# 实习一：数据库应用案例设计

吕杭州 2200013126　　戴思颖 2200094811　　戴傅聪 2100013061

2025 年 4 月 12 日

本次实习的目标是设计咸鱼数据库，包括罗列业务需求、设计 ER 图、设计数据表结构、用 SQL 语句实现业务功能和使用 Flask 进行前端 web 页面开发。

## 1. 步骤一：罗列业务需求

1. **商品交易**：用户发布二手商品信息，其他用户可以浏览、购买
2. **用户管理**：用户注册、登录、个人信息管理，用户信用
3. **消息系统**：买卖双方沟通
4. **订单管理**：交易订单的创建、支付、发货、确认收货
5. **收藏功能**：用户收藏感兴趣的商品

## 2. 步骤二：数据库设计

### 2.1. 数据库实体与联系设计

#### 2.1.1. 主要实体

1. **用户(User)**
   - user_id 用户 ID
   - username 用户名
   - password_hash 密码哈希值
   - phone 手机号
   - email 电子邮箱
   - credit_score 信用分
   - registration_date 注册日期
   - last_login 最后登录时间
   - status 状态 (active 活跃/banned 封禁)
2. **商品(Product)**
   - product_id 商品 ID
   - seller_id (FK → User) 卖家 ID
   - title 商品标题
   - description 商品描述
   - price 售价
   - original_price 原价
   - condition 商品状况 (new 全新/like new 几乎全新/good 良好/fair 一般/poor 较差)
   - location 所在地
   - post_date 发布时间
   - status 状态 (available 可售/reserved 已预订/sold 已售出/removed 已下架)
   - view_count 浏览数
   - fav_count 收藏数量
3. **订单(Order)**

- order_id 订单 ID
- product_id (FK → Product) 商品 ID
- buyer_id (FK → User) 买家 ID
- seller_id (FK → User) 卖家 ID
- order_date 订单日期
- price 成交价格
- status 状态 (pending 待付款/paid 已付款/shipped 已发货/completed 已完成/cancelled 已取消)
- payment_method 支付方式
- shipping_address 收货地址
- tracking_number 物流单号

4. **消息(Message)**
   - message_id 消息 ID
   - sender_id (FK → User) 发送者 ID
   - receiver_id (FK → User) 接收者 ID
   - product_id (FK → Product, nullable) 关联商品 ID（可为空）
   - content 消息内容
   - send_time 发送时间
   - is_read 是否已读

5. **收藏(Favorite)**
   - favorite_id 收藏 ID
   - user_id (FK → User) 用户 ID
   - product_id (FK → Product) 商品 ID
   - add_date 收藏日期

## 2.2. 主要关系

1. **用户-商品**：一对多（一个用户可以发布多个商品）
2. **用户-订单**：一对多（一个用户可以有多个订单作为买家或卖家）
3. **商品-订单**：一对一（一个商品只能对应一个有效订单）
4. **用户-消息**：一对多（一个用户可以发送/接收多条消息）
5. **用户-收藏**：多对多（通过 Favorite 实体实现）

## 2.3. ER 图设计

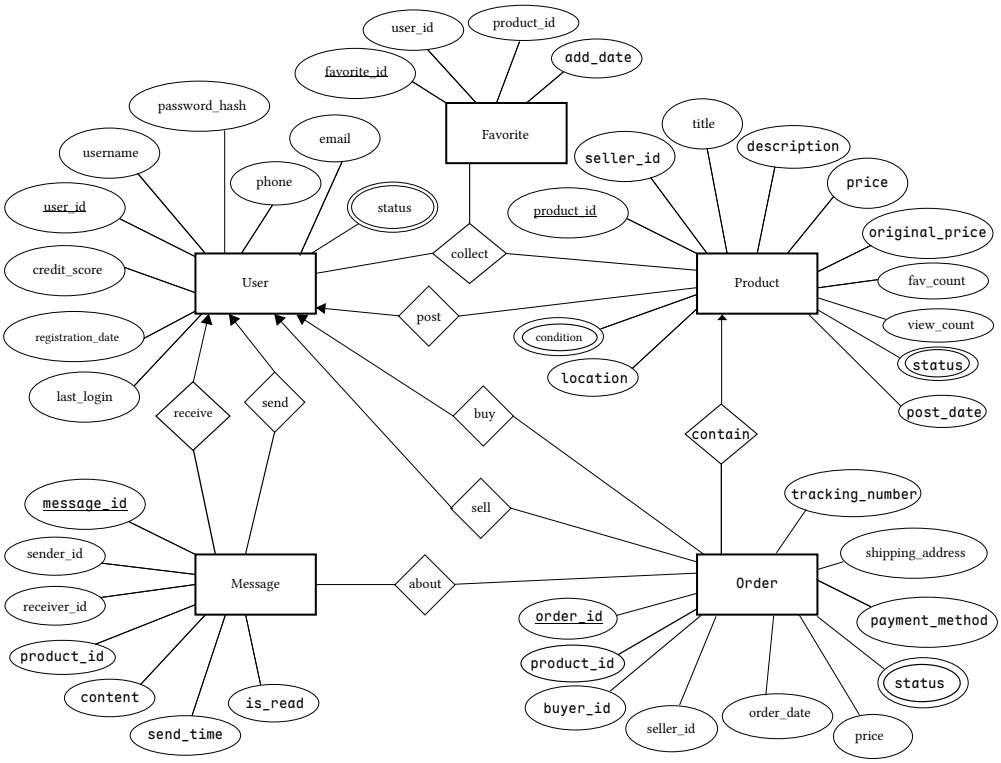


图 1　咸鱼数据库的 ER 图

# 3. 步骤三：创建数据库

```python
import sqlite3
import random
import hashlib
from datetime import datetime
import faker

fake = faker.Faker()

# 连接到 xianyu 数据库
conn = sqlite3.connect('xianyu.db')
cursor = conn.cursor()
```

```python
# 打印指定表的结构（字段信息）
def print_table_schema(table_name):
    cursor.execute(f"PRAGMA table_info({table_name});")
    columns = cursor.fetchall()

    # 获取列的最大宽度
```

```
    max_col_name_len = max(len(col[1]) for col in columns)
    max_type_len = max(len(col[2]) for col in columns)

    # 打印表头
    print(f"\nTable {table_name}\'s schema: ")
    print(f"{'Name'.ljust(max_col_name_len)} | {'Type'.ljust(max_type_len)} | Primary Key")
    print("-" * (max_col_name_len + 3 + max_type_len + 3 + 4))

    # 打印每一列的信息
    for column in columns:
        col_name = column[1]
        col_type = column[2]
        is_primary_key = "Yes" if column[5] else "No"
        print(f"{col_name.ljust(max_col_name_len)} | {col_type.ljust(max_type_len)} | {is_primary_key}")
```

```
# 打印指定表的数据
def print_table_data(table_name):
    try:
        cursor.execute(f"SELECT * FROM {table_name}")
        rows = cursor.fetchall()
        col_names = [desc[0] for desc in cursor.description]

        # 计算每一列的最大宽度（包括字段名和每条记录）
        col_widths = [len(name) for name in col_names]
        for row in rows:
            for i, value in enumerate(row):
                col_widths[i] = max(col_widths[i], len(str(value)) if value is not None else 4)

        # 打印表头
        print(f"\nTable: {table_name}")
        print("-" * (sum(col_widths) + 3 * len(col_widths)))
        header = " | ".join(name.ljust(col_widths[i]) for i, name in enumerate(col_names))
        print(header)
        print("-" * (sum(col_widths) + 3 * len(col_widths)))

        # 打印每一行数据
        if rows:
            for row in rows:
                line = " | ".join(
                    (str(item) if item is not None else "NULL").ljust(col_widths[i])
                    for i, item in enumerate(row)
                )
                print(line)
```

```
        else:
            print("(Empty Table)")
    except sqlite3.Error as e:
        print(f"Failed: {e}")
```

```
def empty_table(table_name):
    try:
        cursor.execute(f"DELETE FROM {table_name}")
        cursor.execute(f"DELETE FROM sqlite_sequence WHERE name='{table_name}'")
        conn.commit()
        print(f"Table {table_name} emptied successfully.")
    except sqlite3.Error as e:
        print(f"Failed to empty table {table_name}: {e}")
```

### 3.1. 创建用户(User)表

```
# 创建用户表 (User)
cursor.execute('''
CREATE TABLE IF NOT EXISTS User (
    user_id INTEGER PRIMARY KEY AUTOINCREMENT,
    username TEXT NOT NULL,
    password_hash TEXT NOT NULL,
    phone TEXT UNIQUE,
    email TEXT UNIQUE,
    credit_score INTEGER DEFAULT 0,
    registration_date TEXT DEFAULT (datetime('now')),
    last_login TEXT,
    status TEXT DEFAULT 'active'
)
''')
print_table_schema('User')
```

```
Table User's schema:
Name              | Type    | Primary Key
-----------------------------------
user_id           | INTEGER | Yes
username          | TEXT    | No
password_hash     | TEXT    | No
phone             | TEXT    | No
email             | TEXT    | No
credit_score      | INTEGER | No
registration_date | TEXT    | No
last_login        | TEXT    | No
status            | TEXT    | No
```

```
empty_table('User')
```

Table User emptied successfully.

```
for _ in range(20):
    username = fake.user_name()
    password = generate_password_hash(fake.password())
    phone = fake.unique.phone_number()
    email = fake.unique.email()
    credit_score = random.randint(300, 850)
    registration_date = fake.date_time_between(start_date='-2y', end_date='now').strftime('%Y-%m-%d %H:%M:%S')
    last_login = fake.date_time_between(start_date=datetime.strptime(registration_date, '%Y-%m-%d %H:%M:%S'), end_date='now').strftime('%Y-%m-%d %H:%M:%S')
    status = random.choice(['active', 'inactive', 'banned'])

    cursor.execute('''
        INSERT INTO User (username, password_hash, phone, email, credit_score, registration_date, last_login, status)
        VALUES (?, ?, ?, ?, ?, ?, ?, ?)
    ''', (username, password, phone, email, credit_score, registration_date, last_login, status))

conn.commit() # 写入数据库中
```

```
print_table_data('User')
```

Too wide. Omitted.

## 3.2. 创建商品(Product)表

说明：`FOREIGN KEY (seller_id) REFERENCES User(user_id) ON DELETE CASCADE` 定义了一个外键约束，并且指定了当被引用的记录（即 User 表中的记录）被删除时所有在子表（如 Product 表）中通过外键与该记录相关联的行也会自动被删除。这是一种级联删除操作。

```
# 创建商品表 (Product)
cursor.execute('''
CREATE TABLE IF NOT EXISTS Product (
    product_id INTEGER PRIMARY KEY AUTOINCREMENT,
    seller_id INTEGER NOT NULL,
    title TEXT NOT NULL,
    description TEXT,
    price REAL NOT NULL,
    original_price REAL,
```

```
    condition TEXT CHECK(condition IN ('new', 'like new', 'good',
'fair', 'poor')),
    location TEXT,
    post_date TEXT DEFAULT (datetime('now')),
    status TEXT DEFAULT 'available' CHECK(status IN ('available',
'reserved', 'sold', 'removed')),
    view_count INTEGER DEFAULT 0,
    fav_count INTEGER DEFAULT 0,
    FOREIGN KEY (seller_id) REFERENCES User(user_id) ON DELETE
CASCADE
)
''')
print_table_schema('Product')
```

```
Table Product's schema:
Name           | Type    | Primary Key
-------------------------------
product_id     | INTEGER | Yes
seller_id      | INTEGER | No
title          | TEXT    | No
description    | TEXT    | No
price          | REAL    | No
original_price | REAL    | No
condition      | TEXT    | No
location       | TEXT    | No
post_date      | TEXT    | No
status         | TEXT    | No
view_count     | INTEGER | No
fav_count      | INTEGER | No
```

```
empty_table('Product')
```

```
Table Product emptied successfully.
```

```
for _ in range(20):
    seller_id = random.randint(1, 5)  # 假设有5个卖家
    title = fake.word().capitalize() + " " + fake.word().capitalize()
# 商品标题
    description = fake.sentence(nb_words=10)  # 商品描述
    price = round(random.uniform(10.0, 1000.0), 2)  # 商品价格
    original_price = round(price * random.uniform(1.0, 1.5), 2)  # 原
价比价格高
    condition = random.choice(['new', 'like new', 'good', 'fair',
'poor'])  # 商品条件
    location = fake.city()  # 商品所在城市
```

```
    post_date = fake.date_this_year().strftime('%Y-%m-%d')  # 发布日期
    status = random.choice(['available', 'reserved', 'sold', 'removed'])  # 商品状态
    view_count = random.randint(0, 500)  # 浏览数
    fav_count = random.randint(0, 100)  # 收藏数

    cursor.execute('''
        INSERT INTO Product (seller_id, title, description, price, original_price, condition, location, post_date, status, view_count, fav_count)
        VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)
    ''', (seller_id, title, description, price, original_price, condition, location, post_date, status, view_count, fav_count))

conn.commit()
```

```
print_table_data('Product')
```

Too wide. Omitted.

### 3.3. 创建订单(Order)表

说明：注意 Order 是一个保留关键字，这里必须通过加下划线等方式避免。

```
# 创建订单表 (Order)
cursor.execute('''
CREATE TABLE IF NOT EXISTS Order_ (
    order_id INTEGER PRIMARY KEY AUTOINCREMENT,
    product_id INTEGER NOT NULL,
    buyer_id INTEGER NOT NULL,
    seller_id INTEGER NOT NULL,
    order_date TEXT DEFAULT (datetime('now')),
    price REAL NOT NULL,
    status TEXT DEFAULT 'pending' CHECK(status IN ('pending', 'paid', 'shipped', 'completed', 'cancelled')),
    payment_method TEXT,
    shipping_address TEXT,
    tracking_number TEXT,
    FOREIGN KEY (product_id) REFERENCES Product(product_id) ON DELETE CASCADE,
    FOREIGN KEY (buyer_id) REFERENCES User(user_id) ON DELETE CASCADE,
    FOREIGN KEY (seller_id) REFERENCES User(user_id) ON DELETE CASCADE
)
''')
```

```
print_table_schema('Order_')
```

```
Table Order_'s schema:
Name             | Type    | Primary Key
---------------------------------
order_id         | INTEGER | Yes
product_id       | INTEGER | No
buyer_id         | INTEGER | No
seller_id        | INTEGER | No
order_date       | TEXT    | No
price            | REAL    | No
status           | TEXT    | No
payment_method   | TEXT    | No
shipping_address | TEXT    | No
tracking_number  | TEXT    | No
```

```
empty_table('Order_')
```

Table Order_ emptied successfully.

```
for _ in range(20):
    product_id = random.randint(1, 20)  # 随机选择产品ID（假设Product表有20条数据）
    buyer_id = random.randint(1, 10)    # 假设有10个买家
    seller_id = random.randint(1, 5)   # 假设有5个卖家
    price = round(random.uniform(10.0, 1000.0), 2)  # 订单价格
    status = random.choice(['pending', 'paid', 'shipped', 'completed', 'cancelled'])  # 订单状态
    payment_method = random.choice(['credit card', 'paypal', 'bank transfer', 'cash'])  # 支付方式
    shipping_address = fake.address().replace("\n", ", ")  # 随机生成地址
    tracking_number = fake.uuid4()  # 随机生成跟踪号

    cursor.execute('''
        INSERT INTO Order_ (product_id, buyer_id, seller_id, order_date, price, status, payment_method, shipping_address, tracking_number)
        VALUES (?, ?, ?, datetime('now'), ?, ?, ?, ?, ?)
    ''', (product_id, buyer_id, seller_id, price, status, payment_method, shipping_address, tracking_number))

# 提交事务
conn.commit()
```

```
print_table_data('Order_')
```

Too wide. Omitted.

### 3.4. 创建消息(Message)表

```
# 创建消息表 (Message)
cursor.execute('''
CREATE TABLE IF NOT EXISTS Message (
    message_id INTEGER PRIMARY KEY AUTOINCREMENT,
    sender_id INTEGER NOT NULL,
    receiver_id INTEGER NOT NULL,
    product_id INTEGER,
    content TEXT NOT NULL,
    send_time TEXT DEFAULT (datetime('now')),
    is_read INTEGER DEFAULT 0,
    FOREIGN KEY (sender_id) REFERENCES User(user_id) ON DELETE CASCADE,
    FOREIGN KEY (receiver_id) REFERENCES User(user_id) ON DELETE CASCADE,
    FOREIGN KEY (product_id) REFERENCES Product(product_id) ON DELETE SET NULL
)
''')
print_table_schema('Message')
```

```
Table Message's schema:
Name        | Type    | Primary Key
-----------------------------
message_id  | INTEGER | Yes
sender_id   | INTEGER | No
receiver_id | INTEGER | No
product_id  | INTEGER | No
content     | TEXT    | No
send_time   | TEXT    | No
is_read     | INTEGER | No
```

```
empty_table('Message')
```

Table Message emptied successfully.

```
for _ in range(20):
    sender_id = random.randint(1, 10)  # 假设有10个用户作为发送者
    receiver_id = random.randint(1, 10)  # 假设有10个用户作为接收者
    product_id = random.randint(1, 20)  # 假设Product表有20个产品
    content = fake.sentence(nb_words=15)  # 随机生成消息内容
```

```
    send_time = fake.date_this_year().strftime('%Y-%m-%d %H:%M:%S')  # 消息发送时间
    is_read = random.choice([0, 1])  # 随机设置消息是否已读（0: 未读, 1: 已读）

    cursor.execute('''
        INSERT INTO Message (sender_id, receiver_id, product_id, content, send_time, is_read)
        VALUES (?, ?, ?, ?, ?, ?)
    ''', (sender_id, receiver_id, product_id, content, send_time, is_read))

# 提交事务
conn.commit()
```

```
print_table_data('Message')
```

Too wide. Omitted.

### 3.5. 创建收藏(Favorite)表

说明：UNIQUE (user_id, product_id) 约束确保了同一个用户不能收藏同一件商品多次。

```
# 创建收藏表 (Favorite)
cursor.execute('''
CREATE TABLE IF NOT EXISTS Favorite (
    favorite_id INTEGER PRIMARY KEY AUTOINCREMENT,
    user_id INTEGER NOT NULL,
    product_id INTEGER NOT NULL,
    add_date TEXT DEFAULT (datetime('now')),
    UNIQUE(user_id, product_id),
    FOREIGN KEY (user_id) REFERENCES User(user_id) ON DELETE CASCADE,
    FOREIGN KEY (product_id) REFERENCES Product(product_id) ON DELETE CASCADE
)
''')
print_table_schema('Favorite')
```

```
Table Favorite's schema:
Name        | Type    | Primary Key
----------------------------
favorite_id | INTEGER | Yes
user_id     | INTEGER | No
product_id  | INTEGER | No
add_date    | TEXT    | No
```

```
empty_table('Favorite')
```

Table Favorite emptied successfully.

```
for _ in range(20):
    user_id = random.randint(1, 10)  # 假设有10个用户
    product_id = random.randint(1, 20)  # 假设有20个商品
    add_date = fake.date_this_year().strftime('%Y-%m-%d')  # 随机生成收藏日期

    # 确保用户和商品的收藏组合唯一
    cursor.execute('''
        INSERT OR IGNORE INTO Favorite (user_id, product_id, add_date)
        VALUES (?, ?, ?)
    ''', (user_id, product_id, add_date))

# 提交事务
conn.commit()
```

```
print_table_data('Favorite')
```

```
Table: Favorite
--------------------------------------------------
favorite_id | user_id | product_id | add_date
--------------------------------------------------
1           | 1       | 4          | 2025-03-05
2           | 3       | 6          | 2025-03-20
3           | 4       | 10         | 2025-03-11
4           | 6       | 9          | 2025-04-02
5           | 1       | 11         | 2025-01-28
6           | 3       | 11         | 2025-03-31
7           | 8       | 3          | 2025-01-21
8           | 6       | 17         | 2025-01-30
9           | 8       | 2          | 2025-04-03
10          | 7       | 8          | 2025-01-21
11          | 8       | 13         | 2025-03-29
12          | 8       | 11         | 2025-03-11
13          | 5       | 3          | 2025-01-08
14          | 10      | 8          | 2025-02-10
15          | 7       | 11         | 2025-01-10
16          | 1       | 19         | 2025-02-24
17          | 10      | 12         | 2025-01-25
18          | 2       | 15         | 2025-01-30
```

```
19          | 5       | 19          | 2025-03-08
20          | 9       | 19          | 2025-03-27
```

# 4. 步骤四：数据库操作

## 4.1. 用户(User)表的操作

```python
def add_user(username, password_hash, phone=None, email=None, credit_score=0):
    cursor.execute('''
    INSERT INTO User (username, password_hash, phone, email, credit_score)
    VALUES (?, ?, ?, ?, ?)
    ''', (username, password_hash, phone, email, credit_score))
    conn.commit()
    return cursor.lastrowid

def get_user(user_id=None, username=None, phone=None, email=None):
    query = 'SELECT * FROM User WHERE '
    conditions = []
    params = []

    if user_id:
        conditions.append('user_id = ?')
        params.append(user_id)
    if username:
        conditions.append('username = ?')
        params.append(username)
    if phone:
        conditions.append('phone = ?')
        params.append(phone)
    if email:
        conditions.append('email = ?')
        params.append(email)

    if not conditions:
        return None

    query += ' AND '.join(conditions)
    cursor.execute(query, params)
    return cursor.fetchone()

def update_user(user_id, username=None, password_hash=None, phone=None, email=None, credit_score=None):
    updates = []
    params = []

    if username:
        updates.append('username = ?')
        params.append(username)
```

```
    if password_hash:
        updates.append('password_hash = ?')
        params.append(password_hash)
    if phone:
        updates.append('phone = ?')
        params.append(phone)
    if email:
        updates.append('email = ?')
        params.append(email)
    if credit_score is not None:
        updates.append('credit_score = ?')
        params.append(credit_score)

    if not updates:
        return False

    params.append(user_id)
    query = 'UPDATE User SET ' + ', '.join(updates) + ' WHERE user_id = ?'
    cursor.execute(query, params)
    conn.commit()
    return cursor.rowcount > 0

def delete_user(user_id):
    cursor.execute('DELETE FROM User WHERE user_id = ?', (user_id,))
    conn.commit()
    return cursor.rowcount > 0
```

```
# 添加用户
user1_id = add_user('john_doe', 'hashed_password123', '1234567890', 'john@example.com')
user2_id = add_user('jane_smith', 'hashed_password456', '0987654321', 'jane@example.com', 50)
print(f"Added users with IDs: {user1_id}, {user2_id}")

# 查询用户
user1 = get_user(user_id=user1_id)
user_by_phone = get_user(phone='1234567890')
print("User1:", user1)
print("User by phone:", user_by_phone)

# 更新用户
update_success = update_user(user1_id, email='john.doe@newexample.com', credit_score=75)
print(f"Update successful: {update_success}")
updated_user = get_user(user_id=user1_id)
print("Updated user:", updated_user)

```

```
# 删除用户
delete_success = delete_user(user2_id)
print(f"Delete successful: {delete_success}")
```

```
Added users with IDs: 1, 2
User1: (1, 'john_doe', 'hashed_password123', '1234567890',
'john@example.com', 0, '2025-04-12 07:56:45', None, 'active')
User by phone: (1, 'john_doe', 'hashed_password123', '1234567890',
'john@example.com', 0, '2025-04-12 07:56:45', None, 'active')
Update successful: True
Updated user: (1, 'john_doe', 'hashed_password123', '1234567890',
'john.doe@newexample.com', 75, '2025-04-12 07:56:45', None, 'active')
Delete successful: True
```

## 4.2. 商品(Product)表的操作

```
def add_product(seller_id, title, description, price,
original_price=None,
                condition=None, location=None):
    cursor.execute('''
    INSERT INTO Product (seller_id, title, description, price,
original_price, condition, location)
    VALUES (?, ?, ?, ?, ?, ?, ?)
    ''', (seller_id, title, description, price, original_price,
condition, location))
    conn.commit()
    return cursor.lastrowid

def get_products(product_id=None, seller_id=None, title=None,
condition=None, status=None, min_price=None, max_price=None):
    query = 'SELECT * FROM Product WHERE '
    conditions = []
    params = []

    if product_id:
        conditions.append('product_id = ?')
        params.append(product_id)
    if seller_id:
        conditions.append('seller_id = ?')
        params.append(seller_id)
    if title:
        conditions.append('title LIKE ?')
        params.append(f'%{title}%')
    if condition:
        conditions.append('condition = ?')
        params.append(condition)
    if status:
        conditions.append('status = ?')
```

```
        params.append(status)
    if min_price:
        conditions.append('price >= ?')
        params.append(min_price)
    if max_price:
        conditions.append('price <= ?')
        params.append(max_price)

    if not conditions:
        query = 'SELECT * FROM Product'
    else:
        query += ' AND '.join(conditions)

    cursor.execute(query, params)
    return cursor.fetchall()

def update_product(product_id, title=None, description=None,
price=None,
                   original_price=None, condition=None,
location=None, status=None):
    updates = []
    params = []

    if title:
        updates.append('title = ?')
        params.append(title)
    if description:
        updates.append('description = ?')
        params.append(description)
    if price:
        updates.append('price = ?')
        params.append(price)
    if original_price:
        updates.append('original_price = ?')
        params.append(original_price)
    if condition:
        updates.append('condition = ?')
        params.append(condition)
    if location:
        updates.append('location = ?')
        params.append(location)
    if status:
        updates.append('status = ?')
        params.append(status)

    if not updates:
        return False

    params.append(product_id)
```

```
    query = 'UPDATE Product SET ' + ', '.join(updates) + ' WHERE
product_id = ?'
    cursor.execute(query, params)
    conn.commit()
    return cursor.rowcount > 0

def delete_product(product_id):
    cursor.execute('DELETE FROM Product WHERE product_id = ?',
(product_id,))
    conn.commit()
    return cursor.rowcount > 0
```

```
# 添加产品
product1_id = add_product(user1_id, 'iPhone 12', 'Like new iPhone 12,
128GB', 599.99, 799.99, 'like new', 'New York')
product2_id = add_product(user1_id, 'MacBook Pro', '2020 MacBook Pro
13', 999.99, 1299.99, 'good', 'New York')
print(f"Added products with IDs: {product1_id}, {product2_id}")

# 查询产品
products = get_products(seller_id=user1_id)
print("All products by seller:", products)
expensive_products = get_products(min_price=800)
print("Expensive products:", expensive_products)

# 更新产品
update_success = update_product(product1_id, price=549.99,
status='reserved')
print(f"Update successful: {update_success}")
updated_product = get_products(product_id=product1_id)
print("Updated product:", updated_product)

# 删除产品
delete_success = delete_product(product2_id)
print(f"Delete successful: {delete_success}")
```

```
Added products with IDs: 1, 2
All products by seller: [(1, 1, 'iPhone 12', 'Like new iPhone 12, 128GB',
599.99, 799.99, 'like new', 'New York', '2025-04-12 07:57:45', 'available',
0, 0), (2, 1, 'MacBook Pro', '2020 MacBook Pro 13', 999.99, 1299.99,
'good', 'New York', '2025-04-12 07:57:45', 'available', 0, 0)]
Expensive products: [(2, 1, 'MacBook Pro', '2020 MacBook Pro 13', 999.99,
1299.99, 'good', 'New York', '2025-04-12 07:57:45', 'available', 0, 0)]
Update successful: True
Updated product: [(1, 1, 'iPhone 12', 'Like new iPhone 12, 128GB', 549.99,
799.99, 'like new', 'New York', '2025-04-12 07:57:45', 'reserved', 0, 0)]
Delete successful: True
```

### 4.3. 订单(Order)表的操作

```
def create_order(product_id, buyer_id, seller_id, price, payment_method=None, shipping_address=None):
    cursor.execute('''
    INSERT INTO Order_ (product_id, buyer_id, seller_id, price, payment_method, shipping_address)
    VALUES (?, ?, ?, ?, ?, ?)
    ''', (product_id, buyer_id, seller_id, price, payment_method, shipping_address))
    conn.commit()
    return cursor.lastrowid

def get_orders(order_id=None, buyer_id=None, seller_id=None, product_id=None, status=None):
    query = 'SELECT * FROM Order_ WHERE '
    conditions = []
    params = []

    if order_id:
        conditions.append('order_id = ?')
        params.append(order_id)
    if buyer_id:
        conditions.append('buyer_id = ?')
        params.append(buyer_id)
    if seller_id:
        conditions.append('seller_id = ?')
        params.append(seller_id)
    if product_id:
        conditions.append('product_id = ?')
        params.append(product_id)
    if status:
        conditions.append('status = ?')
        params.append(status)

    if not conditions:
        query = 'SELECT * FROM Order_'
    else:
        query += ' AND '.join(conditions)

    cursor.execute(query, params)
    return cursor.fetchall()

def update_order(order_id, status=None, payment_method=None, shipping_address=None, tracking_number=None):
    updates = []
    params = []

    if status:
        updates.append('status = ?')
```

```
        params.append(status)
    if payment_method:
        updates.append('payment_method = ?')
        params.append(payment_method)
    if shipping_address:
        updates.append('shipping_address = ?')
        params.append(shipping_address)
    if tracking_number:
        updates.append('tracking_number = ?')
        params.append(tracking_number)

    if not updates:
        return False

    params.append(order_id)
    query = 'UPDATE Order_ SET ' + ', '.join(updates) + ' WHERE order_id = ?'
    cursor.execute(query, params)
    conn.commit()
    return cursor.rowcount > 0


def delete_order(order_id):
    cursor.execute('DELETE FROM Order_ WHERE order_id = ?', (order_id,))
    conn.commit()
    return cursor.rowcount > 0
```

```
# 添加另一个用户用于购买
buyer_id = add_user('buyer_user', 'hashed_password789', '5551234567', 'buyer@example.com')

# 创建订单
order1_id = create_order(product1_id, buyer_id, user1_id, 549.99, 'credit_card', '123 Main St, Anytown')
print(f"Created order with ID: {order1_id}")

# 查询订单
orders = get_orders(buyer_id=buyer_id)
print("Orders by buyer:", orders)

# 更新订单
update_success = update_order(order1_id, status='paid', tracking_number='USPS123456789')
print(f"Update successful: {update_success}")
updated_order = get_orders(order_id=order1_id)
print("Updated order:", updated_order)

```

```python
# 删除订单
delete_success = delete_order(order1_id)
print(f"Delete successful: {delete_success}")
```

```
Created order with ID: 1
Orders by buyer: [(1, 1, 3, 1, '2025-04-12 07:58:30', 549.99, 'pending',
'credit_card', '123 Main St, Anytown', None)]
Update successful: True
Updated order: [(1, 1, 3, 1, '2025-04-12 07:58:30', 549.99, 'paid',
'credit_card', '123 Main St, Anytown', 'USPS123456789')]
Delete successful: True
```

# 5. 步骤五：前端 web 页面开发

## 5.1. 首页


图 2　首页

## 5.2. 我的收藏


图 3　我的收藏

## 5.3. 商品列表


图 4　商品列表

## 5.4. 订单管理


图 5　订单管理

## 5.5. 登录界面


图 6　登录界面