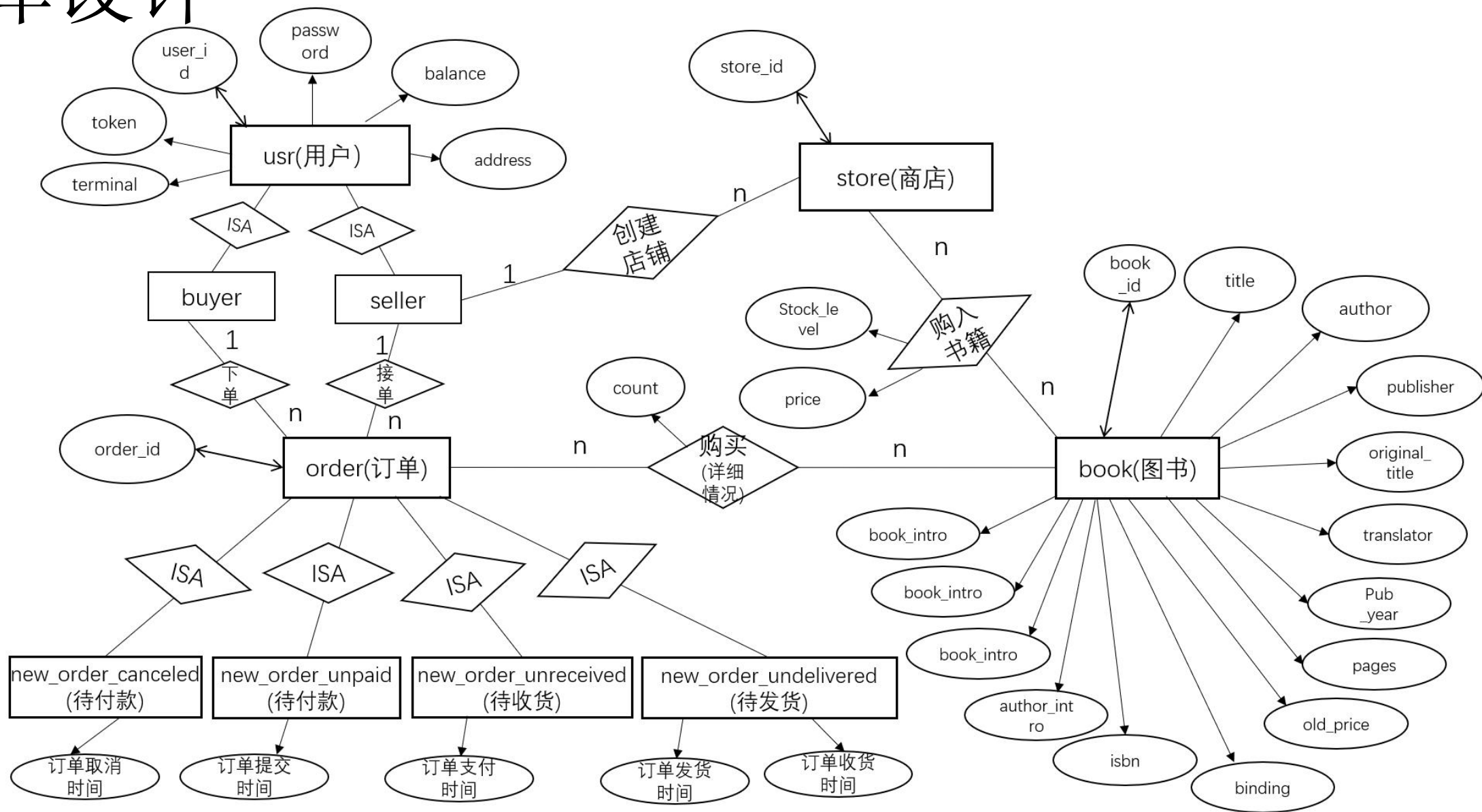


bookstore系统的设计与实现

成员：池欣宁，王文清，何诗雨

数据库设计

ER图



数据库设计

导出的关系模型

用户表(usr)

<u>user_id</u>	password	balance	token	terminal	address
String	String	String	String	String	String

用户-商店表(user_store)

<u>user_id</u>	<u>store_id</u>
String	String

商店表(store)

<u>store_id</u>	<u>book_id</u>	stock_level	price
String	String	Integer	Integer

订单总表(new_order_detail)

<u>order_id</u>	<u>book_id</u>	buyer_id	store_id	price	commit_time
String	String	String	String	Integer	DateTime

待付款订单(new_order_unpaid)

<u>order_id</u>	buyer_id	store_id	price	commit_time
String	String	String	Integer	DateTime

书籍(book)

<u>book_id</u>	title	author	...	picture
Integer	String	String	...	LargeBinary

待发货订单(new_order_undelivered)

<u>order_id</u>	buyer_id	store_id	price	purchase_time
String	String	String	Integer	DateTime

已取消订单(new_order_canceled)

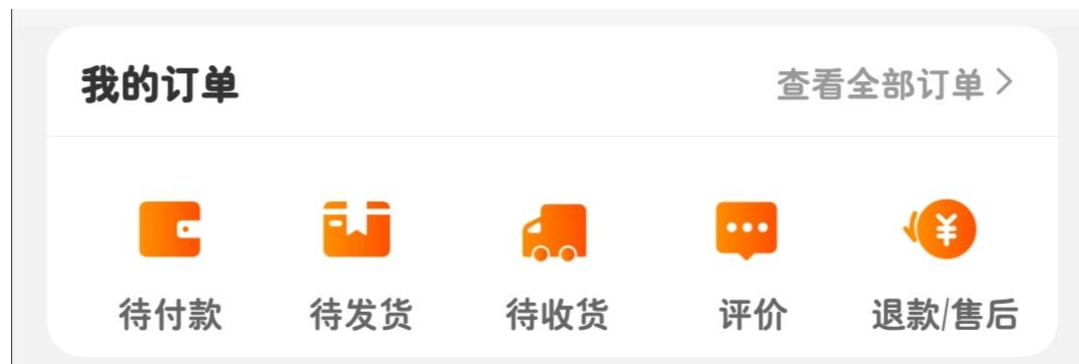
<u>order_id</u>	buyer_id	store_id	price	cancel_time
String	String	String	Integer	DateTime

待收货订单(new_order_unreceived)

<u>order_id</u>	buyer_id	store_id	price	send_time
String	String	String	Integer	DateTime

数据库设计

- new_order_detail分表设计
- 更贴合应用需求，用户会频繁查询已发货订单，待发货订单等



后端功能实现

采用orm形式

- 防止sql注入，保障安全性
- 同时实现事务处理

函数中包含两个事务的实现

——以add_book为例

- 实现两种版本
 - ✓ 支持只上架图书
 - ✓ 支持将书籍添加到book表中并上架图书（该版本可以不运行book.py导入数据，通过add_book函数插入书籍）
- self.commit()位置的思考：
 - a. 第二个版本事务add_book包括添加图书，将书籍添加到商店这两步
 - b. 在每一步做完后都需提交
 - c. 如果两步结束后才commit，就导致第一步的图书还没添加进去，就在做第二步的将书籍添加到商店，而因为store里面的book_id必须存在在book表中

下单和支付的实现

下单

- 保证用户id和storeid存在
- 根据订单信息在store表中查找商户中是否存在对应书籍和足够的库存，若满足则在store表中减少相应书籍的库存
- 向new_order_detail表插入订单信息（订单号，购买书籍信息，买家卖家信息）
- 记录下单时间，将订单信息插入new_order_unpaid

性能分析

- store表k次根据主键查询，k次更新
- new_order_detail表k次插入(k为订单中购买的书本数) new_order_unpaid表一次插入

下单和支付的实现

支付

- 查看用户是否有该待付订单
- 比对用户密码，查询用户余额是否充足
- 若余额充足，付款成功，买家减少余额，卖家增加余额
- 在new_order_unpaid表中删除对应的待付订单信息
- 记录当前时间，在待发货表new_order_undelivered表加入订单信息和付款时间。

性能分析

- new_order_unpaid表一次根据主键order_id查询，一次删除
- user表两次根据主键user_id查询，两次更新（一次买家、一次卖家） new_order_undelivered表一次插入。

后续功能——发货->收货

- 买家收货receive_book

功能实现:

1. 保证传入的参数user_id和order_id在数据库中存在
2. 若订单存在, 卖家存在且匹配, 在待发货表new_order_undelivered删除该订单, 在待收货表new_order_unreceived中添加该订单和发货时间。

性能分析:

usr表一次查询;

new_order_undelivered表一次查询, 一次更新。

new_order_unreceived表一次更新。



- 买家收货receive_book

功能实现:

1. 保证传入的参数user_id和order_id在数据库中存在
2. 若订单存在, 买家存在且匹配, 在待收货表new_order_unreceived中添加买家收货的时间。

性能分析:

usr表一次查询;

new_order_unreceived表一次查询, 一次更新。

特色功能--全类型历史订单查询

```
1 {
2   "buyer_id": "lalala@ecnu.com",
3   "flag": 1
4 }
5
```

body Cookies Headers (4) Test Results

Pretty Raw Preview Visualize JSON ↗

```
1 {
2   "history record": [
3     {
4       "book_list": [
5         {
6           "book_id": 1,
7           "count": 2,
8           "price": 2000
9         }
10      ],
11     "buyer_id": "lalala@ecnu.com",
12     "commit_time": "Sat, 09 Jan 2021 17:35:51 GMT",
13     "order_id": "order1",
14     "status": "未付款",
15     "store_id": "Lemon"
16   },
17   "message": "ok"
18 }
19
```

为支持不同的查询订单需求，函数接口中除buyer_id另增加flag。返回结果可显示下单/购买/收货等时间更符合实际业务要求。

支持查询：

- ✓ 用户所有订单
- ✓ 待付款订单
- ✓ 已付款待发货订单
- ✓ 已发货待收货订单
- ✓ 已收货订单
- ✓ 已取消订单

性能分析

- 查所有订单：new_order_detail表一次查询
- 查询待付款订单：new_order_unpaid表一次查询，对应new_order_detail表k次根据主键查询（k为new_order_unpaid的该用户待付记录数）

扩展功能--手动取消订单

功能实现:

1. 判断是否为待付款订单
2. 若是, 在new_order_unpaid中删除对应订单
3. 判断是否为待发货订单
4. 确定订单未发货后, 在usr表中更新买家余额增加该订单对应款项。
5. 在usr表中更新卖家余额减少该订单对应款项。
6. 在待发列表中删除对应记录。
7. 记录当前时间并将订单信息加入new_order_cancel表中。
8. 在store中将对应的书籍的库存加回。
9. 若不是上述两种情况, 返回无法取消订单

性能分析:

- new_order_unpaid表一次查询, 一次删除, new_order_undelivered表一次查询, 一次删除
- new_order_cancel表一次插入, new_order_detail表一次查询
- store表k次更新 (k为购买书籍数) user表两次根据user_id主键查询, 两次更新 (一次买家、一次卖家)。

扩展功能一 自动取消订单

使用技术:

Redis键空间通知（过期回调） 用户下单之后将订单id作为key，任意值作为值存入redis中，给这条数据设置过期时间，也就是订单超时的时间。

1, 下载安装redis

- 在redis文件中使用redis-server.exe redis.windows.conf启动redis
- 设置redis.windows.conf中的notify-keyspace-events "EX"（为了保证可以发出过期数据的数据） 我们会收到关键事件通知，在keyevent频道中，我们会收到key作为消息。
- 可使用redis-cli.exe --csv psubscribe '*' 测试服务是否打开

2, 在python中使用redis，配置回调函数

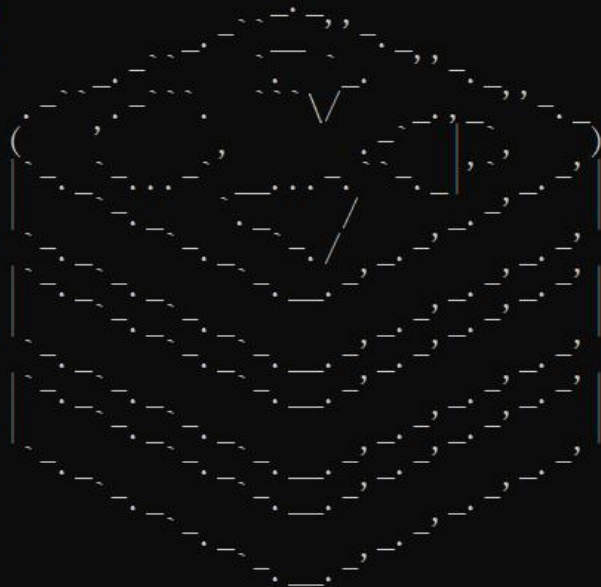
注册回调函数来处理已发布的消息。消息处理程序只接受一个参数即消息。要使用消息处理程序订阅通道或模式，请将通道或模式名称作为关键字参数传递，其值为回调函数。当使用消息处理程序在通道或模式上读取消息时，将创建消息字典并将其传递给消息处理程序。在这种情况下，从get_message()返回None值，因为消息已经处理完毕。

```
D:\program\Redis>redis-server.exe redis.windows.conf
```

```
[16196] 12 Jan 23:13:24.990 # o000o000o000o Redis is starting o000o000o000o
```

```
[16196] 12 Jan 23:13:24.991 # Redis version=5.0.10, bits=64, commit=1c047b68, modified=0, pid=16196, just started
```

```
[16196] 12 Jan 23:13:24.992 # Configuration loaded
```



```
Redis 5.0.10 (1c047b68/0) 64 bit
```

```
Running in standalone mode
```

```
Port: 6379
```

```
PID: 16196
```

```
http://redis.io
```

```
[16196] 12 Jan 23:13:25.002 # Server initialized
```

```
[16196] 12 Jan 23:13:25.003 * DB loaded from disk: 0.001 seconds
```

```
[16196] 12 Jan 23:13:25.004 * Ready to accept connections
```

```
D:\program\Redis>redis-cli.exe -h 127.0.0.1 -p 6379
```

```
127.0.0.1:6379> set 1 1
```

```
OK
```

```
127.0.0.1:6379> get 1
```

```
"1"
```

```
127.0.0.1:6379> setex 1 1 123
```

```
OK
```

```
127.0.0.1:6379>
```

扩展功能—自动取消订单

功能实现:

1. 在buyer()中的new_order函数中将订单号存入redis数据库中, 并设置超时时间
2. 在回调函数auto_cancel_order中设置死循环, 每当接收到一个过期数据, 就将order_id解析出来。
3. 通过这个order_id判断能否找到未支付订单new_order_unpaid中的数据,
4. 如果存在, 将其删除, 在已取消订单new_order_canceled中添加该order和取消时间。
5. 如果不存在, 说明该订单已被支付或者买家主动取消订单, 则什么都不用处理。

性能分析:

- redis数据库一次更新 (插入数据) 。
- 当接收到数据时, 对new_order_unpaid一次查找;
- 当数据存在, 对new_order_unpaid一次更新;
- 对new_order_canceled一次更新;
- 对store进行k次更新, k为订单中图书的数量。

扩展功能一 自动取消订单

```
#连接redis数据库
r=redis.StrictRedis(host='localhost',port=6379,db=0,decode_responses=True)
# 创建pubsub对象, 该对象订阅一个频道并侦听新消息:
pubsub=r.pubsub()
#收到消息的处理函数
def event_handler(msg):
    # print('Handler',msg)
    order_id=str(msg['data'])
    print(order_id)
    #如果能找到订单, 就删除未支付订单
    #添加到已删除订单中
    #将商店中的书籍书加回去
#订阅过期数据
pubsub.psubscribe(**{'__keyevent@0__:expired':event_handler})
#死循环, 当有数据过期时, 调用函数event_handler处理过期数据
while True:
    # print("监控超时订单")
    #获得事件信息, 有结果就会回调函数
    message=pubsub.get_message()
    time.sleep(0.1)
```

POST http://127.0.0.1:5000/buyer/new_order Send

Params Auth Headers (8) Body ● Pre-req. Tests Settings

Query Params

	KEY	VALUE	DESCRIPTION
	Key	Value	Description

Body ▼ 200 OK 2.21 s 231 B

Pretty Raw Preview Visualize JSON

```
1 {
2   "message": "ok",
3   "order_id": "aaa@163.com_Lemon_e5085be6-54e9-11eb-b66c-2016b92fc3c7"
4 }
```



```
D:\这学期\数据管理系统\大作业\项目\DB\be>python app.py
* Serving Flask app "be.serve" (lazy loading)
* Environment: production
  WARNING: This is a development server. Do not use it in a production deployment.
  Use a production WSGI server instead.
* Debug mode: off
2021-01-12 23:21:31,799 [MainThread ] [INFO ] * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
2021-01-12 23:21:44,822 [Thread-1 ] [INFO ] 127.0.0.1 - - [12/Jan/2021 23:21:44] "POST /buyer/new_order HTTP/1.1" 51
7 -
order_id aaa@163.com_Lemon_e5085be6-54e9-11eb-b66c-2016b92fc3c7
2021-01-12 23:21:53,176 [Thread-2 ] [INFO ] 127.0.0.1 - - [12/Jan/2021 23:21:53] "POST /buyer/new_order HTTP/1.1" 20
0 -
```



```
D:\这学期\数据管理系统\大作业\项目\DB\be>python auto_cancel_order.py
aaa@163.com_Lemon_e5085be6-54e9-11eb-b66c-2016b92fc3c7
<init_db.init_database.New_order_unpaid object at 0x00C11530>
订单已删除
订单已添加
count 1
订单已更新
```


搜索

- 实现功能:

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，作者，简介。

- 支持全书库搜索和当前店铺搜索。

- 支持分页显示结果

- 功能实现目标:

尽量避免搜索速度过慢而用户难以忍受的问题，时空代价trade-off

- 后端实现:

1. searchDB表的设计与构建

2. 全文索引

3. 分词

4. 分页查询

基本表结构实现1.0

Basic-Tables

search_author:

search_id	author	book_id
2	美西	1
3	西尔	1

原有Book表直接搜索，4w+数据即使是有索引也会存在过慢的问题

search_tags:

search_id	tags	book_id
1	传记	1
2	纳什	1

search_title:

search_id	title	book_id
1	美丽心灵	1
2	三毛流浪记全集	2 ...

search_book_intro

search_id	book_intro	book_id
1	投资	847
2	投资者	847

搜索功能实现1.0版本:

- 拆分4张字段基本表
- 在author/tags/title/book_intro字段做B-tree索引

搜索功能实现(1) like关键字模糊查询--B-treeIndex

- `SELECT * from search_book_author where author like '%杨红%';`
- 模糊查询的缺点:
- 当用户输入字段前后部分都是`%`, 尤其是相当于搜索`%s`(后缀匹配)时, 我们建立在主键上默认的B树索引即失去了范围查询中的效率优势。
- 以在70w条数据中的全局搜索为例子:

```
39 SELECT count(*) from search_book_intro
40
```

count
690359

❑ `EXPLAIN ANALYZE DISTINCT
book_id FROM search_book_intro
WHERE book_intro like '世界%';`

```
38 EXPLAIN analyze SELECT distinct book_id from search_book_intro where book_intro like '%世界'
39
```

信息 结果 1	
QUERY PLAN	
►	HashAggregate (cost=14639.80..14673.70 rows=3390 width=4) (actual time=69.247..71.614 rows=4013 loops=1)
•	Group Key: book_id
•	Batches: 1 Memory Usage: 465kB
•	-> Gather (cost=1000.00..14630.92 rows=3553 width=4) (actual time=1.708..68.856 rows=4082 loops=1)
•	Workers Planned: 2
•	Workers Launched: 2
•	-> Parallel Seq Scan on search_book_intro (cost=0.00..13275.62 rows=1480 width=4) (actual time=3.240..64.681 rows=1361 loops=1)
•	Filter: (book_intro ~~ '%世界':text)
•	Rows Removed by Filter: 228759
•	Planning Time: 0.096 ms
•	Execution Time: 71.967 ms

搜索功能实现(1) like关键字模糊查询--B-treeIndex

```
3 EXPLAIN ANALYZE SELECT * FROM book where book_intro like '%生活%';
```

信息 结果 1

QUERY PLAN

Seq Scan on book (cost=0.00..7324.74 rows=6069 width=1324) (actual time=1.645..928.123 rows=7910 loops=1)

Filter: (book_intro ~~ '%生活% '::text)

Rows Removed by Filter: 32090

Planning Time: 48.168 ms

Execution Time: 936.401 ms

```
SELECT * FROM book WHERE book_intro like '生活%'
```

QUERY PLAN

Seq Scan on book (cost=0.00..7324.74 rows=4 width=1324) (actual time=0.476..42.866 rows=30 loops=1)

Filter: (book_intro ~~ '生活% '::text)

I Rows Removed by Filter: 39970

Planning Time: 0.293 ms

Execution Time: 42.905 ms

```
SELECT * FROM book WHERE book_intro like '%生活'
```

QUERY PLAN

Seq Scan on book (cost=0.00..7324.74 rows=4 width=1324) (actual time=159.069..159.069 rows=0 loops=1)

Filter: (book_intro ~~ '%生活'::text)

Rows Removed by Filter: 40000

Planning Time: 0.657 ms

Execution Time: 159.089 ms

- EXPLAIN ANALYZE
DISTINCT book_id FROM
search_book_intro WHERE
book_intro like '%生活% '/'%
生活 '/'生活%'
- 可以很明显的看出，无论是前缀匹配还是后缀匹配中，即使是在book_intro字段建立了索引，这个执行时间也很可怕。是用户难以忍受的。

可以很明显的看出，无论是前缀匹配还是后缀匹配中，即使是在book_intro字段建立了索引，这个执行时间也很可怕。是用户难以忍受的。

搜索功能实现(2) 从like到GIN-index: PSQLETEXT全文搜索

- to_tsvector()和to_tsquery()函数

- 函数原理=》

- PSQLETEXT

全文搜索功能
搜索的支持需
要中文分词的
实现!

1. to_tsvector实际上把语句转换成了tsvector(文档格式-包含文档值和角标)

如:

```
SELECT * FROM to_tsvector('parser_name',TEXT)
```

```
Input:SELECT * FROM to_tsvector('parser_name','小明今天去上学')
```

```
Output:'上学': 3 '去': 2 '小明': 1
```

通过output, 我们可以看到“上学”对应位置3, “小明”对应语句的位置1, 我们记住这种格式就行, 后续会用到。

2. to_tsquery(), 实际上就是一个传递的参数格式, 可以搭配中文分词规则, 把一句话拆成多个词传递过去

```
Input:SELECT * FROM to_tsvector('parser_name','小明今天去上学')
```

```
Output:'上学'& '去'& '小明'
```

3. 结合 to_tsvector() 和 to_tsquery(),即可完成本次数据库的关键字全文搜索功能。|

基本表结构实现2.0

对象		search_book_tags@publi...	search_author@public (S...
search_id	author	book_id	tsv_column
2	美西	1	'美西':1
3	西尔	1	'西尔':1
4	尔维	1	'尔维':1
5	维娅	1	
6	娅娜	1	
7	娜萨	1	'萨':1
8	萨	1	'萨':1
9	张乐	2	'张乐':1
10	乐平	2	'乐平':1

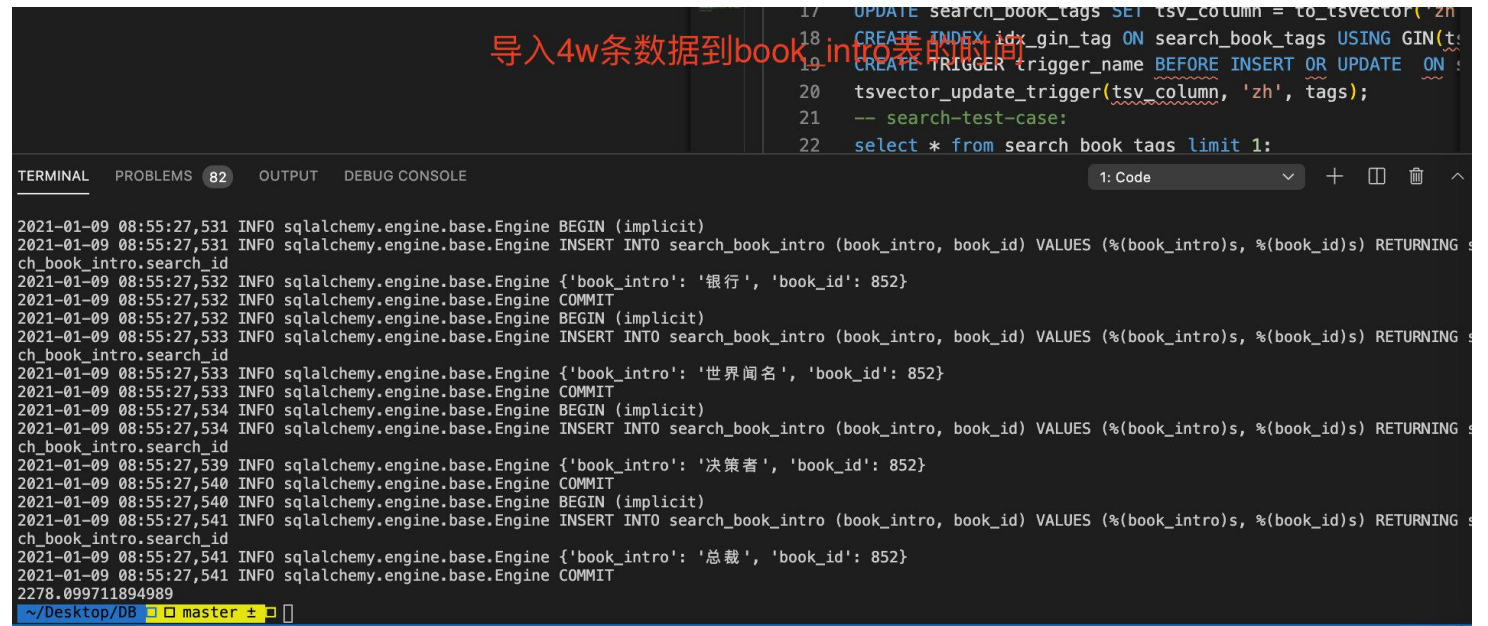
search_id	title	book_id	tsv_column
1	美丽心灵	1	'心灵':2 '美丽':1
2	三毛流浪记	2	'三毛':1 '全集':3 '流浪记':2
3	黑色棉花田	3	'棉花':2 '田':3 '黑色':1
4	中国历代年号	4	'中国':1 '历代':2 '年号':3 '考':4
5	袁氏当国	5	'国':3 '当':2 '袁氏':1
6	一个狗娘舅	6	'养':3 '娘':2 '狗':1 '自白':4
7	新唐书全二	7	'书':3 '全':4 '唐':2 '新':1
8	第二次世界大战回忆录	8	'全':3 '回忆录':2 '第二次世界大战':1
9	学习的革命	9	'学习':1 '革命':2

搜索功能实现2.0版本:

- 在4个基本表的表结构的基础上，执行SQL语句调整表的结构，构建tsv_column列，并在tsv_column上构建GIN——index支持全文索引查询，tsv_column存储的是分好词的相应字段的ts_vector结果。
- ALTER TABLE search_author ADD COLUMN tsv_column tsvector;
- UPDATE search_author SET tsv_column = to_tsvector('zh', coalesce(author, ''));
- CREATE INDEX idx_gin_author ON search_author USING GIN(tsv_column);
- CREATE TRIGGER trigger_name BEFORE INSERT OR UPDATE ON search_book_intro FOR EACH ROW EXECUTE PROCEDURE tsvector_update_trigger(tsv_column, 'zh', book_intro);

搜索功能实现(3) GIN-index构建的依赖-中文分词

- 在搜索系统中，分词很大的影响了了搜索系统的召回率和准确率，以及基础table表的纯粹空间消耗。
- 1. 分词粒度? =>针对不同的字段有不同的分词方式
- 2. postgresql数据库内分词(边查边分) or 数据库外程序分词? =>数据库外程序+PSQL内分词结果存储
- 3. 搜索效率和存储的trade-off



The image shows a code editor window with SQL queries and a terminal window displaying database logs. The code editor has a red annotation: "导入4w条数据到book_intro表的时间" (Time to import 40,000 records into the book_intro table). The SQL queries in the code editor are:

```
17 UPDATE search_book_tags SET tsv_column = to_tsvector('zh',
18 CREATE INDEX idx_gin_tag ON search_book_tags USING GIN(t
19 CREATE TRIGGER trigger_name BEFORE INSERT OR UPDATE ON
20 tsvector_update_trigger(tsv_column, 'zh', tags);
21 -- search-test-case:
22 select * from search_book_tags limit 1;
```

The terminal window shows logs from the sqlalchemy engine, including BEGIN, INSERT, and COMMIT statements, and the results of the SELECT query:

```
2021-01-09 08:55:27,531 INFO sqlalchemy.engine.base.Engine BEGIN (implicit)
2021-01-09 08:55:27,531 INFO sqlalchemy.engine.base.Engine INSERT INTO search_book_intro (book_intro, book_id) VALUES (%(book_intro)s, %(book_id)s) RETURNING s
ch_book_intro.search_id
2021-01-09 08:55:27,532 INFO sqlalchemy.engine.base.Engine {'book_intro': '银行', 'book_id': 852}
2021-01-09 08:55:27,532 INFO sqlalchemy.engine.base.Engine COMMIT
2021-01-09 08:55:27,532 INFO sqlalchemy.engine.base.Engine BEGIN (implicit)
2021-01-09 08:55:27,533 INFO sqlalchemy.engine.base.Engine INSERT INTO search_book_intro (book_intro, book_id) VALUES (%(book_intro)s, %(book_id)s) RETURNING s
ch_book_intro.search_id
2021-01-09 08:55:27,533 INFO sqlalchemy.engine.base.Engine {'book_intro': '世界闻名', 'book_id': 852}
2021-01-09 08:55:27,533 INFO sqlalchemy.engine.base.Engine COMMIT
2021-01-09 08:55:27,534 INFO sqlalchemy.engine.base.Engine BEGIN (implicit)
2021-01-09 08:55:27,534 INFO sqlalchemy.engine.base.Engine INSERT INTO search_book_intro (book_intro, book_id) VALUES (%(book_intro)s, %(book_id)s) RETURNING s
ch_book_intro.search_id
2021-01-09 08:55:27,539 INFO sqlalchemy.engine.base.Engine {'book_intro': '决策者', 'book_id': 852}
2021-01-09 08:55:27,540 INFO sqlalchemy.engine.base.Engine COMMIT
2021-01-09 08:55:27,540 INFO sqlalchemy.engine.base.Engine BEGIN (implicit)
2021-01-09 08:55:27,541 INFO sqlalchemy.engine.base.Engine INSERT INTO search_book_intro (book_intro, book_id) VALUES (%(book_intro)s, %(book_id)s) RETURNING s
ch_book_intro.search_id
2021-01-09 08:55:27,541 INFO sqlalchemy.engine.base.Engine {'book_intro': '总裁', 'book_id': 852}
2021-01-09 08:55:27,541 INFO sqlalchemy.engine.base.Engine COMMIT
2278.099711894989
```

搜索功能实现(3) GIN-index构建的依赖- 外部程序的详细中文分词方式

最后的分词方式：

2-gram+存储作者姓和全称字段(如果全称字段被zh-parser自动分割，也仍然无法提高召回)

Search_book_intro

因为book_intro都是大量的文本，通过对文本提取关键词的方式进行分割，提取关键词的方法有很多，有TF-IDF,TextRank等。

Search_tags

tags是已提取好的标签，直接把tags拆开插入的修改后的表中。

Search_title

title字段和author字段都是短文本，他们最显著的区别是，姓名字段的字与字之间没有明显强烈的上下文关系，如杨红樱三个字与《一个狗娘养的自白》这个标题相比，显然是标题的字与字之间更具有上下文的关系，所以在search_title部分我没有采用形如search_author一样的粒度划分，而是直接采用了zh-parser本身分词功能对与search_title表格的修改。

搜索功能实现(2 & 3) 最终的搜索功能

- GIN-INDEX构建时间仍在可接受的范围内:
针对278549条数据

31
32

select count(*) from sear

count

278549

12
13
14
15

CREATE INDEX idx_gin_auth

ALTER TABLE search_autho
UPDATE search_autho SET

sql

CREATE INDEX idx_gin_author_test

SELECT DISINDIC

38
39

SELECT distinct book_id f
SELECT count(*) from sear

38
39

EXPLAIN analyze SELECT distinct book_id from search_book_intro where book_intro like '%世界'

信息

结果 1

QUERY PLAN

► HashAggregate (cost=14639.80..14673.70 rows=3390 width=4) (actual time=69.247..71.614 rows=4013 loops=1)

• Group Key: book_id

• Batches: 1 Memory Usage: 465kB

• -> Gather (cost=1000.00..14630.92 rows=3553 width=4) (actual time=1.708..68.856 rows=4082 loops=1)

• Workers Planned: 2

• Workers Launched: 2

• -> Parallel Seq Scan on search_book_intro (cost=0.00..13275.62 rows=1480 width=4) (actual time=3.240..64.681 rows=1361 loops=1)

• Filter: (book_intro ~~ '%世界'::text)

• Rows Removed by Filter: 228759

• Planning Time: 0.096 ms

• Execution Time: 71.967 ms

信息

结果 1

sql

message

SELECT distinct book_id from search_book_intro where tsv_column @@ '世界'

OK, Time: 0.006000s

搜索功能实现(4) 分页查询

- `select * from user limit 10000,10;` =》这种分页查询机制，每次都会从数据库第一条记录开始扫描，越往后查询越慢，而且查询的数据越多，也会拖慢总查询速度。
- 优化这种传统分页查询机制的方式
 - 引入search-id的分页查询优化
 - 直接使用GIN INDEX+LIMIT的分页查询优化

37	EXPLAIN analyze SELECT distinct book_id from search_book_intro where tsv_column @@ '世界' LIMIT 285	
38	EXPLAIN analyze SELECT distinct book_id from search_book_intro where book_intro like '%世界%'	
		信息 结果 1
QUERY PLAN		
Limit (cost=7158.72..7161.57 rows=285 width=4) (actual time=5.102..5.191 rows=285 loops=1)		
-> HashAggregate (cost=7158.72..7193.17 rows=3445 width=4) (actual time=5.101..5.155 rows=285 loops=1)		
Batches: 1 Memory Usage: 465kB		
-> Bitmap Heap Scan on search_book_intro (cost=48.00..7149.69 rows=3613 width=4) (actual time=1.021..3.898 rows=3822 loops=1)		
Heap Blocks: exact=2873		
-> Bitmap Index Scan on idx_gin_zh (cost=0.00..47.10 rows=3613 width=0) (actual time=0.635..0.635 rows=3822 loops=1)		
Planning Time: 0.144 ms		信息 结果 1
Execution Time: 5.261 ms		
QUERY PLAN		
Aggregate (cost=12473.93..12473.94 rows=1 width=8) (actual time=9.744..9.745 rows=1 loops=1)		
-> Bitmap Heap Scan on search_book_intro (cost=1960.43..12473.32 rows=245 width=4) (actual time=2.734..9.660 rows=290 loops=1)		
Recheck Cond: (search_id > 200000) AND (search_id < 250000)		
Filter: (book_intro ~~ '%世界':text)		
Rows Removed by Filter: 49709		
-> Bitmap Index Scan on search_book_intro_pkey (cost=0.00..1960.37 rows=47594 width=0) (actual time=2.607..2.607 rows=49999 loops=1)		
Index Cond: ((search_id > 200000) AND (search_id < 250000))		
Planning Time: 0.104 ms		
Execution Time: 9.775 ms		

搜索功能实现(5) 后端代码

- 逻辑上来讲，一家书店拥有的书的数量，跟整个全站搜索库里的书的数量相比是很少的，没有额外进行建表的必要。
- 所以，在SQL查询语句确定了以后，后端实现的逻辑如下：
 - 1. 首先获得book_db库中符合全局搜索的`book_id`
 - 2. 对于全局搜索直接返回
 - 3. 对于本店搜索，则使用子查询，搜索本店和全局检索出的`book_id`重复的部分。

测试结果

pytest:

共计50个test case,
全部测试通过!

```
chixinning@ymac: ~/Desktop/Bookstore/DB $ python3 master.py & bash script/test.sh
===== test session starts =====
platform darwin -- Python 3.8.6, pytest-6.1.1, py-1.9.0, pluggy-0.13.1 -- /usr/local/opt/python@3.8/bin/python3.8
cachedir: .pytest_cache
rootdir: /Users/chixinning/Desktop/Bookstore/DB
collecting ... frontend begin test
* Serving Flask app "be.serve" (lazy loading)
* Environment: production
  WARNING: This is a development server. Do not use it in a production deployment.
  Use a production WSGI server instead.
* Debug mode: off
2021-01-10 13:12:42,068 [Thread-1 ] [INFO ] * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
collected 50 items

fe/test/test_add_book.py::TestAddBook::test_ok PASSED [ 2%]
fe/test/test_add_book.py::TestAddBook::test_error_non_exist_store_id PASSED [ 4%]
fe/test/test_add_book.py::TestAddBook::test_error_exist_book_id PASSED [ 6%]
fe/test/test_add_book.py::TestAddBook::test_error_non_exist_user_id PASSED [ 8%]
fe/test/test_add_funds.py::TestAddFunds::test_ok PASSED [ 10%]
fe/test/test_add_funds.py::TestAddFunds::test_error_user_id PASSED [ 12%]
fe/test/test_add_funds.py::TestAddFunds::test_error_password PASSED [ 14%]
fe/test/test_add_stock_level.py::TestAddStockLevel::test_error_user_id PASSED [ 16%]
fe/test/test_add_stock_level.py::TestAddStockLevel::test_error_store_id PASSED [ 18%]
fe/test/test_add_stock_level.py::TestAddStockLevel::test_error_book_id PASSED [ 20%]
fe/test/test_add_stock_level.py::TestAddStockLevel::test_ok PASSED [ 22%]
fe/test/test_bench.py::test_bench PASSED [ 24%]
fe/test/test_cancel.py::Test_cancel::test_ok_paid PASSED [ 26%]
fe/test/test_cancel.py::Test_cancel::test_false_buyer PASSED [ 28%]
fe/test/test_cancel.py::Test_cancel::test_ok_unpay PASSED [ 30%]
fe/test/test_cancel.py::Test_cancel::test_cannot_cancel_order PASSED [ 32%]
fe/test/test_create_store.py::TestCreateStore::test_ok PASSED [ 34%]
fe/test/test_create_store.py::TestCreateStore::test_error_exist_store_id PASSED [ 36%]
fe/test/test_deliver_book.py::TestDeliverBook::test_ok PASSED [ 38%]
fe/test/test_deliver_book.py::TestDeliverBook::test_authorization_error PASSED [ 40%]
fe/test/test_deliver_book.py::TestDeliverBook::test_order_error PASSED [ 42%]
fe/test/test_login.py::TestLogin::test_ok PASSED [ 44%]
fe/test/test_login.py::TestLogin::test_error_user_id PASSED [ 46%]
fe/test/test_login.py::TestLogin::test_error_password PASSED [ 48%]
fe/test/test_new_order.py::TestNewOrder::test_non_exist_book_id PASSED [ 50%]
fe/test/test_new_order.py::TestNewOrder::test_low_stock_level PASSED [ 52%]
fe/test/test_new_order.py::TestNewOrder::test_ok PASSED [ 54%]
fe/test/test_new_order.py::TestNewOrder::test_non_exist_user_id PASSED [ 56%]
fe/test/test_new_order.py::TestNewOrder::test_non_exist_store_id PASSED [ 58%]
fe/test/test_password.py::TestPassword::test_ok PASSED [ 60%]
fe/test/test_password.py::TestPassword::test_error_password PASSED [ 62%]
fe/test/test_password.py::TestPassword::test_error_user_id PASSED [ 64%]
fe/test/test_payment.py::TestPayment::test_ok PASSED [ 66%]
fe/test/test_payment.py::TestPayment::test_authorization_error PASSED [ 68%]
fe/test/test_payment.py::TestPayment::test_not_suff_funds PASSED [ 70%]
fe/test/test_payment.py::TestPayment::test_repeat_pay PASSED [ 72%]
fe/test/test_receive_book.py::TestReceiveBook::test_ok PASSED [ 74%]
fe/test/test_receive_book.py::TestReceiveBook::test_authorization_error PASSED [ 76%]
fe/test/test_receive_book.py::TestReceiveBook::test_order_error PASSED [ 78%]
fe/test/test_register.py::TestRegister::test_register_ok PASSED [ 80%]
fe/test/test_register.py::TestRegister::test_unregister_ok PASSED [ 82%]
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_search_functions.py::TestSearchFunctions::test_ok_author PASSED [ 88%]
fe/test/test_search_functions.py::TestSearchFunctions::test_non_store_id_err PASSED [ 90%]
fe/test/test_search_functions.py::TestSearchFunctions::test_no_such_book_in_DB_err2 PASSED [ 92%]
fe/test/test_search_functions.py::TestSearchFunctions::test_no_such_book_instore_err PASSED [ 94%]
fe/test/test_search_history_status.py::Test_search_history_status::test_ok PASSED [ 96%]
fe/test/test_search_history_status.py::Test_search_history_status::test_false_buyer PASSED [ 98%]
fe/test/test_search_history_status.py::Test_search_history_status::test_no_record_buyer PASSED [100%]
```

覆盖率

覆盖率93%

DB >	pytest(93%).txt					
1	Name	Stmts	Miss	Branch	BrPart	Cover
2	-----					
3	be/__init__.py	0	0	0	0	100%
4	be/app.py	8	8	2	0	0%
5	be/model1/__init__.py	0	0	0	0	100%
6	be/model1/buyer.py	281	27	102	12	88%
7	be/model1/db_conn.py	30	0	6	0	100%
8	be/model1/error.py	23	3	0	0	87%
9	be/model1/seller.py	95	16	30	4	79%
10	be/model1/store.py	27	4	0	0	85%
11	be/model1/user.py	129	24	38	6	77%
12	be/serve.py	37	1	2	1	95%
13	be/view1/__init__.py	0	0	0	0	100%
14	be/view1/auth.py	42	0	0	0	100%
15	be/view1/buyer.py	82	13	2	0	85%
16	be/view1/seller.py	39	0	0	0	100%
17	fe/__init__.py	0	0	0	0	100%
18	fe/access/__init__.py	0	0	0	0	100%
19	fe/access/auth.py	31	0	0	0	100%
20	fe/access/book.py	70	1	12	2	96%
21	fe/access/buyer.py	61	0	2	0	100%
22	fe/access/new_buyer.py	8	0	0	0	100%
23	fe/access/new_seller.py	8	0	0	0	100%
24	fe/access/seller.py	37	0	0	0	100%
25	fe/bench/__init__.py	0	0	0	0	100%
26	fe/bench/run.py	13	0	6	0	100%
27	fe/bench/session.py	47	0	12	1	98%
28	fe/bench/workload.py	125	1	22	2	98%
29	fe/conf.py	11	0	0	0	100%
30	fe/conftest.py	17	0	0	0	100%
31	fe/test/gen_book_data.py	48	0	16	0	100%
32	fe/test/test_add_book.py	36	0	10	0	100%
33	fe/test/test_add_funds.py	23	0	0	0	100%
34	fe/test/test_add_stock_level.py	39	0	10	0	100%
35	fe/test/test_bench.py	6	2	0	0	67%
36	fe/test/test_cancel.py	49	0	2	0	100%
37	fe/test/test_create_store.py	20	0	0	0	100%
38	fe/test/test_deliver_book.py	64	1	4	1	97%
39	fe/test/test_login.py	28	0	0	0	100%
40	fe/test/test_new_order.py	40	0	0	0	100%
41	fe/test/test_password.py	33	0	0	0	100%
42	fe/test/test_payment.py	60	1	4	1	97%
43	fe/test/test_receive_book.py	70	1	4	1	97%
44	fe/test/test_register.py	35	0	0	0	100%
45	fe/test/test_search_functions.py	52	0	0	0	100%
46	fe/test/test_search_history_status.py	76	0	8	0	100%
47	-----					
48	TOTAL	1900	103	294	31	93%

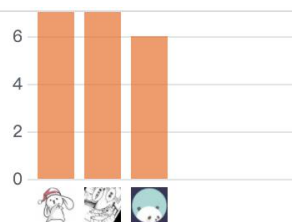
吞吐量&延迟

mac测试吞吐量达18750笔/秒
平均下单延迟0.03秒
平均付款延迟0.022秒

book.py .../access	test_bench.p	pytest(93%).txt	bench_1.11.txt x	workload.py	book.py .../init_db
DB > fe > bench > bench_1.11.txt					
936	INFO:root:TPS_C=17694, NO=0K:440596	Thread_num:943	TOTAL:440596	LATENCY:0.030601230932516463	, P=0K:439551 Thread_num:942 TOTAL:439653 LATENCY:0.022717137404657962
937	INFO:root:TPS_C=17712, NO=0K:441540	Thread_num:944	TOTAL:441540	LATENCY:0.03060174823298685	, P=0K:440487 Thread_num:943 TOTAL:440596 LATENCY:0.022717298422421733
938	INFO:root:TPS_C=17731, NO=0K:442485	Thread_num:945	TOTAL:442485	LATENCY:0.030602238753359833	, P=0K:441424 Thread_num:944 TOTAL:441540 LATENCY:0.02271746072334165
939	INFO:root:TPS_C=17749, NO=0K:443431	Thread_num:946	TOTAL:443431	LATENCY:0.03060272986967315	, P=0K:442362 Thread_num:945 TOTAL:442485 LATENCY:0.02271762344192413
940	INFO:root:TPS_C=17768, NO=0K:444378	Thread_num:947	TOTAL:444378	LATENCY:0.030603244500927464	, P=0K:443300 Thread_num:946 TOTAL:443431 LATENCY:0.022717775594270135
941	INFO:root:TPS_C=17787, NO=0K:445326	Thread_num:948	TOTAL:445326	LATENCY:0.03060378031089631	, P=0K:444239 Thread_num:947 TOTAL:444378 LATENCY:0.022717930462607946
942	INFO:root:TPS_C=17805, NO=0K:446275	Thread_num:949	TOTAL:446275	LATENCY:0.030604290934171566	, P=0K:445179 Thread_num:948 TOTAL:445326 LATENCY:0.022718081081849333
943	INFO:root:TPS_C=17824, NO=0K:447225	Thread_num:950	TOTAL:447225	LATENCY:0.03060481539433102	, P=0K:446120 Thread_num:949 TOTAL:446275 LATENCY:0.0227182317150055
944	INFO:root:TPS_C=17842, NO=0K:448176	Thread_num:951	TOTAL:448176	LATENCY:0.030605319197560654	, P=0K:447061 Thread_num:950 TOTAL:447225 LATENCY:0.022718371066581183
945	INFO:root:TPS_C=17861, NO=0K:449128	Thread_num:952	TOTAL:449128	LATENCY:0.030605818066959067	, P=0K:448003 Thread_num:951 TOTAL:448176 LATENCY:0.02271851024746933
946	INFO:root:TPS_C=17879, NO=0K:450081	Thread_num:953	TOTAL:450081	LATENCY:0.03060629848176536	, P=0K:448945 Thread_num:952 TOTAL:449128 LATENCY:0.022718639048029895
947	INFO:root:TPS_C=17898, NO=0K:451035	Thread_num:954	TOTAL:451035	LATENCY:0.030606777851662475	, P=0K:449888 Thread_num:953 TOTAL:450081 LATENCY:0.02271877016824302
948	INFO:root:TPS_C=17916, NO=0K:451990	Thread_num:955	TOTAL:451990	LATENCY:0.030607237250180514	, P=0K:450832 Thread_num:954 TOTAL:451035 LATENCY:0.022718901135584833
949	INFO:root:TPS_C=17935, NO=0K:452946	Thread_num:956	TOTAL:452946	LATENCY:0.030607677992309498	, P=0K:451777 Thread_num:955 TOTAL:451990 LATENCY:0.022719032762483115
950	INFO:root:TPS_C=17953, NO=0K:453903	Thread_num:957	TOTAL:453903	LATENCY:0.030608149671242663	, P=0K:452723 Thread_num:956 TOTAL:452946 LATENCY:0.02271916402852052
951	INFO:root:TPS_C=17972, NO=0K:454861	Thread_num:958	TOTAL:454861	LATENCY:0.030608619346086025	, P=0K:453670 Thread_num:957 TOTAL:453903 LATENCY:0.022719293656618757
952	INFO:root:TPS_C=17991, NO=0K:455820	Thread_num:959	TOTAL:455820	LATENCY:0.030609079613659874	, P=0K:454618 Thread_num:958 TOTAL:454861 LATENCY:0.022719426832311665
953	INFO:root:TPS_C=18009, NO=0K:456780	Thread_num:960	TOTAL:456780	LATENCY:0.030609556172794865	, P=0K:455567 Thread_num:959 TOTAL:455820 LATENCY:0.022719562147307121
954	INFO:root:TPS_C=18028, NO=0K:457741	Thread_num:961	TOTAL:457741	LATENCY:0.03061005134506849	, P=0K:456516 Thread_num:960 TOTAL:456780 LATENCY:0.022719685042445717
955	INFO:root:TPS_C=18046, NO=0K:458703	Thread_num:962	TOTAL:458703	LATENCY:0.030610545237288907	, P=0K:457466 Thread_num:961 TOTAL:457741 LATENCY:0.0227198326146417
956	INFO:root:TPS_C=18065, NO=0K:459666	Thread_num:963	TOTAL:459666	LATENCY:0.030611020904914264	, P=0K:458417 Thread_num:962 TOTAL:458703 LATENCY:0.02271997921680498
957	INFO:root:TPS_C=18083, NO=0K:460630	Thread_num:964	TOTAL:460630	LATENCY:0.030611510365613288	, P=0K:459369 Thread_num:963 TOTAL:459666 LATENCY:0.022720129575179698
958	INFO:root:TPS_C=18102, NO=0K:461595	Thread_num:965	TOTAL:461595	LATENCY:0.030612021113709018	, P=0K:460321 Thread_num:964 TOTAL:460630 LATENCY:0.022720266400381878
959	INFO:root:TPS_C=18120, NO=0K:462561	Thread_num:966	TOTAL:462561	LATENCY:0.03061253441176155	, P=0K:461273 Thread_num:965 TOTAL:461595 LATENCY:0.02272039071429348
960	INFO:root:TPS_C=18139, NO=0K:463528	Thread_num:967	TOTAL:463528	LATENCY:0.030613056812437638	, P=0K:462226 Thread_num:966 TOTAL:462561 LATENCY:0.0227205146730371
961	INFO:root:TPS_C=18157, NO=0K:464496	Thread_num:968	TOTAL:464496	LATENCY:0.03061358415591007	, P=0K:463180 Thread_num:967 TOTAL:463528 LATENCY:0.02272063646801093
962	INFO:root:TPS_C=18176, NO=0K:465465	Thread_num:969	TOTAL:465465	LATENCY:0.030614121497831805	, P=0K:464134 Thread_num:968 TOTAL:464496 LATENCY:0.022720749930457362
963	INFO:root:TPS_C=18194, NO=0K:466435	Thread_num:970	TOTAL:466435	LATENCY:0.030614645731803482	, P=0K:465089 Thread_num:969 TOTAL:465465 LATENCY:0.02272086329686693
964	INFO:root:TPS_C=18213, NO=0K:467406	Thread_num:971	TOTAL:467406	LATENCY:0.0306151789489519	, P=0K:466045 Thread_num:970 TOTAL:466435 LATENCY:0.022720977372094837
965	INFO:root:TPS_C=18231, NO=0K:468378	Thread_num:972	TOTAL:468378	LATENCY:0.030615722904685276	, P=0K:467002 Thread_num:971 TOTAL:467406 LATENCY:0.02272108930834725
966	INFO:root:TPS_C=18250, NO=0K:469351	Thread_num:973	TOTAL:469351	LATENCY:0.030616242896089444	, P=0K:467960 Thread_num:972 TOTAL:468378 LATENCY:0.02272120212095436
967	INFO:root:TPS_C=18269, NO=0K:470325	Thread_num:974	TOTAL:470325	LATENCY:0.030616758202388704	, P=0K:468919 Thread_num:973 TOTAL:469351 LATENCY:0.022721324070535766
968	INFO:root:TPS_C=18287, NO=0K:471300	Thread_num:975	TOTAL:471300	LATENCY:0.030617250396065616	, P=0K:469879 Thread_num:974 TOTAL:470325 LATENCY:0.022721444651415578
969	INFO:root:TPS_C=18306, NO=0K:472276	Thread_num:976	TOTAL:472276	LATENCY:0.030617750978121316	, P=0K:470840 Thread_num:975 TOTAL:471300 LATENCY:0.022721566347566958
970	INFO:root:TPS_C=18324, NO=0K:473253	Thread_num:977	TOTAL:473253	LATENCY:0.030618259967036803	, P=0K:471801 Thread_num:976 TOTAL:472276 LATENCY:0.022721677699770487
971	INFO:root:TPS_C=18343, NO=0K:474231	Thread_num:978	TOTAL:474231	LATENCY:0.030618766422138562	, P=0K:472762 Thread_num:977 TOTAL:473253 LATENCY:0.022721779458483044
972	INFO:root:TPS_C=18361, NO=0K:475210	Thread_num:979	TOTAL:475210	LATENCY:0.030619255825790958	, P=0K:473724 Thread_num:978 TOTAL:474231 LATENCY:0.022721880968607885
973	INFO:root:TPS_C=18380, NO=0K:476190	Thread_num:980	TOTAL:476190	LATENCY:0.030619782838625344	, P=0K:474687 Thread_num:979 TOTAL:475210 LATENCY:0.022721981681664307
974	INFO:root:TPS_C=18398, NO=0K:477171	Thread_num:981	TOTAL:477171	LATENCY:0.03062031314996335	, P=0K:475651 Thread_num:980 TOTAL:476190 LATENCY:0.022722079749920365
975	INFO:root:TPS_C=18417, NO=0K:478153	Thread_num:982	TOTAL:478153	LATENCY:0.03062082984379053	, P=0K:476615 Thread_num:981 TOTAL:477171 LATENCY:0.022722176922341444
976	INFO:root:TPS_C=18435, NO=0K:479136	Thread_num:983	TOTAL:479136	LATENCY:0.03062136053485669	, P=0K:477580 Thread_num:982 TOTAL:478153 LATENCY:0.022722385318947727
977	INFO:root:TPS_C=18454, NO=0K:480120	Thread_num:984	TOTAL:480120	LATENCY:0.030621887313557854	, P=0K:478545 Thread_num:983 TOTAL:479136 LATENCY:0.02272258424911376
978	INFO:root:TPS_C=18472, NO=0K:481105	Thread_num:985	TOTAL:481105	LATENCY:0.030622427486401798	, P=0K:479511 Thread_num:984 TOTAL:480120 LATENCY:0.022722783291686487
979	INFO:root:TPS_C=18491, NO=0K:482091	Thread_num:986	TOTAL:482091	LATENCY:0.030622954607370415	, P=0K:480477 Thread_num:985 TOTAL:481105 LATENCY:0.02272297175540786
980	INFO:root:TPS_C=18509, NO=0K:483078	Thread_num:987	TOTAL:483078	LATENCY:0.030623466989689523	, P=0K:481444 Thread_num:986 TOTAL:482091 LATENCY:0.02272316353258332
981	INFO:root:TPS_C=18528, NO=0K:484066	Thread_num:988	TOTAL:484066	LATENCY:0.030623971605953215	, P=0K:482412 Thread_num:987 TOTAL:483078 LATENCY:0.022723356941955988
982	INFO:root:TPS_C=18546, NO=0K:485055	Thread_num:989	TOTAL:485055	LATENCY:0.0306244755033092	, P=0K:483381 Thread_num:988 TOTAL:484066 LATENCY:0.022723550524295536
983	INFO:root:TPS_C=18565, NO=0K:486045	Thread_num:990	TOTAL:486045	LATENCY:0.030624991607952484	, P=0K:484350 Thread_num:989 TOTAL:485055 LATENCY:0.02272373268906102
984	INFO:root:TPS_C=18583, NO=0K:487036	Thread_num:991	TOTAL:487036	LATENCY:0.030625521357846505	, P=0K:485320 Thread_num:990 TOTAL:486045 LATENCY:0.02272391440239055
985	INFO:root:TPS_C=18602, NO=0K:488028	Thread_num:992	TOTAL:488028	LATENCY:0.030626048520051685	, P=0K:486291 Thread_num:991 TOTAL:487036 LATENCY:0.02272409512205026
986	INFO:root:TPS_C=18620, NO=0K:489021	Thread_num:993	TOTAL:489021	LATENCY:0.030626596952791244	, P=0K:487262 Thread_num:992 TOTAL:488028 LATENCY:0.02272426575793329
987	INFO:root:TPS_C=18639, NO=0K:490015	Thread_num:994	TOTAL:490015	LATENCY:0.030627138399135608	, P=0K:488233 Thread_num:993 TOTAL:489021 LATENCY:0.022724426793058413
988	INFO:root:TPS_C=18657, NO=0K:491010	Thread_num:995	TOTAL:491010	LATENCY:0.030627699397090535	, P=0K:489204 Thread_num:994 TOTAL:490015 LATENCY:0.022724578725624228
989	INFO:root:TPS_C=18676, NO=0K:492006	Thread_num:996	TOTAL:492006	LATENCY:0.030628266143335952	, P=0K:490175 Thread_num:995 TOTAL:491010 LATENCY:0.02272472092533366
990	INFO:root:TPS_C=18694, NO=0K:493003	Thread_num:997	TOTAL:493003	LATENCY:0.03062882087160187	, P=0K:491146 Thread_num:996 TOTAL:492006 LATENCY:0.022724852580539353
991	INFO:root:TPS_C=18713, NO=0K:494001	Thread_num:998	TOTAL:494001	LATENCY:0.030629422013524442	, P=0K:492118 Thread_num:997 TOTAL:493003 LATENCY:0.022724982620757433
992	INFO:root:TPS_C=18731, NO=0K:495000	Thread_num:999	TOTAL:495000	LATENCY:0.030630044592269742	, P=0K:493090 Thread_num:998 TOTAL:494001 LATENCY:0.022725100348845527
993	INFO:root:TPS_C=18750, NO=0K:496000	Thread_num:1000	TOTAL:496000	LATENCY:0.030630671510292636	, P=0K:494063 Thread_num:999 TOTAL:495000 LATENCY:0.02272521724797258

开发流程

Excluding merges, **3 authors** have pushed **20 commits** to master and **20 commits** to all branches. On master, **0 files** have changed and there have been **0 additions** and **0 deletions**.



9 Pull requests merged by 2 people

add auto cancel

#13 merged 2 hours ago

数据库增加cancel表 history cancel对应修改

#12 merged 2 days ago

add_book的加书功能删除 需运行book.py加书 book属性增加picture 目前覆盖率90%

#11 merged 2 days ago

add_book search_history完善 优化数据库结构

#10 merged 3 days ago

手动取消订单 search history 完善add_book用户输入price 接口

#9 merged 3 days ago

add deliver and receive book

#8 merged 4 days ago

ORM完善 基础功能全部测试通过

#7 merged 4 days ago

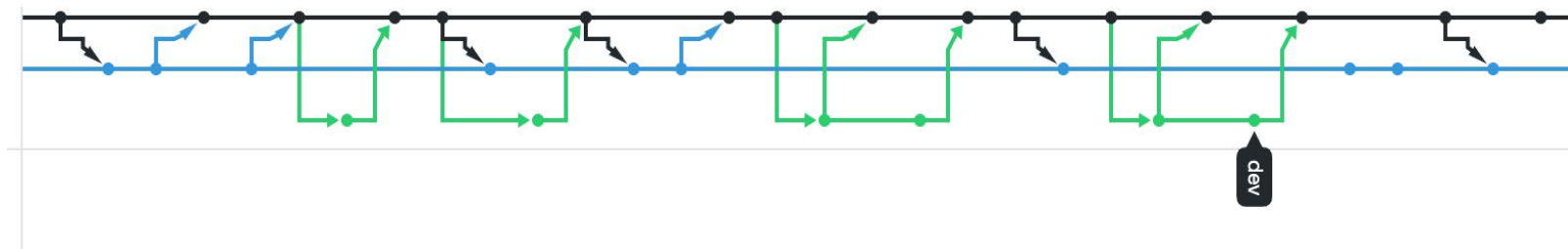
orm

#6 merged 4 days ago

test完善

#5 merged 5 days ago

- 版本控制
- 测试驱动开发
- 应用驱动优化(表结构的修改与完善)
如：在new_order_detail中新增buyer_id/store_id,
新增用户订单canel表



感谢聆听

成员：池欣宁，王文清，何诗雨