数据库设计：校园订餐系统

一、 实验目的

1. 掌握数据库系统的基本设计流程（需求分析、概念结构设计、逻辑结构设计）。
2. 使用 MySQL 进行表结构的创建、索引优化及外键约束的设置。
3. 掌握视图（View）与触发器（Trigger）的使用。
4. 通过 Python Flask 框架连接数据库，实现前后端交互，完成一个完整的全栈应用。

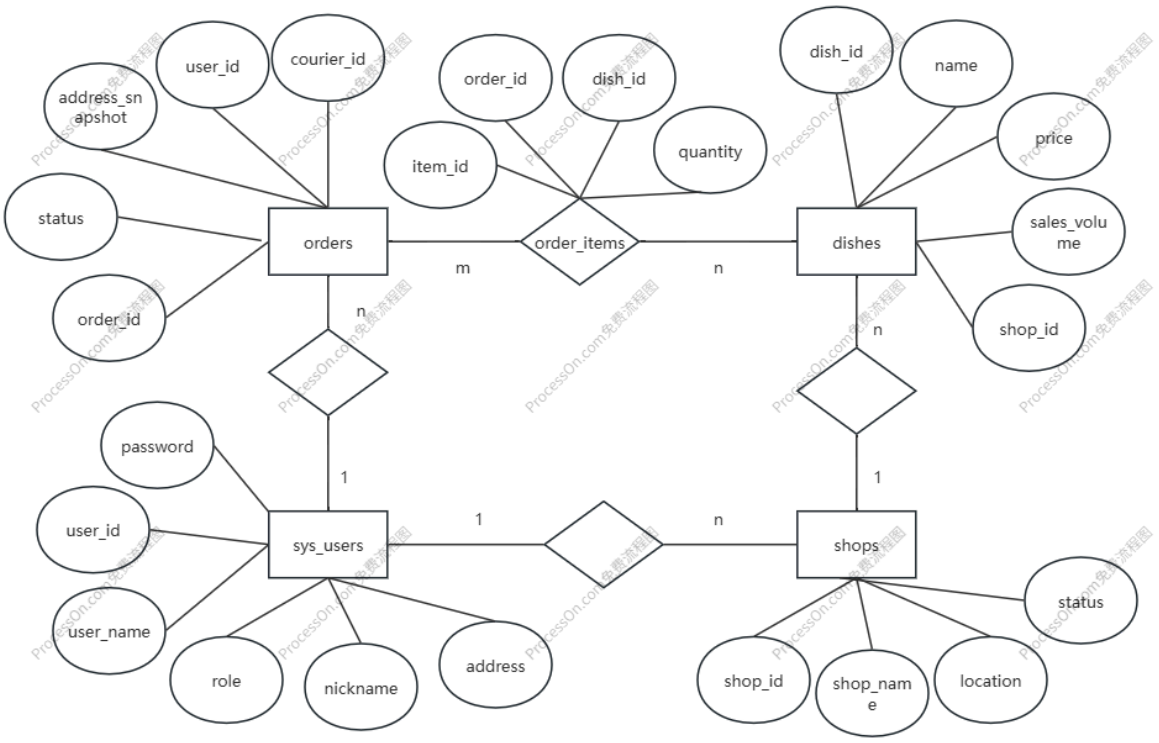
二、 需求分析

本系统旨在解决校园内学生订餐、商家接单及骑手配送的信息化管理问题。系统包含三种用户角色：

1. **消费者 (Consumer)**：
    - 浏览所有营业中的店铺。
    - 进入特定店铺查看菜单并下单。
    - 查看个人历史订单及其状态（待接单、配送中、已完成）。
2. **商家 (Merchant)**：
    - 管理店铺信息（本项目简化为后台预设）。
    - 实时查看店铺的订单详情（包括菜品、顾客信息、配送状态）。
3. **配送员 (Courier)**：
    - 在“抢单大厅”查看所有待配送订单。
    - 执行抢单操作，并在送达后更新订单状态。

三、 数据库设计

3.1 概念结构设计



- **用户 (sys_users)** 与 **店铺 (shops)** 存在 1:N 关系（商家）。
- **店铺 (shops)** 与 **菜品 (dishes)** 存在 1:N 关系。
- **用户** 与 **订单 (orders)** 存在 1:N 关系（作为消费者或配送员）。
- **订单** 与 **菜品** 存在 M:N 多对多关系（通过 `order_items` 中间表实现）。

3.2 逻辑结构设计

本系统共设计了 5 张核心数据表：

1. **系统用户表 (`sys_users`)**：
   - 存储所有角色的登录信息。
   - 核心字段：`user_id` (PK), `username`, `password`, `role` (不同角色有不同权限), `address`, `nickname`。

user_id # int	username ABC varchar(50)	password ABC varchar(255)	phone ABC varchar(20)	address ABC varchar(100)	role ABC varchar(20)	create_time datetime	nickname ABC varchar(50)
1	student	123456	19512345678	东园五栋	consumer	2026-01-14 15:3	小明
2	rider	123456	13987654321	(Null)	courier	2026-01-14 15:3	闪电
3	boss	123456	13700137000	(Null)	merchant	2026-01-14 15:3	富贵
4	boss1	123456	13623041051	(Null)	merchant	2026-01-14 20:3	一凡打卤面
5	boss2	123456	18087960152	(Null)	merchant	2026-01-14 20:3	蒙自源

2. **店铺表 (`shops`)**：
   - 存储商家店铺信息。
   - 核心字段：`shop_id` (PK), `shop_name`, `location`, `status`。

shop_id # int	owner_id # int	shop_name ABC varchar(100)	location ABC varchar(100)	status # tinyint(1)
1	3	湘小悦	东饭三楼201档口	1
2	4	一凡打卤面	西园食堂一楼101	1
3	5	蒙自源	东园食堂二楼205	1

3. **菜品表 (`dishes`)**:
    - 存储具体的菜单项。
    - 核心字段: `dish_id` (PK), `shop_id` (FK), `name`, `price`, `sales_volume` (销量)。

dish_id # int	shop_id # int	name varchar(100)	price # decimal(10,2)	sales_volume # int	is_available # tinyint(1)
1	1	孜然土豆包菜饭	14.00	201	1
2	1	洋葱炒牛肉饭	18.00	122	1
3	2	自选打卤面	12.00	200	1
4	2	自选米饭	12.00	150	1
5	2	牛腩面	18.00	500	1
6	2	肉酱面	15.00	800	1
7	3	原味鸡汤米线	16.00	300	1
8	3	番茄肥牛米线	20.00	120	1
9	3	土豆泥肉酱拌米线	17.00	600	1
10	3	小酥肉米线	15.00	80	1

4. **订单主表 (`orders`)**:
    - 存储订单头部信息。
    - 核心字段: `order_id` (PK), `user_id` (FK), `courier_id` (FK), `status` (0:待发货, 1:配送中, 2:已完成, 4:已取消), `address_snapshot` (地址快照)。

order_id # bigint	user_id # int	shop_id # int	courier_id # int	total_amount # decimal(10,2)	status # int	address_snapshot varchar(100)	create_time datetime
1	1	1	2	14.00	2	默认宿舍地址	2026-01-14 15:5
2	1	1	2	18.00	2	默认宿舍地址	2026-01-14 17:3
3	1	1	2	18.00	2	东园五栋	2026-01-14 17:4

5. **订单详情表 (`order_items`)**:
    - 存储订单包含的具体菜品。
    - 核心字段: `item_id` (PK), `order_id` (FK), `dish_id` (FK), `quantity`。

item_id # bigint	order_id # bigint	dish_id # int	quantity # int	price_snapshot # decimal(10,2)
1	1	1	1	14.00
2	2	2	1	18.00
3	3	2	1	18.00

以下是创建表的代码:

```sql
-- Set names to ensure encoding matches
SET NAMES utf8mb4;

-- ============================================
-- 1. 系统用户表 (sys_users)
-- 包含：消费者、商家、配送员。通过 role 区分
-- ============================================
DROP TABLE IF EXISTS `sys_users`;
CREATE TABLE `sys_users` (
  `user_id` int(11) NOT NULL AUTO_INCREMENT COMMENT '用户ID',
```

```sql
  `username` varchar(50) NOT NULL COMMENT '登录账号',
  `password` varchar(255) NOT NULL COMMENT '密码',
  `phone` varchar(20) NOT NULL COMMENT '电话',
  `address` varchar(100) DEFAULT NULL COMMENT '收货地址(消费者用)',
  `role` varchar(20) NOT NULL COMMENT '角色: consumer/merchant/courier',
  `create_time` datetime DEFAULT CURRENT_TIMESTAMP,
  PRIMARY KEY (`user_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COMMENT='用户综合表';

-- ============================================
-- 2. 商家店铺表 (shops)
-- ============================================
DROP TABLE IF EXISTS `shops`;
CREATE TABLE `shops` (
  `shop_id` int(11) NOT NULL AUTO_INCREMENT,
  `owner_id` int(11) NOT NULL COMMENT '关联商家用户ID',
  `shop_name` varchar(100) NOT NULL COMMENT '店铺名称',
  `location` varchar(100) DEFAULT NULL COMMENT '店铺位置',
  `status` tinyint(1) DEFAULT '1' COMMENT '1营业, 0休息',
  PRIMARY KEY (`shop_id`),
  CONSTRAINT `fk_shop_owner` FOREIGN KEY (`owner_id`) REFERENCES `sys_users`
(`user_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COMMENT='店铺表';

-- ============================================
-- 3. 菜品表 (dishes)
-- 注：按照要求，去掉了库存字段，只保留销量
-- ============================================
DROP TABLE IF EXISTS `dishes`;
CREATE TABLE `dishes` (
  `dish_id` int(11) NOT NULL AUTO_INCREMENT,
  `shop_id` int(11) NOT NULL,
  `name` varchar(100) NOT NULL COMMENT '菜名',
  `price` decimal(10,2) NOT NULL COMMENT '价格',
  `sales_volume` int(11) DEFAULT '0' COMMENT '销量',
  `is_available` tinyint(1) DEFAULT '1' COMMENT '1上架, 0下架',
  PRIMARY KEY (`dish_id`),
  CONSTRAINT `fk_dish_shop` FOREIGN KEY (`shop_id`) REFERENCES `shops`
(`shop_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COMMENT='菜品表';

-- ============================================
-- 4. 订单主表 (orders)
-- ============================================
DROP TABLE IF EXISTS `orders`;
CREATE TABLE `orders` (
  `order_id` bigint(20) NOT NULL AUTO_INCREMENT,
  `user_id` int(11) NOT NULL COMMENT '下单用户',
  `shop_id` int(11) NOT NULL COMMENT '目标店铺',
  `courier_id` int(11) DEFAULT NULL COMMENT '配送员ID',
  `total_amount` decimal(10,2) NOT NULL COMMENT '总金额',
  `status` int(11) DEFAULT '0' COMMENT '0未发货, 1配送中, 2已完成, 4已取消',
  `address_snapshot` varchar(100) NOT NULL COMMENT '地址快照',
  `create_time` datetime DEFAULT CURRENT_TIMESTAMP,
  PRIMARY KEY (`order_id`),
```

```
  CONSTRAINT `fk_order_user` FOREIGN KEY (`user_id`) REFERENCES `sys_users`
(`user_id`),
  CONSTRAINT `fk_order_shop` FOREIGN KEY (`shop_id`) REFERENCES `shops`
(`shop_id`),
  CONSTRAINT `fk_order_courier` FOREIGN KEY (`courier_id`) REFERENCES `sys_users`
(`user_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COMMENT='订单主表';

-- ============================================
-- 5. 订单详情表 (order_items)
-- ============================================
DROP TABLE IF EXISTS `order_items`;
CREATE TABLE `order_items` (
  `item_id` bigint(20) NOT NULL AUTO_INCREMENT,
  `order_id` bigint(20) NOT NULL,
  `dish_id` int(11) NOT NULL,
  `quantity` int(11) NOT NULL COMMENT '购买数量',
  `price_snapshot` decimal(10,2) NOT NULL COMMENT '下单时单价',
  PRIMARY KEY (`item_id`),
  CONSTRAINT `fk_item_order` FOREIGN KEY (`order_id`) REFERENCES `orders`
(`order_id`),
  CONSTRAINT `fk_item_dish` FOREIGN KEY (`dish_id`) REFERENCES `dishes`
(`dish_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COMMENT='订单详情表';
```

3.3 视图与触发器实现

(1) 视图设计 (`v_order_details`)

为了简化后端多表查询的复杂度，设计了一个全能视图，关联了 `orders`, `users` (两次关联：顾客和骑手), `shops`, `order_items`, `dishes` 五张表。 从而直接获取带有中文昵称、店铺名、菜名的完整订单信息，极大简化了业务代码。

order_id	user_id	shop_name	shop_address	courier_id	courier_name	courier_phone	total_amount	status	status_text	user_address	
1	1	湘小悦	东饭三楼201档口	2	闪电	13987654321	14.00	2	已完成	默认宿舍地址	2(
2	1	湘小悦	东饭三楼201档口	2	闪电	13987654321	18.00	2	已完成	默认宿舍地址	2(
3	1	湘小悦	东饭三楼201档口	2	闪电	13987654321	18.00	2	已完成	东园五栋	2(

```
ALTER TABLE `sys_users` ADD COLUMN `nickname` varchar(50) DEFAULT NULL COMMENT
'昵称/真实姓名';

CREATE OR REPLACE VIEW `v_order_details` AS
SELECT
    o.order_id,

    -- 1. 顾客信息 (专门连一次 sys_users 表，起名叫 u_customer)
    o.user_id,
    u_customer.nickname AS customer_nickname,

    -- 2. 店铺信息
    s.shop_name,
    s.location AS shop_address,

    -- 3. 配送员信息 (专门连一次 sys_users 表，起名叫 u_courier)
    o.courier_id,
    u_courier.nickname AS courier_nickname,
```

```
    u_courier.phone AS courier_phone,

    -- 4. 菜品信息
    d.name AS dish_name,

    -- 5. 订单基础信息
    o.total_amount,
    o.status,
    CASE o.status
        WHEN 0 THEN '待发货'
        WHEN 1 THEN '配送中'
        WHEN 2 THEN '已完成'
        WHEN 4 THEN '已取消'
        ELSE '未知'
    END AS status_text,
    o.address_snapshot AS user_address,
    o.create_time

FROM `orders` o
-- 关联店铺
JOIN `shops` s ON o.shop_id = s.shop_id
-- 关联菜品（通过详情表）
JOIN `order_items` oi ON o.order_id = oi.order_id
JOIN `dishes` d ON oi.dish_id = d.dish_id
-- 第一次关联用户表：查顾客
JOIN `sys_users` u_customer ON o.user_id = u_customer.user_id
-- 第二次关联用户表：查骑手（用 LEFT JOIN，因为刚下单可能没骑手）
LEFT JOIN `sys_users` u_courier ON o.courier_id = u_courier.user_id;
```

(2) 触发器设计（`Trigger`）

设计了自动化触发器以维护数据一致性：

- `trg_add_sales`：当 `order_items` 插入新记录时，自动增加对应菜品的 `sales_volume`。
- `trg_rollback_sales`：当订单被取消（详情被删除）时，自动回滚（扣减）对应菜品的销量。

```
DELIMITER $$
-- 触发器 1: 下单后，自动增加销量
DROP TRIGGER IF EXISTS `trg_add_sales`$$
CREATE TRIGGER `trg_add_sales`
AFTER INSERT ON `order_items`
FOR EACH ROW
BEGIN
    UPDATE `dishes`
    SET `sales_volume` = `sales_volume` + NEW.quantity
    WHERE `dish_id` = NEW.dish_id;
END$$
-- 触发器 2: 删除订单详情后，自动扣减销量（回滚）
DROP TRIGGER IF EXISTS `trg_rollback_sales`$$
CREATE TRIGGER `trg_rollback_sales`
AFTER DELETE ON `order_items`
FOR EACH ROW
BEGIN
    UPDATE `dishes`
    SET `sales_volume` = `sales_volume` - OLD.quantity
```

```
    WHERE `dish_id` = OLD.dish_id;
END$$

DELIMITER ;
```

四、 系统实现 (代码与界面)

4.1 开发环境

- **数据库**：MySQL 8.0 (通过 Navicat 管理)
- **后端语言**：Python 3.11.7
- **Web框架**：Flask
- **前端样式**：HTML5 + Picnic CSS (轻量级UI库)

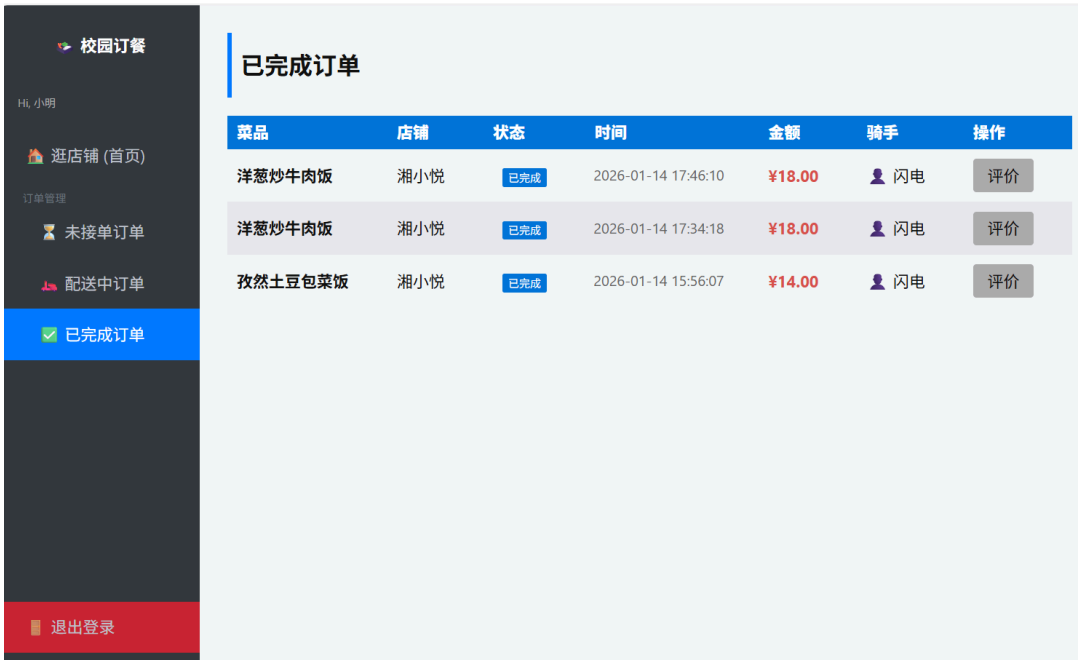
4.2 核心功能展示

1. **登录模块**：支持三种角色登录，根据 `role` 字段自动跳转不同主页。

2. **消费者模块**：
    - **店铺列表**：展示所有入驻商家。

- **点餐界面**：进入店铺后可以看到菜品价格及销量并点餐。

- **订单管理**：分类查看未接单/配送中/已完成的订单。

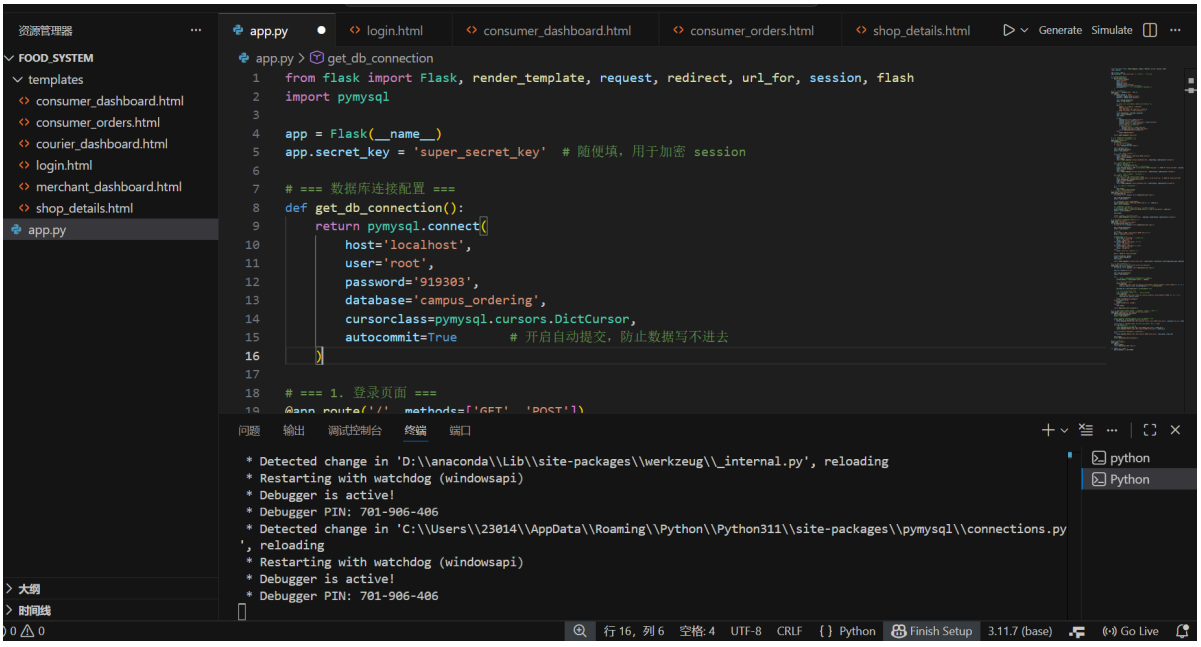


3. **商家模块**：
    - 实时显示本店订单，包含顾客昵称和具体菜品。

4. **配送员模块**：
    - 抢单大厅与配送状态更新。

4.3 前后端代码实现

本次实验重点在于数据库设计，前后端只是方便展示使用，故完整代码会放在github里，这里不再赘述。



五、 实验结果与测试

设计过程中分两次插入了一些测试用户，消费者student，配送员rider，三个商家boss、boss1、boss2，以下是插入代码：

```
-- 1. 插入一个“消费者”
INSERT INTO sys_users (username, password, nickname, phone, address, role)
VALUES ('student', '123456', '小明', '19512345678', '东园五栋', 'consumer');

-- 2. 插入一个“配送员”
INSERT INTO sys_users (username, password, nickname, phone, role)
VALUES ('rider', '123456', '闪电', '13987654321', 'courier');
```

```sql
-- 3. 插入一个“商家”
-- 注意：商家的 address 可以为空，因为他有店铺地址
INSERT INTO sys_users (username, password, nickname, phone, role)
VALUES ('boss', '123456', '富贵', '13700137000', 'merchant');

-- 4. 给“富贵”开一家店
-- 我们利用子查询自动找到刚才插入的 'boss' 的 ID
INSERT INTO shops (owner_id, shop_name, location, status)
VALUES (
    (SELECT user_id FROM sys_users WHERE username = 'boss'),
    '湘小悦',
    '东饭三楼201档口',
    1
);

-- 5. 给这家店上架两个菜
INSERT INTO dishes (shop_id, name, price, sales_volume)
VALUES
    ((SELECT shop_id FROM shops WHERE shop_name = '湘小悦'), '孜然土豆包菜饭',
14.00, 200),
    ((SELECT shop_id FROM shops WHERE shop_name = '湘小悦'), '洋葱炒牛肉饭', 18.00,
120);
```

```sql
-- 第一步：创建两个新商家的账号
INSERT INTO sys_users (username, password, nickname, phone, role)
VALUES
('boss1', '123456', '一凡打卤面', '13623041051', 'merchant'),
('boss2', '123456', '蒙自源', '18087960152', 'merchant');

-- 第二步：给这两个商家开店
-- (这里使用子查询自动获取刚才插入的用户ID，防止出错)
INSERT INTO shops (owner_id, shop_name, location, status)
VALUES
(
    (SELECT user_id FROM sys_users WHERE username = 'boss1'),
    '一凡打卤面',
    '西园食堂一楼101',
    1
),
(
    (SELECT user_id FROM sys_users WHERE username = 'boss2'),
    '蒙自源',
    '东园食堂二楼205',
    1
);

-- 第三步：给“一凡打卤面”上架菜品
INSERT INTO dishes (shop_id, name, price, sales_volume)
VALUES
((SELECT shop_id FROM shops WHERE shop_name = '一凡打卤面'), '自选打卤面', 12.00,
200),
((SELECT shop_id FROM shops WHERE shop_name = '一凡打卤面'), '自选米饭', 12.00,
150),
((SELECT shop_id FROM shops WHERE shop_name = '一凡打卤面'), '牛腩面', 18.00, 500),
((SELECT shop_id FROM shops WHERE shop_name = '一凡打卤面'), '肉酱面', 15.00, 800);
```

```sql
-- 第四步：给“蒙自源”上架菜品
INSERT INTO dishes (shop_id, name, price, sales_volume)
VALUES
((SELECT shop_id FROM shops WHERE shop_name = '蒙自源'), '原味鸡汤米线', 16.00,
300),
((SELECT shop_id FROM shops WHERE shop_name = '蒙自源'), '番茄肥牛米线', 20.00,
120),
((SELECT shop_id FROM shops WHERE shop_name = '蒙自源'), '土豆泥肉酱拌米线', 17.00,
600),
((SELECT shop_id FROM shops WHERE shop_name = '蒙自源'), '小酥肉米线', 15.00, 80);
```

通过预置测试数据（student, boss, boss1,boss2, rider），模拟了完整的业务闭环：

1. **下单**：消费者下单“番茄肥牛米线”，数据库 `orders` 表生成记录，状态为 `0`。

Table Profile | Begin Transaction | Cell Editor | Filter & Sort | Columns | Data Profiling | Tools

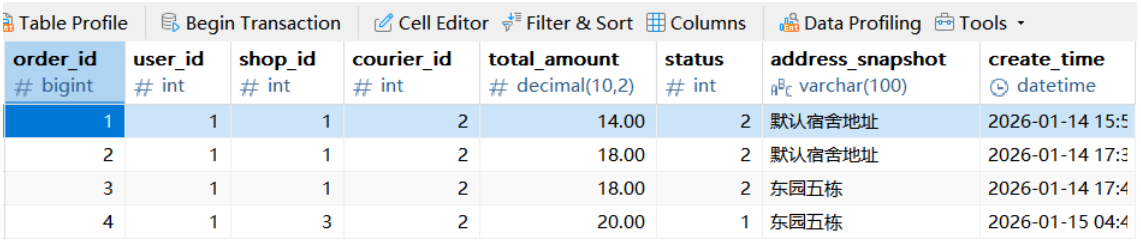
order_id # bigint	user_id # int	shop_id # int	courier_id # int	total_amount # decimal(10,2)	status # int	address_snapshot varchar(100)	create_time datetime
1	1	1	2	14.00	2	默认宿舍地址	2026-01-14 15:5
2	1	1	2	18.00	2	默认宿舍地址	2026-01-14 17:3
3	1	1	2	18.00	2	东园五栋	2026-01-14 17:4
4	1	3	(Null)	20.00	0	东园五栋	2026-01-15 04:4

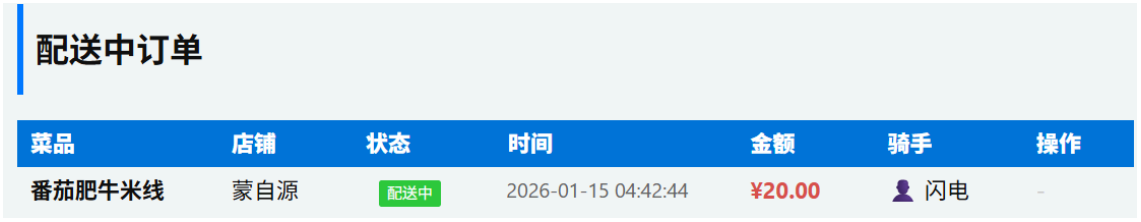
2. **触发器验证**：检查 `dishes` 表，发现“番茄肥牛米线”销量自动 +1。

Table Profile | Begin Transaction | Cell Editor | Filter & Sort | Columns | Data Profiling

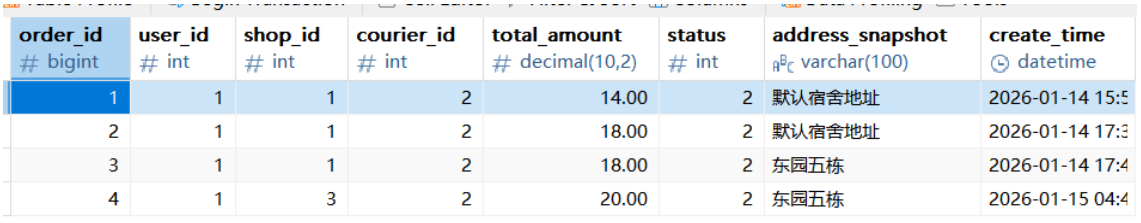
dish_id # int	shop_id # int	name varchar(100)	price # decimal(10,2)	sales_volume # int	is_available # tinyint(1)
1	1	孜然土豆包菜饭	14.00	201	1
2	1	洋葱炒牛肉饭	18.00	122	1
3	2	自选打卤面	12.00	200	1
4	2	自选米饭	12.00	150	1
5	2	牛腩面	18.00	500	1
6	2	肉酱面	15.00	800	1
7	3	原味鸡汤米线	16.00	300	1
8	3	番茄肥牛米线	20.00	121	1
9	3	土豆泥肉酱拌米线	17.00	600	1
10	3	小酥肉米线	15.00	80	1

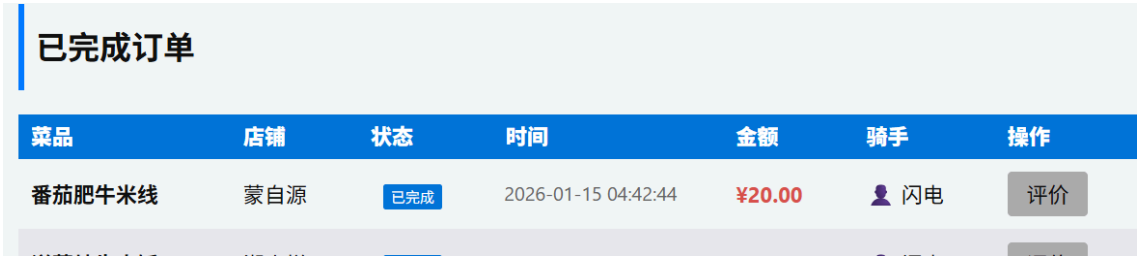
3. **接单**：骑手点击“抢单”，订单状态变更为 `1`，消费者可以在配送中订单中查看订单相关信息。





4. **送达**：骑手点击“确认送达”，状态变更为 `2`。





5. **视图验证**：商家后台成功看到该订单显示为“已完成”。

六、 实验总结与体会

本次实验设计并实现了一个校园订餐系统的数据库。

1. **数据库设计的规范性**：深刻理解了第三范式（3NF），通过将订单拆分为“主表”和“详情表”解决了数据冗余问题。
2. **视图**：在项目初期，查询订单详情需要编写极长的 JOIN 语句，容易出错。引入 `v_order_details` 视图后，后端代码变得非常简洁，大大提高了开发效率。
3. **数据一致性的维护**：通过触发器实现了销量的自动计算，避免了在应用层编写复杂的更新逻辑，保证了数据的准确性。
4. **全栈开发的整合**：学会了使用 Python Flask 连接 MySQL，通过前后端展示功能。