上海交通大學

SHANGHAI JIAO TONG UNIVERSITY

数据库系统课程设计

DATABASE SYSTEM PROJECT

“饿了吗”在线订餐平台设计

ELEMA ONLINE FOOD DELIVERY SYSTEM

课题报告

PROJECT REPORT

目 录

一、项目简介

1. 课题介绍

本系统采用关系型数据库模式，基于 Apache、PHP 和 MySQL 的开发环境，我们设计了“饿了吗”在线订餐平台。整个平台的设计思路借鉴了现有的网络订餐平台“饿了么”的流程设计和外观，为用户在该平台上的注册登录、餐厅查询和餐食预定等给予帮助，同时为餐厅经营者提供管理菜单、接受订单和管理订单状态等服务。

数据库系统由用户、餐厅和订单三部分组成，每部分都尽量涵盖了完成操作所需要的详细信息。

同时我们借助已有服务提供商的 API 接口，实现了基于高德地图的餐饮查询系统，并利用已有 twitter 设计模板美化了用户界面，在网站页面的设计中实现了合理的操作逻辑。

2. 相关文档

[1]《A First Course in Database Systems》 机械工业出版社
[2] php 在线教程 W3SCHOOL
[3] CSS 在线教程 W3SCHOOL
[4] jQuery 在线教程 W3SCHOOL
[5] bootstrap 官方文档 GetBootstrap

二、系统架构

1. 基本结构

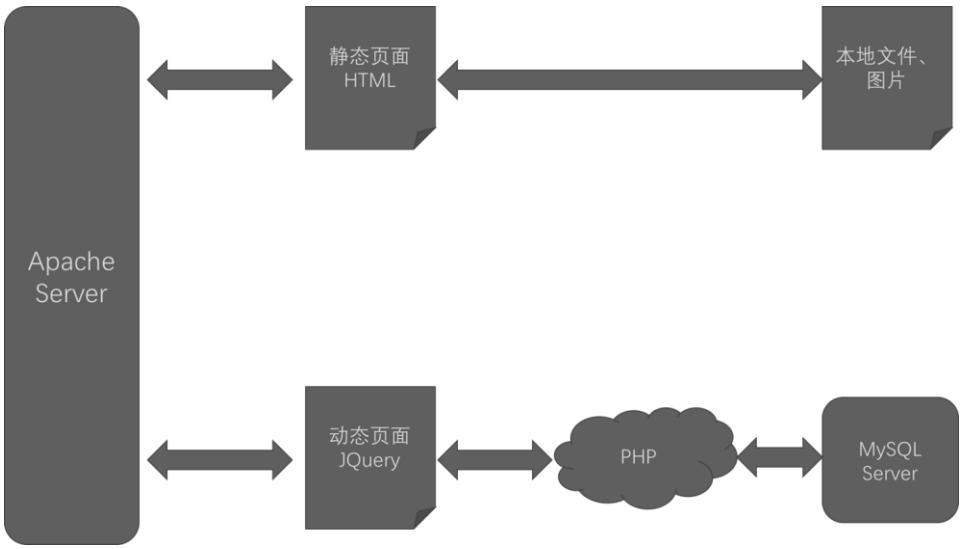

图 1. 平台基本结构

我们平台系统的基本结构如上图所示。

服务器上部署的 MySQL 服务器通过封装好的用户界面操作平台与 PHP 脚本进行连接。在这部分，我们直接利用了 AMPPS 集成环境下的平台：PHPMyAdmin。作为一个可视化工具，它可以更好地进行对数据库的基本操作（如修改用户权限，监控数据库状态等），同时进行访问和控制。

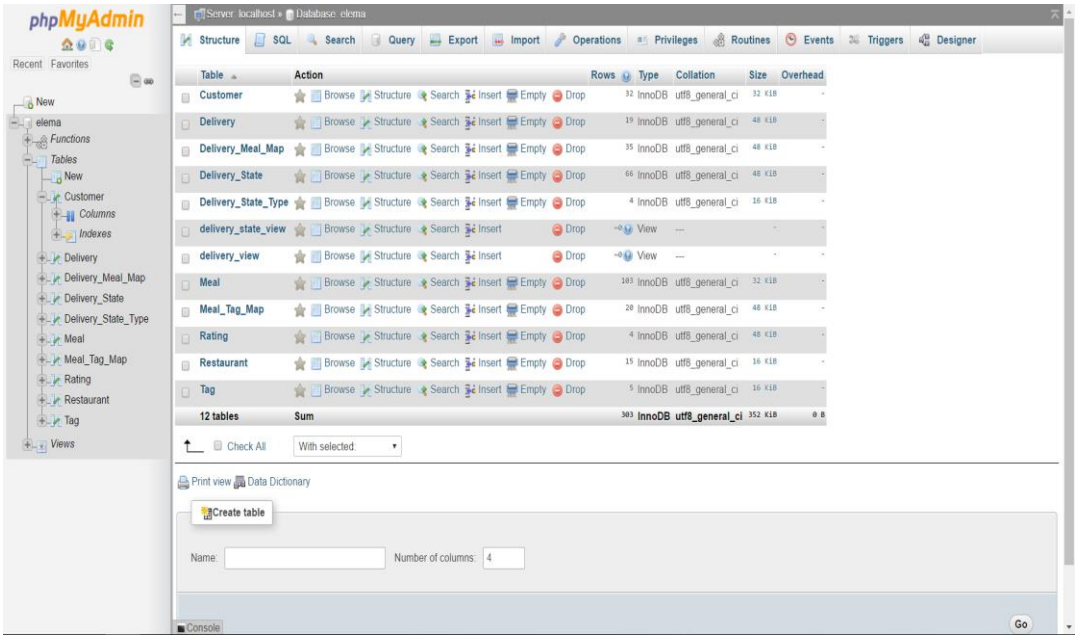

图 2. PHPMyAdmin 界面

为了建立更友好的交互界面，我们在静态 HTML 之外，利用 JQuery 与 PHP 连接完成了动态页面的设计。用户在动态页面中完成的动作（填写表单、点击按钮等）由 JQuery 执行后将数据发送给后台的 PHP 进行处理，最终将数据的增删和更改在数据库中储存。

所以总结来说我们的系统主要分为 3 部分：

a) 数据库层：存放了用户信息、餐厅信息、菜品信息、外卖信息和配送状态的具体数据；

b) 数据服务层：用于负责处理用户请求，与数据库进行连接，并接受上层发来的结果对数据库进行更新或修改；

c) 用户图形界面层：用户或餐厅管理员通过图形界面发起相应的指令请求，并接收来自下面一层返回的信息。

依据上面的基本架构，我们的编程工作主要包含了三部分内容：数据库的编写（SQL），数据服务层通信脚本语言代码的编写（PHP），用户图形界面层脚本语言的编写（HTML+CSS+JQuery）。

2. 特殊结构

本系统借鉴了“饿了么”平台的餐厅选择理念，通过调用开放的高德地图 API 对搜索地点周围一定距离内的餐厅进行显示，以保证搜索到的餐厅

可以满足餐厅属性中有关配送距离的要求。搜索调用 API 的具体实现在目录./js/index.js 文件中。基本思路是首先利用高德地图返回的地点的经纬度坐标，将这一坐标交付给 PHP 脚本./api/customer/delievery.php，脚本调取数据库中满足配送距离的所有餐厅后发送给 JS 脚本，再通过另外一个 JS 脚本将所有的餐厅信息按照一定的顺序列举出来。

3. 系统 E/R 模型设计

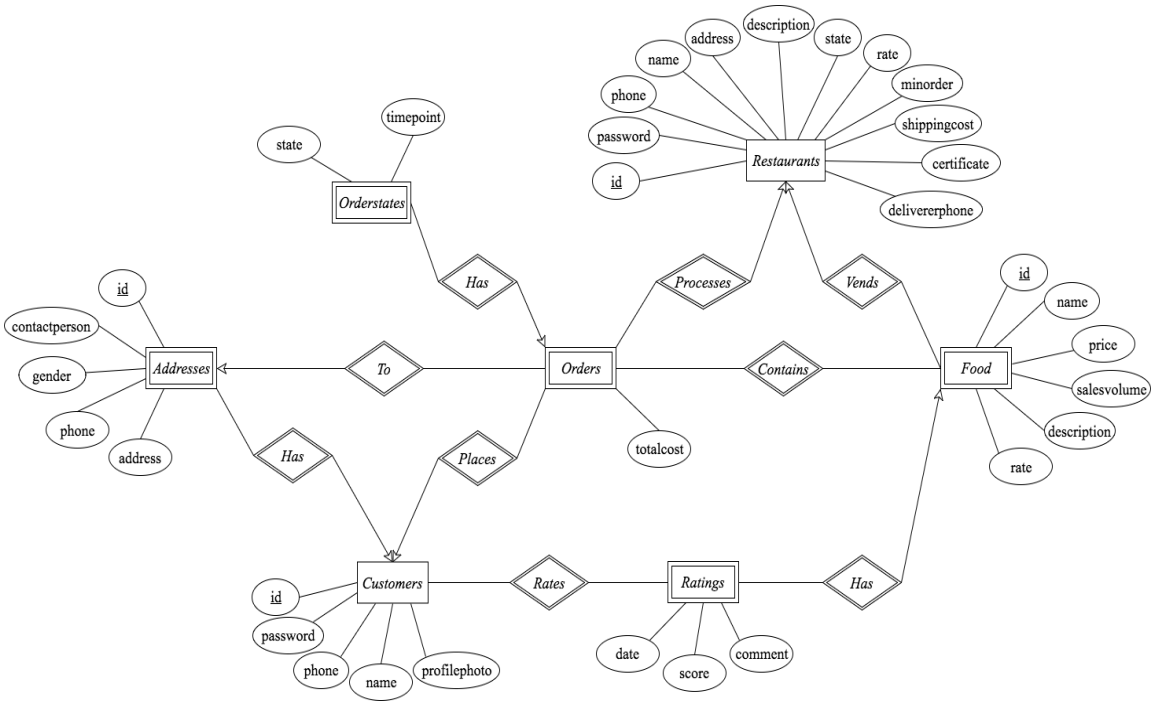

图 3. 系统 E/R 模型设计

需要声明的是，虽然我们设计之初规划了整个 E/R 模型，但最后实现还是有所差别。拿地址信息举例来说，我们曾经将其作为一个实体来对待，因为一个用户可以拥有多个地址。后来，为了简化设计，我们考虑到用户在网页上查询的第一个步骤就是输入地址，所以我们直接将该地址作为用户此次点单的送餐地址来处理，因此地址变成了 Order 的一个属性。

三、数据库模式

1. 用户信息

Customer（用户信息表）的主键由唯一指定的用户 ID：usr_id 构成，同时用户拥有自己的密码，手机号码，用户昵称，另外用户的微信号和邮箱作为可空属性。

在与其他 schema 的关联上，Customer 与 Delivery 是一对多的关系，代表一个用户可以点多单外卖，之间的虚线表示并非强连接，即 Delivery 不

一定要必须相应的 Customer 存在。另外一个 Customer 与 Rating（评分表）也是一对多的关系，代表一个用户可做出多项评分。

2. 餐厅信息

Restaurant（餐厅信息表）类似地以餐馆 ID：restaurant_id 作为唯一主键，同时一个餐厅拥有名称、配送范围、城市、街道、街道号、订单起送价格和用于管理自身数据库的密码，还有一个管理用户管理的 rate 属性作为餐馆的评分。

在与其他 schema 的关联上，一个餐馆可以对应多份 Delivery，一个餐馆可以拥有很多种类 Food。

3. 菜品信息

Food（食物信息表）在拥有 food_id 作为主键的同时，还有关联的餐馆的 restaurant_id 作为外键。Food 自身有食物名称、价格、卖出份数、评分和可选的描述信息属性。

在与其他 schema 的关联上，一个 Food 可以显示在很多个。订单的的列表 Delivery_Food_List 中，同时也唯一属于一个 Restaurant，在评分 Rating 中一种食物可以接受来自多个用户的评分，所以也是一对多的关系。

4. 外卖订单信息

Delivery（外卖信息表）代表了一份外卖，通常是由一个用户向一个餐馆发起的含有多份食物的订单，所以其中包含了用户 ID 和餐馆 ID 作为外键，同时每一份外卖有一个自动生成的 delivery_id 作为主键。注意到我们并没有在用户的信息中加入地理信息，而是将其作为属性放在了外卖信息表中，这代表了一个用户不一定只能在同一个地方点外卖。同时一份外卖会包含一个 total_price 作为订单价格。特别地，我们加入了配送员的概念，每一份外卖都有一个唯一的配送员，所以也将其作为了外键。

在与其他 schema 的联系上，首先一个用户或一个餐馆都可以有多个 delivery，另外每一个配送员会负责多份订单的配送。

同时我们需要记录外卖的配送状态，因为配送状态分为“已接单”，“制作中”，“配送中”，“已接单”和“订单完成”五个状态，如果都在 Delivery 中更新会导致冗余，所以我们将配送状态作为了一个单独的 schema，将对应的 delivery_id 作为外键，同时有 state 表示状态，time_point 用于记录时间戳。为了使顾客可见这一状态，我们创建了一个 View：Customer2Delivery 来为顾客查看订单状态提供视图。

更为重要的是，一份 Delivery 中拥有多份食物，同样地为了减小冗余，我们加入了新的 schema：Delivery_Food_List 用于标识一个订单内的食物类

型，所以也是一个一对多的关系。

5. 评分信息

这是我们系统额外的一部分，用于记录用户对每一种食物做出的评价。由于一个用户可以做出多份评价且一种食物可以接受多份评价，所以都是多对一的关系。同时 Rating 有时间戳和 rate 作为必需属性，而 comment 作为文字内容是可选属性。

6. 各类图关系表

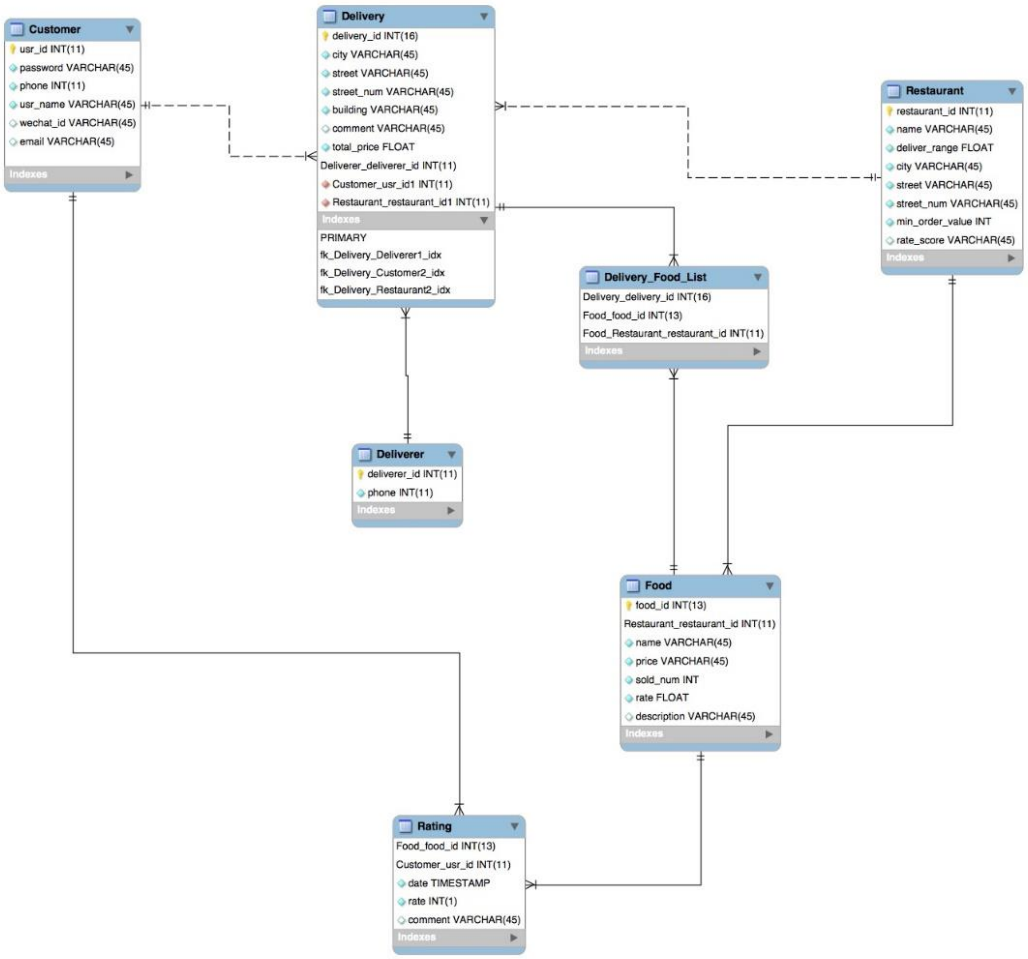

图 4. 数据库的 EER 模型

7. 系统中的约束

系统里的约束一部分是在数据库实现，第一类是外键约束。如新建外卖实体，这个外卖属性中的用户 ID 和餐馆 ID 一定要是有意义的。

```
--
-- Constraints for table `Delivery`
--
ALTER TABLE `Delivery`
  ADD CONSTRAINT `fk_User_has_Restaurant_Restaurant1` FOREIGN KEY (`
    Restaurant_restaurant_id`) REFERENCES `Restaurant` (`restaurant_id_pk`) ON
    DELETE NO ACTION ON UPDATE NO ACTION,
  ADD CONSTRAINT `fk_delivery_users1` FOREIGN KEY (`Customer_customer_id`)
    REFERENCES `Customer` (`customer_id_pk`) ON DELETE NO ACTION ON UPDATE NO
    ACTION;
```

图 5. 外键约束举例

另一类通过触发器实现，主要是在插入前对各个值进行检查。诸如在餐馆决定添加餐品时，其价格要符合实际意义（大于 0）；而新建一个商户时，它的属性也要符合实际意义，可见下图。

```
--
-- Triggers `Restaurant`
--
DELIMITER $$
CREATE TRIGGER `restaurant_constraints` BEFORE INSERT ON `restaurant`
 FOR EACH ROW BEGIN
    DECLARE msg VARCHAR(255);
    IF !(NEW.shipping_cost >= 0 &&
        NEW.min_order_value >= 0 &&
        NEW.max_delivery_range >= 0 &&
        NEW.position_lat >= 0 &&
        NEW.position_long >= 0)
    THEN
        SET msg = "DIE: You inserted a resctricted VALUE";
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = msg;
    END IF;
END
$$
DELIMITER ;
```

图 6. 触发器实现举例

这类触发器实现的约束有如下四种：

a) 订单里的某一个种类菜品的数量一定是大于零的；
b) 每一个菜品的价格一定是大于等于零的；
c) 一个菜品的 rate 是在 0~5 之间的数字；
d) 一个餐馆自身的配送费价格大于等于零、最小订单价格大于等于零、最大配送范围大于等于零、经纬度大于等于零。

而除此之外，我们对这个系统的约束还有一部分在数据接口层实现，如 javascript 代码中，当用户进行注册时，会做很多检查：密码是否高于 6 位，所有必要的注册信息是否都填写了，通过输入第二次来核对确认的信息是否不同等。为了方便起见，这些约束并没有在数据库底层实现，而是放到了数据库和前端的连接处。

```
/* Check that there is data in all fields */
var emptyFields = $('#task-register input').filter(function(){ return !this.value });
if(emptyFields.length > 0) {
  $(emptyFields[0]).popover(
    {content: 'Please fill in this field', placement: 'bottom'}).popover('show');
  return;
}

/* Check if password matches with its control */
if(params.pw != pw2) {
  $('#task-register #form-password-control').popover(
    {content: 'The passwords do not match', placement: 'bottom'}).popover('show');
  return;
}

/* Check if the phone matches with its control */
if(params.phone != phone2) {
  $('#task-register #form-phone-control').popover(
    {content: 'The phone numbers do not match', placement: 'bottom'}).popover('show');
  return;
}

/* Check if password is long enough */
if(params.pw.length < 6) {
  $('#task-register #form-password').popover(
    {content: 'The password must have at least 6 characters', placement: 'bottom'}).popover('show');
  return;
}
```

图 7. main.js 中对用户注册的检查

再比如，用户在进行下单页面前，系统应检查订单的金额是否达到了起送价格，这种逻辑的实现也是在对应的 javascript 代码中：

```
/* Click handler for checkout button */
$('.btn-checkout').click(function() {
  /* Query value */
  cart.getCartValue(restaurant, function(value) {
    /* If value is high enough, forward to next page */
    if(value > 0 && value >= minOrderValue) {
      leaveTo('order.php?restaurant=' + restaurant);

    }

    /* If value is too low, show error */
    else {
      showErrorOverlay('Price Too Low', '多点一些吧，未达到起送价格，商家会不开心的！');

    }
  });
});
```

图 8. meals.js 中对订单价格的检查

四、用户界面

1. 顾客界面

顾客的用户界面主要包括地址查询界面（首页）、餐馆展示界面、餐品列表界面、下单界面、注册登录界面（不登录无法下单，除开首页外，之前的界面在顶栏都有登录的接口）、个人信息界面（可查看订单状态、给订单进行评分、查看历史订单）等。除此之外，在侧边栏有一个进入帮助信息界面的入口，展示了使用本平台的各类提示。侧边栏的下方有一些联系方式

和分享页面的入口。最后，顾客的用户界面还包括，在下单成功或发生故障后响应的提示界面。

2. 商户界面

商户的用户界面包括注册登录界面、订单处理界面（依次更改订单的状态）、菜单管理界面（可查询、添加、更改、删除商品）、商户信息界面（可以更改信息和图标）。

为了方便阅览，所有的用户界面并没有包含在报告中，详见附件。

五、事务流程

1. 用户订餐流程

a) 用户访问首页，输入自己当前所在位置，或者选择快速定位栏中的历史位置；
b) 调用高德地图 API 对输入的位置进行查找，获得该位置的经纬度信息，再调取数据库中满足距离范围要求的餐馆，在餐馆选择页挑选餐馆；
c) 用户进入餐馆菜单目录，进行菜品挑选，可以通过底部栏的按钮进行排序和查询，下单前订单总价需要满足最低起送价格；
d) 页面跳转至确认下单页面，确认下单时若用户未登陆，系统会提示进行登陆；
e) 用户输入用户名和密码后登陆；
f) 登陆成功后页面返回确认页面，确认后页面提示提交成功，并返回到用户个人信息页面显示订单信息。

2. 用户注册流程

a) 用户访问首页，搜索位置后进入餐馆页面，点击右上角登陆/注册按钮进行注册；
b) 输入用户的个人信息，当个人各项信息满足后页面提示创建账户成功。

3. 用户评价流程

a) 用户进入个人信息页，页面右侧显示已完成订单和待评价订单；
b) 用户选择任一待评价订单，在跳出的订单评价表中选择评分，并依情况加入一定的评价内容。

4. 餐馆处理订单流程

a) 餐馆登陆管理页面；

b) 餐馆在“待接收的订单”中查看用户提交的订单，选择 process 则代表餐馆已接单；
c) 餐馆在“制作中的订单中”查看已经在制作的订单，制作完成后选择 in delivery 则代表进入配送状态；
d) 餐馆在“配送中的订单”中查看正在配送的订单，选择 complete 则代表用户已接单，该订单完成；
e) 餐馆在“已完成的订单”中查看已经完成的订单信息。

5. 餐馆管理菜品

a) 餐馆登陆管理页面；
b) 餐馆在“菜单管理”中查看已有的菜品信息，点开菜品名以查看具体的菜品信息或修改；
c) 餐馆在“菜单管理”中选择添加菜品，输入新加入的菜品信息后自动返回菜单管理页面，可以查看新加入的菜品信息。

6. 餐馆更改信息

a) 餐馆登陆管理页面；
b) 餐馆在“餐馆信息”中查看已有的餐馆信息，并对需要修改的具体信息进行修改；
c) 修改完成后点击保存，则页面返回管理页面，并可再次查看新的餐馆信息。

六、项目总结

1. 收获与心得

经过数周围绕系统设计的讨论和一个月时间的集中开发，我们的系统终于在答辩验收的前一天完成。不得不感慨，这是一个十分庞大的项目。虽然整个过程十分辛苦，并在期末考试周继续熬夜开发的压力下我们两个人也有过不少苦衷，但是经过努力和不断地尝试，总体来说我们已经尽最大能力完善了这一个系统。

从对一个问题的分析，到对应数据库的建模分析和用例场景分析，我们都有了更为贴近现实的斟酌。在前期大量的时间中我们反复讨论数据库本身的设计对系统整体功能的影响，而且在此过程中我们多次询问老师意见，收获了很多有用的建议。在数据库编写过程中，我们对 SQL 语言有了更为深入的体会与了解，同时通过实践也加深了我们对数据库这门课的知识掌握。

虽说课题名为“数据库课程设计”，但实际上包含的内容已远超过 SQL

语言和其与 PHP 之间的交互，在我们看来，前端和后端之间的通信，尤其是 JavaScript 的设计也是苦难之一。我们利用自身已有的前端和后端代码编写经验对系统的用户界面和控制逻辑进行了优化，使得系统尽可能用户友好，同时也使我们对前后端语言有了更为熟练的掌握。

此外，我们还收获了对工程编码规范的认识。我们在前期代码编写中，因为是两个人分开进行，所以在代码合并时出现了很多代码一致性上面的麻烦。经过讨论后我们决定先进行更为合理的分工，同时对整体工程的环境、编码方式和组织结构进行统一规划。

最后，在整个项目进行过程中我们进行了大量的学习，通过网络资源对解决问题的办法进行搜索，加强了独立解决问题的能力，尤其是 AMPPS 平台的掌握和应用，大大提升了我们的效率。

2. 问题与不足

由于开发时间不足，后期我们没有办法实现一些我们在一开始规划的功能，如允许每个用户保留多个地址，因为这样做的话地址既与用户多对一关联，同时要与订单关联，会造成逻辑上的复杂性。我们退而求其次，通过把地址信息从用户属性中剔除，将其只与订单本身一对一关联。再比如，我们也曾想过为每个配送员再添加一个控制界面，这样每个配送员可以对配送中的订单进行处理，与现实中的场景更为接近。同样因为开发时间原因我们没有能完成这一部分的工作。

配置环境的过程中我们发现 MySQL 对中文的支持性不好，虽然表面上支持 gbk 和 utf8 的编码格式，但是在配置过程中我们屡次出现了中文在页面上显示为乱码的情形。我们花了大量的时尝试间解决这一问题，并且网络上因为个人配置的环境变量都有差异的原因解决的方法众说纷纭，且这一问题我们仍然有待解决。

由于数据来源匮乏且获取方法较为繁琐，我们只在某些地点插入了餐厅的数据，并没有在更大地理范围上进行过搜索餐厅的测试。可能数据量没有达到老师的期望。

和现实中的“饿了么”相比，我们的系统还存在一些漏洞，如缺乏支付接口、没有红包和“满减”设计、商户对订单的处理不受监控等。值得一提的是开始我们尝试加入短信验证注册的功能，并且也获取了阿里大于的短信验证码 API 许可。但是后来发现这一 API 无法在本地主机上调用，也就是说只有在拥有域名的情况下该 API 才可以被正常调用。我们为了解决这一问题特地去租借了云服务器和域名（elema.info），但是因为国家网络部门的备案时间过长最终没有实现这一功能。