

数据库实践课程报告

题目： 饱了么外卖平台

姓名： 朱勇赤

学号： 10152130131

完成时间： 2018-6-25

华东师范大学计算机科学技术系

目 录(供参考, 可根据实际情况调整)

一. 系统需求分析.....	1
1. 系统描述.....	1
2. 数据存储需求.....	1
3. 系统常做的查询与更新.....	2
4. 应用程序功能.....	3
二. 数据库概念设计.....	6
1. 确定实体和属性.....	6
2. E-R 图.....	6
三. 数据库逻辑结构设计.....	8
1. 关系模式设计.....	8
2. 基本表设计.....	9
四. 数据库物理设计和实施.....	12
1. 数据库的创建.....	12
2. 创建基本表.....	12
3. 触发器设计.....	18
4. 存储过程设计.....	18
5. 视图设计.....	19
6. 其他数据库对象的设计.....	20
7. 防止 SQL 注入.....	20
五. 应用程序设计.....	21
1. 开发及运行环境介绍.....	21
2. 主要功能设计.....	22
3. 主要界面.....	28
六. 心得体会.....	39

(此目录由 WORD 自动生成。正文中的各类标题只须更改内容, 不要更改格式。最后在本页中右击上方的目录区域, 选择“更新域”命令。)

一. 系统需求分析

1. 系统描述

随着网络与智能手机的发展，餐品销售模式由传统的店铺经营逐渐发展到网络经营，越来越多的消费者选择网上订餐。网上外卖平台系统能够保证餐品销售信息的准确性，减少用户花费，提高配送效率。

网上外卖平台系统包括菜品查询、购买、评价等用户功能，以及菜品添加、删除、修改等商家管理功能。本系统面向的用户分为普通用户、商家和骑手三类。商家负责出售，管理菜品；普通用户可以查找菜品并下订单；骑手则可选择订单接单并进行配送。

网上外卖平台系统跨越了时间和空间的限制，给餐饮行业带来了变革，也给消费者带来了便捷。

2. 数据存储需求

外卖平台系统数据库需要存储如下信息---

用户：用户名、密码、地址、电话号码

骑手：骑手名（账号）、密码、电话号码

商家：商家名（账号）、密码、地址

菜品：菜品编号、菜名、价格、所属商家名

购物车：菜品编号、商家名、用户名

订单：订单编号、用户名、商家名、骑手名、总价、运费

订单明细：订单编号、菜品编号

骑手评价：订单编号、骑手名、用户名、评价内容

商家评价：订单编号、商家名、用户名、评价内容

其中---

一种菜品只属于一个商家，每一个商家可以售卖多种菜品；

若干属于相同用户与商家的购物车条目构成一个购物车；

一个用户可拥有多个购物车，一个商家可产生多个购物车；

一对用户与商家只可拥有一个购物车，一个购物车只对应一对用户与商家；

一个订单由一个购物车生成；

若干拥有相同订单编号的订单明细条目构成了一个订单的内容；

一个用户可撰写多条商家评价，一个商家可拥有多条评价，一个商家评价只属于一对用户与商家；

一个订单只能对商家评论一次，一个商家评价对应一个订单；

一个用户可撰写多条骑手评价，一个骑手可拥有多条评价，一个骑手评价只属于一对用户与骑手；

一个订单只能对骑手评论一次，一个骑手评价对应一个订单；

3. 系统常做的查询与更新

经常做的查询，或许对创建索引有影响的：

- 查询含有某些关键字词的菜品
- 查询含有某些关键字词的商店
- 查询某用户在某商店选购的菜品（即购物车）
- 查询所有未被接取的订单
- 骑手查看所有对自己的评价
- 商家查看所有对自己的评价

- 在选定的商店内查询包含某些关键字词的菜品

根据经常做的查询，需要创建有关视图的：

- 查找已被接单，未完成的属于本用户的订单
- 查找已完成，未被评价的属于本用户的订单

关于更新：

- 用户修改自己的密码；
- 菜品更改价格以及菜名；
- 骑手接单后，被更新为已接单，用户和商家将会看见配送者，其他骑手不可再次接单该订单；
- 用户收到订单后确认送达，订单被更新为已完成；
- 用户确认送达后，对于订单的配送以及菜品分别评价，商家和骑手的评价会被更新。

4. 应用程序功能

用户界面的功能如下：

1) 个人中心：

- a. 用户注册：注册用户名不能重复，电话号码需要符合格式；
- b. 用户登录：依据数据库中记录的用户名与密码信息进行验证，符合后方可登录；
- c. 个人信息维护：登录后密码可被修改；

2) 查询菜品以及店铺：

- a. 查询菜品：查询含有某些关键字词的菜品，结果显示菜品的编号，名称，价格，店铺，选择对应结果后进入所在店铺；
 - b. 查询店铺：查询含有某些关键字词的店铺，结果显示店铺的名称，地址，选择对应结果后进入该店铺；
- 3) 进入店铺：在店铺的子页面可查询属于该店铺的所有菜品，查询含有某些关键字词的菜品，结果显示菜品的编号，名称，价格。可选择菜品加入购物车，并可查看购物车；
- 4) 购物车：
 - a. 查看：显示所有加入了购物车的菜品的 id，名称，价格，页面右上角显示购物车菜品的总价；
 - b. 删除：选中菜品可将菜品从购物车中移除，同时显示的总价会发生变化；
 - c. 全部结算：将购物车所有条目删除，并生成对应的订单条目以及订单明细条目，结算时提示用户订单总价；
- 5) 确认订单送达：用户可查看自己的订单是否有骑手接单，骑手名是什么，以及选中对应订单可以确认送达；
- 6) 评价订单：
 - a. 评价骑手：可选中已送达订单对配送该订单的骑手进行评价；
 - b. 评价商家：可选中已送达订单对该订单的商家进行评价；

商家界面的功能如下：

- 1) 查看在售菜品：
 - a. 主页面：显示所有属于该商家的菜品编号，名称，价格；
 - b. 修改商品信息：更改菜品的名称以及价格
 - c. 下架商品：将菜品从商家在售菜品中移除
- 2) 上架菜品：输入新菜品的名称，价格，系统自动分配编号，菜品即可成为商家在售菜品
- 3) 查看未完成订单：
 - a. 主界面：查看用户已下单但还没确认完成的订单的编号，下单用户，配送骑手，骑手电话，总价，配送费等

- b. 查看明细：选中未完成订单可查看该订单包含哪些菜品
- 4) 查看评价：显示已完成订单编号以及对应的用户评价，但不显示用户名

骑手界面的功能如下：

- 1) 接取订单：显示所有未被接取的订单编号，用户名，用户地址，商家名，商家地址等
- 2) 查看已接订单：显示所有已被接取未送达的订单编号，用户名，用户地址，用户电话，商家名，商家地址等
- 3) 查看评价：显示已完成订单编号以及对应的用户评价，但不显示用户名

二. 数据库概念设计

1. 确定实体和属性

分析外卖平台系统的系统需求，将系统中设计的人、物进行抽象，得到了系统的实体如下：

- 1) 用户信息实体集。属性包括：用户名、登录密码、电话号码、地址
- 2) 商家实体集。属性包括：商家名、登录密码、地址
- 3) 骑手实体集。属性包括：骑手名、登录密码、电话号码
- 4) 菜品实体集。属性包括：菜品编号、菜名、价格、所属商家名
- 5) 订单实体集。属性包括：订单编号、用户名、商家名、骑手名、总价、运费、是否送达、是否被评价
- 6) 订单详情实体集。属性包括：订单编号、菜品编号
- 7) 购物车实体集。属性包括：菜品编号、商家名、用户名
- 8) 商家评论实体集。属性包括：订单编号、骑手名、用户名、评价内容
- 9) 骑手评论实体集。属性包括：订单编号、商家名、用户名、评价内容

2. E-R 图

系统 E-R 图如图 2-1 所示：

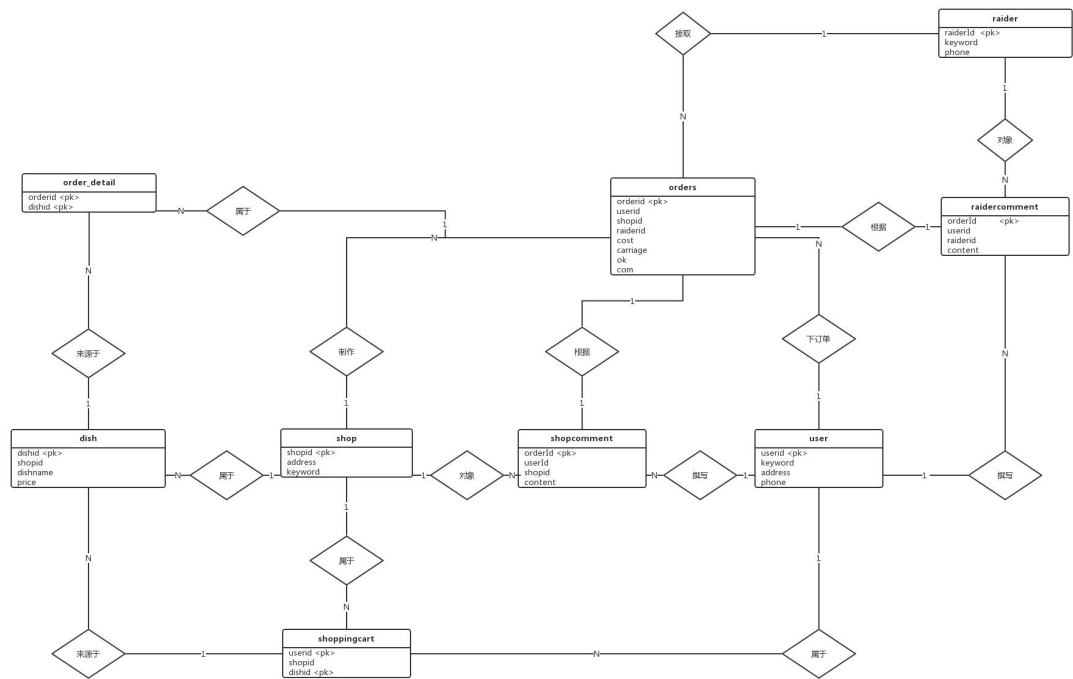


图 2-1 E-R 图

三. 数据库逻辑结构设计

1. 关系模式设计

根据概念结构设计得到的 E-R 图和转换规则，得到如下关系模式（主键用下划线标出，外键予以说明）：

- 用户表（用户名、登录密码、电话号码、地址）
- 商家表（商家名、登录密码、地址）
- 骑手表（骑手名、登录密码、电话号码）
- 菜品表（菜品编号、菜名、价格、商家名）
外键： 商家名 参照 商家表
- 订单表（订单编号、用户名、商家名、骑手名、总价、运费、是否送达、是否被评价）
外键： 用户名 参照 用户表
商家名 参照 商家表
骑手名 参照 骑手表
- 订单详情表（订单编号、菜品编号）
外键： 订单编号 参照 订单表
菜品编号 参照 菜品表
- 购物车表（菜品编号、商家名、用户名）
外键： 用户名 参照 用户表
商家名 参照 商家表
菜品编号 参照 菜品表
- 商家评论表（订单编号、骑手名、用户名、评价内容）
外键： 用户名 参照 用户表
订单编号 参照 订单表
骑手名 参照 骑手表
- 骑手评论表（订单编号、商家名、用户名、评价内容）
外键： 用户名 参照 用户表

订单编号 参照 订单表

商家名 参照 商家表

2. 基本表设计

基本表设计如表 3-1~3-*所示。

表 3-1：用户信息表的设计

属性名	数据类型	是否可空	列约束	默认值	键	解释
userid	varchar(20)	否			主键	用户名
Address	varchar(20)	否				地址
phone	varchar(20)	否				用户电话
keyword	varchar(20)	否				用户密码

表 3-2：商店表的设计

属性名	数据类型	是否可空	列约束	默认值	键	解释
Shopid	varchar(20)	否			主键	商店名
address	varchar(20)	否				地址
keyword	varchar(20)	否				商店密码

表 3-3：骑手表的设计

属性名	数据类型	是否可空	列约束	默认值	键	解释
raiderid	varchar(20)	否			主键	骑手名
phone	varchar(20)	否				电话
keyword	varchar(20)	否				骑手密码

表 3-4：菜品表的设计

属性名	数据类型	是否可空	列约束	默认值	键	解释
dishid	varchar(20)	否			主键	商店名

dishname	varchar(20)	否				地址
shopid	varchar(20)	否				商店密码
price	Decimal(10, 2)	否	Price>=0			菜品价格

表 3-5: 购物车表的设计

属性名	数据类型	是否可空	列约束	默认值	键	解释
Shopid	varchar(20)	否				商店名
Userid	varchar(20)	否			主键	用户名
dishid	varchar(20)	否			主键	菜品编号

表 3-6: 订单表的设计

属性名	数据类型	是否可空	列约束	默认值	键	解释
Orderid	varchar(20)	否			主键	菜品编号
Shopid	varchar(20)	否				商店名
Userid	varchar(20)	否				用户名
raiderid	varchar(20)	是				骑手名
cost	Doubel(10)	否				总价
carriage	Doubel(10)	否		5		运费
OK	Char(1)	否				是否被接单 (0 否/1 是)
Com	Char(1)	否				是否被评价 (0 否/1 是)

表 3-7: 订单明细表的设计

属性名	数据类型	是否可空	列约束	默认值	键	解释
Orderid	varchar(20)	否				商店名

Dishid	varchar(20)	否			主键	菜品编号
--------	-------------	---	--	--	----	------

表 3-8：商家评价表的设计

属性名	数据类型	是否可空	列约束	默认值	键	解释
Shopid	varchar(20)	否				商店名
Userid	varchar(20)	否			主键	用户名
orderid	varchar(20)	否			主键	菜品编号
content	varchar(255)	是				评价内容

表 3-9：骑手评价表的设计

属性名	数据类型	是否可空	列约束	默认值	键	解释
Shopid	varchar(20)	否				商店名
Userid	varchar(20)	否			主键	用户名
orderid	varchar(20)	否			主键	菜品编号
content	varchar(255)	是				评价内容

四. 数据库物理设计和实施

1. 数据库的创建

基于 MySQL 和 Navicat Premium 建立外卖平台系统的数据库，命名为 baoleme

（由于 Mysql 文件创建与存储方式与原表形式不一致故无法填写）

2. 创建基本表

```
-- -----  
-- Table structure for dish  
-- -----  
  
CREATE TABLE `dish` (  
  `dishid` int(11) NOT NULL AUTO_INCREMENT,  
  `dishname` varchar(255) CHARACTER SET utf8mb4 COLLATE  
utf8mb4_0900_ai_ci NULL DEFAULT NULL,  
  `shopid` varchar(11) CHARACTER SET utf8mb4 COLLATE utf8mb4_0900_ai_ci  
NULL DEFAULT NULL,  
  `price` decimal(10, 2) NULL DEFAULT NULL,  
  PRIMARY KEY (`dishid`) USING BTREE,  
  INDEX `shopid` (`shopid`) USING BTREE,  
  CONSTRAINT `dish_ibfk_1` FOREIGN KEY (`shopid`) REFERENCES `shop`  
(`shopid`) ON DELETE RESTRICT ON UPDATE RESTRICT  
)  
  
-- -----  
-- Table structure for order_detail  
-- -----
```

```
CREATE TABLE `order_detail` (  
  `orderid` int(11) NOT NULL,  
  `dishid` int(11) NOT NULL,  
  PRIMARY KEY (`orderid`, `dishid`) USING BTREE,  
  INDEX `dishid` (`dishid`) USING BTREE,  
  CONSTRAINT `order_detail_ibfk_1` FOREIGN KEY (`orderid`) REFERENCES  
`orders` (`orderid`) ON DELETE RESTRICT ON UPDATE RESTRICT,  
  CONSTRAINT `order_detail_ibfk_2` FOREIGN KEY (`dishid`) REFERENCES  
`dish` (`dishid`) ON DELETE RESTRICT ON UPDATE RESTRICT  
)
```

```
--  
-- Table structure for orders  
--
```

```
CREATE TABLE `orders` (  
  `orderid` int(11) NOT NULL AUTO_INCREMENT,  
  `userid` varchar(11) CHARACTER SET utf8mb4 COLLATE utf8mb4_0900_ai_ci  
NULL DEFAULT NULL,  
  `shopid` varchar(11) CHARACTER SET utf8mb4 COLLATE utf8mb4_0900_ai_ci  
NULL DEFAULT NULL,  
  `cost` double(10, 2) NULL DEFAULT NULL,  
  `carriage` double(10, 0) NULL DEFAULT NULL,  
  `raiderid` varchar(11) CHARACTER SET utf8mb4 COLLATE  
utf8mb4_0900_ai_ci NULL DEFAULT NULL,  
  `ok` char(1) CHARACTER SET utf8mb4 COLLATE utf8mb4_0900_ai_ci NULL  
DEFAULT '0',  
  `com` char(1) CHARACTER SET utf8mb4 COLLATE utf8mb4_0900_ai_ci  
NULL DEFAULT '0',  
  PRIMARY KEY (`orderid`) USING BTREE,  
  INDEX `userid` (`userid`) USING BTREE,
```

```
INDEX `shopid`(`shopid`) USING BTREE,
INDEX `raiderid`(`raiderid`) USING BTREE,
CONSTRAINT `orders_ibfk_1` FOREIGN KEY (`userid`) REFERENCES `user`
(`userid`) ON DELETE RESTRICT ON UPDATE RESTRICT,
CONSTRAINT `orders_ibfk_2` FOREIGN KEY (`shopid`) REFERENCES `shop`
(`shopid`) ON DELETE RESTRICT ON UPDATE RESTRICT,
CONSTRAINT `orders_ibfk_3` FOREIGN KEY (`raiderid`) REFERENCES
`raider` (`raiderid`) ON DELETE RESTRICT ON UPDATE RESTRICT
)
```

```
-- -----
-- Table structure for raider
```

```
-- -----
CREATE TABLE `raider` (
  `raiderid` varchar(11) CHARACTER SET utf8mb4 COLLATE
utf8mb4_0900_ai_ci NOT NULL,
  `phone` varchar(255) CHARACTER SET utf8mb4 COLLATE
utf8mb4_0900_ai_ci NULL DEFAULT NULL,
  `keyword` varchar(255) CHARACTER SET utf8mb4 COLLATE
utf8mb4_0900_ai_ci NULL DEFAULT NULL,
  PRIMARY KEY (`raiderid`) USING BTREE
)
```

```
-- -----
-- Table structure for raider_comment
```

```
-- -----
DROP TABLE IF EXISTS `raider_comment`;
CREATE TABLE `raider_comment` (
  `userid` varchar(11) CHARACTER SET utf8mb4 COLLATE utf8mb4_0900_ai_ci
NOT NULL,
```



```

    `raiderid`    varchar(11)    CHARACTER SET    utf8mb4    COLLATE
utf8mb4_0900_ai_ci NULL DEFAULT NULL,
    `orderid` int(11) NOT NULL,
    `content`    varchar(255)    CHARACTER SET    utf8mb4    COLLATE
utf8mb4_0900_ai_ci NULL DEFAULT NULL,
    PRIMARY KEY (`orderid`) USING BTREE,
    INDEX `userid`(`userid`) USING BTREE,
    INDEX `raiderid`(`raiderid`) USING BTREE,
    CONSTRAINT `raider_comment_ibfk_1` FOREIGN KEY (`userid`)
REFERENCES `user` (`userid`) ON DELETE RESTRICT ON UPDATE RESTRICT,
    CONSTRAINT `raider_comment_ibfk_2` FOREIGN KEY (`raiderid`)
REFERENCES `raider` (`raiderid`) ON DELETE RESTRICT ON UPDATE
RESTRICT,
    CONSTRAINT `raider_comment_ibfk_3` FOREIGN KEY (`orderid`)
REFERENCES `orders` (`orderid`) ON DELETE RESTRICT ON UPDATE
RESTRICT
)

```

```

-- -----
-- Table structure for shop
-- -----

```

```

CREATE TABLE `shop` (
    `shopid`    varchar(255)    CHARACTER SET    utf8mb4    COLLATE
utf8mb4_0900_ai_ci NOT NULL,
    `address`    varchar(255)    CHARACTER SET    utf8mb4    COLLATE
utf8mb4_0900_ai_ci NULL DEFAULT NULL,
    `keyword`    varchar(255)    CHARACTER SET    utf8mb4    COLLATE
utf8mb4_0900_ai_ci NULL DEFAULT NULL,
    PRIMARY KEY (`shopid`) USING BTREE
)

```

-- Table structure for shop_comment

```
CREATE TABLE `shop_comment` (
  `userid` varchar(255) CHARACTER SET utf8mb4 COLLATE
utf8mb4_0900_ai_ci NOT NULL,
  `shopid` varchar(255) CHARACTER SET utf8mb4 COLLATE
utf8mb4_0900_ai_ci NOT NULL,
  `content` varchar(255) CHARACTER SET utf8mb4 COLLATE
utf8mb4_0900_ai_ci NULL DEFAULT NULL,
  `orderid` int(11) NOT NULL,
  PRIMARY KEY (`orderid`) USING BTREE,
  INDEX `userid`(`userid`) USING BTREE,
  INDEX `shopid`(`shopid`) USING BTREE,
  CONSTRAINT `shop_comment_ibfk_1` FOREIGN KEY (`userid`)
REFERENCES `user` (`userid`) ON DELETE RESTRICT ON UPDATE RESTRICT,
  CONSTRAINT `shop_comment_ibfk_2` FOREIGN KEY (`shopid`)
REFERENCES `shop` (`shopid`) ON DELETE RESTRICT ON UPDATE
RESTRICT,
  CONSTRAINT `shop_comment_ibfk_3` FOREIGN KEY (`orderid`)
REFERENCES `orders` (`orderid`) ON DELETE RESTRICT ON UPDATE
RESTRICT
)
```

-- Table structure for shopcart

```
CREATE TABLE `shopcart` (
```

```

`userid`    varchar(255)    CHARACTER SET    utf8mb4    COLLATE
utf8mb4_0900_ai_ci NOT NULL,
`shopid`    varchar(255)    CHARACTER SET    utf8mb4    COLLATE
utf8mb4_0900_ai_ci NOT NULL,
`dishid` int(11) NOT NULL,
PRIMARY KEY (`dishid`, `userid`) USING BTREE,
INDEX `userid`(`userid`) USING BTREE,
INDEX `shopid`(`shopid`) USING BTREE,
CONSTRAINT `shopcart_ibfk_1` FOREIGN KEY (`userid`) REFERENCES
`user` (`userid`) ON DELETE RESTRICT ON UPDATE RESTRICT,
CONSTRAINT `shopcart_ibfk_2` FOREIGN KEY (`shopid`) REFERENCES
`shop` (`shopid`) ON DELETE RESTRICT ON UPDATE RESTRICT,
CONSTRAINT `shopcart_ibfk_3` FOREIGN KEY (`dishid`) REFERENCES
`dish` (`dishid`) ON DELETE RESTRICT ON UPDATE RESTRICT
)

```

```
-- -----
```

```
-- Table structure for user
```

```
-- -----
```

```

CREATE TABLE `user` (
  `userid`    varchar(255)    CHARACTER SET    utf8mb4    COLLATE
utf8mb4_0900_ai_ci NOT NULL,
  `address`    varchar(255)    CHARACTER SET    utf8mb4    COLLATE
utf8mb4_0900_ai_ci NULL DEFAULT NULL,
  `phone`    varchar(255)    CHARACTER SET    utf8mb4    COLLATE
utf8mb4_0900_ai_ci NULL DEFAULT NULL,
  `keyword`    varchar(255)    CHARACTER SET    utf8mb4    COLLATE
utf8mb4_0900_ai_ci NULL DEFAULT NULL,
  PRIMARY KEY (`userid`) USING BTREE
)

```

3. 触发器设计

```
CREATE TRIGGER t_price
BEFORE
INSERT
ON `dish`
FOR EACH ROW
BEGIN
    IF new.price <0 THEN
        SET new.price = 0;
    END IF;
END
```

因为 mysql 检查列约束但列约束并不起作用，所以需要设置触发器来代替 check 约束；

触发器 T_price： 即当店铺商家商品价格小于 0 时，将商品价格更改为 0，之后将该条数据插入。

4. 存储过程设计

```
CREATE PROCEDURE `SumCart` ( IN users VARCHAR ( 20 ), IN shop
VARCHAR ( 20 ), OUT summ INT ) BEGIN
SELECT
    sum( price )
FROM
    shopcart
    JOIN dish ON dish.dishid = shopcart.dishid
WHERE
    shopcart.shopid = shop
    AND userid = USER;
```

```
SET summ = sum( price );
```

```
DELETE
```

```
FROM
```

```
    shopcart
```

```
WHERE
```

```
    userid = users
```

```
    AND shopid = shop;
```

```
END;
```

创建存储过程 SumCart: 将对应 in 变量 user 与 shop 的 shopcart 表中符合条件条目计算总价存入 out 变量 summ, 之后全部删除对应条目。

该存储过程实现了将购物车物品全部结算, 是较为复杂的过程, 开发过程中如果不使用存储过程将会在应用程序开发中写入大量 sql 语句, 不仅容易出错, 而且工程的严密性会出问题。

5. 视图设计

```
Create view 'Uncom'
```

```
As
```

```
Select  *
```

```
From orders
```

```
Where ok='1' and com='0'
```

创建视图 Uncom: 即在订单表 orders 中选择出 ok='1' (已完成) 且 com='0' (未评价) 的条目, 在用户查找历史订单并评价时, 只需从 Uncom 视图中寻找对应 Useid 的条目, 简化了 sql 语句且加快了搜索速度

```
Create view 'Unok'
```

```
As
```

```
Select  *
```

```
From orders
```

```
Where ok='0'
```

创建视图 Uncom: 即在订单表 orders 中选择出 ok='0' (未完成)。在用户查找未完成订单并评价时, 只需从 Unok 视图中寻找对应 Useid 的条目, 简化了 sql 语句且加快了搜索速度。

6. 其他数据库对象的设计

暂无

7. 防止 SQL 注入

SQL 注入是对 Python 与 MySQL 进行动态数据校验时, 用户故意输入非法字段, 从而绕过数据校验的行为。

SQL 注入的中心思想就是人为的输入 SQL 语句中的特殊字符串, 绕过验证。"--" 在 MySQL 中为注释字符, 通过此方法可以屏蔽部分代码, 从而绕过验证。

使用 MySQL 的内置方法校验输入字符串的合法性, 提高安全性。

例如:

#执行 sql 语句

sql='select * from user where name=%s and password=%s'#注意%s 没有加引号

五. 应用程序设计

1. 开发及运行环境介绍

系统使用的开发工具:

Navicat Premium 12

JetBrains Pycharm 2018

开发环境:

Python 3.5

WxPython

Mysql 8.0

2. 主要功能设计

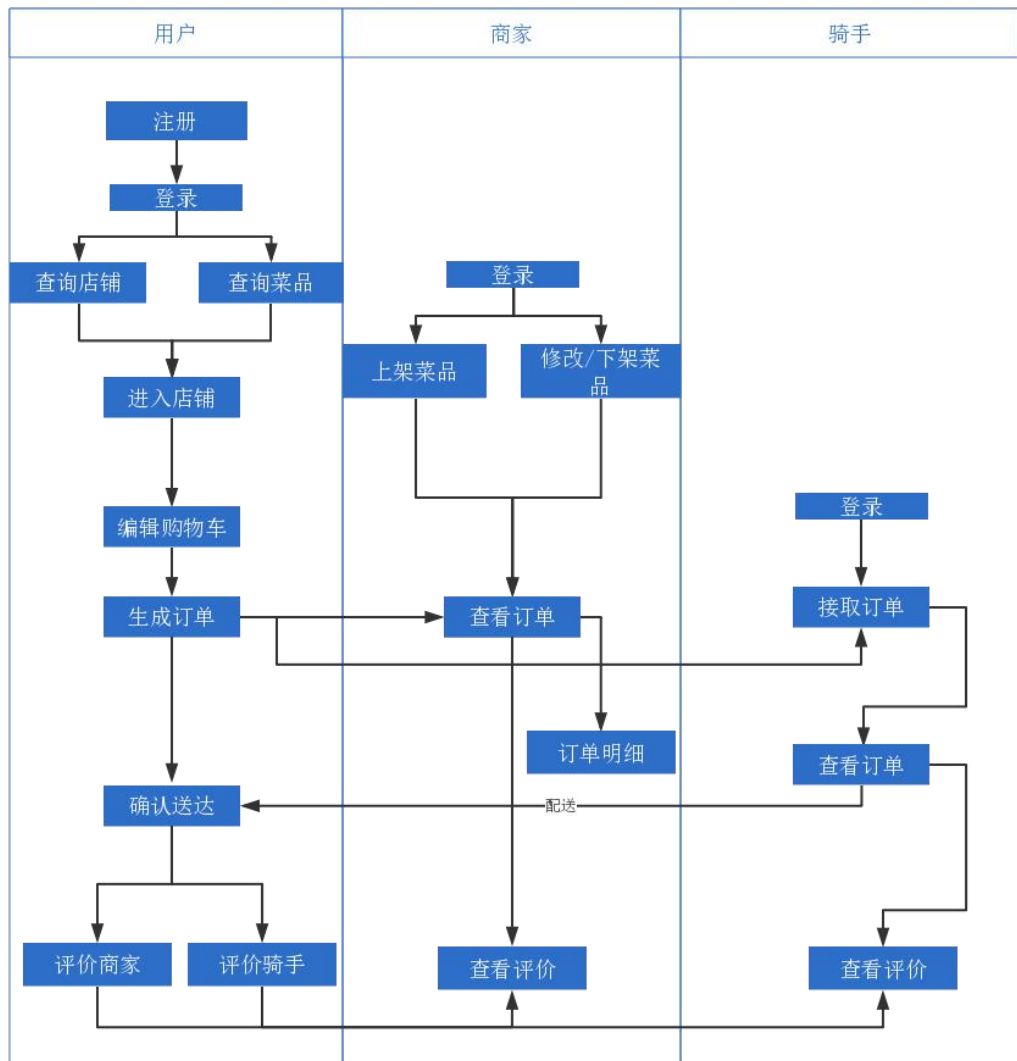


图 5-0 外卖平台角色功能与流程图

外卖平台分为三个角色，各自所能使用的所有功能与角色之间的交互如图 5-0 所示。

项目部分功能已在数据库对象设计中有所解释，因篇幅所限以下列举典型功能（增删改查）的实现。

1、登录功能:

```
conn = pymysql.connect("localhost", "root", "password", "baoleme",
charset='utf8')

cursor = conn.cursor()

data = (user, self.pwd)

try:

    sql = "select * from user where userid = %s and keyword = %s"

    result = cursor.execute(sql, data)

    #获取用户名与密码相同的用户表条目数

    if result == 0:

        print('用户名或密码错误')

        dial = wx.MessageDialog(None, '用户名或密码错误!', '错误
信息', wx.YES_NO) # 创建一个带按钮的对话框, 语法是(self, 内容, 标题, ID)

        dial.ShowModal() # 显示对话框

    else:

        print('登陆成功')

        self.Close()

        Mydialogmu.OnCreate(None)

    conn.commit()
```

2、注册功能:

```
conn = pymysql.connect("localhost", "root", "password", "baoleme",
charset='utf8')

cursor = conn.cursor()

data = (student_phone, order_id, order_money, order_way)

try:

    print(data)

    sql = "insert into user (userid,address,phone,keyword)
values(%s,%s,%s,%s)"
```

```

        t = cursor.execute(sql, data)

        print(t)

        conn.commit()

        if t == 1:

            dial = wx.MessageDialog(None, '注册成功，请登录！', '结果', wx.YES_NO) # 创建一个带按钮的对话框，语法是(self, 内容, 标题, ID)

            dial.ShowModal() # 显示对话框

            self.Close()

        #else:

            #dial = wx.MessageDialog(None, 'id 重复!', '结果', wx.YES_NO) # 创建一个带按钮的对话框，语法是(self, 内容, 标题, ID)

            #dial.ShowModal() # 显示对话框

        except:

            #出错即说明插入失败，id 有重复

            conn.rollback()

            dial = wx.MessageDialog(None, 'id 重复!', '结果', wx.YES_NO)

            # 创建一个带按钮的对话框，语法是(self, 内容, 标题, ID)

            dial.ShowModal() # 显示对话框

        finally:

            cursor.close()

            conn.close()

```

3、查询功能：

```

conn = pymysql.connect("localhost", "root", "password",
                        "baoleme", charset='utf8')

cursor = conn.cursor()

name=self.t1.GetValue()

print(name)

name='%'+name+'%'

try:

```

```

sql = "SELECT * FROM dish WHERE dishname LIKE %s "
t = cursor.execute(sql,name)
#print(t)
rs = cursor.fetchall()
#print(rs)
h = 80
for row in rs:
    h = h + 20
    dish_id = str(row[0])
    dishname = str(row[1])
    shopname = str(row[2])
    price = str(row[3])
    self.m_radioBtn5      =      wx.RadioButton(self.panel,
wx.ID_ANY, dish_id, (20,h))
    wx.StaticText(self.panel, -1, dishname, (120, h))
    wx.StaticText(self.panel, -1, shopname, (220, h))
    wx.StaticText(self.panel, -1, price, (320, h))
    #self.panel.Refresh()
    self.Bind(wx.EVT_RADIOBUTTON, self.OnRadio)
except:
    conn.rollback()
finally:
    cursor.close()
    conn.close()

```

4、下架功能（删除）：

```

conn = pymysql.connect("localhost", "root", "password", "baoleme",
charset='utf8')

cursor = conn.cursor()

# name = self.t1.GetValue()

```

```

        # print(name)

        # name = '%' + name + '%'

        try:

            global shop

            sql = "delete FROM dish WHERE dishid=%s "

            t = cursor.execute(sql, self.idon)

            print(t)

            print(self.idon)

            conn.commit()

            dial = wx.MessageDialog(None, '成功下架!', '结果',
wx.YES_NO)

            dial.ShowModal() # 显示对话框

            self.Close()

        except:

            conn.rollback()

        finally:

            cursor.close()

            conn.close()

```

修改功能:

```

conn = pymysql.connect("localhost", "root", "password",
"baoleme", charset='utf8')

cursor = conn.cursor()

global id

print(id)

name = self.t1.GetValue().encode('utf8') #注意 GetValue() 获取的是
unicode 编码,

price = self.t2.GetValue().encode('utf8') #你使用的#coding=utf8, 那就

```

对获取的数据.encode('utf8')重新编码

```
data = (name, price,id)
```

```
try:
```

```
    sql = "update dish set dishname=%s ,price=%s where dishid=%s"
```

```
    cursor.execute(sql, data)
```

```
    conn.commit() #提交给后台数据库
```

```
    dial = wx.MessageDialog(None, '成功修改!', '结果', wx.YES_NO)
```

```
# 创建一个带按钮的消息框, 语法是(self, 框中内容, 框标题, ID)
```

```
    dial.ShowModal() # 显示对话框
```

```
    self.Close()
```

```
except:
```

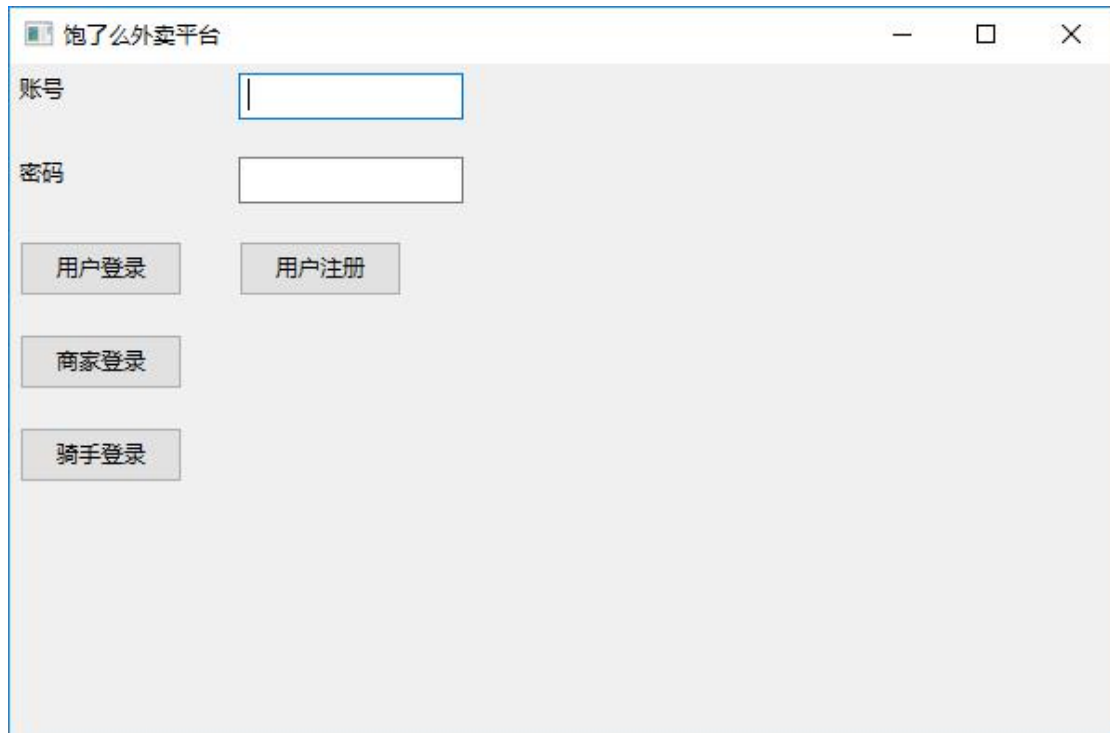
```
    conn.rollback()
```

```
finally:
```

```
    cursor.close()
```

```
    conn.close()
```

3. 主要界面



饱了么外卖平台

账号

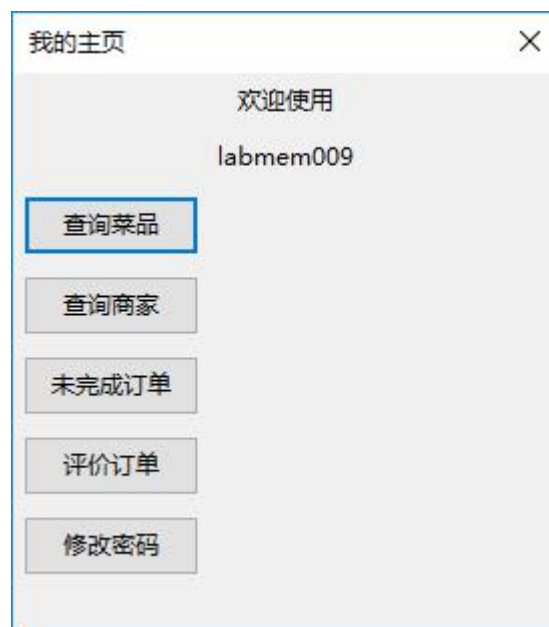
密码

用户登录 用户注册

商家登录

骑手登录

图 5-1 登录界面



我的主页

欢迎使用
labmem009

查询菜品

查询商家

未完成订单

评价订单

修改密码

图 5-2 用户主页

订单信息

×

查询菜名 (查询所有则置空) :

菜品id	菜名	商家	价格
<input checked="" type="radio"/> 20	板烧鸡腿堡	金拱门	14.50
<input type="radio"/> 23	麦辣鸡翅	金拱门	8.00
<input type="radio"/> 24	上校鸡块	KFC	10.50
<input type="radio"/> 27	劲辣鸡腿堡	KFC	18.00
<input type="radio"/> 31	原凉色鸡腿	河东饭店	5.00
<input type="radio"/> 40	烤鸡翅	撸串烧烤	5.00
<input type="radio"/> 47	黄焖鸡米饭	丽娃饭店	10.00
<input type="radio"/> 48	腐竹黄焖鸡米饭	丽娃饭店	12.00
<input type="radio"/> 49	麦乐鸡块	金拱门	8.00

图 5-3 查询菜品页面

订单信息

×

查询店名 (查询所有则置空) :

商家名	地址
<input type="radio"/> 金拱门	环球港B2

图 5-4 查询商家页面

店铺 ×

查询菜名 (查询所有则置空) :

菜品id	菜名	价格
<input type="radio"/> 20	板烧鸡腿堡	14.50
<input type="radio"/> 21	巨无霸汉堡	20.00
<input type="radio"/> 22	薯条	6.00
<input type="radio"/> 23	麦辣鸡翅	8.00
<input checked="" type="radio"/> 49	麦乐鸡块	8.00

图 5-5 选购商家页面

购物车 ×

菜品总价为: 28.50元

菜品id	菜品名	单价	商店
<input checked="" type="radio"/> 20	板烧鸡腿堡	14.50	金拱门
<input type="radio"/> 22	薯条	6.00	金拱门
<input type="radio"/> 49	麦乐鸡块	8.00	金拱门

图 5-6 购物车页面

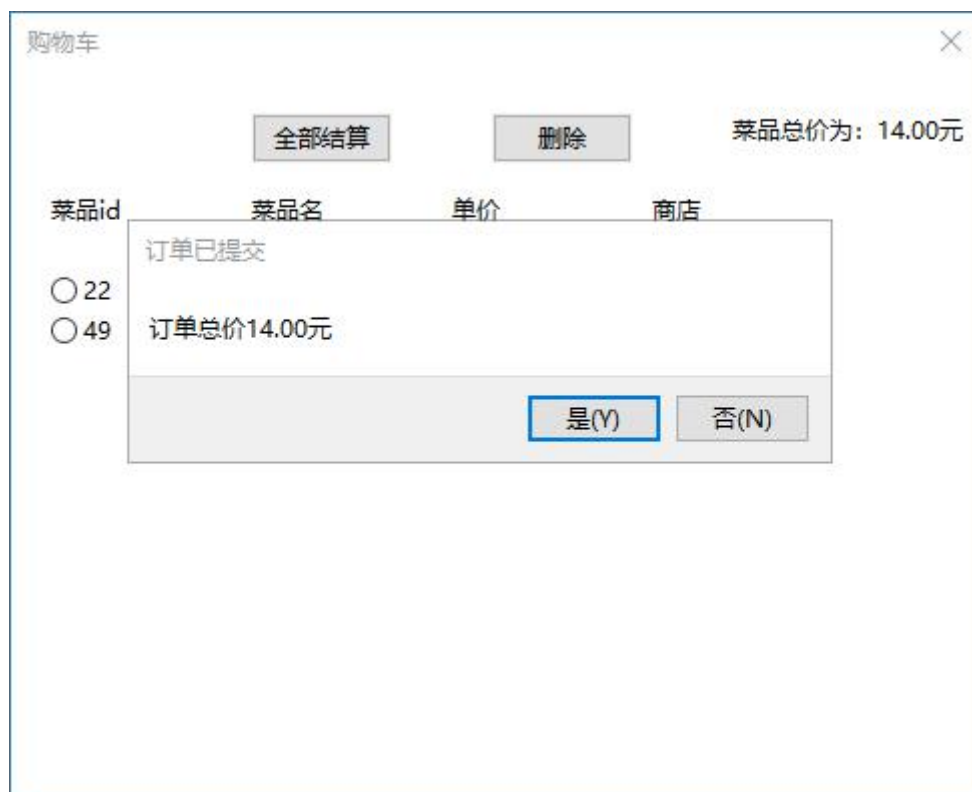


图 5-7 下单页面

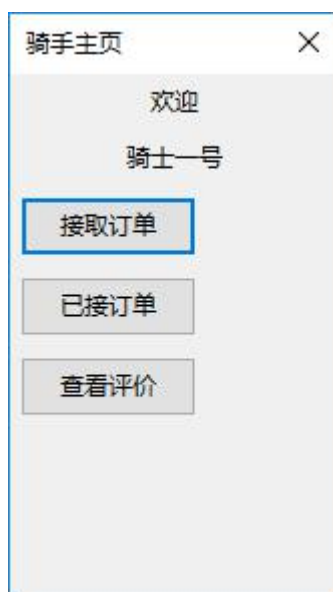


图 5-8 骑手主页

订单 ×

接受

订单编号	下单用户	用户电话	用户地址	接单商店	商店地址
● 18	labmem009	111111	中山北路	金拱门	环球港B2

图 5-9 接收订单

查看订单 ×

订单编号	下单用户	用户电话	用户地址	接单商店	商店地址
18	labmem009	111111	中山北路	金拱门	环球港B2

图 5-10 已接订单

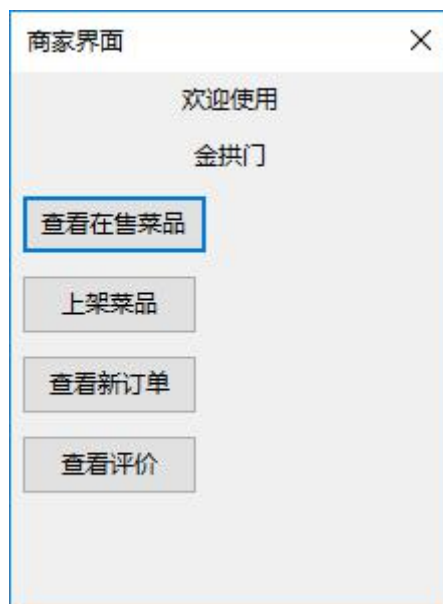


图 5-11 商家主页



图 5-12 商家在售商品

The screenshot shows a '在售菜品' (On-Sale Dishes) window with a table of dishes. A modal '菜品修改' (Dish Modification) window is open, allowing editing of dish 49. The new name is '麦乐鸡' and the new price is '13'.

菜品号	菜名	价格
<input type="radio"/> 20	板烧鸡腿堡	14.50
<input type="radio"/> 21		
<input type="radio"/> 22		
<input type="radio"/> 23		
<input checked="" type="radio"/> 49		

菜品修改

请输入新名称: 麦乐鸡

请输入新价格: 13

修改

图 5-13 修改在售商品

The screenshot shows an '上架菜品' (Shelf Dishes) window. It contains input fields for '请输入菜名:' (Please enter dish name) with the value '麦香鸡' and '请输入价格:' (Please enter price) with the value '6'. There is an '上架' (Shelf) button at the bottom.

上架菜品

请输入菜名: 麦香鸡

请输入价格: 6

上架

图 5-14 上架商品



图 5-15 商家查看订单



图 5-16 商家查看订单明细

订单

×

确认送达

订单编号	订单金额	配送骑手	接单商家
<input checked="" type="radio"/> 18	14.0	骑士一号	金拱门

图 5-17 用户确认送达

订单


×

评价菜品

评价配送

订单编号	订单金额	配送骑手	接单商店
<input type="radio"/> 18	14.0	骑士一号	金拱门

图 5-18 用户评价已完成订单



评价配送

评价: 很慢

确定

图 5-19 用户评价配送



评价菜品

评价: 难吃

确定

图 5-20 用户评价商家

查看评价		✕	
订单编号		评价详情	
16		快	
17		快	
18		很慢	

图 5-21 骑手查看评价

查看评价		✕	
订单编号		评价详情	
17		好吃	
18		难吃	

图 5-22 商家查看评价

六. 心得体会

1. 在数据库设计方面:

一是部分表主键的确定,如购车表本来设计三个属性均为主键,后发现菜品编号和用户名可以唯一确定一个条目,如两评价表都只需订单编号为主键即可,不需要用户,商家或是骑手来辅助确定;

二是菜品是否需要编号这一属性,原先的考虑是菜品也可以通过菜品名以及商店名共同唯一确定,但考虑到有修改菜品名称这一需求所以设置了自增属性菜品编号来作为菜品表的主键;

三是对于订单的状态是否需要新建表来存储,即订单被接取后从原表删除并添加至已被评价的订单表,后发现设计过于冗余,这种操作会添加两到三个表,且经常为空。最后选择对订单表添加 `ok`, `com` 两属性分别标志是否已完成,是否被评价;

四是外键选择是禁止删除,级联还是 `set null` 三种,最后为了数据路稳定性选择了禁止删除。

2. 应用开发方面:

一是 `python` 中的转义符 `'%'` 与 `sql` 的通配符 `'%'` 相同导致的查询语句编写问题,解决方法是将通配符与输入字段先连接为新字符串再传入 `sql` 语句;

二是在 `sql` 语句执行时使用 `try--except---catch` 结构来防止发生错误;

三是数据库表名称不可使用 `mysql` 的保留字段不然会执行 `sql` 语句出现 1064 错误;

四是编码问题,在连接数据库时使用 `utf8` 编码,在获取输入字段时也要使用 `utf8` 编码。这些问题不仅和 `sql` 有关也和开发应用的语言 (`python`) 的本身特性有关。

经过这次数据库应用的开发使我了解到,在以后开发数据库应用时需要注意所使用的数据库以及开发语言的特性与细节的问题。