48001
  - test\_add\_book\_stock\_level\_user\_af65ff4d-54ec-11eb-b3f05cd0fde9518d08b9d3a3
  - buyer\_10\_created/b0c-54eb-11eb-8a6d-acde48001122
  - test\_add\_books\_seller\_id\_ac3d43e-54ec-11eb-bd5d-a
  - test\_add\_books\_seller\_id\_ac87709c-54ec-11eb-9fc-a
  - test\_add\_books\_seller\_id\_ac87709c-54ec-11eb-9fc-a
  - test\_add\_books\_seller\_id\_ae6ff5fe-54ec-11eb-a0fc-ac

Figure: user-table

# 重要表设计：new-order

The screenshot shows a database management interface with a sidebar on the left and a main query window on the right.

**Left Sidebar:**

- oceanbase
- quishi
- bookstore2021
- postgres
- public
  - Tables:
    - book
    - ha\_health\_check
    - invert\_index
    - new\_order
    - new\_order\_detail
    - store
    - user\_store
    - users
  - Views
  - Materialized Views
  - Functions
  - Queries
  - Backups
- sqlgame
- QuishiSun-Sqlite2pg

**Main Window:**

Objects: new\_order@public (bookstore2021)

Table: new\_order

order_id	user_id	store_id	status	total_price	order_time
test_new_order_buyer_id_0	test_new_order_buyer_id_d87	test_new_order_store_id_d82b8afa-54ed-11eb-81e3-acde48001122	1	6066322	1610466195
test_send_books_buyer_id_1	test_send_books_buyer_id_e1	test_send_books_store_id_a44b07e-54ec-11eb-b9c3-acde48001122	1	474468	1610466237
test_send_books_buyer_id_2	test_send_books_buyer_id_f	test_send_books_store_id_fb9eeb8-54ec-11eb-bed2-acde48001122	3	1939639	1610466242
test_send_books_buyer_id_3	test_send_books_buyer_id_0	test_send_books_store_id_0e6e7298-54ed-11eb-beaf-acde48001122	1	8701629	1610466301
test_send_books_buyer_id_4	test_send_books_buyer_id_2	test_send_books_store_id_220e6f3a-54ed-11eb-9d52-acde48001122	1	1142700	1610466307
test_send_books_buyer_id_5	test_send_books_buyer_id_3	test_send_books_store_id_31b6769a-54ed-11eb-8078-acde48001122	1	4431931	1610466355
test_send_books_buyer_id_6	test_send_books_buyer_id_4	test_send_books_store_id_424bc5e6-54ed-11eb-9550-acde48001122	1	1012470	1610466359
test_send_books_buyer_id_7	test_send_books_buyer_id_4	test_send_books_store_id_44a3d6e4-54ed-11eb-b467-acde48001122	1	459141	1610466362

Figure: new-order

# 重要表设计：inverted-index

The screenshot shows a database interface with a sidebar on the left and a main table view on the right.

**Left Sidebar:**

- oceanbase
- qulish
- bookstore2021
- postgres
- public
  - book
  - ha\_health\_check
  - invert\_index
  - new\_order
  - new\_order\_detail
  - store
  - user\_store
  - users
- Views
- Materialized Views
- Functions
- Queries
- Backups
- sqlgame
- QulishSun-Sqlite2pg

**Main View:**

Table: search\_key

	search_key	search_id	book_id	book_title	book_author
1	A Beautiful Mind : Genius, Schi	1	1000067	美丽心灵	西尔维娅·普萨
2	A Beautiful Mind : Genius, Schizophrenia and Recovery	2	1000067	美丽心灵	西尔维娅·普萨
3	西	3	1000067	美丽心灵	西尔维娅·普萨
4	A Beautiful Mind : Genius, Schizophrenia and Recovery in the Life of a	4	1000067	美丽心灵	西尔维娅·普萨
5	A Beautiful Mind : Genius, Schiz	5	1000067	美丽心灵	西尔维娅·普萨
6	A Beautiful Mind : Genius, Sch	6	1000067	美丽心灵	西尔维娅·普萨
7	A Beautiful Mind :	7	1000067	美丽心灵	西尔维娅·普萨
8	A Beautiful Mind : Genius, Schizophrenia and	8	1000067	美丽心灵	西尔维娅·普萨
9	A Beautiful Mind	9	1000067	美丽心灵	西尔维娅·普萨
10	西尔维娅·普	10	1000067	美丽心灵	西尔维娅·普萨
11	A Beautiful Mind : Genius, Schizophrenia and Recovery in the Life of	11	1000067	美丽心灵	西尔维娅·普萨
12	A Beautiful Mind : Genius, Schizoph	12	1000067	美丽心灵	西尔维娅·普萨
13	A Beautiful Mind : Genius, Schizophrenia	13	1000067	美丽心灵	西尔维娅·普萨
14	A Beautif	14	1000067	美丽心灵	西尔维娅·普萨
15	A Beautiful Mind : Genius, Schizophrenia and Recovery in the Li	15	1000067	美丽心灵	西尔维娅·普萨
16	A Beautiful Mind : Genius, Schizophrenia and R	16	1000067	美丽心灵	西尔维娅·普萨
17	A Beauti	17	1000067	美丽心灵	西尔维娅·普萨
18	A Beautiful Mind : Genius, Schizophrenia and Rec	18	1000067	美丽心灵	西尔维娅·普萨
19	A Beautiful Mind : Genius, Schizophrenia and Recovery in the Lif	19	1000067	美丽心灵	西尔维娅·普萨
20	心	20	1000067	美丽心灵	西尔维娅·普萨
21	A Beautiful Mind : Genius, Schizophrenia and Recovery in the Life o	21	1000067	美丽心灵	西尔维娅·普萨
22	美	22	1000067	美丽心灵	西尔维娅·普萨
23	A Beautiful Mind : Genius, Schizophrenia and Recovery in the Life of a Nobel Lau	23	1000067	美丽心灵	西尔维娅·普萨
24	A Beautiful Mind : Genius, Sc	24	1000067	美丽心灵	西尔维娅·普萨
25	A Beautiful Mind : Genius, Schizophrenia and Recovery in the Life of a Nobel La	25	1000067	美丽心灵	西尔维娅·普萨
26	西尔维	26	1000067	美丽心灵	西尔维娅·普萨
27	A Beautiful Mind :	27	1000067	美丽心灵	西尔维娅·普萨
28	A	28	1000067	美丽心灵	西尔维娅·普萨
29	A Beautiful Mind : Genius, Schizophrenia and Re	29	1000067	美丽心灵	西尔维娅·普萨

Figure: 倒排索引

# 索引设计

## Mongodb-book-index (B-tree)

Book-index 的作用：加速查询与推荐

The screenshot shows the MongoDB Compass interface connected to the 'bookstore' database on 'localhost:27017'. The left sidebar lists collections: admin, bookstore, book, history\_order, config, game, local, and test. The 'book' collection is selected. The main window displays the 'bookstore.book' collection with 100 documents, totaling 16.0MB with an average size of 163.6KB. There are 5 indexes, totaling 136.0KB with an average size of 27.2KB. The 'Indexes' tab is active, showing the following index definitions:

Name and Definition	Type	Size	Usage	Properties	Drop
\$_id	REGULAR	20.5 KB	0 since Tue Jan 12 2021	UNIQUE	
auth	REGULAR	20.5 KB	157 since Tue Jan 12 2021		
book	REGULAR	20.5 KB	6174 since Tue Jan 12 2021	UNIQUE	
pub	REGULAR	20.5 KB	157 since Tue Jan 12 2021		
tag	REGULAR	573 KB	157 since Tue Jan 12 2021		

# 索引设计 Cont'd

## Mongodb-history-index (B-tree)

history-index 的作用：加速历史信息查询（买家/卖家）

The screenshot shows the MongoDB Compass interface for the `bookstore.history_order` collection. The left sidebar lists databases and collections, with `history_order` selected. The main pane displays the collection details: 135 documents, 325.7KB total size, and 2.4KB average size. It also shows 3 indexes, totaling 124.0KB and an average size of 41.3KB. The `Indexes` tab is active, showing three regular indexes:

Name and Definition	Type	Size	Usage	Properties	Drop
<code>_id_</code>	REGULAR	36.9 KB	0 since Tue Jan 12 2021	UNIQUE	
<code>stores</code>	REGULAR	45.1 KB	12 since Tue Jan 12 2021		
<code>users</code>	REGULAR	45.1 KB	24 since Tue Jan 12 2021		

## 1 数据库设计

## 2 基本功能设计与实现

- 基本功能实现：用户权限功能
- 基本功能实现：卖家功能
- 基本功能实现：买家功能

## 3 进阶功能设计与实现

## 4 接口测试与性能分析

## 5 小组协作与分工

## 6 演示与提问

# 用户权限

## 功能实现：注册

- (1) 根据 user\_id 在 users 表中查询用户是否已经存在
- (2) 若不存在，则插入新用户 (user\_id, password, balance, token, terminal) 到 users 表中

**性能:** 一次查询 users table，一次插入 users table，访问数据库两次

## 功能实现：登陆

- (1) 根据 user\_id 在 users 表中获取密码 password
- (2) 将获取到的 password 与用户输入的密码对比
- (3) 更新用户的 token, terminal

**性能:** 一次查询 users table, 一次更新 users table，访问数据库两次

# 用户权限 Cont'd

## 功能实现：登出

- (1) 根据 user\_id 在 users table 中查询，判断登录信息是否失效
- (2) 更新用户 token

**性能:** 一次查询 users table, 一次插入 users table, 访问数据库两次

## 功能实现：注销

- (1) 根据 user\_id 在 users table 中查询该用户是否存在
- (2) 删除 users table 中该用户条目

**性能:** 一次查询 users table, 一次更新 users table, 访问数据库两次

## 功能实现：更改密码

- (1) 根据 user\_id 在 users table 中查询用户原先密码
- (2) 判断用户原先密码和用户新密码是否相同
- (3) 若不同，则更新 users table 中该用户的 password

性能：一次查询 users table，一次插入 users table，访问数据库两次

# 卖家功能

## 功能实现：创建店铺

- (1) 分别在 users table 和 store table 中查询 users\_id 和 store\_id 是否已经存在
- (2) 若不存在，插入用户 user\_id 和 store\_id 到 user\_store table
- (3) 若不同，则更新 users table 中该用户的 password

**性能:** 一次查询 users table, 一次查询 store table, 一次插入 user\_store table, 访问数据库三次

## 功能实现：添加库存

- (1) 检查 user\_id, store\_id, book\_id 是否存在
- (2) 根据 store\_id, book\_id 寻找该店家某本书的库存，并在 store table 中更新

**性能:** 一次查询 users\_table, 一次查询 user\_store table, 一次查询 store table, 一次更新 store table, 访问数据库四次

## 功能实现：上架图书

- (1) 检查 user\_id, store\_id, book\_id 是否存在
- (2) 根据 book\_id 在 Mongodb 的 book collection 中查询 book 是否存在
- (3) 根据 book\_json\_str 分离出作者名，作者国籍，书籍关键词，书的标题，书的作者等，插入倒排表 invert\_index table
- (4) (store\_id, book\_id, stock\_level, price) 插入 store table

**性能：**一次查询 users table, 一次查询 store table, 一次插入 user\_store table, 访问数据库三次

## 功能实现：卖家发货

- (1) 检查 store\_id,book\_id 是否存在
- (2) 根据 order\_id 在 new\_order table 中更新订单状态

**性能:** 一次查询 user\_store table, 一次查询 store table , 一次更新 new\_order table , 访问数据库三次

## 功能实现：充值

- (1) 根据 user\_id 获取用户密码
- (2) 校验用户密码与用户输入的密码做对比
- (3) 若密码一致，则更新该用户在 users table 中的余额

性能：一次查询 users table，一次更新 users table，访问数据库两次

## 功能实现：下单

- (1) 检查 user\_id, store\_id 是否存在
- (2) 根据订单信息 (store\_id, book\_id) 在 store 表中查找商户中是否存在对应书籍和足够的库存。
- (3) 若库存足够，则更新 store table 库存
- (4) 创建新订单信息，将 (order\_id, book\_id, count, price) 插入 new\_order\_detail table
- (5) 创建该笔订单，计算该笔订单总价 total\_price，将 (order\_id, store\_id, user\_id, total\_price, order\_time) 插入 new\_order table，同时将订单号 order\_id 添加到 unpaid\_order 数组

**性能:** 一次查询 users table, 一次查询 user\_store table, 一次查询 store table, 一次更新 store table, 一次插入 new\_order\_detail table, 一次插入 new\_order table, 访问数据库六次

## 功能实现：付款

- (1) 根据 order\_id 在 new\_order table 中查询订单信息
- (2) 查询订单是否超时
- (3) 若订单未超时，则根据 buyer\_id 在 users table 中获取买家余额和密码
- (4) 根据 store\_id 在 user\_store table 中查询卖家 seller\_id
- (5) 在 users table 中更新买家的余额
- (6) 在 new\_order table 中更新订单状态 status=2

**性能:** 一次查询 new\_order table, 一次查询 users table, 一次查询 user\_store table, 一次更新 users , 一次更新 new\_order table, 访问数据库五次

## 功能实现：收货

- (1) 根据 order\_id 在 new\_order table 中查询对应的订单状态，买家 id
- (2) 检查订单状态是否为已发货，订单 id 与买家 id 是否对应
- (3) 若符合条件，则更新卖家余额，更新订单状态为已收货并且加入历史记录

**性能:** 一次查询 new\_order table，一次查询 user\_store table，两次更新 users table，一次查询 users table，访问数据库五次

## 功能实现：取消订单

- (1) 根据 order\_id 在 new\_order table 中查询订单状态是否为已下单
  - (2) 检查用户是否存在，订单号是否存在
  - (3) 从全局 unpaid\_orders 字典中删除该订单
  - (4) 根据 order\_id 从 new\_order table 中删除该订单，从 new\_order\_detail table 中删除订单信息
  - (5) 根据 store\_id 和 book\_id 回滚库存 stock\_level
  - (6) 将订单信息加入 MongoDB 的历史记录 history\_order collection 中
- 性能:** 一次查询 new\_order table, 一次查询 user\_store table, 两次更新 users table, 一次查询 users table, 访问数据库五次

## 1 数据库设计

## 2 基本功能设计与实现

## 3 进阶功能设计与实现

- 自动取消订单
- OCR 搜书
- 迷你推荐系统

## 4 接口测试与性能分析

## 5 小组协作与分工

## 6 演示与提问

# 自动订单取消

```
from be.model.order import Order
import datetime
import time

def get_time_stamp(): #get timestamp for auto cancelling orders that are not paid in time
    cur_time_stamp = time.time()
    return int(cur_time_stamp)

time_limit = 30 # 订单存活时间
unpaid_orders = {}

#优点：通过维护全局数组to_be_paid，没有额外新启线程，代价降到最低
def add_unpaid_order(orderID):
    unpaid_orders[orderID] = get_time_stamp()
    print("add successfully")
    print(unpaid_orders)
    return 200, "ok"

def delete_unpaid_order(orderID):
    try:
        unpaid_orders.pop(orderID)
        print(unpaid_orders)
    except BaseException as e:
        return 530, "{}".format(str(e))
    return 200, "ok"
```

# 自动订单取消 Cont'd

```
def check_order_time(order_time):
    cur_time = get_time_stamp()
    time_diff = cur_time - order_time
    if time_diff > time_limit:
        return False
    else:
        return True

def time_exceed_delete():
    del_temp=[]
    o = Order()
    print("new cycle start")
    for (oid,tim) in unpaid_orders.items():
        if check_order_time(tim) == False:
            del_temp.append(oid) # remember, not to append the index of the array, we need the orderID
    for oid in del_temp:
        delete_unpaid_order(oid)
        o.cancel_order(oid)
    return @
```

## 自动订单取消 Cont'd

Figure: 配置定时任务

## 以图搜图的一种实现方法

- 为什么不用哈希方法进行图像搜索？
- 开销太大了！

## 以图搜图的一种实现方法

- 为什么不用哈希方法进行图像搜索？
- 开销太大了！

# OCR 测试

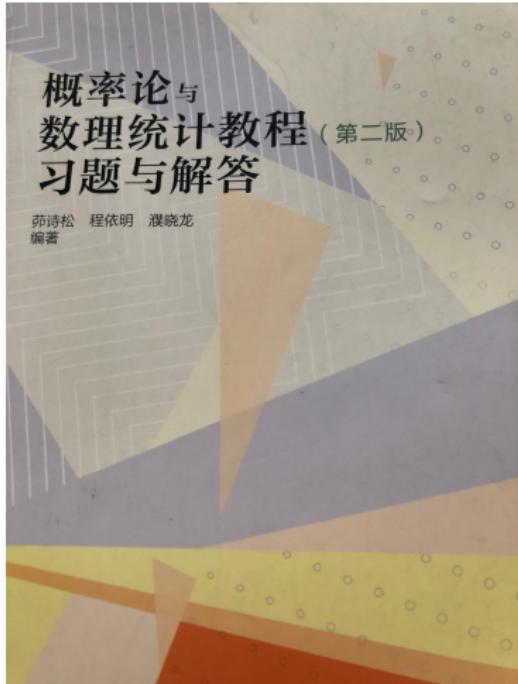


Figure: 书本封面示例

POST http://localhost:5000/buyer/upload

Params Authorization Headers (9) Body  Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL

KEY VALUE

png math.png X

Key Value

Body Cookies Headers (4) Test Results

Pretty Raw Preview Visualize JSON

```
1 {
2   "message": "ok",
3   "result": [
4     200,
5     "ok",
6     [
7       {
8         "author": "茆诗松,程依明,潘晓龙",
9         "bid": "g11",
10        "title": "概率论与数理统计教程习题与解答"
11      }
12    ]
13  ]
14 }
```

Figure: Json 形式的 OCR 结果

# OCR 测试 Cont'd

```
[‘概率论’, ‘解答’, ‘数理统计’, ‘习题’, ‘教程’]  
概率论  
[{"bid": "gll", "title": "概率论与数理统计教程习题与解答", "author": "茆诗松,程依明,濮晓龙"}, {"bid": "016855", "title": "概率论与数理统计教程", "author": "茆诗松,程依明,濮晓龙"}]  
解答  
[{"bid": "gll", "title": "概率论与数理统计教程习题与解答", "author": "茆诗松,程依明,濮晓龙"}]  
数理统计  
[{"bid": "gll", "title": "概率论与数理统计教程习题与解答", "author": "茆诗松,程依明,濮晓龙"}, {"bid": "016855", "title": "概率论与数理统计教程", "author": "茆诗松,程依明,濮晓龙"}]  
习题  
[{"bid": "gll", "title": "概率论与数理统计教程习题与解答", "author": "茆诗松,程依明,濮晓龙"}]  
教程  
[{"bid": "gll", "title": "概率论与数理统计教程习题与解答", "author": "茆诗松,程依明,濮晓龙"}, {"bid": "016855", "title": "概率论与数理统计教程", "author": "茆诗松,程依明,濮晓龙"}]
```

Figure: 通过 TextRank 提取词组

# OCR 测试 Cont'd

Untitled Request

POST http://localhost:5000/buyer/upload

Send Save

Params Authorization Headers (9) Body Pre-request Script Tests Settings Cookies Co

Body Authorization Headers (4) Test Results

Status: 200 OK Time: 7.90 s Size: 524 B Save Response

KEY VALUE DESCRIPTION \*\*\* Bulk Edit

png math.png X

Pretty Raw Preview Visualize JSON

```
1 {
2     "message": "ok",
3     "result": [
4         200,
5         "ok",
6         [
7             {
8                 "author": "茆诗松,程依明,濮晓龙",
9                 "bid": "g11",
10                "title": "概率论与数理统计教程习题与解答"
11            },
12            {
13                "author": "茆诗松,程依明,濮晓龙",
14                "bid": "01685",
15                "title": "概率论与数理统计教程"
16            }
17        ]
18    ]
}
```

Figure: OCR 搜书结果

# 迷你推荐系统

```
def recommend(self, user_id):
    try:
        code, message, orders = self.history_order(user_id)
        if code != 200:
            return error.error_non_exist_user_id(user_id)

        reco = {}
        labels = []
        for order in orders:
            boughtbooks = order['books']
            for boughtbook in boughtbooks:
                eachbook = self.mongo['book'].find_one({'id': boughtbook['book_id']},
                                                       {'_id': 0, 'id': 1, 'title': 1, 'author': 1, 'tags': 1, 'publisher': 1})
                labels += eachbook['tags']
                reco[eachbook['id']] = eachbook
                reco[eachbook['id']]['tags'] = []
                books = self.mongo['book'].find({'$or': [{'author': eachbook['author']},
                                                          {'publisher': eachbook['publisher']},
                                                          {'tags': {'$elemMatch': {'$in': eachbook['tags']}}}]},
                                                 {'_id': 0, 'id': 1, 'title': 1, 'author': 1, 'tags': 1})
                for book in books:
                    if book['id'] not in reco.keys():
                        reco[book['id']] = book
    print(len(reco))
    labels = list(set(labels))
    print(labels)
```

Figure: 小型推荐系统的实现

我们基于用户搜索/购买书本的历史，实现了一个小型的推荐系统

## 1 数据库设计

## 2 基本功能设计与实现

## 3 进阶功能设计与实现

## 4 接口测试与性能分析

- 功能接口测试
- 代码覆盖率
- 性能分析与提升

## 5 小组协作与分工

## 6 演示与提问

# 接口测试

## 接口测试详情

- (1) 35 个基础测试全部通过
- (2) 为进阶功能编写了 37 个测试

```
fe/test/test_order.py::TestSendBooks::test_processing_order_sent PASSED [ 62%]
fe/test/test_order.py::TestSendBooks::test_processing_order_receive PASSED [ 63%]
fe/test/test_order.py::TestSendBooks::test_history_order PASSED [ 65%]
fe/test/test_order.py::TestSendBooks::test_history_order_send PASSED [ 66%]
fe/test/test_order.py::TestSendBooks::test_history_order_receive PASSED [ 68%]
fe/test/test_order.py::TestSendBooks::test_recommend_empty PASSED [ 69%]
fe/test/test_order.py::TestSendBooks::test_recommend_order PASSED [ 78%]
fe/test/test_order.py::TestSendBooks::test_seller_processing_order_ok PASSED [ 72%]
fe/test/test_order.py::TestSendBooks::test_seller_processing_order_sent PASSED [ 73%]
fe/test/test_order.py::TestSendBooks::test_seller_processing_order_receive PASSED [ 75%]
fe/test/test_order.py::TestSendBooks::test_seller_history_order PASSED [ 76%]
fe/test/test_order.py::TestSendBooks::test_seller_history_order_sent PASSED [ 77%]
fe/test/test_order.py::TestSendBooks::test_seller_history_order_receive PASSED [ 79%]
fe/test/test_order.py::TestSendBooks::test_get_book_info_ok PASSED [ 88%]
fe/test/test_order.py::TestSendBooks::test_get_book_info_error PASSED [ 81%]
fe/test/test_order.py::TestSendBooks::test_search_in_store_ok PASSED [ 83%]
fe/test/test_order.py::TestSendBooks::test_search_in_store_page_ok PASSED [ 84%]
fe/test/test_password.py::TestPassword::test_ok PASSED [ 86%]
fe/test/test_password.py::TestPassword::test_error_password PASSED [ 87%]
fe/test/test_password.py::TestPassword::test_error_user_id PASSED [ 88%]
fe/test/test_payment.py::TestPayment::test_order PASSED [ 98%]
fe/test/test_payment.py::TestPayment::test_authorization_error PASSED [ 91%]
fe/test/test_payment.py::TestPayment::test_not_enough_funds PASSED [ 93%]
fe/test/test_payment.py::TestPayment::test_repeat_pay PASSED [ 96%]
fe/test/test_register.py::TestRegister::test_register_ok PASSED [ 95%]
fe/test/test_register.py::TestRegister::test_unregister_ok PASSED [ 97%]
fe/test/test_register.py::TestRegister::test_unregister_error_authorization PASSED [ 98%]
fe/test/test_register.py::TestRegister::test_unregister_error_exist_user_id PASSED [ 100%]

=====
===== 72 passed in 186.45s (0:03:06) =====
```

# 接口测试

## 接口测试详情

- (1) 35 个基础测试全部通过
- (2) 为进阶功能编写了 37 个测试

```
fe/test/test_order.py::TestSendBooks::test_processing_order_sent PASSED [ 62%]
fe/test/test_order.py::TestSendBooks::test_processing_order_receive PASSED [ 63%]
fe/test/test_order.py::TestSendBooks::test_history_order PASSED [ 65%]
fe/test/test_order.py::TestSendBooks::test_history_order_send PASSED [ 66%]
fe/test/test_order.py::TestSendBooks::test_history_order_receive PASSED [ 68%]
fe/test/test_order.py::TestSendBooks::test_recommend_empty PASSED [ 69%]
fe/test/test_order.py::TestSendBooks::test_recommend_order PASSED [ 70%]
fe/test/test_order.py::TestSendBooks::test_seller_processing_order_ok PASSED [ 72%]
fe/test/test_order.py::TestSendBooks::test_seller_processing_order_sent PASSED [ 73%]
fe/test/test_order.py::TestSendBooks::test_seller_processing_order_receive PASSED [ 75%]
fe/test/test_order.py::TestSendBooks::test_seller_history_order PASSED [ 76%]
fe/test/test_order.py::TestSendBooks::test_seller_history_order_sent PASSED [ 77%]
fe/test/test_order.py::TestSendBooks::test_seller_history_order_receive PASSED [ 79%]
fe/test/test_order.py::TestSendBooks::test_get_book_info_ok PASSED [ 80%]
fe/test/test_order.py::TestSendBooks::test_get_book_info_error PASSED [ 81%]
fe/test/test_order.py::TestSendBooks::test_search_in_store_ok PASSED [ 83%]
fe/test/test_order.py::TestSendBooks::test_search_in_store_page_ok PASSED [ 84%]
fe/test/test_password.py::TestPassword::test_ok PASSED [ 86%]
fe/test/test_password.py::TestPassword::test_error_password PASSED [ 87%]
fe/test/test_password.py::TestPassword::test_error_user_id PASSED [ 88%]
fe/test/test_payment.py::TestPayment::test_order PASSED [ 90%]
fe/test/test_payment.py::TestPayment::test_authorization_error PASSED [ 91%]
fe/test/test_payment.py::TestPayment::test_not_enough_funds PASSED [ 93%]
fe/test/test_payment.py::TestPayment::test_repeat_pay PASSED [ 94%]
fe/test/test_register.py::TestRegister::test_register_ok PASSED [ 95%]
fe/test/test_register.py::TestRegister::test_unregister_ok PASSED [ 97%]
fe/test/test_register.py::TestRegister::test_unregister_error_authorization PASSED [ 98%]
fe/test/test_register.py::TestRegister::test_unregister_error_exist_user_id PASSED [ 100%]

=====
===== 72 passed in 186.45s (0:03:06) =====
```

# 覆盖率测试

Name	Stmts	Miss	Branch	BrPart	Cover
<hr/>					
be\__init__.py	0	0	0	0	100%
be\app.py	4	4	2	0	0%
be\model\__init__.py	0	0	0	0	100%
be\model\buyer.py	227	52	92	18	73%
be\model\db_conn.py	29	0	8	0	100%
be\model\error.py	25	2	0	0	92%
be\model\nlp.py	75	3	32	3	94%
be\model\ocr.py	0	0	0	0	100%
be\model\order.py	34	8	10	2	73%
be\model\seller.py	153	29	72	11	76%
be\model\store.py	41	3	0	0	93%
be\model\times.py	24	4	0	0	83%
be\model\user.py	200	39	72	10	77%
be\serve.py	43	6	4	2	83%
be\tasks.py	2	0	0	0	100%
be\view\__init__.py	0	0	0	0	100%
be\view\auth.py	60	0	0	0	100%
be\view\buyer.py	76	0	2	0	100%
be\view\seller.py	50	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	46	0	0	0	100%
fe\access\book.py	70	1	12	2	96%
fe\access\buyer.py	77	0	2	0	100%
fe\access\new_buyer.py	14	0	0	0	100%

# 覆盖率测试 Cont'd

fe\access\new_seller.py	8	0	0	0	100%
fe\access\seller.py	47	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\gen_book_data.py	50	0	16	0	100%
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_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_order.py	240	0	0	0	100%
fe\test\test_password.py	33	0	0	0	100%
fe\test\test_payment.py	60	1	4	1	97%
fe\test\test_register.py	31	0	0	0	100%
<hr/>					
TOTAL	2124	155	388	52	90%

Figure: 覆盖率:90%

# 性能分析

我们对实现的各功能接口做了吞吐量测试

## 性能

峰值吞吐量：44798

```
1962 INFO:root:TPS_C=43893, NO=UN:4/z2/o Thread_num:970 TOTAL:4/z2/o LATENCY:0.013895/4440/33032 , P=UN:4/z2/o Thread_num:970 TOTAL:4/z2/o LATENCY:0.0884694/2821361713
1963 INFO:root:TPS_C=43788, NO=OK:473253 Thread_num:977 TOTAL:473253 LATENCY:0.013895/9608/261572 , P=OK:471699 Thread_num:976 TOTAL:472276 LATENCY:0.0884694/2821361713
1964 INFO:root:TPS_C=43748, NO=OK:474231 Thread_num:978 TOTAL:474231 LATENCY:0.013895/182758/260882 , P=OK:472468 Thread_num:977 TOTAL:473253 LATENCY:0.088468668857187865
1965 INFO:root:TPS_C=43796, NO=OK:475210 Thread_num:979 TOTAL:475210 LATENCY:0.013894/401818/1561187 , P=OK:473422 Thread_num:978 TOTAL:474231 LATENCY:0.08846798356674288
1966 INFO:root:TPS_C=43844, NO=OK:476198 Thread_num:980 TOTAL:476198 LATENCY:0.013893/6138089/818183 , P=OK:473855 Thread_num:979 TOTAL:475218 LATENCY:0.088467484808973857
1967 INFO:root:TPS_C=43891, NO=OK:477171 Thread_num:981 TOTAL:477171 LATENCY:0.013892/829252173599 , P=OK:475549 Thread_num:980 TOTAL:476198 LATENCY:0.088466399562371623
1968 INFO:root:TPS_C=43939, NO=OK:478153 Thread_num:982 TOTAL:478153 LATENCY:0.013892/85024694182 , P=OK:476314 Thread_num:981 TOTAL:477171 LATENCY:0.0884655454914645848
1969 INFO:root:TPS_C=43987, NO=OK:479136 Thread_num:983 TOTAL:479136 LATENCY:0.013891/274667566749 , P=OK:477288 Thread_num:982 TOTAL:478153 LATENCY:0.0884649144312161982
1970 INFO:root:TPS_C=44834, NO=OK:480120 Thread_num:984 TOTAL:480120 LATENCY:0.013898/502558/224879 , P=OK:478247 Thread_num:983 TOTAL:479136 LATENCY:0.0884642178108368867
1971 INFO:root:TPS_C=44882, NO=OK:481185 Thread_num:985 TOTAL:481185 LATENCY:0.013887/27679348725 , P=OK:479215 Thread_num:984 TOTAL:480128 LATENCY:0.08846344584337346
1972 INFO:root:TPS_C=44138, NO=OK:482891 Thread_num:986 TOTAL:482891 LATENCY:0.013888/945934191183 , P=OK:480184 Thread_num:985 TOTAL:481185 LATENCY:0.08846271555474995
1973 INFO:root:TPS_C=44178, NO=OK:483878 Thread_num:987 TOTAL:483878 LATENCY:0.013888/17278249596 , P=OK:481153 Thread_num:986 TOTAL:482891 LATENCY:0.08846198318171219
1974 INFO:root:TPS_C=44225, NO=OK:484866 Thread_num:988 TOTAL:484866 LATENCY:0.013887/3968618777477 , P=OK:482123 Thread_num:987 TOTAL:483878 LATENCY:0.088461258745725865
1975 INFO:root:TPS_C=44273, NO=OK:485855 Thread_num:989 TOTAL:485855 LATENCY:0.013886/625443745896 , P=OK:483993 Thread_num:988 TOTAL:484866 LATENCY:0.08846518189191484
1976 INFO:root:TPS_C=44321, NO=OK:486845 Thread_num:990 TOTAL:486845 LATENCY:0.013885/845186567707 , P=OK:484064 Thread_num:989 TOTAL:485855 LATENCY:0.088459785551296936
1977 INFO:root:TPS_C=44368, NO=OK:487836 Thread_num:991 TOTAL:487836 LATENCY:0.013885/86612355756 , P=OK:485836 Thread_num:990 TOTAL:486045 LATENCY:0.088459852897395828
1978 INFO:root:TPS_C=44416, NO=OK:488828 Thread_num:992 TOTAL:488828 LATENCY:0.013888/284445843733 , P=OK:486088 Thread_num:991 TOTAL:487836 LATENCY:0.08845832823567294
1979 INFO:root:TPS_C=44464, NO=OK:489821 Thread_num:993 TOTAL:489821 LATENCY:0.013883/51425643499 , P=OK:486988 Thread_num:992 TOTAL:488828 LATENCY:0.08845758756195296
1980 INFO:root:TPS_C=44512, NO=OK:490815 Thread_num:994 TOTAL:490815 LATENCY:0.013882/763811531463 , P=OK:487953 Thread_num:993 TOTAL:489821 LATENCY:0.0884568589966384145
1981 INFO:root:TPS_C=44559, NO=OK:491810 Thread_num:995 TOTAL:491810 LATENCY:0.013882/16648595749 , P=OK:488926 Thread_num:994 TOTAL:490815 LATENCY:0.088456138335418859
1982 INFO:root:TPS_C=44687, NO=OK:492866 Thread_num:996 TOTAL:492866 LATENCY:0.013881/274785104739 , P=OK:489899 Thread_num:995 TOTAL:491818 LATENCY:0.088455481691423482
1983 INFO:root:TPS_C=44655, NO=OK:493803 Thread_num:997 TOTAL:493803 LATENCY:0.013888/53887985819 , P=OK:490873 Thread_num:996 TOTAL:492866 LATENCY:0.088454675865373461
1984 INFO:root:TPS_C=44728, NO=OK:494881 Thread_num:998 TOTAL:494881 LATENCY:0.013879/8658483733 , P=OK:491848 Thread_num:997 TOTAL:493083 LATENCY:0.088453945374189682
1985 INFO:root:TPS_C=44758, NO=OK:495800 Thread_num:999 TOTAL:495800 LATENCY:0.013879/83825493347 , P=OK:492824 Thread_num:998 TOTAL:494881 LATENCY:0.088465233947911745
1986 INFO:root:TPS_C=44798, NO=OK:496800 Thread_num:1000 TOTAL:496800 LATENCY:0.013879/83825493347 , P=OK:493881 Thread_num:999 TOTAL:495800 LATENCY:0.088452519489057137
```

Figure: 吞吐量一览



# 性能提升

我们采取了 RETURNING 关键字减少了不必要的查询

## 性能提升

- (1) 同时进行 SELECT 和 UPDATE 操作
- (2) 作用于 INSERT,UPDATE,DELETE
- (3) DELETE 返回操作前的数据, INSERT,UPDATE 返回操作后的数据

对应的 MongoDB 操作: `FindOneAndUpdate...`

## 云数据库

(1) 阿里云 RDS

(2) 阿里云 MongoDB

## 云数据库

- (1) 阿里云 RDS
- (2) 阿里云 MongoDB

The screenshot shows the Data Management DMS (DMS) interface. On the left, there's a sidebar with sections for '新增实例 / 批量导入', '实例地址、名称、数据', and '已登录实例 (1)' which lists 'pgm-uf61plxx443ypa8d' and 'postgres'. The main area has tabs for '全部功能', '单库查询', '数据导入', 'SQL结果集导出', '实例管理', '用户管理', '安全规则', and '操作审计'. The 'SQL public' tab is selected, showing a SQL console with the following code:

```
1 SELECT * FROM "book" LIMIT 100;
```

The results table shows data from the 'book' table:

序号	id	title	author	publisher
1	1361264	撒哈拉的故事	三毛	皇冠出版社
2	1036490	万水千山走遍	三毛	哈尔滨出版社
3	6710437	撒哈拉的故事	三毛	北京十月文艺出版社
4	1029111	哭泣的骆驼	三毛	哈尔滨出版社
5	26997048	撒哈拉的故事	三毛	北京十月文艺出版社

At the bottom, there's a message: 【消息】：执行成功。当前返回 [100] 行，耗时 [460ms]。注意：由于管理员配置，单次查询结果最多返回3,000条，如果需要更多行数，请走【数据导出】工单。创建导出工单。

Figure: 云数据库一览

1 数据库设计

2 基本功能设计与实现

3 进阶功能设计与实现

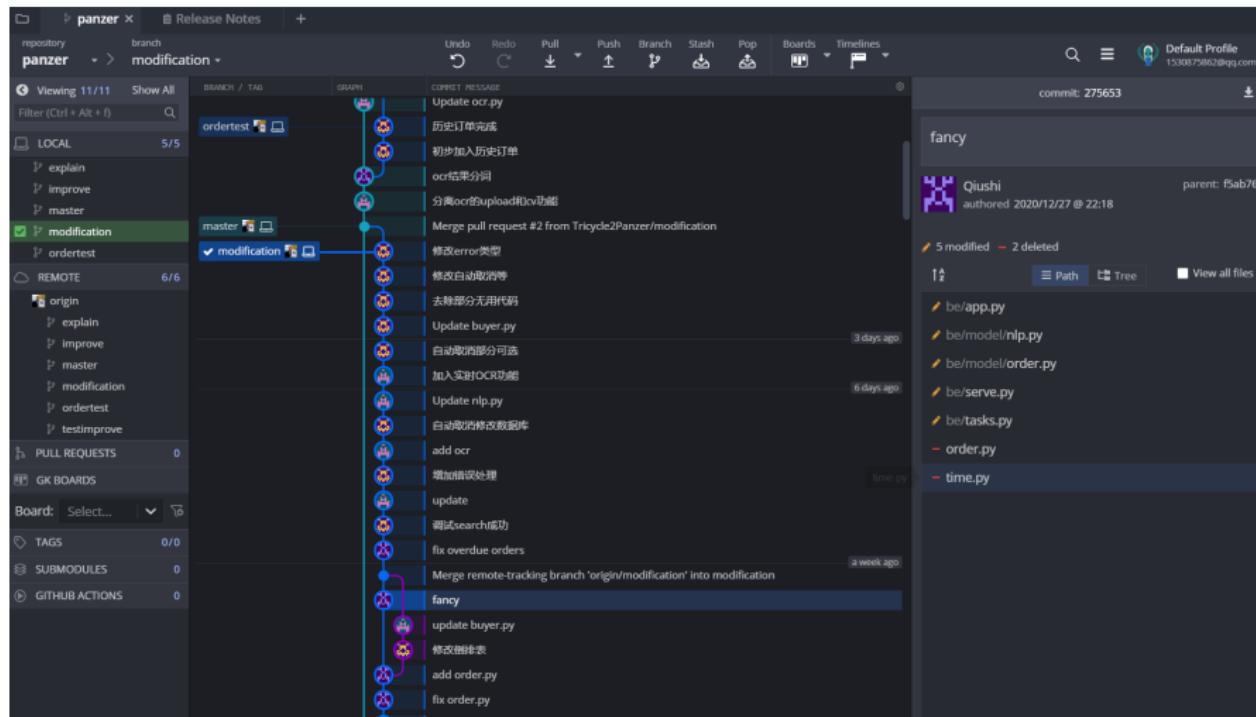
4 接口测试与性能分析

5 小组协作与分工

- Git 协作与版本控制
- 小组分工情况

6 演示与提问

# Git 协作与版本控制



# Git 协作与版本控制

The screenshot shows a GitHub repository interface for the 'panzer' project. The main view displays a timeline of commits and pull requests across various branches:

- improve**: The active branch, showing a series of commits from 'ChiachenCheng' (authored 2021/1/8 @ 19:39) and 'GitHub' (committed 2021/1/8 @ 19:39). The commit message is "Merge pull request #4 from Tricycle2Panzer/testimprove Testimprove, 第一批新加的test".
  - Commit details: 3 modified, 1 added. Files: fe/access/buyer.py, fe/access/seller.py, fe/test/gen\_book\_data.py, fe/test/test\_send\_books.py.
- testimprove**: A branch merged into 'improve'. Commit message: "Merge pull request #5 from Tricycle2Panzer/improve".
- ordertest**: A branch merged into 'improve'. Commit message: "Merge pull request #3 from Tricycle2Panzer/ordertest".
- modification**: A branch merged into 'improve'. Commit message: "Merge pull request #2 from Tricycle2Panzer/modification".
- master**: A branch merged into 'improve'. Commit message: "Merge pull request #1 from Tricycle2Panzer/master".

**Local** branch status: 5/5. **Remote** branch status: 6/6 (origin).

**PULL REQUESTS**: 0.

**GK BOARDS**: Board selection dropdown.

**TAGS**: 0/0.

**SUBMODULES**: 0.

**GITHUB ACTIONS**: 0.

# 小组分工情况

## 分工明细

- (1) 郑佳辰：数据库设计，推荐系统，倒排表及搜索系统，历史订单，性能优化，接口测试，产品经理
- (2) 孙秋实：订单自动取消，基本功能实现，接口测试，吞吐量测试，项目上云，slides 制作
- (3) 汤琼：接口测试，基本功能实现，以图搜书，实时 OCR，postman 测试，收发货功能实现，吞吐量测试

Git 版本控制和项目文档维护由三位同学一起完成。

# 小组分工情况

## 分工明细

- (1) 郑佳辰：数据库设计，推荐系统，倒排表及搜索系统，历史订单，性能优化，接口测试，产品经理
- (2) 孙秋实：订单自动取消，基本功能实现，接口测试，吞吐量测试，项目上云，slides 制作
- (3) 汤琼：接口测试，基本功能实现，以图搜书，实时 OCR，postman 测试，收发货功能实现，吞吐量测试

Git 版本控制和项目文档维护由三位同学一起完成。

# 小组分工情况

## 分工明细

- (1) 郑佳辰: 数据库设计, 推荐系统, 倒排表及搜索系统, 历史订单, 性能优化, 接口测试, 产品经理
- (2) 孙秋实: 订单自动取消, 基本功能实现, 接口测试, 吞吐量测试, 项目上云, slides 制作
- (3) 汤琼: 接口测试, 基本功能实现, 以图搜书, 实时 OCR, postman 测试, 收发货功能实现, 吞吐量测试

Git 版本控制和项目文档维护由三位同学一起完成。

1 数据库设计

2 基本功能设计与实现

3 进阶功能设计与实现

4 接口测试与性能分析

5 小组协作与分工

6 演示与提问

## *Demo and Questions!*