一、数据库基本表定义

本系统数据库中共包含8个基本表，具体定义如下：

1.1 用户表 (users)

属性名	中文	数据类型	备注
user_name	用户名	varchar(32)	主键
password_md5	密码	varchar(32)	MD5加密
nickname	昵称	varchar(32)	
avatar_url	头像链接	varchar(256)	
phone	手机号	varchar(32)	
intro	简介	text	
create_time	创建时间	timestamp	

1.2 商品表 (products)

属性名	中文	数据类型	备注
product_id	商品ID	varchar(32)	主键
product_title	商品标题	varchar(32)	
description	商品描述	text	
img_url	图片链接	varchar(256)	
price	价格	decimal(10,2)	
status	状态	varchar(32)	active/sold/deleted
owner_id	卖家ID	varchar(32)	外键 (关联users)
category_id	分类ID	varchar(32)	外键 (关联categories)
create_time	发布时间	timestamp	
update_time	更新时间	timestamp	

1.3 订单表 (orders)

属性名	中文	数据类型	备注
order_id	订单ID	varchar(32)	主键
order_status	订单状态	varchar(32)	
buyer_id	买家ID	varchar(32)	外键 (关联users)

属性名	中文	数据类型	备注
seller_id	卖家ID	varchar(32)	外键 (关联users)
product_id	商品ID	varchar(32)	外键 (关联products)
created_time	下单时间	timestamp	

1.4 消息表 (message)

属性名	中文	数据类型	备注
message_id	消息ID	varchar(32)	主键
sender_id	发送者ID	varchar(32)	外键 (关联users)
receiver_id	接收者ID	varchar(32)	外键 (关联users)
time	发送时间	timestamp	
content	消息内容	text	

1.5 评论表 (comment)

属性名	中文	数据类型	备注
comment_id	评论ID	varchar(32)	主键
user_id	评论者ID	varchar(32)	外键 (关联users)
product_id	商品ID	varchar(32)	外键 (关联products)
time	评论时间	timestamp	
rating	评分	int unsigned	
content	评论内容	text	

1.6 商品分类表 (categories)

属性名	中文	数据类型	备注
category_id	分类ID	varchar(32)	主键
category_name	分类名称	varchar(32)	

1.7 收藏夹表 (favorites)

属性名	中文	数据类型	备注
favorite_id	收藏夹ID	varchar(32)	主键
user_id	用户ID	varchar(32)	外键 (关联users)
created_time	创建时间	timestamp	

属性名	中文	数据类型	备注
name	收藏夹名称	varchar(64)	

1.8 收藏项表 (favorite_item)

属性名	中文	数据类型	备注
favorite_id	收藏夹ID	varchar(32)	外键 (关联favorites)
product_id	商品ID	varchar(32)	外键 (关联products)
created_time	收藏时间	timestamp	

注： favorite_item 表的主键为 (favorite_id, product_id) 的组合主键

二、 触发器与存储过程的设计与实现说明

2.1 触发器设计与实现说明

本项目为了保证数据的一致性与完整性，在数据库层面设计了触发器。这些触发器有效减少了后端代码的复杂度，同时确保了关键业务逻辑的原子性。

2.1.1 订单与商品状态联动触发器

- 触发时机：`orders` 表发生 `INSERT` 操作之后。
- 功能描述：当用户下单生成新的订单记录时，触发器自动根据订单中的 `product_id`，将 `products` 表中对应商品的状态 (`status`) 更新为 `'sold'` (已售出)。
- 设计意义：确保“下单”与“下架”这两个动作的原子性，防止在高并发场景下出现“一货多卖”的情况。

```
DELIMITER $$

CREATE TRIGGER `after_order_insert`
AFTER INSERT ON `orders`
FOR EACH ROW
BEGIN
    -- 当生成新订单时，自动将对应商品标记为已售出
    UPDATE products
    SET status = 'sold'
    WHERE product_id = NEW.product_id;
END$$

DELIMITER ;
```

2.1.2 收藏夹与评论级联清理触发器

- **触发时机**：`products` 表发生 `UPDATE` 操作之后。
- **功能描述**：本项目采用“软删除”策略。当商品状态变为 `'deleted'` 时，该触发器会执行两步清理操作：
    1. 从 `favorite_item` 表中删除所有关联该商品的收藏记录。

2. 从 `comment` 表中删除该商品下的所有评论。

- **设计意义**：维护数据的整洁性，确保商品“下架”后，与其相关的交互数据（收藏、评论）同步失效

```
DELIMITER $$

CREATE TRIGGER `after_product_delete`
AFTER UPDATE ON `products`
FOR EACH ROW
BEGIN
    IF NEW.status = 'deleted' THEN
        DELETE FROM favorite_item WHERE product_id = NEW.product_id;
        DELETE FROM comment WHERE product_id = NEW.product_id;
    END IF;
END$$

DELIMITER ;
```

2.1.3 数据更新时间自动维护触发器

- 触发时机：`products` 表发生 `UPDATE` 操作之前 。
- 功能描述：每当商品信息（如价格、描述、标题）被修改时，触发器自动将当前系统时间赋值给 `update_time` 字段。
- 设计意义：自动化维护审计字段，无需后端代码显式传入时间，确保 `update_time` 永远反映数据的最后一次变动时间。

```
DELIMITER $$

CREATE TRIGGER `before_product_update`
BEFORE UPDATE ON `products`
FOR EACH ROW
BEGIN
    -- 自动更新最后修改时间为当前时间
    SET NEW.update_time = NOW();
END$$

DELIMITER ;
```

2.1.4 收藏夹删除前置清理触发器

- **触发时机**：`favorites` 表发生 `DELETE` 操作之前。
- **功能描述**：当用户删除一个收藏夹时，该触发器会在物理删除文件夹记录之前，先自动清空 `favorite_item` 表中属于该文件夹的所有收藏项。
- **设计意义**：实现逻辑上的级联删除。由于 `favorite_item` 表依赖于 `favorites` 表的主键，如果直接删除文件夹会导致外键约束冲突或产生“孤儿数据”。该触发器保证了删除操作的原子性和数据的完整性。

- **实现代码**:

```
DELIMITER $$

CREATE TRIGGER `before_favorite_delete`
BEFORE DELETE ON `favorites`
FOR EACH ROW
BEGIN
    DELETE FROM favorite_item WHERE favorite_id = OLD.favorite_id;
END$$

DELIMITER ;
```

2.2 存储过程设计与实现说明

本系统后端采用 Python 的 Flask 框架，利用 PyMySQL 库与 MySQL 数据库进行交互。数据存取逻辑封装在后端的各个处理函数中。系统通过定义 API 接口接收前端请求，在后端执行 SQL 语句完成业务逻辑，并将结果以 JSON 格式返回。

2.2.1 用户模块

(1) 用户注册

- **请求路径**: `/api/register`
- **请求方法**: `POST`
- **请求数据格式**:
  - `username`: 字符串格式，用户设置的登录账号。
  - `password`: 字符串格式，用户设置的密码。
- **返回数据格式**: JSON 对象，包含 `message` 提示信息。

实现代码:

```
@auth_bp.route("/register", methods=["POST"])
def register():
    data = request.json
    user_name = data.get("username")
    password = data.get("password")

    conn = get_db_connection()
    cursor = conn.cursor()
    try:
        # 检查用户是否已存在
        cursor.execute("SELECT user_name FROM users WHERE user_name = %s",
(user_name,))
        if cursor.fetchone():
            return jsonify({"message": "用户名已存在"}), 409

        # 插入新用户
        hashed_password = md5(password)
```

```
        cursor.execute("""
            INSERT INTO users (user_name, password_md5, create_time)
            VALUES (%s, %s, NOW())
        """, (user_name, hashed_password))
        conn.commit()
        return jsonify({"message": "注册成功"}), 201
    except Exception as e:
        conn.rollback()
        return jsonify({"message": "服务器内部错误"}), 500
    finally:
        cursor.close()
        conn.close()
```

(2) 用户登录

- **请求路径**: `/api/login`
- **请求方法**: `POST`
- **请求数据格式**:
  - `username`: 字符串格式，登录账号。
  - `password`: 字符串格式，登录密码。
- **返回数据格式**:
  - `token`: 字符串格式，用于身份验证的令牌。
  - `message`: 操作结果提示。

实现代码:

```
@auth_bp.route("/login", methods=["POST"])
def login():
    data = request.json
    user_name = data.get("username")
    password = data.get("password")

    conn = get_db_connection()
    cursor = conn.cursor(pymysql.cursors.DictCursor)
    try:
        # 验证用户名与密码
        cursor.execute("SELECT user_name, password_md5 FROM users WHERE user_name
= %s", (user_name,))
        user = cursor.fetchone()

        if user and md5(password) == user["password_md5"]:
            token = generate_token(user_name)
            return jsonify({"token": token, "message": "登录成功"}), 200
        else:
            return jsonify({"message": "用户名或密码错误"}), 401
    finally:
        cursor.close()
        conn.close()
```

2.2.2 商品模块

(1) 发布商品

- **请求路径**: `/api/create_product`
- **请求方法**: `POST`
- **请求数据格式**:
  - `name`: 字符串，商品标题。
  - `price`: 浮点数，商品价格。
  - `category_id`: 字符串，商品所属分类ID。
  - `description`: 字符串，商品描述。
  - `image_url`: 字符串，图片路径。
- **返回数据格式**: 包含 `product_id` 和成功信息。

实现代码:

```
@product_bp.route("/create_product", methods=["POST"])
def create_product():
    # Token 验证略...
    data = request.json
    new_id = generate_uuid()

    conn = get_db_connection()
    cursor = conn.cursor()
    try:
        sql = """
            INSERT INTO products
            (product_id, product_title, price, img_url, description, category_id,
owner_id, status, create_time, update_time)
            VALUES (%s, %s, %s, %s, %s, %s, %s, 'active', NOW(), NOW())
        """
        cursor.execute(sql, (new_id, data.get("name"), data.get("price"),
                             data.get("image_url"), data.get("description"),
                             data.get("category_id"), user_name))
        conn.commit()
        return jsonify({"message": "商品发布成功", "product_id": new_id}), 201
    finally:
        cursor.close()
        conn.close()
```

(2) 商品搜索

- **请求路径**: `/api/search_products`
- **请求方法**: `GET`
- **请求参数**: `keyword` (URL参数)，表示搜索关键词。
- **返回数据格式**: `products` 数组，包含匹配的商品列表。

实现代码:

```
@product_bp.route("/search_products", methods=["GET"])
def search_products():
    keyword = request.args.get("keyword", "").strip()
    conn = get_db_connection()
    cursor = conn.cursor(pymysql.cursors.DictCursor)
    try:
        # 使用 LIKE 进行模糊查询
        sql = """
            SELECT p.*, u.nickname as seller_name, u.avatar_url as seller_avatar
            FROM products p
            LEFT JOIN users u ON p.owner_id = u.user_name
            WHERE p.status = 'active'
            AND (p.product_title LIKE %s OR p.description LIKE %s)
            ORDER BY p.create_time DESC
        """
        search_pattern = f"%{keyword}%"
        cursor.execute(sql, (search_pattern, search_pattern))
        products = cursor.fetchall()
        # 数据格式化略...
        return jsonify({"products": result_list, "message": "搜索完成"}), 200
    finally:
        cursor.close()
        conn.close()
```

(3) 修改商品信息

- **请求路径**: `/api/modify_product`
- **请求方法**: `POST`
- **功能说明**: 允许卖家修改已发布商品的信息（如价格、描述）。后端会校验操作者是否为该商品的拥有者（Owner）。
- **返回数据格式**: 成功/失败提示。

实现代码:

```
@product_bp.route("/modify_product", methods=["POST"])
def modify_product():
    # Token 验证略
    data = request.json
    product_id = data.get("id")

    conn = get_db_connection()
    cursor = conn.cursor()
    try:
        sql = """
            UPDATE products
            SET product_title = %s, price = %s, img_url = %s, description = %s,
category_id = %s, update_time = NOW()
            WHERE product_id = %s AND owner_id = %s
        """
        affected = cursor.execute(sql, (data.get("name"), data.get("price"),
```

```
data.get("image_url"),
                                        data.get("description"),
data.get("category_id"),
                                        product_id, user_name))
        conn.commit()
        if affected == 0:
            return jsonify({"message": "修改失败：无权操作或商品不存在"}), 403
        return jsonify({"message": "商品修改成功"}), 200
    finally:
        cursor.close()
        conn.close()
```

(4) 删除商品 (软删除)

- **请求路径**: /api/delete_product/<product_id>
- **请求方法**: DELETE
- **功能说明**: 采用"软删除"策略，不物理删除记录，而是将状态标记为 deleted，以保留历史订单数据。
- **返回数据格式**: 成功/失败提示。 **实现代码**:

```
@product_bp.route("/delete_product/<product_id>", methods=["DELETE"])
def delete_product(product_id):
    conn = get_db_connection()
    cursor = conn.cursor()
    try:
        sql = "UPDATE products SET status = 'deleted', update_time = NOW() WHERE
product_id = %s AND owner_id = %s"
        affected = cursor.execute(sql, (product_id, user_name))
        conn.commit()

        if affected == 0:
            return jsonify({"message": "删除失败"}), 403
        return jsonify({"message": "商品已删除"}), 200
    finally:
        cursor.close()
        conn.close()
```

2.2.3 订单模块

(1) 购买商品 (事务处理)

- **请求路径**: /api/buy_product/<product_id>
- **请求方法**: POST
- **功能说明**: 通过数据库事务保证订单生成与商品状态变更的一致性。
- **返回数据格式**: 包含生成的 order_id。

实现代码:

```
@order_bp.route("/buy_product/<product_id>", methods=["POST"])
def buy_product(product_id):
    # Token 验证略
    conn = get_db_connection()
    cursor = conn.cursor(pymysql.cursors.DictCursor)
    try:
        # 1. 检查商品状态
        cursor.execute("SELECT * FROM products WHERE product_id = %s",
(product_id,))
        product = cursor.fetchone()
        if not product or product['status'] != 'active':
            return jsonify({"message": "商品已售出或下架"}), 409

        # 2. 开启事务：插入订单
        order_id = generate_uuid()
        insert_sql = """
            INSERT INTO orders
            (order_id, order_status, buyer_id, seller_id, product_id,
created_time)
            VALUES (%s, 'completed', %s, %s, %s, NOW())
        """
        cursor.execute(insert_sql, (order_id, user_name, product['owner_id'],
product_id))

        conn.commit() # 提交事务
        return jsonify({"message": "购买成功", "order_id": order_id}), 200
    except Exception as e:
        conn.rollback() # 回滚
        return jsonify({"message": "交易失败"}), 500
    finally:
        cursor.close()
        conn.close()
```

(2) 获取我的订单列表

- **请求路径**: `/api/get_orders`
- **请求方法**: `GET`
- **功能说明**: 查询当前登录用户的所有订单，通过 `JOIN` 操作关联 `products` 表，同时返回商品的标题、图片等详细信息。
- **返回数据格式**: `orders` 数组。

实现代码:

```
@order_bp.route("/get_orders", methods=["GET"])
def get_orders():
    # Token 验证略
    conn = get_db_connection()
    cursor = conn.cursor(pymysql.cursors.DictCursor)
    try:
        sql = """
```

```
            SELECT o.order_id, o.order_status, o.created_time,
                   p.product_title, p.img_url, p.price, o.seller_id
            FROM orders o
            JOIN products p ON o.product_id = p.product_id
            WHERE o.buyer_id = %s
            ORDER BY o.created_time DESC
        """
        cursor.execute(sql, (user_name,))
        orders = cursor.fetchall()

        return jsonify({"orders": orders, "message": "获取成功"}), 200
    finally:
        cursor.close()
        conn.close()
```

2.2.4 互动模块

(1) 发布评论

- **请求路径**: `/api/publish_comment`
- **请求方法**: `POST`
- **请求数据格式**:
  - `product_id`: 被评论的商品ID。
  - `content`: 评论内容。
  - `rate`: 评分（1-5）。
- **返回数据格式**: 成功提示信息。

实现代码:

```
@interaction_bp.route("/publish_comment", methods=["POST"])
def publish_comment():
    # Token 验证略
    data = request.json
    conn = get_db_connection()
    cursor = conn.cursor()
    try:
        comment_id = generate_uuid()
        sql = """
            INSERT INTO comment
            (comment_id, user_id, product_id, time, rating, content)
            VALUES (%s, %s, %s, NOW(), %s, %s)
        """
        cursor.execute(sql, (comment_id, user_name, data.get("product_id"),
                             data.get("rate", 5), data.get("content")))
        conn.commit()
        return jsonify({"message": "评论发布成功"}), 201
    finally:
        cursor.close()
        conn.close()
```

(2) 获取收藏夹列表

- **请求路径**: `/api/get_favorites`
- **请求方法**: `GET`
- **功能说明**: 执行复杂的三表连接查询。先通过 `users` 找到 `favorites`，再通过 `favorite_item` 关联出具体的 `products` 信息。
- **返回数据格式**: `favorites` 数组（包含商品详情）。

实现代码:

```
@interaction_bp.route("/get_favorites", methods=["GET"])
def get_favorites():
    # Token 验证略
    conn = get_db_connection()
    cursor = conn.cursor(pymysql.cursors.DictCursor)
    try:
        sql = """
            SELECT p.* FROM favorites f
            JOIN favorite_item fi ON f.favorite_id = fi.favorite_id
            JOIN products p ON fi.product_id = p.product_id
            WHERE f.user_id = %s
        """
        cursor.execute(sql, (user_name,))
        favorites = cursor.fetchall()
        return jsonify({"favorites": favorites}), 200
    finally:
        cursor.close()
        conn.close()
```

(3) 获取私信列表

- **请求路径**: `/api/get_msgs`
- **请求方法**: `GET`
- **功能说明**: 获取当前用户的完整对话流。查询条件包含逻辑或（OR），即检索“我发送的”或者“我接收的”所有消息，并按时间倒序排列。
- **返回数据格式**: `messages` 数组。 **实现代码**:

```
@interaction_bp.route("/get_msgs", methods=["GET"])
def get_msgs():
    # Token 验证略...
    conn = get_db_connection()
    cursor = conn.cursor(pymysql.cursors.DictCursor)
    try:
        # 使用 OR 逻辑查询双向消息
        sql = """
            SELECT m.*, u.nickname as sender_nickname, u.avatar_url as
sender_avatar
            FROM message m
            JOIN users u ON m.sender_id = u.user_name
```

```
            WHERE m.receiver_id = %s OR m.sender_id = %s
            ORDER BY m.time DESC
        """
        cursor.execute(sql, (user_name, user_name))
        msgs = cursor.fetchall()
        return jsonify({"messages": msgs}), 200
    finally:
        cursor.close()
        conn.close()
```

(4) 删除评论

- **请求路径**: `/api/delete_comment/<comment_id>`
- **请求方法**: `DELETE`
- **功能说明**: 允许用户删除自己发布的评论。
- **返回数据格式**: 成功/失败提示。

实现代码:

```
@interaction_bp.route("/delete_comment/<comment_id>", methods=["DELETE"])
def delete_comment(comment_id):
    # Token 验证略
    conn = get_db_connection()
    cursor = conn.cursor(pymysql.cursors.DictCursor)
    try:
        cursor.execute("SELECT user_id FROM comment WHERE comment_id = %s",
(comment_id,))
        comment = cursor.fetchone()
        if not comment or comment['user_id'] != user_name:
            return jsonify({"message": "无权删除"}), 403

        cursor.execute("DELETE FROM comment WHERE comment_id = %s", (comment_id,))
        conn.commit()
        return jsonify({"message": "评论已删除"}), 200
    finally:
        cursor.close()
        conn.close()
```

2.2.5 收藏夹管理模块

本系统支持多收藏夹管理功能，用户可以创建自定义收藏夹，并将商品分类收藏。

(1) 创建与获取收藏夹

- **请求路径**: `/api/create_favorite_folder` (POST) / `/api/favorite_folders` (GET)
- **功能说明**: 用户可以创建自定义命名的收藏夹；系统支持按创建时间倒序展示用户的收藏夹列表。

实现代码 (创建收藏夹):

```
@interaction_bp.route("/create_favorite_folder", methods=["POST"])
def create_favorite_folder():
    # Token 验证略
    folder_name = request.json.get("name")
    folder_id = generate_uuid()

    conn = get_db_connection()
    cursor = conn.cursor()
    try:
        sql = "INSERT INTO favorites (favorite_id, user_id, name, created_time)
VALUES (%s, %s, %s, NOW())"
        cursor.execute(sql, (folder_id, user_name, folder_name))
        conn.commit()
        return jsonify({"message": "创建成功", "id": folder_id}), 201
    finally:
        cursor.close()
        conn.close()
```

(2) 收藏商品到指定文件夹

- **请求路径**: `/api/favorite_product`
- **请求方法**: `POST`
- **请求数据**: `product_id`, `folder_id`
- **功能说明**: 将商品关联到指定的收藏夹中。系统会检查是否重复收藏以保证幂等性。

实现代码:

```
@interaction_bp.route("/favorite_product", methods=["POST"])
def add_favorite_product():
    # Token 验证略
    data = request.json
    conn = get_db_connection()
    cursor = conn.cursor()
    try:
        cursor.execute("SELECT * FROM favorite_item WHERE favorite_id = %s AND
product_id = %s",
                       (data.get("folder_id"), data.get("product_id")))
        if cursor.fetchone():
            return jsonify({"message": "已收藏"}), 200

        sql = "INSERT INTO favorite_item (favorite_id, product_id, created_time)
VALUES (%s, %s, NOW())"
        cursor.execute(sql, (data.get("folder_id"), data.get("product_id")))
        conn.commit()
        return jsonify({"message": "收藏成功"}), 201
    finally:
        cursor.close()
        conn.close()
```

(3) 删除收藏夹

- **请求路径**: `/api/delete_favorite_folder/<folder_id>`
- **请求方法**: `DELETE`
- **功能说明**: 删除指定收藏夹及其所有包含的商品项。采用级联删除策略，确保数据完整性。
- **实现代码**:

```
@interaction_bp.route("/delete_favorite_folder/<folder_id>", methods=["DELETE"])
def delete_favorite_folder(folder_id):
    # Token 验证略
    conn = get_db_connection()
    cursor = conn.cursor()
    try:
        cursor.execute("SELECT * FROM favorites WHERE favorite_id = %s AND user_id
= %s", (folder_id, user_name))
        if not cursor.fetchone(): return jsonify({"message": "无权操作"}), 403

        cursor.execute("DELETE FROM favorite_item WHERE favorite_id = %s",
(folder_id,))
        cursor.execute("DELETE FROM favorites WHERE favorite_id = %s",
(folder_id,))
        conn.commit()
        return jsonify({"message": "收藏夹已删除"}), 200
    finally:
        cursor.close()
        conn.close()
```

三、系统各项功能数据库端操作主要代码与结果说明

本章节重点展示系统核心业务逻辑在数据库层面的实现。后端通过 PyMySQL 驱动执行 SQL 语句，实现了数据的增删改查及完整性维护。

3.1 发布新商品

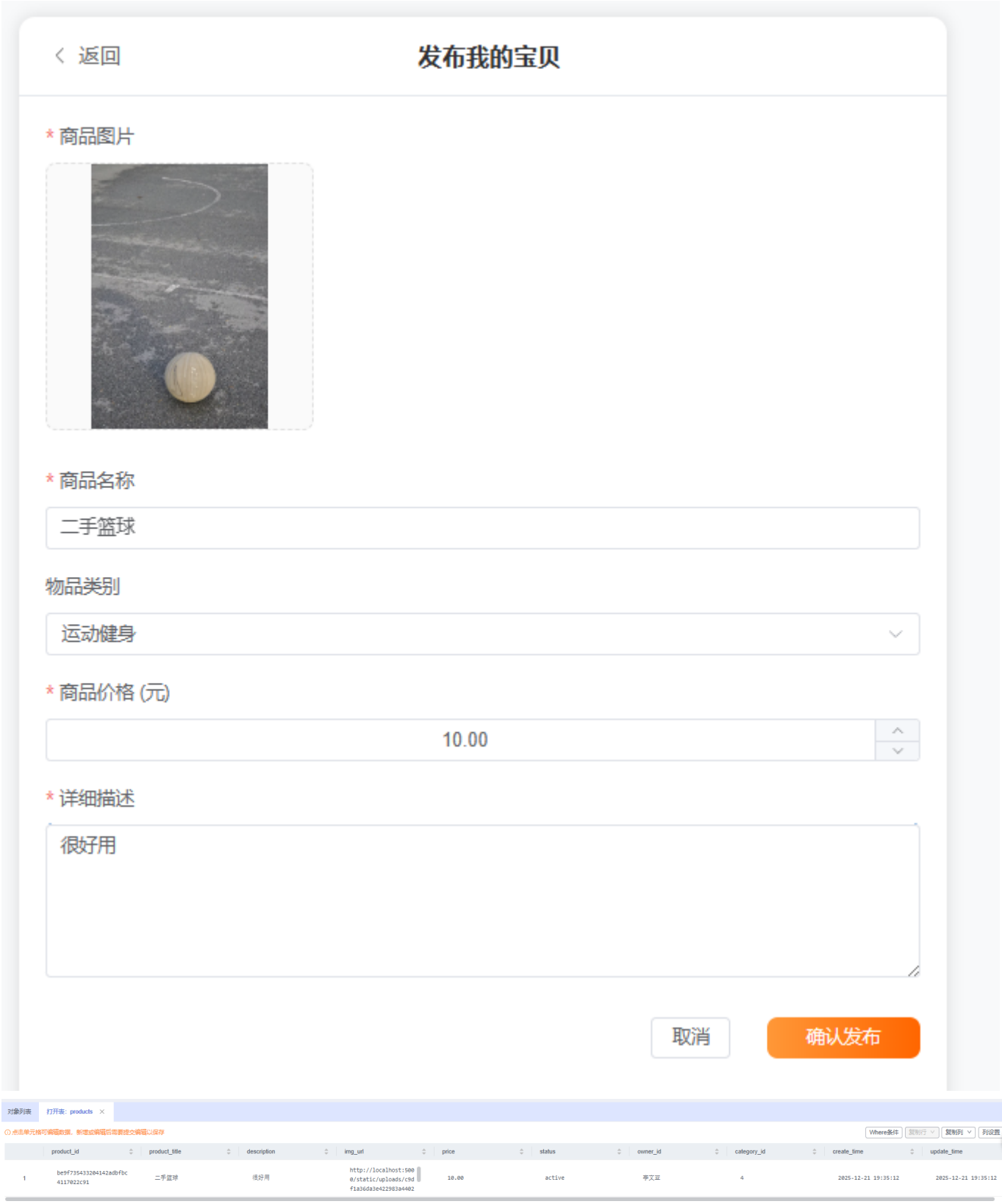
功能描述： 用户在前端填写商品信息（标题、价格、图片、描述）并提交。后端接收请求后，生成唯一的 UUID 作为 `product_id`，并将数据插入 `products` 表。同时，系统利用数据库函数 `NOW()` 自动记录商品的发布时间 (`create_time`) 和最后修改时间 (`update_time`)。

核心代码实现:

```
sql = """
    INSERT INTO products
    (product_id, product_title, price, img_url, description, category_id,
owner_id, status, create_time, update_time)
    VALUES (%s, %s, %s, %s, %s, %s, %s, 'active', NOW(), NOW())
"""
cursor.execute(sql, (new_id, data.get("name"), data.get("price"),
                     data.get("image_url"), data.get("description"),
```

```
                data.get("category_id"), user_name))
```

执行结果： 在前端“发布商品”页面填写信息并点击提交。



3.2 修改商品信息

功能描述： 卖家可以随时修改自己发布的商品信息。后端在执行 `UPDATE` 操作时，增加了 `WHERE owner_id = %s` 条件，确保只有商品的发布者才能修改该记录，实现了数据库层面的权限控制。同时，`update_time` 字段

会通过触发器或 SQL 语句自动更新为当前时间。

核心代码实现：

```
sql = """
    UPDATE products
    SET product_title = %s, price = %s, img_url = %s, description = %s,
category_id = %s, update_time = NOW()
    WHERE product_id = %s AND owner_id = %s
"""
cursor.execute(sql, (data.get("name"), data.get("price"), data.get("image_url"),
                     data.get("description"), data.get("category_id"),
                     product_id, user_name))
```

执行结果： 用户修改了某个商品的价格和描述。(在图片的例子里是上一个例子中的篮球)

修改后的数据库记录显示更新成功，且 `update_time` 字段已变更为当前时间。

3.3 商品模糊搜索

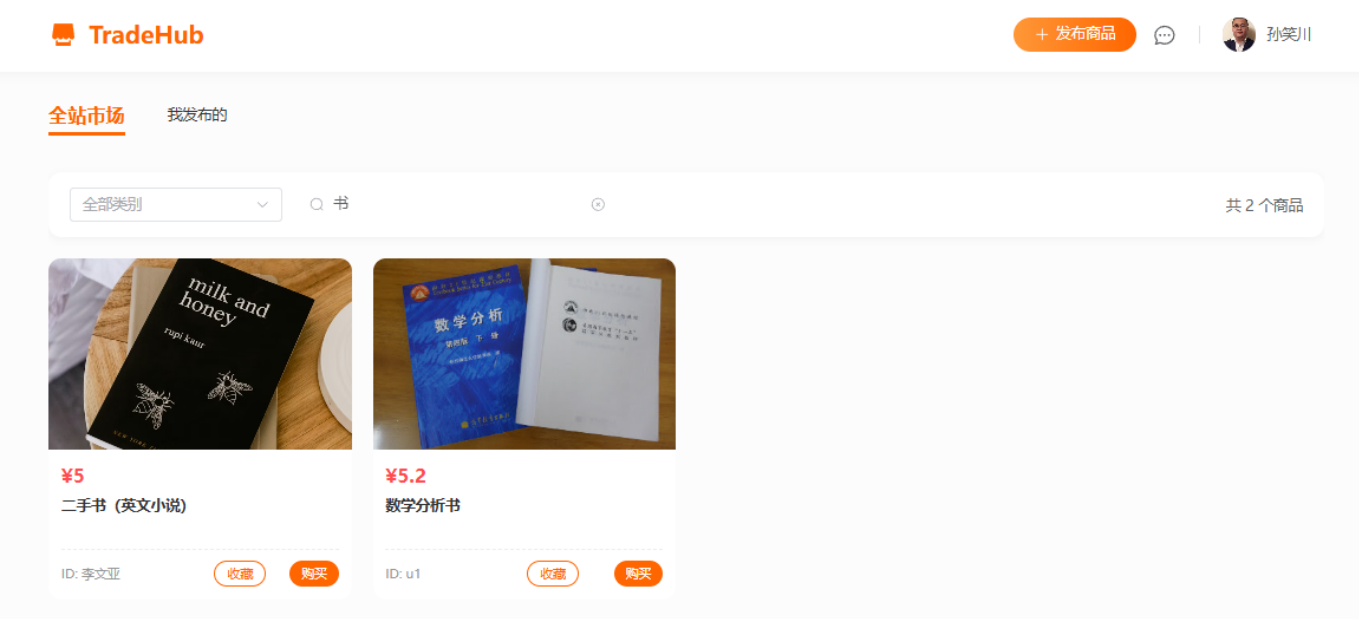
功能描述： 系统支持根据关键词搜索商品。为了在展示商品列表时能够同时显示卖家的头像和昵称，后端使用了 `LEFT JOIN` 将 `products` 表与 `users` 表关联。

核心代码实现：

```
sql = """
    SELECT p.*, u.nickname as seller_name, u.avatar_url as seller_avatar
```

```
    FROM products p
    LEFT JOIN users u ON p.owner_id = u.user_name
    WHERE p.status = 'active'
    AND (p.product_title LIKE %s OR p.description LIKE %s)
    ORDER BY p.create_time DESC
"""
search_pattern = f"%{keyword}%"
cursor.execute(sql, (search_pattern, search_pattern))
```

执行结果说明： 在前端搜索框输入关键词（例如“书”）。



4.4 删除商品

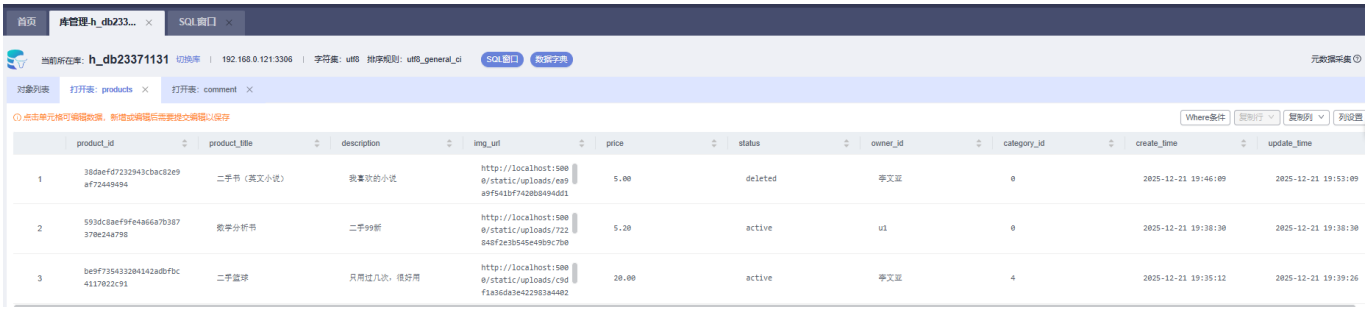
功能描述： 为了保留历史交易数据，商品删除采用“软删除”策略。后端执行 `UPDATE` 操作将商品状态标记为 `deleted`。同时，数据库触发器 `after_product_delete` 会自动监测此状态变化，一旦发现商品被删除，立即级联删除该商品在 `favorite_item` 表（收藏）和 `comment` 表（评论）中的所有记录，保证数据的整洁性。

核心代码实现：

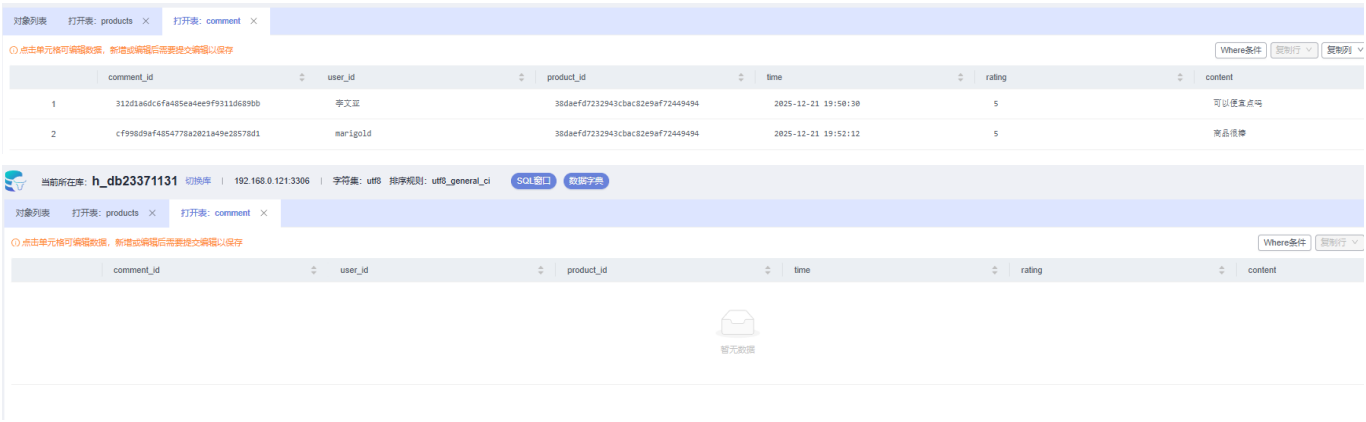
```
# SQL 逻辑：执行软删除，不物理删除记录
# 触发器联动：状态变为 'deleted' 后，触发器自动清理收藏和评论
sql = "UPDATE products SET status = 'deleted', update_time = NOW() WHERE
product_id = %s AND owner_id = %s"
affected = cursor.execute(sql, (product_id, user_name))
```

执行结果： 用户删除了一个已发布的商品。

删除后，该商品在数据库中的状态变为 `deleted`。



同时，触发器自动清理了该商品在收藏夹和评论表中的相关记录



3.5 购买商品

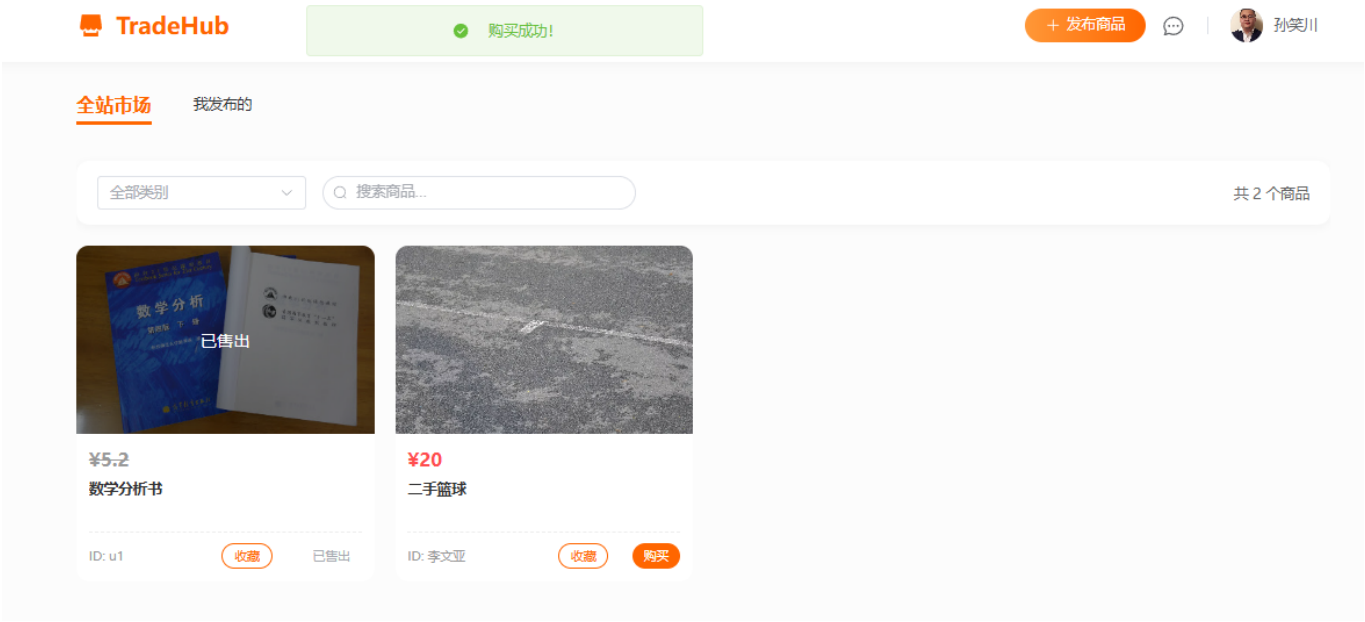
功能描述： 这是系统的核心交易功能。为了保证订单生成与商品下架的原子性，后端代码开启了数据库事务（Transaction）。操作仅需向 `orders` 表插入一条记录，数据库后置触发器 `after_order_insert` 会自动捕获该操作，并将 `products` 表中对应商品的状态更新为 `sold`，防止超卖。

核心代码实现：

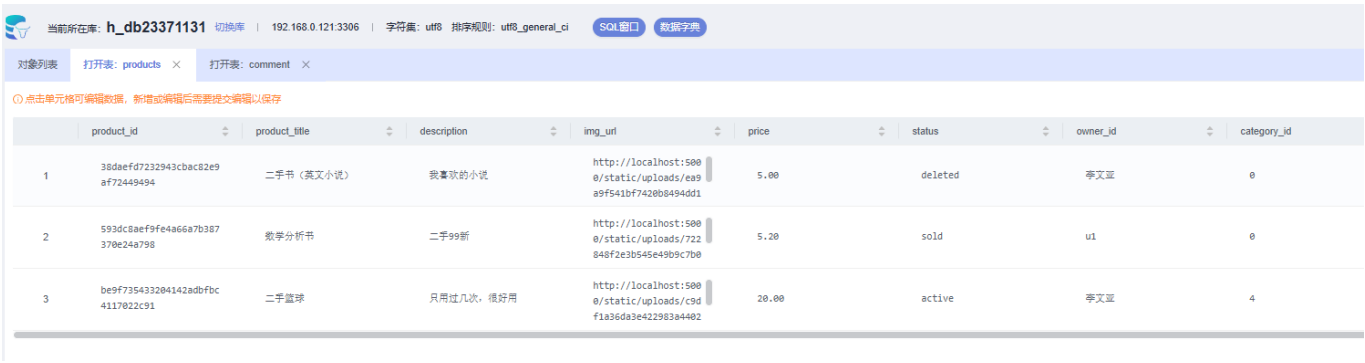
```
try:
    # 1. 开启事务，插入订单记录
    # Python端无需显式更新商品状态，完全交由数据库触发器处理
    insert_sql = """
        INSERT INTO orders
        (order_id, order_status, buyer_id, seller_id, product_id, created_time)
        VALUES (%s, 'completed', %s, %s, %s, NOW())
    """
    cursor.execute(insert_sql, (order_id, user_name, seller_id, product_id))

    # 2. 提交事务 (Commit)
    # 此时触发器被激活，自动锁定并更新商品状态为 'sold'
    conn.commit()
except Exception as e:
    conn.rollback()
```

执行结果： 在前端点击购买按钮。

前端显示已售出



数据库中对应商品状态变为 `sold`，订单记录成功插入。



3.6 查看我的订单

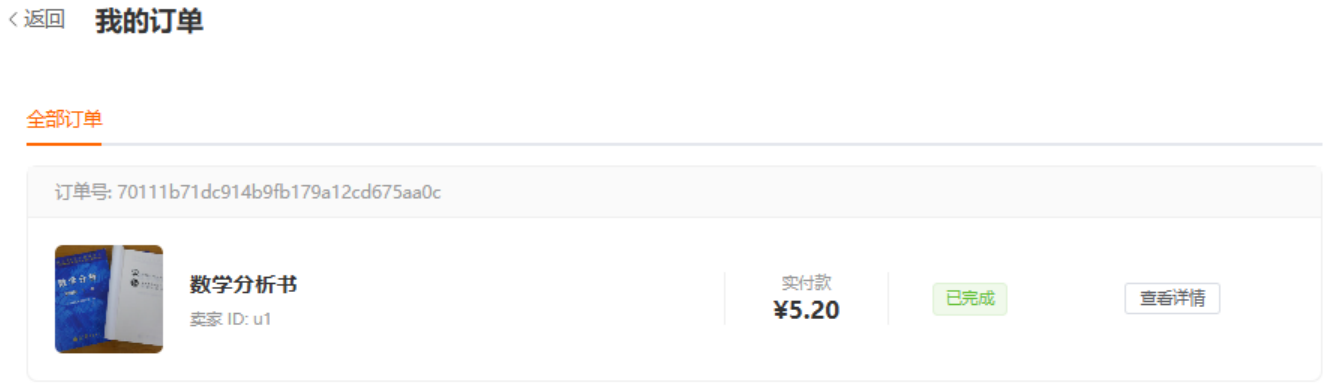
功能描述： 用户查看订单列表时，不仅需要知道订单号和状态，还需要看到所购买商品的详细信息（如图片、标题、价格）。后端使用 `JOIN` 语句将 `orders` 表与 `products` 表进行内连接，一次性获取所有必要数据，并按订单创建时间倒序排列。

核心代码实现：

```
sql = """
    SELECT o.order_id, o.order_status, o.created_time,
           p.product_title, p.img_url, p.price, o.seller_id
    FROM orders o
    JOIN products p ON o.product_id = p.product_id
    WHERE o.buyer_id = %s
```

```
    ORDER BY o.created_time DESC
"""
cursor.execute(sql, (user_name,))
```

执行结果： 用户进入“我的订单”页面。



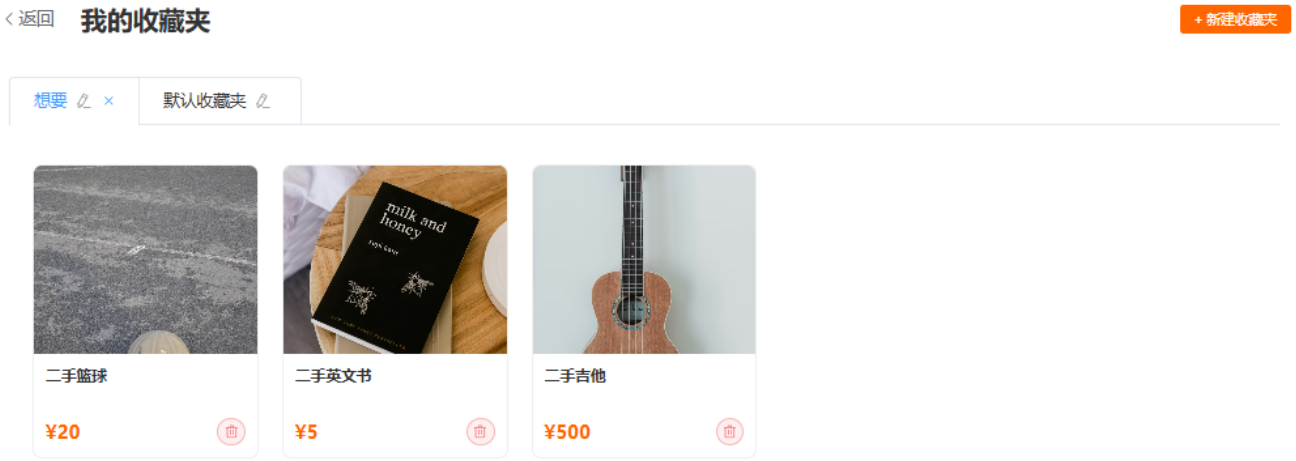
3.7 查看收藏详情

功能描述： 收藏系统采用“多文件夹”结构。为了展示某个特定收藏夹内的商品详情，后端需要执行复杂的 **三表连接查询**。路径为：从 `favorites` 表（确认归属）关联到 `favorite_item` 表（中间表），再关联到 `products` 表（获取商品具体信息），最后按收藏时间倒序排列。

核心代码实现：

```
sql = """
    SELECT p.product_id, p.product_title as name, p.price, p.img_url,
fi.created_time
    FROM favorites f
    JOIN favorite_item fi ON f.favorite_id = fi.favorite_id
    JOIN products p ON fi.product_id = p.product_id
    WHERE f.favorite_id = %s AND f.user_id = %s
    ORDER BY fi.created_time DESC
"""
cursor.execute(sql, (folder_id, user_name))
```

执行结果说明： 用户点击进入名为“考研资料”的收藏夹。



3.8 删除收藏夹

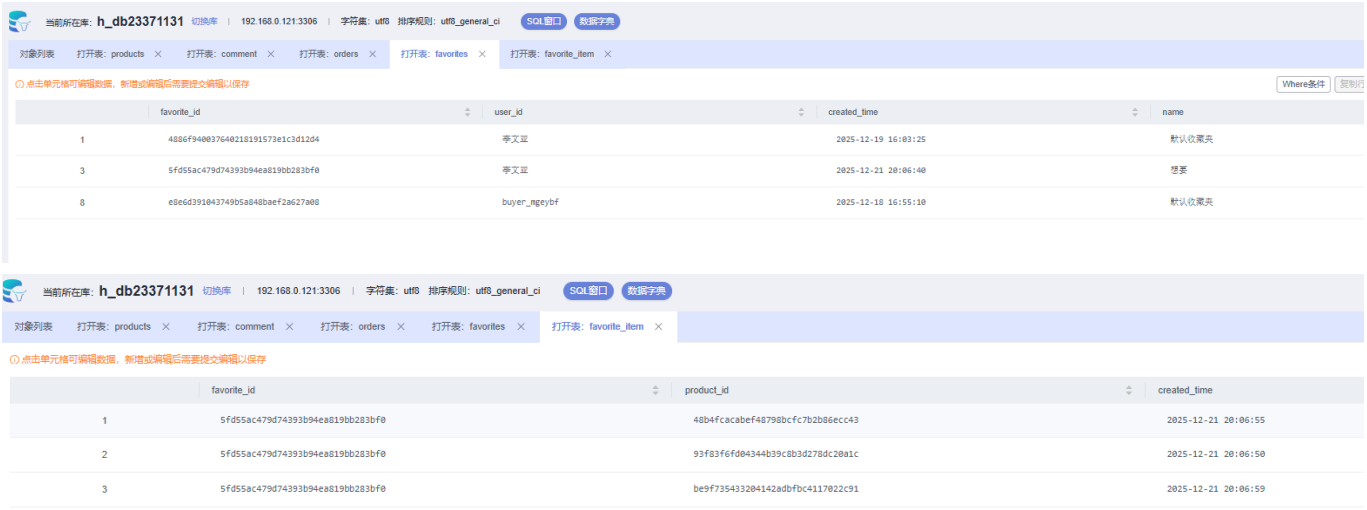
功能描述： 当用户删除收藏夹时，必须保证数据的一致性，防止 `favorite_item` 表中出现指向不存在文件夹的“孤儿数据”。系统利用数据库的 `BEFORE DELETE` 触发器，在删除 `favorites` 表记录之前，自动先清空 `favorite_item` 表中属于该文件夹的所有条目，实现了逻辑闭环。

核心代码实现：

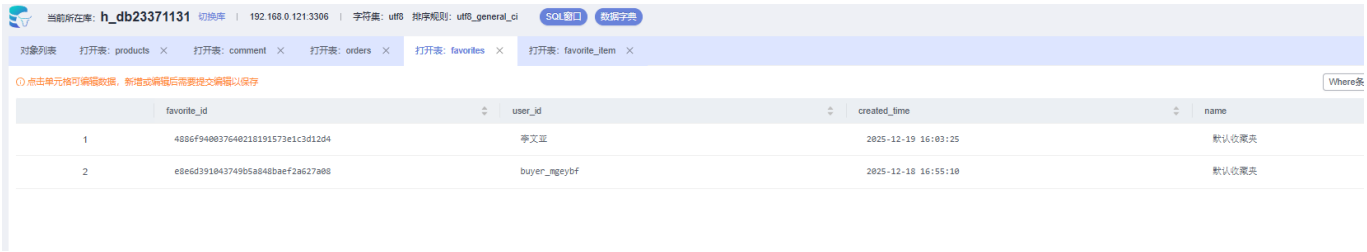
```
# 级联删除交给触发器处理
sql = "DELETE FROM favorites WHERE favorite_id = %s AND user_id = %s"
cursor.execute(sql, (folder_id, user_name))
```

执行结果说明： 用户删除了一个包含多件商品的收藏夹。

收藏夹删除前的favorite表与favorite_item表内容



删除后的favorite表与favorite_item表内容，可以看到相关收藏项已被触发器自动清理



3.9 消息系统

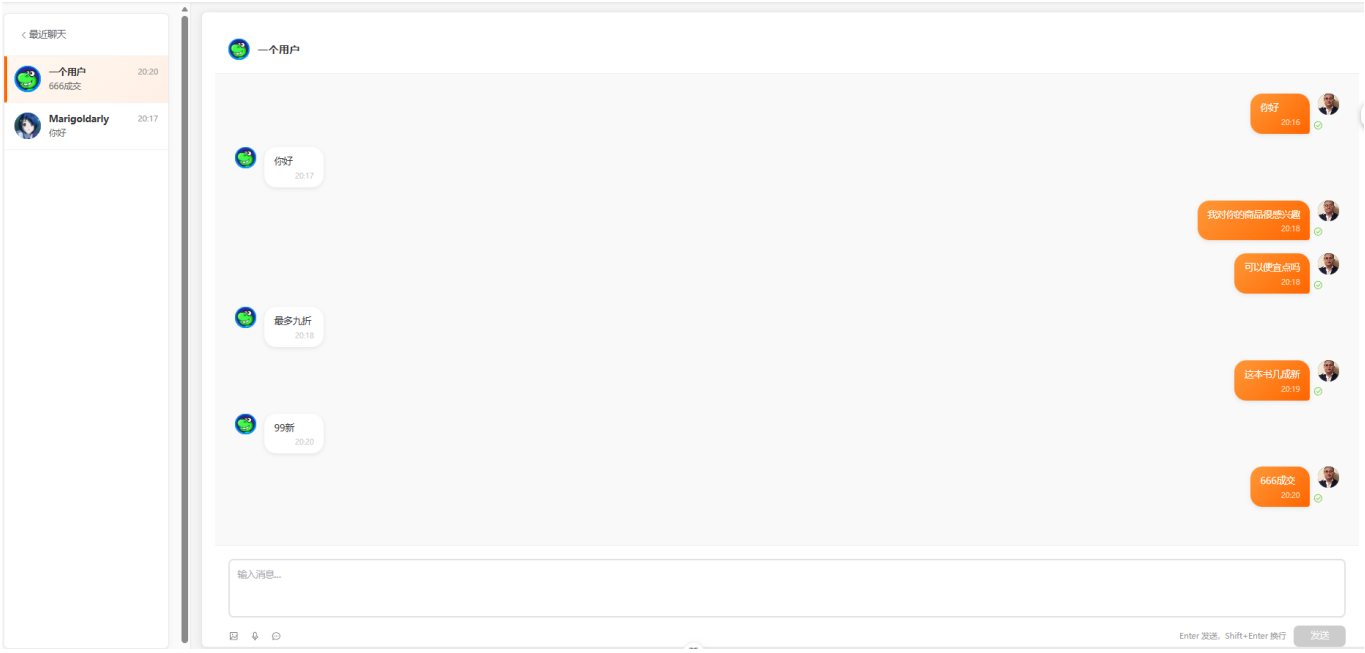
功能描述： 私信系统需要构建完整的对话流。这要求数据库查询不仅要找到"别人发给我的"消息，还要找到"我发给别人"的消息。后端使用 `WHERE ... OR ...` 复合条件进行筛选，并 `JOIN` 用户表获取对方的头像信息，最后按时间倒序排列，从而在前端还原出类似微信的聊天体验。

核心代码实现：

```
sql = """
    SELECT m.*, u.nickname as sender_nickname, u.avatar_url as sender_avatar
    FROM message m
    JOIN users u ON m.sender_id = u.user_name
    WHERE m.receiver_id = %s OR m.sender_id = %s
    ORDER BY m.time DESC
"""
cursor.execute(sql, (user_name, user_name))
```

执行结果说明：

用户进入私信页面，查看与某个卖家的完整对话记录



数据库里储存了所有信息的详细记录

33	26e11a216c544c329cff35afe408f684	buaa	marigold	2025-12-21 16:05:45	你好
34	d46a642ec34e44ab8cd5a965583fca79	李文亚	u1	2025-12-21 20:16:50	你好
35	36a63a1b91f34b79a62aea2788412597	marigold	李文亚	2025-12-21 20:17:06	你好
36	69cf27787bae49f3b98b4d94f3eeb4cf	u1	李文亚	2025-12-21 20:17:51	你好
37	7d14b1da54f74c329f2acdc85056be44	李文亚	u1	2025-12-21 20:18:04	我对你的商品很感兴趣
38	289c97c58f364a699560c3a3ab8c67a7	李文亚	u1	2025-12-21 20:18:11	可以便宜点吗
39	ef0bdad51a3541fd8881c9f6774b3dad	u1	李文亚	2025-12-21 20:18:33	最多九折
40	9d086886cbb44da895e71e9bac2001a4	李文亚	u1	2025-12-21 20:19:57	这本书几成新
41	f38b88a534c64544a08eb75858349165	u1	李文亚	2025-12-21 20:20:15	99新
42	542e0fbd50f24e06b5a6c8f0f02c2d57	李文亚	u1	2025-12-21 20:20:25	666成交

四、组员大作业总结

王宇博

在本次大作业中，我主要承担了系统功能架构设计、数据库后端实现以及核心业务逻辑代码编写的任务。

在系统架构与功能设计阶段，我基于需求分析将复杂的电商系统解耦为用户、商品、订单、互动四大核心模块。我重点梳理了各模块间的数据流转逻辑（如“下单-库存-状态”的联动），结合华家璇同学设计的ER模型，进一步完成了从概念模型到物理模型的落地转化，确立了系统的整体技术路线。

在数据库后端实现方面，我遇到的最大挑战在于如何高效完成数据库的实现与优化。为了实现高效且一致的数据存储，我不仅严格遵循第三范式（3NF）进行表结构设计，还深入使用了数据库的特性。例如，针对“购买商品”这一核心业务，我利用事务机制保证了订单生成与商品下架的原子性；针对“删除操作”，我设计了触发器实现了收藏夹与评论的级联清理，有效避免了脏数据的产生。这些实践让我深刻理解了如何通过数据库手段来保障数据的完整性与业务的鲁棒性。

在后端代码实现方面，我基于 Python 的 Flask 框架构建了系统的 API 服务。面对复杂的业务需求，我采用了蓝图技术对代码进行模块化管理，极大地提升了代码的可维护性。在处理前后端交互时，我设计了规范的 API接口，并实现了多表连接查询（JOIN）与复杂条件筛选，成功解决了商品与卖家信息聚合展示、双向私信列表等开发难点。

通过本次大作业，我完成了从理论学习到工程实践的跨越。我不仅熟练掌握了 SQL 复杂查询与 Python 后端开发技术，更培养了系统化的工程思维，学会了如何在多表关联等实际场景中灵活应用数据库原理，为今后的技

术进阶打下了坚实基础。

华家璇

在本次大作业中，我主要承担了数据库设计、部分数据库操作的实现以及展示用简易前端的实现工作。

在数据库设计方面，我运用数据库课程所学的理论知识，完成了系统的整体架构与模块划分。首先，通过绘制ER图明确了系统中的核心实体、属性及其相互关系，为后续的数据库表结构设计奠定了清晰的基础。其次，通过绘制数据流图，系统地描述了数据在系统中的流动、处理与存储过程。在此基础上，我严格依据3NF对所有的表结构进行了规范化设计与优化，通过消除传递依赖和部分依赖，有效减少了数据冗余，确保了数据库结构的合理性与高效性。其主要难点在于将课上所学真正应用到实际问题上，构建出一个合理高效的数据库关系模式，其中如何精准把握实体关系并使其完全符合3NF要求是一大难点，我通过与同学讨论，并严格按照定义优化了关系模式，使其规范到3NF。

在数据库操作的实现方面，我主要负责使用Python编写与数据库交互的部分逻辑。具体任务包括实现与华为TaurusDB的稳定对接，编写可靠的数据连接与操作代码实现用户管理功能，主要难点在于网络通信相关知识，通过网络上的资料即可学习。 在前端实现方面，我构建了一个简易但功能完整的展示界面。该前端的主要目的是直观地呈现数据库的查询结果，因此我重点实现了数据的可视化展示功能。通过整合数据，将后端返回的数据以更清晰、易懂的形式展现出来，提升了系统的可交互性与结果的可读性。

最终，我圆满完成了所承担的各项工作。系统功能设计文档齐全规范，数据库运行稳定高效，前端界面简洁实用，三者协同工作良好，整体系统达到了预期的设计目标。

实现这次大作业的过程，使我获得了宝贵的实践经验。我深刻体会到数据库设计理论对实际项目的指导意义，并熟练掌握了Python操作数据库、SQL优化及前后端协同开发的全流程。这个过程不仅巩固了我的专业知识，也极大地锻炼了我分析问题、解决复杂技术难题和进行系统化工程实践的能力。