长 沙 学 院

软件工程基础实训Ⅱ 实训

|  |  |
| --- | --- |
| **题目** | **基于C/S架构的餐饮管理系统** |
| **学院** | **计算机科学与工程** |
| **专业(班级)** | **软件工程（21软件01）** |
| **姓名** | **易志望** |
| **学号** | **B20210304117** |
| **指导教师** | **杨刚，潘怡** |
| **起止日期** | **2023.06.05～2023.06.16** |

实训任务书

**课程名称：软件工程基础实训II**

**实训题目：基于C/S架构的餐饮管理系统**

### 已知技术参数和设计要求：

1. 问题描述（功能要求）：

本次案例将完成一个基于C/S架构的餐饮管理系统。

（1）.进行餐饮管理系统的功能设计。

（2）.设计餐饮管理系统的数据库。

（3）.利用JAVA的GUI类设计开发友好的人机接口。

（4）.利用JAVA语言设计监听用户操作和JDBC进行数据库的处理。

餐饮管理系统主要有6大模块：

（1）.员工管理：对员工实现增删改查。

（2）.客户管理：对客户实现增删改查。

（3）.餐台管理：对餐台实现增删改查。

（4）.菜品管理：对菜品分类、菜品实现增删改查。

（5）.点菜管理：服务员对某客户、某一空闲餐台实行开台，同时实现点菜，将餐台号与所点的菜品对应起来，分别显示出来，并记录开台时间。

（6）.结账管理：收银员对某一餐台通过统计显示消费的菜品清单统计出消费金额，通过手动输入实收金额进行找零的计算，并显示，完成结账的操作，并记录成统计数据。

基础实训II有2周时间40学时，学生基本必做任务：独立完成基于C/S架构的餐饮管理系统。

选做扩展：菜品推荐、销售统计。

**2. 运行环境要求：**

（1）客户端：

windows操作系统，JDK1.6，Idea或Eclipse。

（2）服务器：

MySQL5.6或 SQL Server 数据库服务器。

3. 技术要求：

1） 掌握软件工程的需求分析和系统设计方法。

2) 掌握数据库的分析与设计，完成餐饮管理系统数据库的设计。

3) 掌握JAVA的GUI设计方法，完成登录、主界面、菜品分类管理界面、菜品管理界面等人机接口。

4） 掌握JAVA面向对象程序设计方法，设计系统所需的各种类。

5) 掌握JDBC方法，对数据库进行操作，完成主程序和菜品分类、菜品管理等功能。

### 实训工作量：

40课时

### 工作计划：

1.班级

21软件1-4

2.课时及教室安排

见实训课程工作计划表。

实训报告

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 学生姓名 | 易志望 | 学号 | B20210304117 | 班级 | 21软件01 |
| 专业 | 软件工程 | | 指导教师姓名 | 杨刚、潘怡 | |
| 项目名称 | 基于C/S架构的餐饮管理系统 | | | | |
| **实训过程**：（说明实训各阶段的工作过程，包括需求分析、系统设计、编码实现、系统测试、交付实施等阶段的主要工作。）  需求分析：  根据餐饮行业的特点和该企业的实际情况，本餐饮管理系统以餐饮业务为基础，突出管理，从专业角度出发，提供科学有效的管理模式。  能够针对中餐因其菜品的多样化和特色化的服务提供标准化的管理。  能够提供符合餐饮企业自身要求的较科学的标准化、流程化管理，解决餐饮行业专业人才欠缺的问题。  能够依据订餐、点菜、结账等环节的繁重化、复杂化问题，实现强化管理、降低成本、堵漏节流等效益。  能够针对企业的经营现状做出科学的分析，使得企业对市场的应变能力得到提高。  本餐饮管理系统采用Java语言进行开发，JDK采用1.7，开发工具使用Eclipse，数据库使用mysql8.0.  系统设计：  根据该企业的具体情况，系统主要功能设计有6大部分，分别为用户管理、员工管理、客户管理、餐台管理、菜品管理、菜品分类管理、业务管理、结账管理、支付管理。如图所示。  1.用户管理：对用户实现增删改查。  2.员工管理：对员工实现增删改查。  3.客户管理：对客户实现增删改查。  4.餐台管理：对餐台实现增删改查。  5.菜品管理：对菜品实现增删改查。  6.菜品分类管理：对菜品分类实现增删改查。  7.业务管理： 服务员对某客户、某一空闲餐台实行开台，同时实现点菜，将餐台号与所点的菜品对应起来，分别显示出来，并记录开台时间。  8.结账管理：显示未支付的订单。  9.支付管理；收银员对某一餐台通过统计显示消费的菜品清单并统计出消费金额，通过手动输入实收金额进行找零的计算，并显示，完成结账的操作，并记录成统计数据。  编码实现：利用Java的GUI设计人机交互界面；利用Java的JDBC所提供的方法对数据库进行操纵。  系统测试：测试系统的完备性、实用性和可靠性。  交付实施：经测试成功后对系统进行维护和运营。 | | | | | |
| **实训成果**：  实现了餐饮管理系统的基本功能，界面美观、人机交互友好 | | | | | |
| **实训总结**：（包括心得体会、存在的问题和改进方向。）  通过这两周的实训，我对Java的GUI编程和JDBC操作数据库有了更深刻的认识，也提高了我的编程能力和解决问题的能力。在实训过程中，我主要完成了以下几个方面的工作：  - 根据企业的具体情况，设计了一个餐饮管理系统的界面，包括用户管理、员工管理、客户管理、餐台管理、菜品管理、菜品分类管理、结账管理、支付管理等功能模块。  - 利用Java Swing组件和布局管理器，实现了各个功能模块的界面布局和交互效果，使得界面美观、简洁、易用。  - 利用JDBC技术，连接MySQL数据库，实现了各个功能模块的数据增删改查操作，保证了数据的完整性和一致性。  - 利用数据结构和算法，实现了业务管理模块的核心逻辑，包括开台、点菜、结账等功能，满足了餐饮业务的需求。  - 利用异常处理机制，对可能出现的错误进行了捕获和处理，提高了系统的稳定性和安全性。  在实训过程中，我也遇到了一些问题和困难，主要有以下几点：  - 在设计界面时，需要考虑到不同分辨率下的显示效果，以及不同操作系统下的兼容性问题，这需要对Swing组件的属性和方法有较深入的了解。  - 在操作数据库时，需要注意SQL语句的正确性和效率，以及事务的控制和回滚，这需要对数据库的基本知识和操作有较熟练的掌握。  - 在实现业务逻辑时，需要考虑到多种情况和异常，以及多线程下的并发问题，这需要对业务需求和编程思想有较清晰的理解。  改进措施：  - 在设计界面时，参考了网上的一些优秀案例和教程，学习了如何使用Swing组件和布局管理器来实现自适应和跨平台的界面效果。  - 在操作数据库时，利用了一些工具软件和插件，如Navicat和MyBatis等，来辅助编写和测试SQL语句，并优化了部分查询语句，提高了数据库的性能。  - 在实现业务逻辑时，多次测试和调试，发现并修复了一些bug，并使用了同步锁等机制来解决多线程下的并发问题。  总之，在这次实训中，我收获了很多知识和技能，也锻炼了我的动手能力和团队协作能力。我感谢老师和同学们给予我的指导和帮助。我相信这次实训对我的今后学习和工作都会有很大的帮助。 | | | | | |

基于C/S架构的餐饮管理系统

设计说明书

作者：B20210304117 易志望

计算机科学与工程学院

2023年 06 月 15日

**摘要**

随着我国市场经济的不断发展，国民生活水平的不断提高，进入酒店等高等消费场所的人数也与日俱增。传统的手工点菜方式由于其难计算、难查找、难更改、易出错、效率低等缺点已逐渐退出了酒店等高等消费场所的服务管理平台。层出不穷的各类餐厅点菜管理系统也应运而生，呈现出多元化的发展。为了更好的满足广大消费者的多元化消费需求和不同层次的消费水平，提高餐厅的服务管理质量，提高餐厅工作人员的工作效率，在多方面考察、分析、研究现有酒店点菜管理系统的基础之上，以提高消费者的满意程度及商家的服务水平和市场竞争力为目标，致力于开发出一套可视化程度高、功能全面、集分析管理于一体的点餐管理系统，极具有市场价值。

本系统包含客户管理、员工管理、菜品管理、餐台管理、点菜管理和结账管理六大基本模块。客户管理实现对客户信息的增删改查；员工管理实现对员工信息的增删改查；菜品管理实现对菜品信息的增删改查；餐台管理实现对餐台信息的增删改查；点菜管理实现服务员对客户的人数进行开台，并进行点菜和退菜，结账管理实现对订单的查询和修改功能。

**关键词：**java ，C/S架构，餐饮管理系统

目录

[已知技术参数和设计要求： 2](#_Toc28137)

[1. 问题描述（功能要求）： 2](#_Toc26491)

[2. 运行环境要求： 2](#_Toc24787)

[3. 技术要求： 2](#_Toc32080)

[实训工作量： 3](#_Toc7743)

[工作计划： 3](#_Toc26734)

[1 引言 1](#_Toc4699)

[1.1 编写目的 1](#_Toc3290)

[1.2 参考资料 1](#_Toc16251)

[2 需求规约 1](#_Toc24685)

[2.1 功能需求 1](#_Toc13700)

[2.2 界面需求 8](#_Toc10020)

[2.3 数据需求 8](#_Toc30087)

[3 系统设计 12](#_Toc32456)

[3.1 运行环境 13](#_Toc11063)

[3.2 系统静态结构设计 13](#_Toc7431)

[3.3 人机接口设计 14](#_Toc13097)

[3.3.6 用户接口5（客户管理界面）设计 18](#_Toc31550)

[3.4 登录模块设计说明 25](#_Toc23551)

[26](#_Toc17779)

[3.5 员工管理模块设计说明 26](#_Toc483)

[3.6 客户管理模块设计说明 29](#_Toc20059)

[3.7 餐台管理模块设计说明 31](#_Toc24332)

[3.8 菜品管理模块设计说明 33](#_Toc6241)

[3.9 业务管理模块设计说明 35](#_Toc280)

[3.10 支付管理模块设计说明 36](#_Toc17412)

[4 数据库设计 37](#_Toc4733)

[4.1 数据库环境说明 37](#_Toc30216)

[4.2 数据库的命名规则 37](#_Toc24687)

[4.3 逻辑结构设计 38](#_Toc7248)

[4.4 物理结构设计 40](#_Toc2271)

[4.4.1 数据库表 40](#_Toc28244)

[4.5 安全性设计 45](#_Toc22909)

[4.5.1 防止用户直接操作数据库的方法 45](#_Toc11929)

[4.5.2 用户帐号密码的加密方法 45](#_Toc28656)

[5 测试用例设计 45](#_Toc7316)

[附录 49](#_Toc27806)

[附录1 程序运行结果 49](#_Toc17107)

[附录2 程序源代码 57](#_Toc21673)

# 1 引言

## 1.1 编写目的

本文档是“餐饮管理系统”的软件设计说明书。编写目的是：明确软件需求、概要设计、详细设计和功能测试用例，为软件后续开发和维护提供指导。本文档的读者对象是需求分析人员、系统设计人员、系统开发人员、系统测试人员。

## 1.2 参考资料

[1] 徐转运, 张扬, 王淼. Java高级程序设计[M]. 北京:清华大学出版社, 2014.

[2] 史济民，顾春华，李昌武（著）；《软件工程——原理、方法与应用》；第

二版；高等教育出版社；2008：276 ~ 293

# 2 需求规约

## 2.1 功能需求

根据该企业的具体情况，系统主要功能设计有9大部分，分别为用户管理、员工管理、客户管理、餐台管理、菜品管理、菜品分类管理、业务管理、结账管理、支付管理。如图所示。

1.用户管理：对用户实现增删改查。

2.员工管理：对员工实现增删改查。

3.客户管理：对客户实现增删改查。

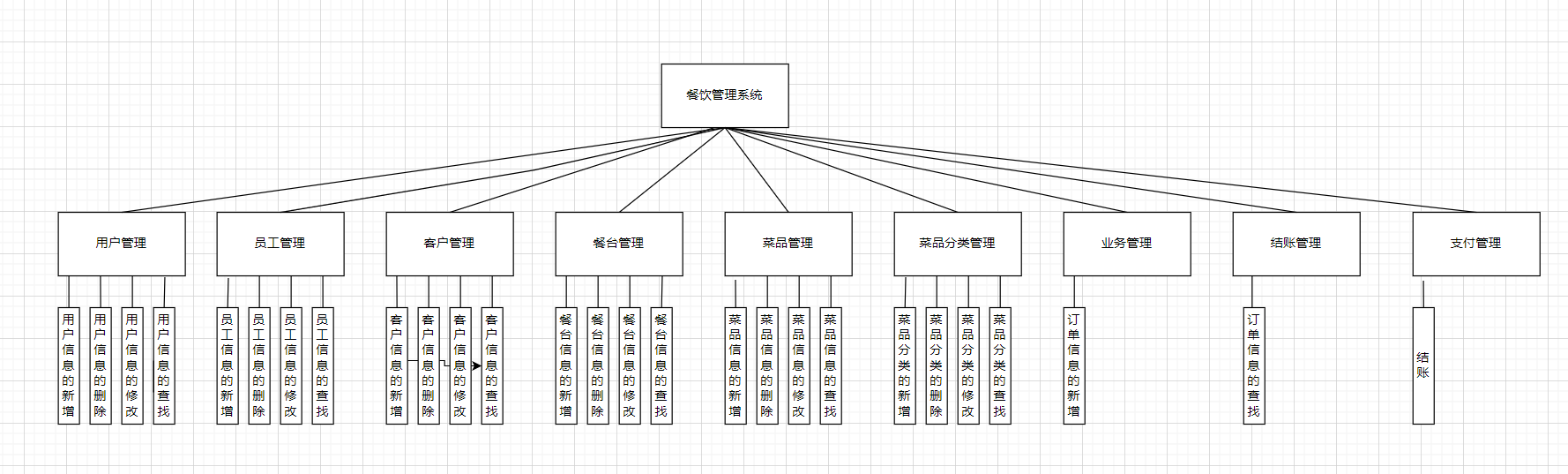
4.餐台管理：对餐台实现增删改查并实现点菜。

5.菜品管理：对菜品实现增删改查。

6.菜品分类管理：对菜品分类实现增删改查。

7.结账管理：显示未支付的订单。

8.支付管理；收银员对某一餐台通过统计显示消费的菜品清单并统计出消费金额，通过手动输入实收金额进行找零的计算，并显示，完成结账的操作，并记录成统计数据。



2.1.1 用户登录用例说明

用例名称：用户登录。

用例描述：餐厅管理员登录餐厅管理系统。

前置条件：管理员输入账号和密码。

后置条件：管理员登录成功。

活动步骤：

1. 管理员输入账号和密码
2. 点击登录按钮，管理员登录成功。

异常处理：

1. 用户账号或密码为空，用例终止。
2. 用户输入账号不存在或密码不正确，用例终止。
3. 用户输入账号或密码格式不正确，用例终止。

2.1.2 用户管理用例说明

用例名称：用户管理。

用例描述：餐厅管理员管理账号。

前置条件：用户输入账号、密码。

后置条件：用户添加，删除，查询成功。

活动步骤：

1. 管理员输入账号和密码。
2. 点击添加，删除，查询按钮，

异常处理：

1. 管理员输入账号或密码为空，用例终止。
2. 管理员输入账号或密码格式不对，用例终止。

2.2.3 员工管理用例说明

用例名称：员工管理。

用例描述：餐厅管理员对员工信息进行增删改查。

前置条件：餐厅管理员对员工信息进行筛选。

后置条件：餐厅管理员对员工信息进行一系列操作。

活动步骤：

查询操作：

1. 餐厅管理员在文本框内输入员工信息，包括员工id，员工姓名，若未输入员工信息，系统将查询到所有员工。
2. 餐厅管理员查询的信息将在表格中显示。

新增操作：

1. 餐厅管理员在文本框输入员工信息
2. 餐厅管理员点击添加按钮，完成添加
3. 员工信息添加成功后立即更新查询后的表格。

修改操作：

1. 餐厅管理员输入员工信息
2. 餐厅管理员点击修改按钮，完成修改
3. 员工信息修改成功后立即更新查询后的表格。

删除操作：

1. 餐厅管理员在文本框中输入该员工的id或点击该信息行。
2. 餐厅管理员点击删除按钮。
3. 员工信息删除后立即更新查询后表格。

异常处理：

新增操作：若输入员工信息有误，系统将提示具体哪一部分出错。

修改操作：若输入要修改的信息有误，系统将提示修改错误，将不会对表格信息进行更新。

2.2.4 客户管理用例

用例名称：客户管理。

用例描述：餐厅管理员对客户信息进行增删改查。

前置条件：餐厅管理员对客户信息进行筛选。

后置条件：餐厅管理员对客户信息进行一系列操作。

活动步骤：

查询操作：

1. 餐厅管理员在文本框内输入客户信息，包括客户id，客户姓名，若未输入客户信息，系统将查询到所有客户。
2. 餐厅管理员查询的信息将在表格中显示

新增操作：

1. 餐厅管理员在文本框输入客户信息
2. 餐厅管理员点击添加按钮，完成添加
3. 客户信息添加成功后立即更新查询后的表格

修改操作：

1. 餐厅管理员输入客户信息
2. 餐厅管理员点击修改按钮，完成修改
3. 客户信息修改成功后立即更新查询后的表格。

删除操作：

1. 餐厅管理员在文本框中输入该客户的id或点击该信息行。
2. 餐厅管理员点击删除按钮。
3. 客户信息删除后立即更新查询后表格。

异常处理：

新增操作：若输入客户信息有误，系统将提示具体哪一部分出错。

修改操作：若输入要修改的信息有误，系统将提示修改错误，将不会对表格信息进行更新。

2.2.5 菜品管理用例

用例名称：菜品管理。

用例描述：餐厅管理员对菜品信息进行增删改查。

前置条件：餐厅管理员对菜品信息进行筛选。

后置条件：餐厅管理员对菜品信息进行一系列操作。

活动步骤：

查询操作：

1. 餐厅管理员在文本框内输入菜品信息，包括菜品id，菜名，若未输入菜品信息，系统将查询到所有菜品。
2. 餐厅管理员查询的信息将在表格中显示

新增操作：

1. 餐厅管理员在文本框输入菜品信息

2. 餐厅管理员点击添加按钮，完成添加

3. 菜品信息添加成功后立即更新查询后的表格

修改操作：

1. 餐厅管理员输入菜品信息
2. 餐厅管理员点击修改按钮，完成修改

3. 菜品信息修改成功后立即更新查询后的表格。

删除操作：

1. 餐厅管理员在文本框中输入该菜品的id或点击该信息行。
2. 餐厅管理员点击删除按钮。
3. 菜品信息删除后立即更新查询后表格。

异常处理：

新增操作：若输入菜品信息有误，系统将提示具体哪一部分出错。

修改操作：若输入要修改的信息有误，系统将提示修改错误，将不会对表格信息进行更新。

2.2.6 菜品分类管理用例

用例名称：菜品分类管理。

用例描述：餐厅管理员对菜品分类信息进行增删改查。

前置条件：餐厅管理员对菜品分类信息进行筛选。

后置条件：餐厅管理员对菜品分类信息进行一系列操作。

活动步骤：

查询操作：

1. 餐厅管理员在文本框内输入菜品分类信息，包括菜品id，分类名，若未输入菜品信息，系统将查询到所有菜品。
2. 餐厅管理员查询的信息将在表格中显示

新增操作：

1. 餐厅管理员在文本框输入菜品分类信息

2. 餐厅管理员点击添加按钮，完成添加

3. 菜品分类信息添加成功后立即更新查询后的表格。

修改操作：

1. 餐厅管理员输入菜品分类信息
2. 餐厅管理员点击修改按钮，完成修改

3. 菜品分类信息修改成功后立即更新查询后的表格。

删除操作：

1. 餐厅管理员在文本框中输入该菜品分类的id或点击该信息行。
2. 餐厅管理员点击删除按钮。
3. 菜品分类信息删除后立即更新查询后表格。

异常处理：

新增操作：若输入菜品信息有误，系统将提示具体哪一部分出错。

修改操作：若输入要修改的信息有误，系统将提示修改错误，将不会对表格信息进行更新。

2.2.7 餐台管理用例

用例名称：餐台管理。

用例描述：餐厅管理员对餐台信息进行增删改查。

前置条件：餐厅管理员对餐台信息进行筛选。

后置条件：餐厅管理员对餐台信息进行一系列操作。

活动步骤：

查询操作：

1. 餐厅管理员在文本框内输入餐台号，若未输入餐台号，系统将查询到所有餐台。
2. 餐厅管理员查询的信息将在表格中显示

新增操作：

1. 餐厅管理员在文本框输入餐台信息

2. 餐厅管理员点击添加按钮，完成添加

3. 餐台信息添加成功后立即更新查询后的表格。

修改操作：

1. 餐厅管理员输入餐台信息
2. 餐厅管理员点击修改按钮，完成修改

3. 餐台信息修改成功后立即更新查询后的表格。

删除操作：

1. 餐厅管理员在文本框中输入该餐台的id或点击该信息行。
2. 餐厅管理员点击删除按钮。
3. 餐台信息删除后立即更新查询后表格。

异常处理：

修改操作：若输入要修改的信息有误，系统将提示修改错误，将不会对表格信息进行更新。

2.2.8 点菜管理用例

用例名称：点菜管理。

用例描述：餐厅管理员对客户进行开台，点菜，下单操作。

前置条件；餐厅管理员输入客户的客户名和人数，对客户进行开台操作。

后置条件：餐厅管理员为客户完成点单，下单操作。

活动步骤：

1. 餐厅管理员输入客户名以及人数，对客户进行开台操作。
2. 系统将为该用户分配空闲的餐台。
3. 餐厅管理员点击选择菜品，进入新界面，新界面包含菜品信息，提供给客户去选菜。
4. 选菜完成后点击加入后再点击下单，完成下单，生成订单。

异常处理：

1. 餐厅管理员输入的人数大于现有空闲餐台的所能容纳的最大人数时，系统将提示找不到合适的餐台。

2.2.9 结账管理用例

用例名称：结账管理。

用例描述：餐厅管理员对订单进行查询的操作。

前置条件：餐厅管理员对订单进行查询。

后置条件：餐厅管理员查看未支付订单。

活动步骤：

1. 餐厅管理员点击刷新，对订单进行查询。
2. 在文本框中显示未支付订单。

2.2.10 支付管理用例

用例名称：支付管理。

用例描述：餐厅管理员对订单进行查询，删除，结账的操作。

前置条件：餐厅管理员输入餐桌号和客户名，对订单进行查询。

后置条件：餐厅管理员对所选订单进行结账处理。

活动步骤：

1. 餐厅管理员输入餐桌号和客户名，对订单进行查询。
2. 在文本框中显示订单的应付金额与折后价格以及实付金额。
3. 餐厅管理员在实付金额处填写客户支付的金额，点击支付，系统将对该客户进行找零操作，同时更新查询后的表格。

异常处理：餐厅管理员输入金额小于折后价格，将提醒支付失败。

## 2.2 界面需求

1. 界面友好，操作简单。

2. 界面美观，用户体验感良好。

## 2.3 数据需求

表2.1 用户实体user

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 列名 | 类型 | 长度 | 是否为空 | 唯一标识 | 说明 |
| id | int | 10 | no | yes | 序号 |
| username | varchar | 20 | no | no | 用户名 |
| password | varchar | 20 | no | no | 密码 |

表2.2 员工实体employees

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 列名 | 类型 | 长度 | 是否为空 | 唯一标识 | 说明 |
| id | int | 10 | no | yes | 序号 |
| name | varchar | 20 | no | no | 用户名 |
| sex | varchar | 2 | yes | no | 性别 |
| birthday | datatime | 8 | yes | no | 出生日期 |
| identityID | varchar | 18 | yes | no | 身份证号 |
| address | varchar | 40 | yes | no | 家庭住址 |
| tel | varchar | 11 | yes | no | 电话 |
| position | varchar | 4 | no | no | 职位 |
| freeze | varchar | 4 | no | no | 是否在职 |

表2.3 客户实体customer

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 列名 | 类型 | 长度 | 是否为空 | 唯一标识 | 说明 |
| id | int | 10 | no | yes | 序号 |
| name | varchar | 20 | no | no | 用户名 |
| sex | varchar | 4 | yes | no | 性别 |
| company | varchar | 20 | yes | no | 单位 |
| tel | varchar | 11 | yes | no | 电话 |
| cardID | varchar | 10 | no | no | 贵宾卡号 |

表2.4 菜品分类实体category

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 列名 | 类型 | 长度 | 是否为空 | 唯一标识 | 说明 |
| id | int | 10 | no | yes | 序号 |
| name | varchar | 20 | no | no | 名称 |
| describ | varchar | 20 | yes | no | 描述 |

表2.5 菜品信息实体dish

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 列名 | 类型 | 长度 | 是否为空 | 唯一标识 | 说明 |
| id | int | 10 | no | yes | 序号 |
| name | varchar | 20 | no | no | 菜品名 |
| categoryId | int | 10 | no | no | 类别编号 |
| pic | Blob | 50 | no | no | 图片 |
| code | varchar | 8 | no | no | 菜品代码 |
| unit | varchar | 4 | yes | no | 单位 |
| price | double | 6 | yes | no | 价格 |
| status | varchar | 4 | yes | no | 状态 |

表2.6 餐台实体desk

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 列名 | 类型 | 长度 | 是否为空 | 唯一标识 | 说明 |
| id | int | 10 | no | yes | 序号 |
| no | varchar | 8 | no | yes | 餐台编号 |
| seating | int | 4 | no | no | 座位数 |
| status | varchar | 10 | no | no | 状态 |

表2.7 订单实体order

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 列名 | 类型 | 长度 | 是否为空 | 唯一标识 | 说明 |
| id | int | 10 | no | yes | 序号 |
| orderNo | varchar | 20 | no | yes | 订单编号 |
| deskId | int | 10 | no | no | 餐台号 |
| createtime | date | 40 | no | no | 就餐日期时间 |
| money | double | 6 | no | no | 金额 |
| customerId | int | 10 | no | no | 客户编号 |
| status | varchar | 4 | no | no | 状态 |
| number | int | 4 | no | no | 就餐人数 |

表2.8 订单明细实体orderitem

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 列名 | 类型 | 长度 | 是否为空 | 唯一标识 | 说明 |
| id | int | 10 | no | yes | 序号 |
| orderId | int | 10 | no | no | 订单编号 |
| dishId | int | 10 | no | no | 菜品序号 |
| amount | double | 4 | no | no | 菜品数量 |

# 3 系统设计

餐饮管理系统的总体设计思路是建立一个集用户管理、员工管理、客户管理、餐台管理、菜品管理、菜品分类管理、业务管理、结账管理、支付管理功能于一体的系统，以提高餐厅的运营效率和管理水平。

下面是每个功能实现的思路：

用户管理：系统应能够管理餐厅的用户信息，包括用户名和密码。系统可以通过系统添加、修改和删除用户信息。在实现上可以设计一个用户管理模块，提供用户信息录入和查询功能，并将用户信息存储在数据库中。

员工管理：系统应能够管理餐厅的员工信息，包括用户名，性别，出生日期，身份证号，家庭住址，电话，职位，是否在职。管理员可以通过系统添加、修改和删除员工信息。在实现上，可以设计一个员工管理模块，提供员工信息录入和查询功能，并将员工信息存储在数据库中。

客户管理：系统应能够管理餐厅的客户信息，包括用户名，性别，单位，电话，贵宾卡好。管理员可以通过系统添加、修改和删除客户信息。在实现上，可以设计一个客户管理模块，提供客户信息录入和查询功能，并将客户信息存储在数据库中。

餐台管理：系统应能够管理餐厅的餐台信息，包括餐台编号，座位数，状态(空闲、使用中)。管理员可以通过系统添加、修改和删除餐台信息。在实现上，可以设计一个餐台管理模块，提供餐台信息录入和查询功能，并将餐台信息存储在数据库中。

菜品管理：系统应能够管理餐厅的菜品信息，包括菜品名，类别编号，图片，菜品代码，单位，价格，状态(可用、不可用)。管理员可以通过系统添加、修改和删除菜品信息。在实现上，可以设计一个菜品管理模块，提供菜品信息录入和查询功能，并将菜品信息存储在数据库中。

菜品分类管理：系统应能够管理餐厅的菜品分类信息，包括名称，描述。管理员可以通过系统添加、修改和删除菜品分类信息。在实现上，可以设计一个菜品分类管理模块，提供菜品分类信息录入和查询功能，并将菜品分类信息存储在数据库中。

点菜管理：系统应具备接收、处理和跟踪订单的能力。用户可以通过系统下单，同时也可以通过系统查看订单状态和历史记录。在实现上，可以采用前后端分离的架构，前端提供用户订单输入界面和查询界面，后端负责处理订单逻辑和与数据库的交互。

结账管理：系统应具备接收、处理和跟踪订单的能力。用户可以通过结账管理找到未结账的订单。在实现上，可以设计一个结账管理模块，提供订单查询功能，通过数据库查询未支付的订单。

支付管理：系统应具备接收、处理和跟踪订单的能力。用户可以通过系统查询应收金额，通过输入实收金额后，计算找零。在实现上，可以通过数据库查询应收金额，并比较与实收金额的大小，通过计算给出找零。

总体而言，餐饮管理系统的设计思路是通过集成各个功能模块，实现订单、餐台、员工、客户、菜品、找零的全面管理，从而提高餐厅的运营效率和管理水平。

## 3.1 运行环境

操作系统：Windows 11

数据库：MySQL 8.0.33

## 3.2 系统静态结构设计

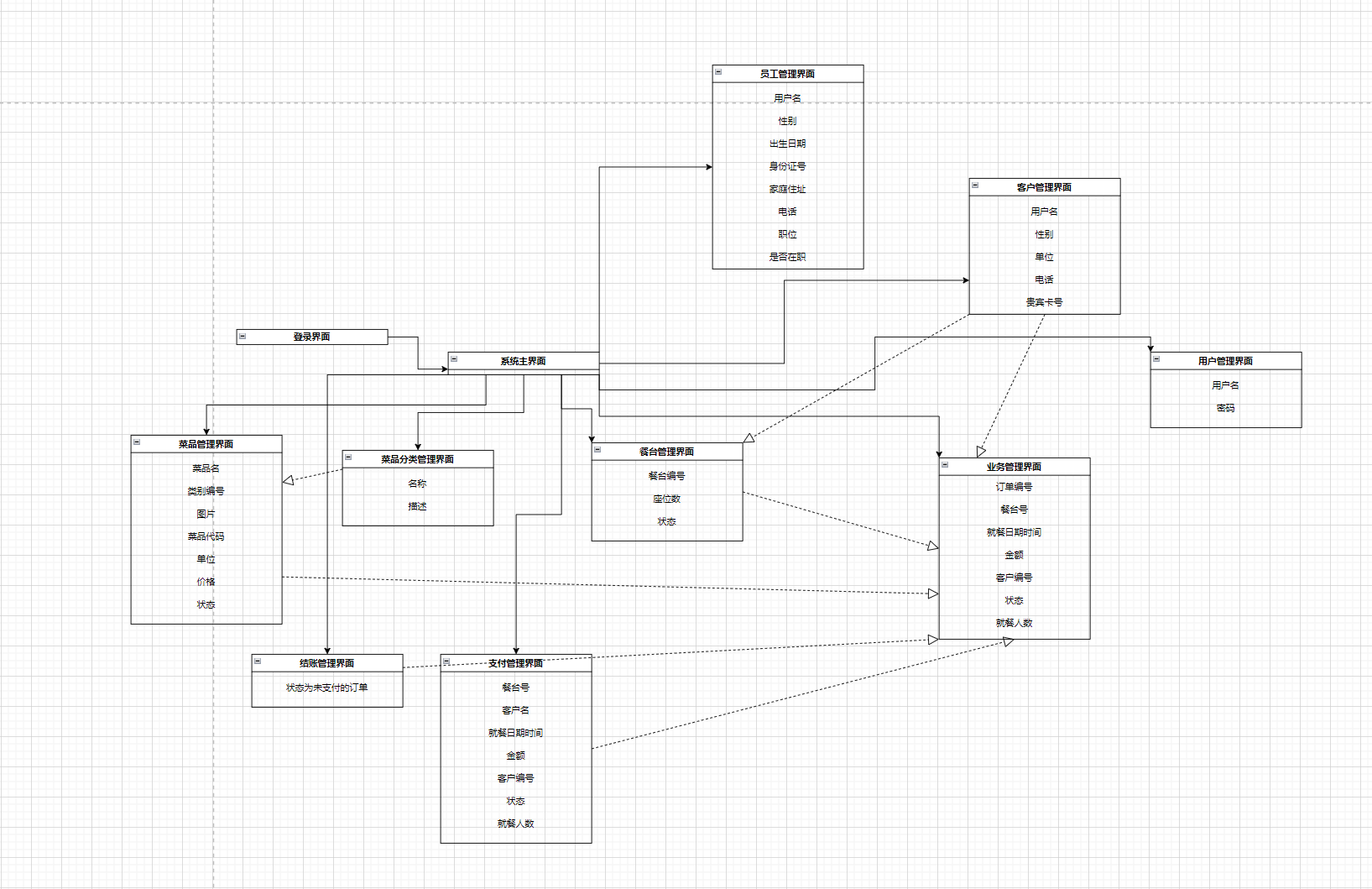


图3.1 餐饮管理系统总体类图

系统静态结构说明

餐厅管理员通过主界面可以跳转至不同的界面，在员工管理界面，管理员可以实现对员工信息的增删改查，在客户管理界面，管理员可以实现对客户信息的增删改查，在菜品管理界面，管理员可以实现对菜品信息的增删改查，在菜品分类管理界面，管理员可以实现对菜品分类信息的增删改查，在餐台界面，管理员可以实现对餐台信息的增删改查，在业务管理界面，管理员可以为顾客提供点菜服务，点菜完成后生成新的订单，在支付管理界面，管理员可以对订单进行查询和结账操作。

## 3.3 人机接口设计

3.3.1 设计应用类

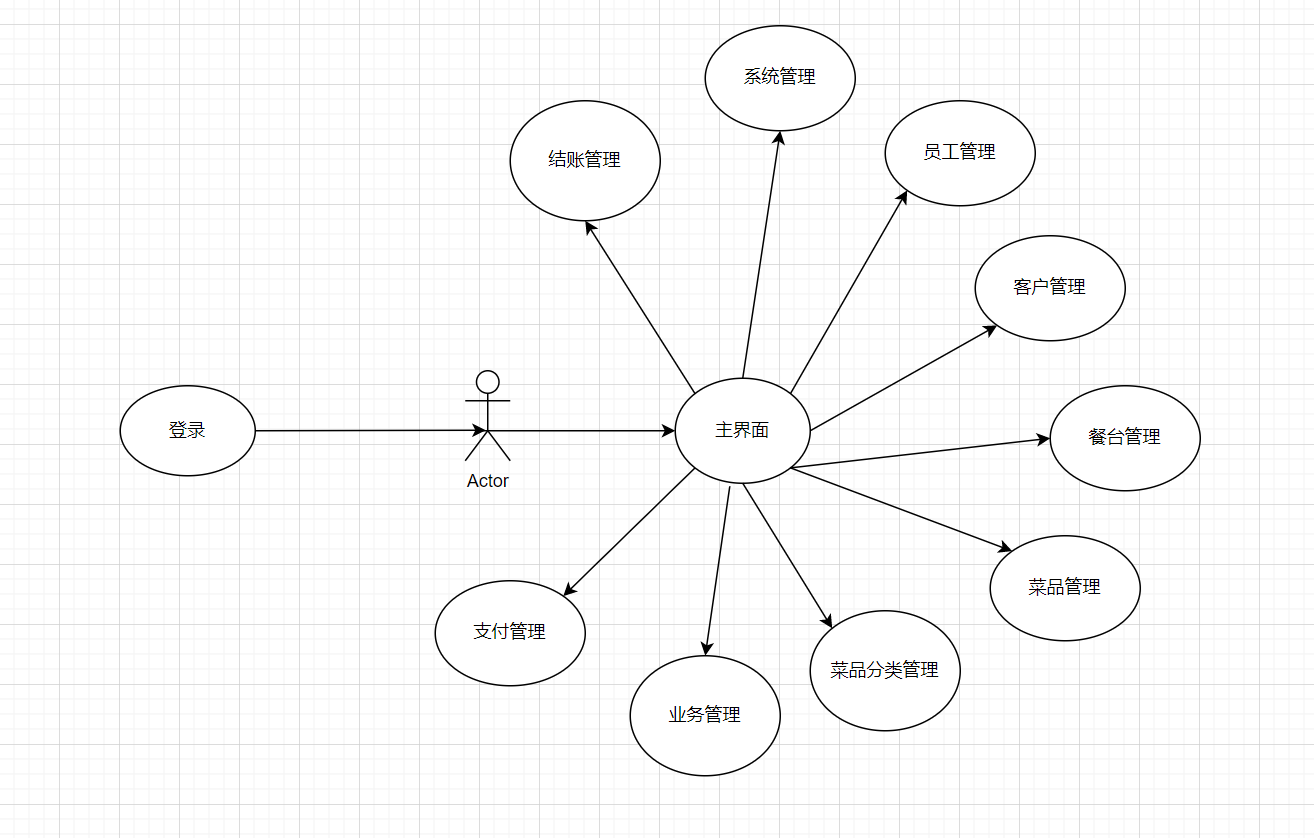


图3.3 餐饮管理系统设计类图

3.3.2 用户接口1（登录界面）设计

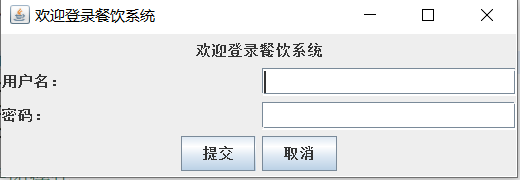
**界面主要控件：**

JTextField，输入框，接收账号和密码。

JButton，按钮，触发登录注册事件。

**实现功能：**

1. 餐厅管理员输入账号密码，点击登录按钮跳转餐厅管理界面；账号设置为10位数字，密码长度为8到16位的数字或字母，字母区分大小写。



3.3.3 用户接口2（餐饮管理系统主界面）设计



**界面主要控件：**

JButton，按钮，触发点击事件。

**实现功能：**

1. 点击员工管理，跳转至员工管理界面。
2. 点击客户管理，跳转至客户管理界面。
3. 点击餐台管理，跳转至餐台管理界面。
4. 点击菜品管理，跳转至菜品管理界面。
5. 点击菜品分类管理，跳转至菜品分类管理界面。
6. 点击点菜管理功能，跳转点菜管理界面
7. 点击结账管理，跳转至结账管理界面。
8. 点击结账管理，跳转至支付管理界面。
9. 点击系统管理，跳转至系统管理界面。

3.3.4 用户接口3（系统管理界面）设计

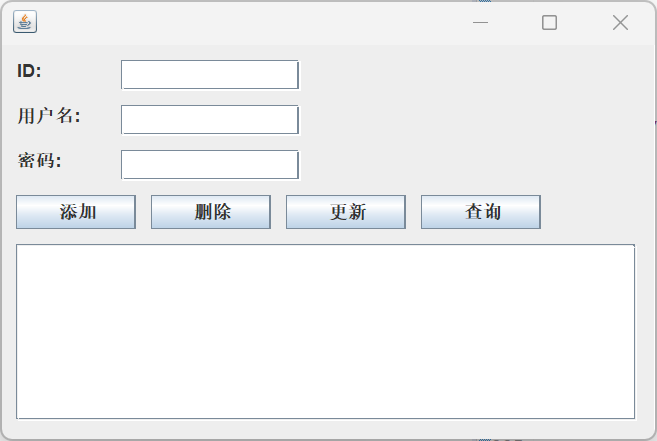
**界面主要控件：**

JButton，按钮，触发点击事件。

JTextField，输入框，接收用户id，用户名和密码。

**实现功能：**

1. 点击添加按钮，将进行用户的增添。
2. 通过输入信息，点击删除按钮，将用户信息删除。
3. 通过输入ID，点击查询按钮，将查询用户信息。



3.3.5 用户接口4（员工管理界面）设计

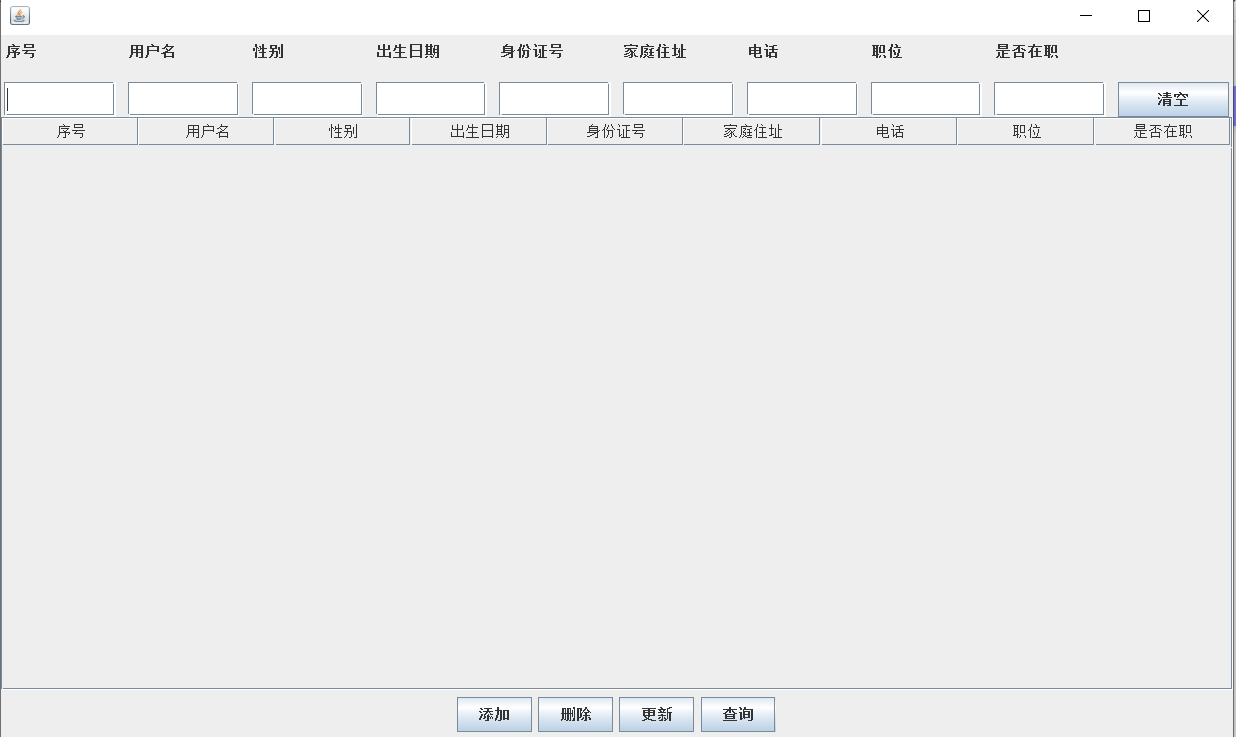
**界面主要控件：**

JButton，按钮，触发点击事件。

JTextField，输入框，接收员工姓名，性别，生日，身份证号，地址，电话，职位，是否在职。

**实现功能：**

1. 点击添加按钮，将对进行员工的增添。
2. 点击表格中的某个员工后再点击删除员工，将对该员工进行删除。
3. 通过输入信息，点击保存按钮，将对员工信息进行修改。



## 3.3.6 用户接口5（客户管理界面）设计

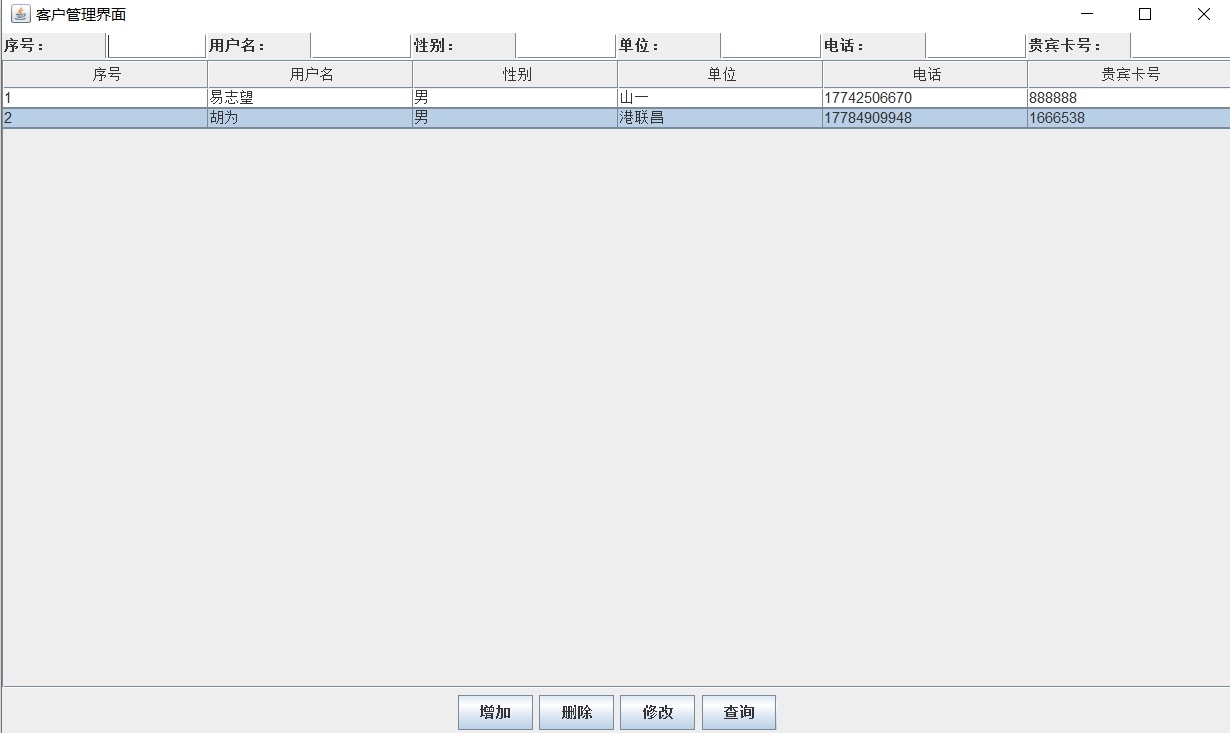
**界面主要控件：**

JButton，按钮，触发点击事件。

JTextField，输入框，接收客户id，用户名，性别，单位，电话，贵宾卡号。

**实现功能：**

1. 点击查询按钮，将对客户进行查询。
2. 点击添加按钮，将对客户进行增添。
3. 点击表格中的某个客户后再点击删除客户，将对该客户进行删除。
4. 点击表格，将对客户信息进行修改。



3.3.7 用户接口6（餐台管理界面）设计

**界面主要控件：**

JButton，按钮，触发点击事件。

JTextField，输入框，接收餐台号，编号，座位数，状态。

**实现功能：**

1. 点击添加按钮，将对进行餐台的增添。
2. 点击表格中的某个餐台后再点击删除餐台，将对该餐台进行删除。
3. 点击表格，将对餐台信息进行修改。



3.3.8 用户接口7（菜品管理界面）设计

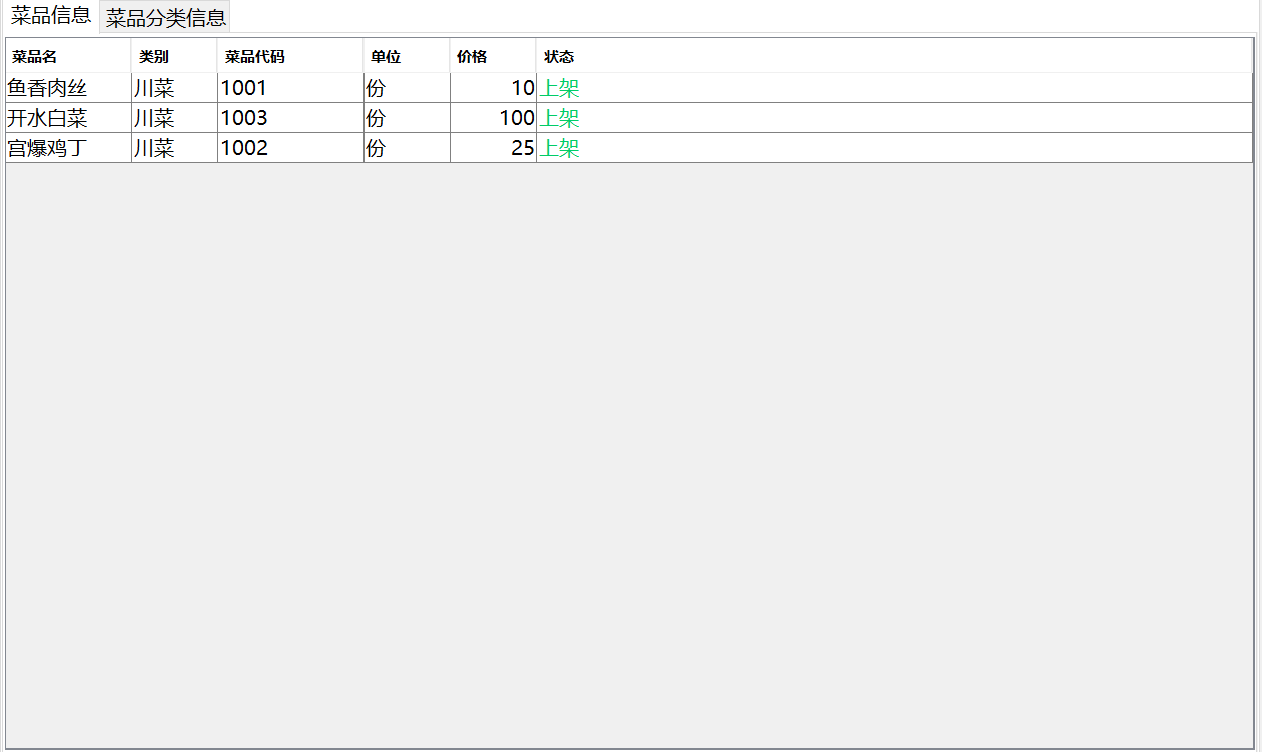
**界面主要控件：**

JButton，按钮，触发点击事件。

JTextField，输入框，接收菜品名称，分类，图片，编码，单位，价格，状态。

**实现功能：**

1. 点击保存按钮，将对进行菜品的增添。
2. 点击表格中的某个菜品后再点击删除菜品，将对该菜品进行删除。
3. 点击表格，将对菜品信息进行修改。

****

3.3.9 用户接口8（菜品分类管理界面）设计

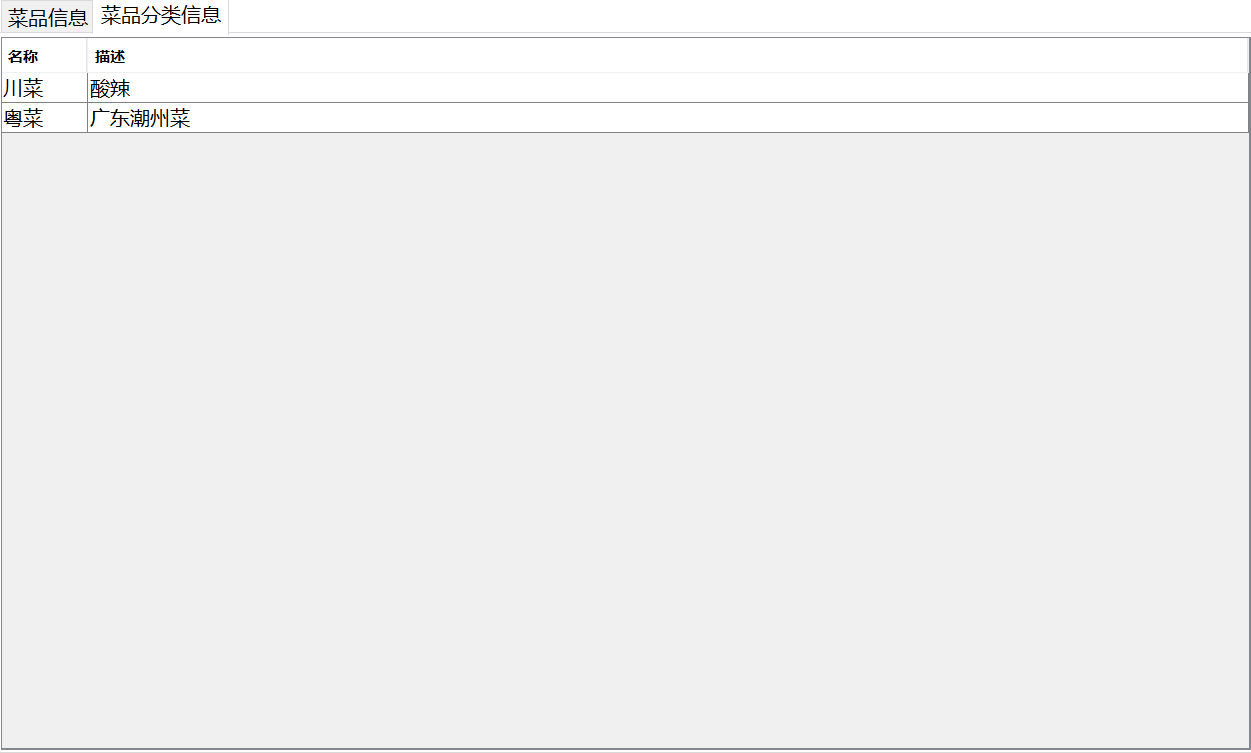
**界面主要控件：**

JButton，按钮，触发点击事件。

JTextField，输入框，接收菜品名称，描述。

**实现功能：**

1. 点击添加按钮，将对进行菜品的增添。
2. 点击表格中的某个菜品分类后再点击删除菜品，将对该菜品分类进行删除。
3. 点击表格，将对菜品分类信息进行修改。

****

3.3.10 用户接口9（点菜管理界面）设计

**界面主要控件：**

界面1：

JButton，按钮，触发点击事件。

JTextField，输入框，接收客户名，餐桌号和就餐人数。

界面2：

JButton，按钮，触发点击事件。

JTextField，输入框，接收菜名。

JComboBox，对菜品信息进行筛选。

**实现功能：**

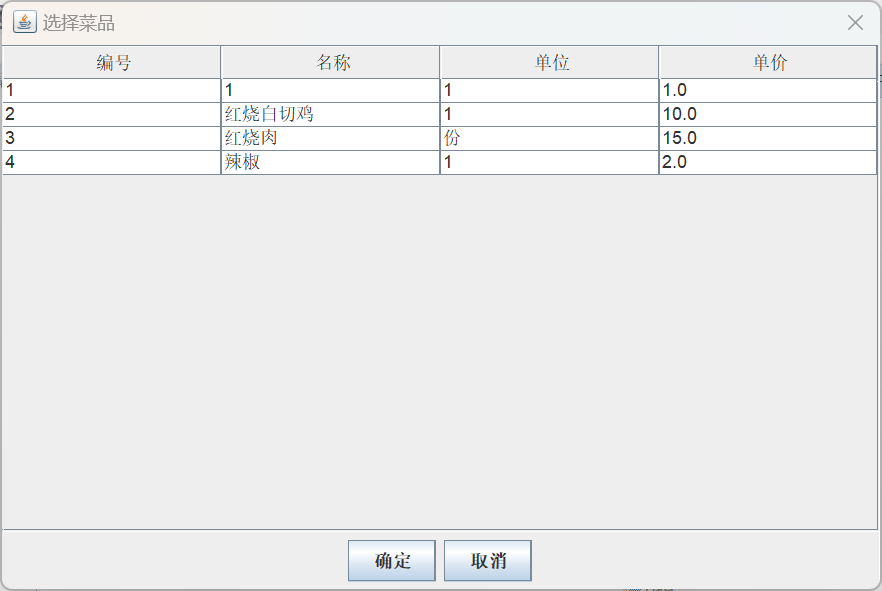
界面1：

1. 选择餐桌号，客户编号和就餐人数，点击选择菜品。
2. 点击选择菜品跳转至界面2

界面2：

1. 选择菜品，点击确认，菜品将添加到订单中

****

****

3.3.11 用户接口10（支付管理界面）设计

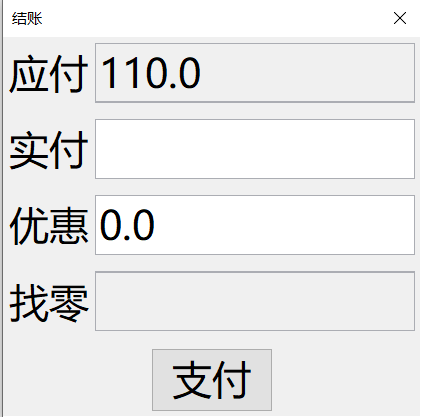
**界面主要控件：**

JButton，按钮，触发点击事件。

JTextField，输入框，接收客户名，餐桌号，应收金额，实收金额。

**实现功能：**

1. 选择餐桌号，客户名，对应到相应订单。
2. 输入实收金额，输出找零金额。
3. 点击确认，将对订单进行支付操作。



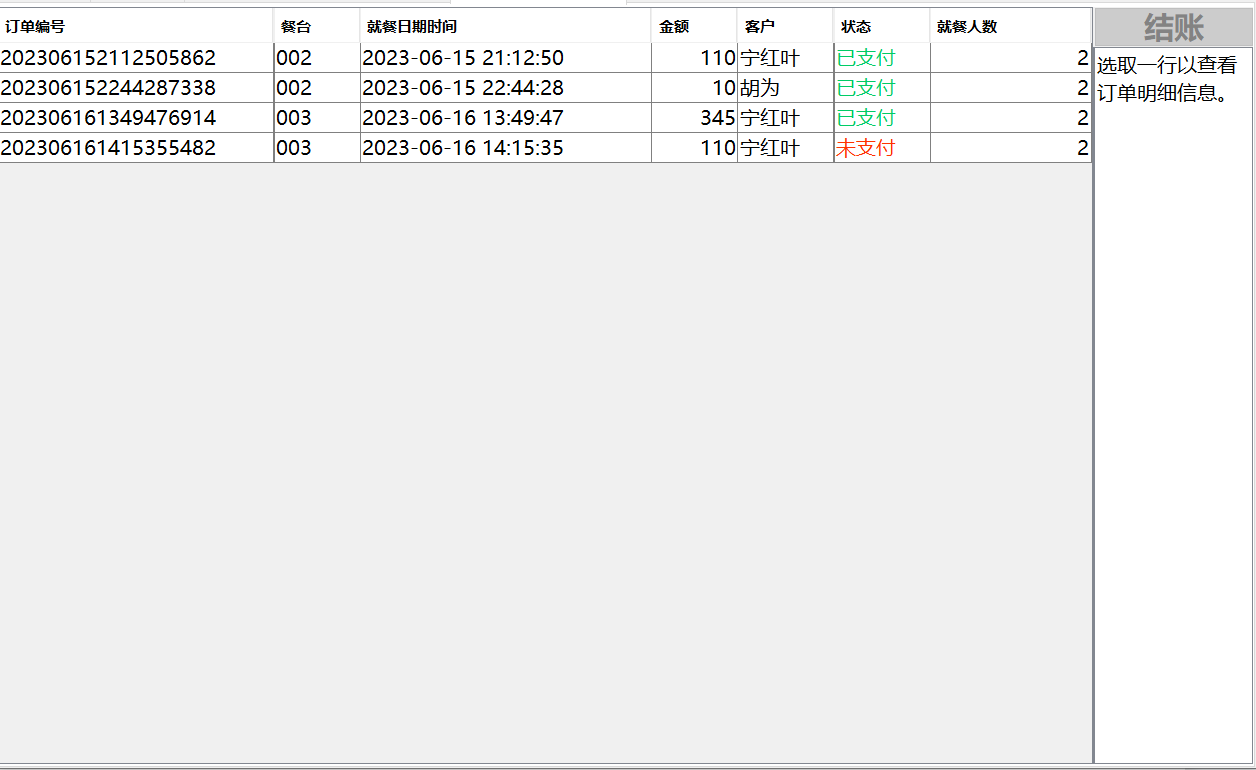
3.3.12 用户接口11（结账管理界面）设计

**界面主要控件：**

JButton，按钮，触发点击事件。

**实现功能：**

1. 点击刷新，显示未支付状态的订单。



## 3.4 登录模块设计说明

（1）功能：该模块主要是管理员对餐饮管理系统进行登录操作。

（2）测试要点：

1. 输入的账号或密码为空将提示账号或密码为空。

2. 输入的账号密码错误则提示账号密码有误。

（3）程序流程图如图所示：

## wps

登录模块程序流程图

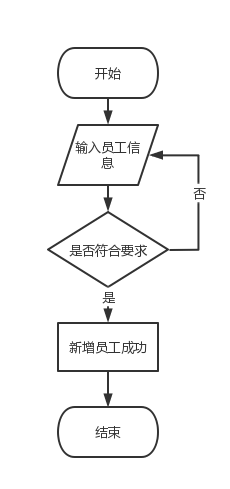
## 3.5 员工管理模块设计说明

（1）功能：实现管理员对员工的增删改查。

1. 新增员工，程序流程图如图所示：

测试要点：

1. 输入员工的信息，若员工信息无误且完整，点击添加，则提示新增成功。
2. 若信息不完整，且信息有误，系统会给出提醒。



新增员工信息流程图

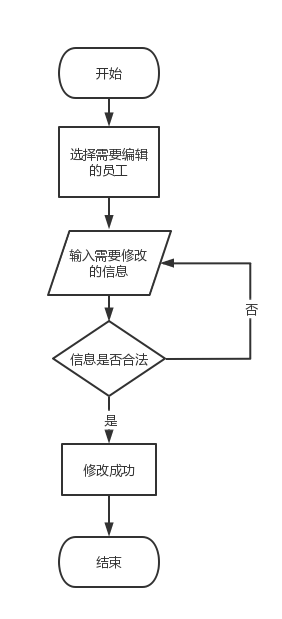
1. 删除员工，程序流程图如图所示：



删除员工流程图

测试要点：

1. 点击表格中员工的信息，点击删除，系统提示删除成功。
2. 修改员工，程序流程图如图所示：



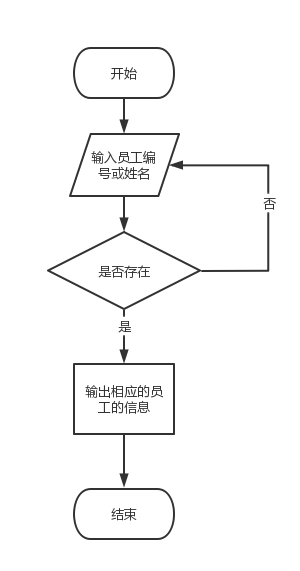
修改员工信息流程图

测试要点：

1. 在文本框中填写要修改的信息，若输入的信息合法则提示修改成功
2. 若输入的信息不合法，则提示信息不合法。
3. 查询员工，程序流程图如图所示：

测试要点：

1. 在文本框中输入员工id，点击查询按钮。
2. 查询信息将在表格中显示。

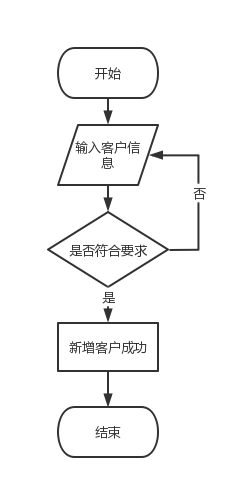


查询员工信息流程图

## 3.6 客户管理模块设计说明

（1）功能：实现管理员对客户的增删改查。

1. 新增客户，程序流程图如图所示：



新增客户流程图

测试要点：

1. 输入客户的信息，若客户信息无误且完整，点击添加，则提示新增成功。
2. 若信息不完整，且信息有误，系统会给出提醒。
3. 删除客户，程序流程图如图3.19所示：



图3.19删除客户流程图

测试要点：

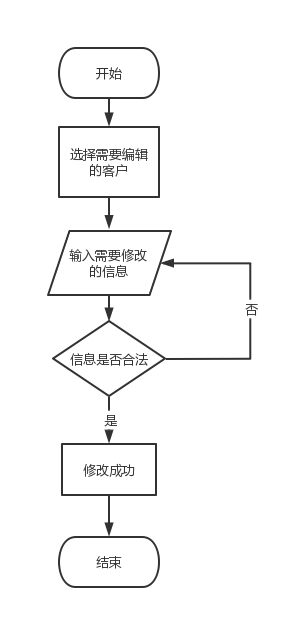
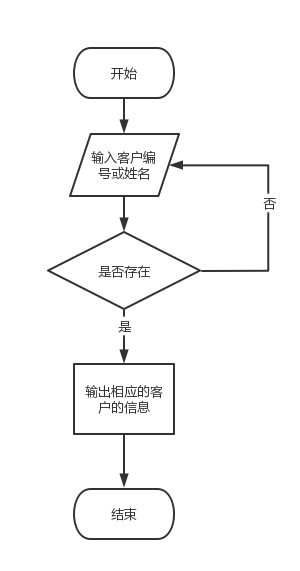
1. 点击表格中客户的信息，点击删除，系统提示删除成功。
2. 修改客户，程序流程图如图所示：

测试要点：

1. 在文本框中填写要修改的信息，点击保存，若输入的信息合法则提示修改成功。
2. 若输入的信息不合法，则提示信息不合法。
3. 查询客户，程序流程图如图所示：

测试要点：

1. 在文本框中输入客户的姓名和id，点击查询按钮。
2. 查询信息将在表格中显示。

修改客户流程图 查询客户流程图

## 3.7 餐台管理模块设计说明

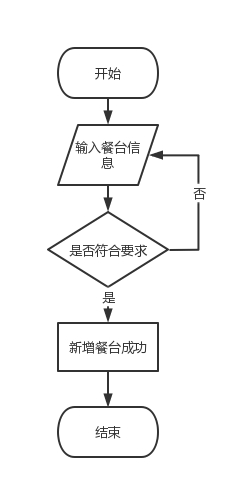


图3.22新增餐台流程图

测试要点：

1. 输入餐台的座位数，若输入为数字且合法，点击添加，则新增餐台成功，否则提示新增失败。
2. 删除餐台，程序流程图如图3.23所示：



图3.23删除餐台流程图

测试要点：

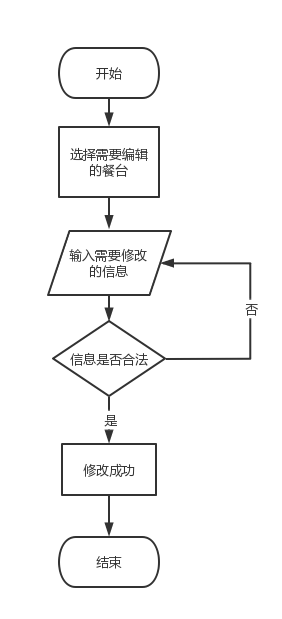
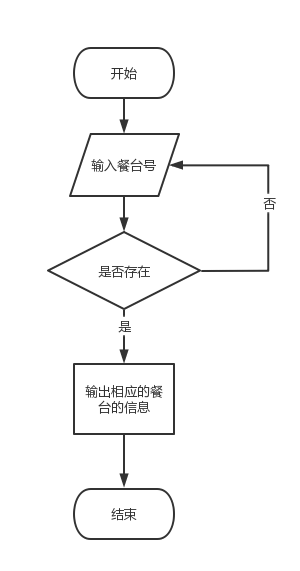
1. 点击餐台信息，点击删除按钮，系统提示删除成功
2. 修改餐台，程序流程图如所示：

测试要点：

1. 在文本框中填写要修改的信息，点击保存，若输入的信息合法则提示修改成功。
2. 若输入的信息不合法，则提示信息不合法。
3. 查询餐台，程序流程图如图所示：

测试要点：

1. 在文本框中输入餐台号，点击查询按钮。
2. 查询信息将在表格中显示。

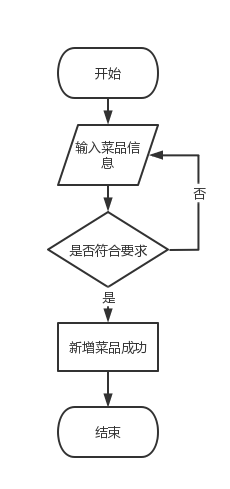
 

修改餐台流程图 查询餐台流程图

## 3.8 菜品管理模块设计说明

（1）功能：实现管理员对菜品信息的增删改查

1. 新增菜品，程序流程图如图所示：



新增菜品程序流程图

测试要点：

1. 输入菜品的信息，点击添加，若菜品信息无误且完整，则提示新增成功。
2. 若信息不完整，且信息有误，系统会给出提醒。
3. 删除菜品，程序流程图如图3.27所示：



图3.27删除菜品流程图

测试要点：

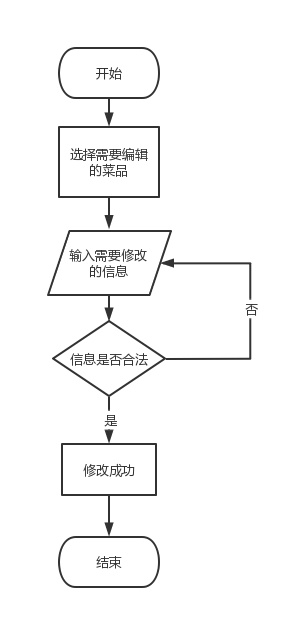
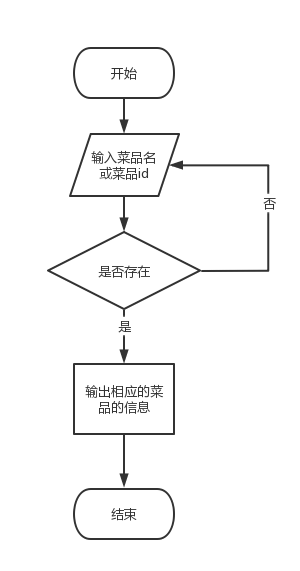
1. 点击菜品信息，点击删除按钮，系统提示删除成功。
2. 修改菜品，程序流程图如图3.28所示：

测试要点：

1. 在文本框中填写要修改的信息，若输入的信息合法则提示修改成功。
2. 若输入的信息不合法，则提示信息不合法。
3. 查询菜品，程序流程图如图3.29所示：

测试要点：

1. 在文本框中输入菜品名或菜品id，点击查询按钮。
2. 查询信息将在表格中显示。

删除菜品程序流程图 查询菜品程序流程图

## 3.9 业务管理模块设计说明

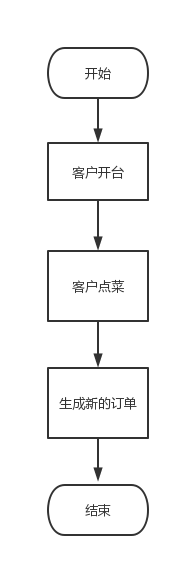
（1）功能：实现管理员对客户进行开台，点单操作。

（2）测试要点：

1. 输入客户的客户号，餐桌号以及人数，为客户开台。

2. 客户选菜，生成新的订单。

（3）程序流程图如图3.30所示：



业务管理程序流程图

## 3.10 支付管理模块设计说明

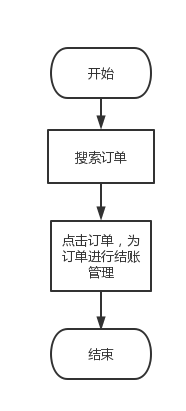
（1）功能：实现管理员对订单进行结账管理。

（2）测试要点：

1. 在输入框中输入餐桌号和客户名，定位到相应订单。

2. 在文本框中输入实收金额，找零

（3）程序流程图如图3.31所示：



支付管理流程图

# 4 数据库设计

## 4.1 数据库环境说明

数据库版本：MySQL 8.0.28

运行环境：Windows 11

## 4.2 数据库的命名规则

数据库名：cysjk

数据表命名规范：

（1）采用26个英文字母（区分大小写）和0-9的自然数（经常不需要）加上下划线’\_’组成，命名简洁明确，多个单词用下划线’\_’分隔。

（2）全部小写命名，禁止出现大写。

（3）禁止使用数据库关键字，如：name,time, datetime,password等。

（4）表名称不应该取得太长（一般不超过三个英文单词）。

（5）表的名称一般使用名词或者动宾短语。

（6）用单数形式表示名称。

（7）表必须填写描述信息（使用SQL语句建表时）。

字段命名规范：

（1）采用26个英文字母（区分大小写）和0-9个自然数（经常不需要）加下划线’\_’组成，命名简洁明确，多个单词用下划线’\_’分隔。

（2）全部小写命名，禁止出现大写。

（3）字段必须填写描述信息。

（4）禁止使用数据库关键字。

（5）字段名称一般采用名词或动宾短语。

（6）采用字段的名称必须是易于理解，一般不超过三个英文单词。

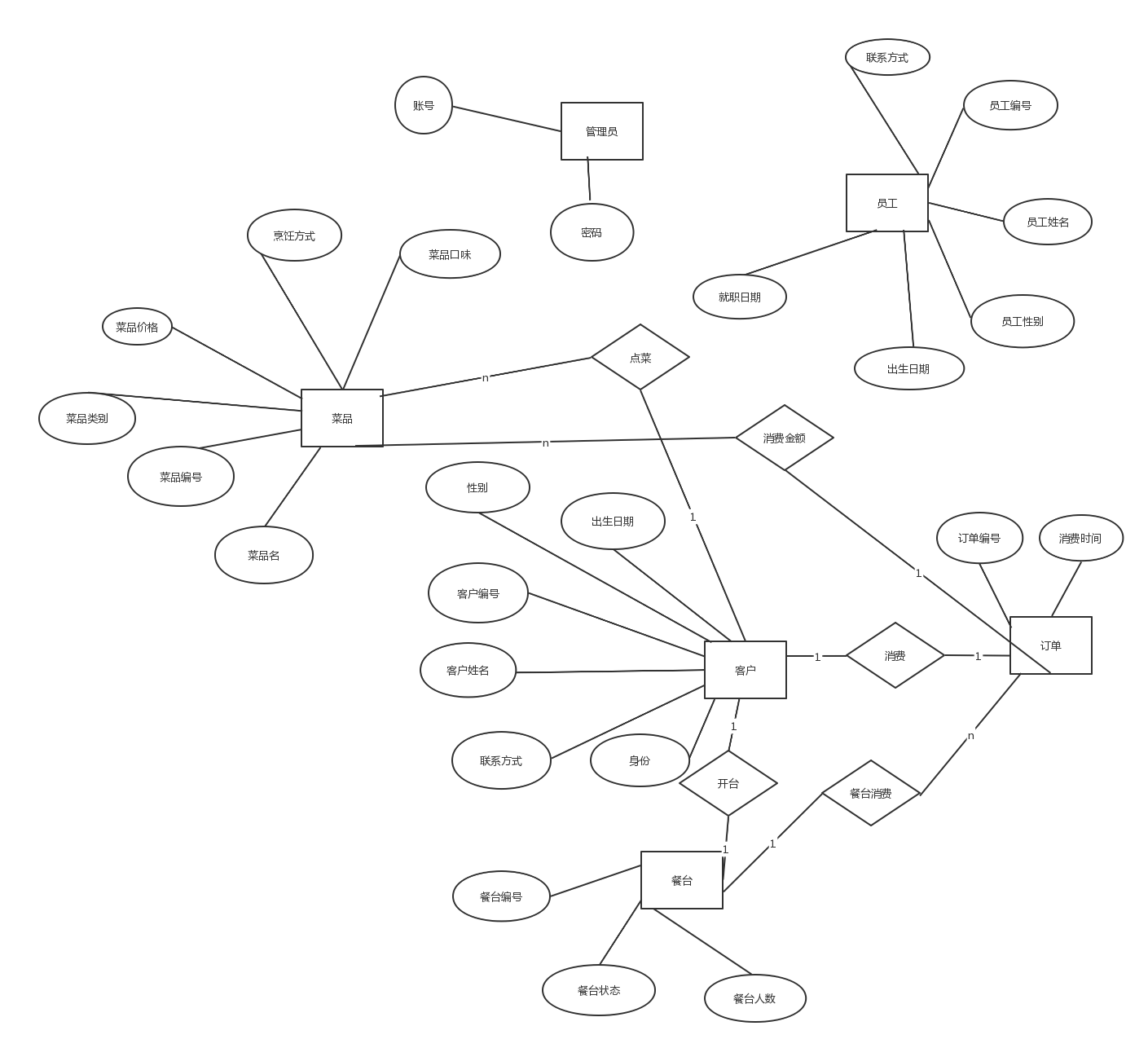
（7）在命名表的列时，不要重复表的名称。

（8）不要在列的名称中包含数据类型。

（9）字段命名使用完整命名，禁止缩写。

## 4.3 逻辑结构设计

E-R图



关系模型（主键用下划线标注，双下划线表示既是主键又是外键）

管理员（用户名，密码）

客户（客户编号，客户姓名，性别，单位，电话，贵宾卡号）

菜品（菜品编号，菜品名，类别编号，图片，菜品代码，单位，价格，状态）

员工（员工编号，员工姓名，员工性别，出生日期，身份证号，家庭住址，电话，职位，是否在职）

餐台（餐台编号，餐台状态，座位数）

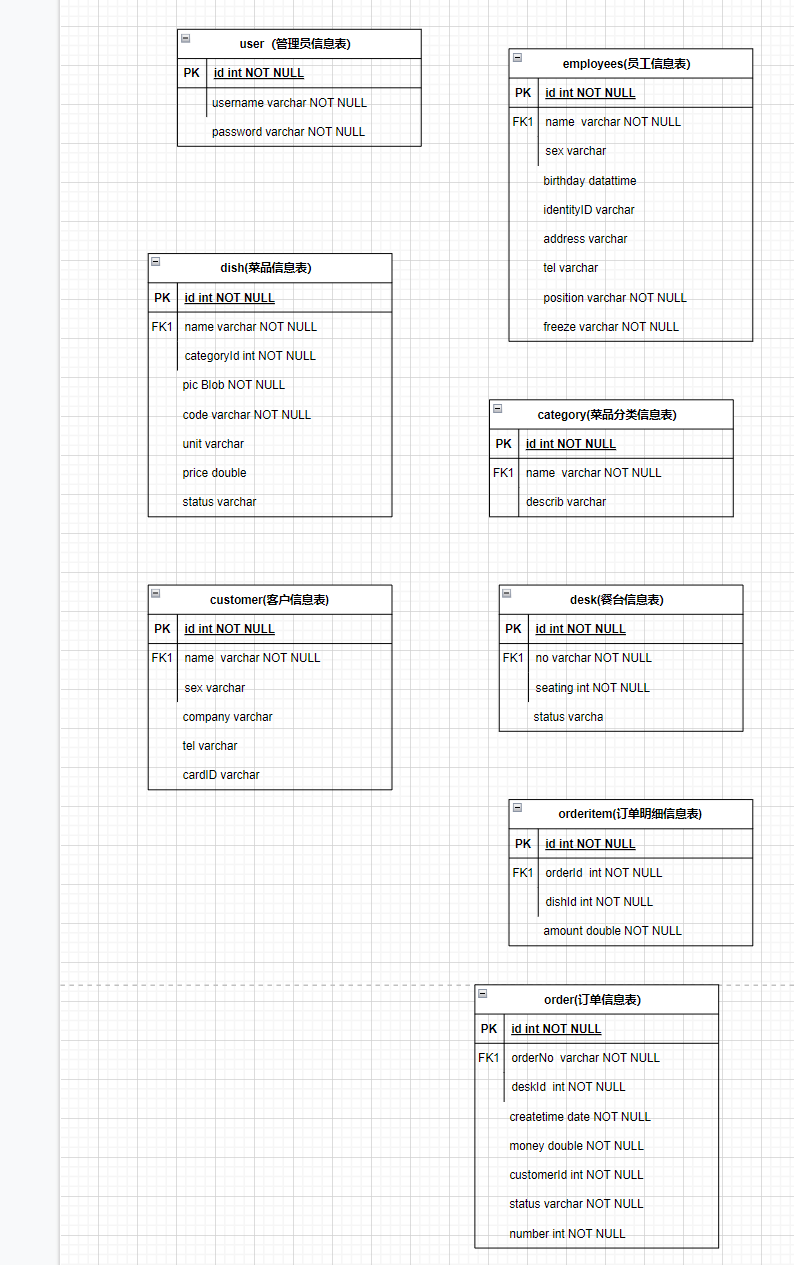
点单（订单编号，菜品编号，数量）

订单（订单编号，餐台编号，客户编号，开台时间，金额，状态，人数）

范式分析

所列关系均为第三范式，因为它消除了非主属性对主属性的传递依赖

逻辑模型图



## 4.4 物理结构设计

### 4.4.1 数据库表

表4.1 表汇总

|  |  |
| --- | --- |
| 表 名 | 功 能 说 明 |
| user | 存储管理员账号密码 |
| customer | 存储客户信息 |
| table | 存储餐台信息 |
| order | 存储订单信息 |
| employees | 存储员工信息 |
| dish | 存储菜品信息 |
| category | 存储菜品分类信息 |
| orderitem | 存储订单明细信息 |

表4.2 管理员信息表

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 表 名 | | user | | | | | |
| 数据库用户 | | 数据库管理员 | | | | | |
| 主 键 | | id | | | | | |
| 其他排序字段 | | 无 | | | | | |
| 索引字段 | | 无 | | | | | |
| 序号 | 字段名称 | 数据类型（精度范围） | 允许为空Y/N | 唯一Y/N | 区别度 | 默认值 | 约束条件/说明 |
| 1 | username | char | N | N |  | 无 | 10位数字 |
| 2 | password | char | N | N |  | 无 | 8-16位数字或字母 |
| sql脚本 | | CREATE TABLE `user` (  `id` int NOT NULL primary key,  `adname` char(20) NOT NULL,  `passstr` char(20) DEFAULT NULL  ) | | | | | |
| 备注 | |  | | | | | |

表4.3 客户信息表

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 表 名 | | customer | | | | | |
| 数据库用户 | | 数据库管理员 | | | | | |
| 主 键 | | id | | | | | |
| 其他排序字段 | | 无 | | | | | |
| 索引字段 | | 无 | | | | | |
| 序号 | 字段名称 | 数据类型（精度范围） | 允许为空Y/N | 唯一Y/N | 区别度 | 默认值 | 约束条件/说明 |
| 1 | id | char | N | Y |  | 无 | 无 |
| 2 | name | char | Y | N |  | 无 | 无 |
| 3 | sex | char | Y | N |  | 无 | 无 |
| 4 | company | date | Y | N |  | 无 | 无 |
| 5 | tel | char | Y | N |  | 无 | 无 |
| 6 | cardID | char | Y | N |  | 无 | 是、否中的一个 |
| sql脚本 | | CREATE TABLE `customer` (  `id` int NOT NULL primary key,  `name` char(10) DEFAULT NULL,  `sex` char(1) DEFAULT NULL,  `company` date DEFAULT NULL,  `tel` char(11) DEFAULT NULL,  `cardID` char(1) DEFAULT NULL  ) | | | | | |
| 备注 | |  | | | | | |

表4.4 员工信息表

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 表 名 | | employees | | | | | |
| 数据库用户 | | 数据库管理员 | | | | | |
| 主 键 | | id | | | | | |
| 其他排序字段 | | 无 | | | | | |
| 索引字段 | | 无 | | | | | |
| 序号 | 字段名称 | 数据类型（精度范围） | 允许为空Y/N | 唯一Y/N | 区别度 | 默认值 | 约束条件/说明 |
| 1 | id | int | N | Y |  | 无 | 无 |
| 2 | name | char | Y | N |  | 无 | 无 |
| 3 | sex | char | Y | N |  | 无 | 无 |
| 4 | birthday | date | Y | N |  | 无 | 无 |
| 5 | identityID | char | Y | N |  | 无 | 无 |
| 6 | address | char | Y | N |  | 无 | 无 |
| sql脚本 | | CREATE TABLE `employees` (  `id` int NOT NULL primary key,  `name` char(10) DEFAULT NULL,  `sex` char(1) DEFAULT NULL,  `birthday` char(11) DEFAULT NULL,  `identityID` date DEFAULT NULL,  `address` date DEFAULT NULL ,  ） | | | | | |
| 备注 | |  | | | | | |

表4.5 菜品信息表

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 表 名 | | dishes | | | | | |
| 数据库用户 | | 数据库管理员 | | | | | |
| 主 键 | | id | | | | | |
| 其他排序字段 | | 无 | | | | | |
| 索引字段 | | 无 | | | | | |
| 序号 | 字段名称 | 数据类型（精度范围） | 允许为空Y/N | 唯一Y/N | 区别度 | 默认值 | 约束条件/说明 |
| 1 | id | int | N | Y |  | 无 | 无 |
| 2 | name | char | Y | N |  | 无 | 无 |
| 3 | categoryID | int | Y | N |  | 无 | 无 |
| 4 | pic | Blob | Y | N |  | 无 | 无 |
| 5 | code | char | Y | N |  | 无 | 无 |
| 6 | unit | char | Y | N |  | 无 | 无 |
| sql脚本 | | CREATE TABLE `dishes` (  `id` int NOT NULL,  `name` char(10) DEFAULT NULL,  `categoryID` int DEFAULT NULL,  `pic` Blob DEFAULT NULL,  `code` char(10) DEFAULT NULL,  `unit` char(10) DEFAULT NULL, | | | | | |
| 备注 | |  | | | | | |

表4.6 餐台信息表

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 表 名 | | desk | | | | | |
| 数据库用户 | | 数据库管理员 | | | | | |
| 主 键 | | id | | | | | |
| 其他排序字段 | | 无 | | | | | |
| 索引字段 | | 无 | | | | | |
| 序号 | 字段名称 | 数据类型（精度范围） | 允许为空Y/N | 唯一Y/N | 区别度 | 默认值 | 约束条件/说明 |
| 1 | id | int | N | Y |  | 无 | 无 |
| 2 | no | char | Y | N |  | 无 | 无 |
| 3 | seating | int | Y | N |  | 无 | 无 |
| sql脚本 | | CREATE TABLE `desk` (  `id` int NOT NULL primary key,  `no` char(10) DEFAULT NULL,  `seating` int DEFAULT NULL, | | | | | |
| 备注 | |  | | | | | |

表4.7 订单信息表

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 表 名 | | order | | | | | |
| 数据库用户 | | 数据库管理员 | | | | | |
| 主 键 | | orderno | | | | | |
| 其他排序字段 | | 无 | | | | | |
| 索引字段 | | 无 | | | | | |
| 序号 | 字段名称 | 数据类型（精度范围） | 允许为空Y/N | 唯一Y/N | 区别度 | 默认值 | 约束条件/说明 |
| 1 | orderno | char | N | Y |  | 无 | 外键 |
| 2 | deskid | char | Y | N |  | 无 | 外键 |
| 3 | createtime | datetime | Y | N |  | 无 | 无 |
| 4 | money | int | Y | N |  | 无 | 无 |
| 5 | customerid | char | Y | N |  | 无 | 外键 |
| 6 | state | char | Y | N |  | 无 | 无 |
| 7 | number | int | Y | N |  | 无 | 无 |
| sql脚本 | | CREATE TABLE `orders` (  `orderno` char(20) NOT NULL COMMENT '订单编号' primay key,  `deskid` char(23) DEFAULT NULL COMMENT '餐台号',  `createtime` datetime DEFAULT NULL COMMENT '就餐时间',  `money` int DEFAULT NULL COMMENT '消费金额',  `customerid` char(23) DEFAULT NULL COMMENT '客户编号',  `state` char(3) DEFAULT NULL COMMENT '状态',  `number` int DEFAULT NULL COMMENT '就餐人数',  ) | | | | | |
| 备注 | |  | | | | | |

表4.8 菜品订单信息表

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 表 名 | | orderitem | | | | | |
| 数据库用户 | | 数据库管理员 | | | | | |
| 主 键 | | orderid | | | | | |
| 其他排序字段 | | 无 | | | | | |
| 索引字段 | | 无 | | | | | |
| 序号 | 字段名称 | 数据类型（精度范围） | 允许为空Y/N | 唯一Y/N | 区别度 | 默认值 | 约束条件/说明 |
| 1 | orderid | int | N | Y |  | 无 | 外键 |
| 2 | dishid | int | N | Y |  | 无 | 外键 |
| 3 | amount | int | N | N |  | 无 | 无 |
| sql脚本 | | CREATE TABLE `orderdishes` (  `orderid` int NOT NULL COMMENT '订单编号' primary key,  `dishid` int NOT NULL COMMENT '菜品号',  `amount` int DEFAULT NULL COMMENT '数量',  ) | | | | | |
| 备注 | |  | | | | | |

## 4.5 安全性设计

4.5.1 防止用户直接操作数据库的方法

（1）用户标识和鉴定

（2）存取控制

（3）数据加密

### 4.5.2 用户帐号密码的加密方法

1. 防止用户直接操作数据库的方法。用户只能用帐号登录到应用软件，通过应用软

件访问数据库，而没有其他途径操作数据库。

2. 用户帐号密码的加密方法。对用户帐号的密码进行加密处理，确保在任何地方都

不会出现密码的明文。

# 5 测试用例设计

表5.1 登录-注册测试用例

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 功能 | 用例编号 | 操作步骤  （输入条件） | 测试数据 | 预期结果 | 实际结果/判定 |
| 登录 | yl-001 | 输入‘用户名’和‘密码’ | admin,123456 | 登录成功 | 通过 |
| yl-002 | aaa,111 | 登录失败 | 通过 |
| yl-003 | bbb,222 | 登录失败 | 通过 |
| yl-004 | aaa,空 | 登录失败 | 通过 |
| yl-005 | 空,111 | 登录失败 | 通过 |
| yl-006 | bbb,一个空格 | 登录失败 | 通过 |
| yl-007 | 一个空格，222 | 登录失败 | 通过 |

表5.2 员工管理测试用例

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 功能 | 用例编号 | 操作步骤  （输入条件） | 测试数据 | 预期结果 | 实际结果/判定 |
| 增加员工 | y2-001 | 点击添加按  钮，输入‘姓  名’，‘性别’，‘出生日期’，‘地址’，‘电话号码’，‘地址’，‘职位’，‘是否在职’ | 1,1,1111-11-11,1,1,1,1,1 | 新增成功 | 通过 |
| 删除员工 | y2-002 | 点击删除按钮 | 1 | 删除成功 | 通过 |
| 修改员工 | y2-003 | 点击表格 | 1->A | 修改成功 | 通过 |
| 查询员工 | y2-004 | 输入员工编  号或姓名，点击查询  按钮 | 1 | 查询成功 | 通过 |

表5.3 客户管理测试用例

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 功能 | 用例编号 | 操作步骤  （输入条件） | 测试数据 | 预期结果 | 实际结果/判定 |
| 增加客户 | y3-001 | 点击添加按  钮，输入‘姓  名’，‘性别’，‘出生日期’，‘电话号码’，‘贵宾卡号’ | 梅西，男，巴萨，123456，666 | 新增成功 | 通过 |
| 删除客户 | y3-002 | 点击删除按钮 | 梅西 | 删除成功 | 通过 |
| 修改客户 | y3-003 | 点击表格 | 123456->111111 | 修改成功 | 通过 |
| 查询客户 | y3-004 | 输入员工编  号或姓名，点击查询  按钮 | 梅西 | 查询成功 | 通过 |

表5.4 菜品管理测试用例

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 功能 | 用例编号 | 操作步骤  （输入条件） | 测试数据 | 预期结果 | 实际结果/判定 |
| 增加菜品 | y4-001 | 点击添加按  钮，输入‘菜品名’，‘图片’，‘代码’，‘份数’，‘价格’，‘状态’ | 红烧肉，3，图片，3，1，15.0，可用 | 新增成功 | 通过 |
| 删除菜品 | y4-002 | 点击删除按钮 | 红烧肉 | 删除成功 | 通过 |
| 修改菜品 | y4-003 | 点击表格 | 3->4 | 修改成功 | 通过 |
| 查询菜品 | y4-004 | 输入菜品编  号或菜名，点击查询  按钮 | 红烧肉 | 查询成功 | 通过 |

表5.5 餐台管理测试用例

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 功能 | 用例编号 | 操作步骤  （输入条件） | 测试数据 | 预期结果 | 实际结果/判定 |
| 增加餐台 | y5-001 | 点击添加按  钮，输入‘座位编号’，’座位数‘，’状态‘ | 1，1，空闲 | 新增成功 | 通过 |
| 删除餐台 | y5-002 | 点击删除按钮 | 1 | 删除成功 | 通过 |
| 修改餐台 | y5-003 | 点击表格 | 1->10 | 修改成功 | 通过 |
| 查询餐台 | y5-004 | 输入菜品编  号或菜名，点击查询  按钮 | 1 | 查询成功 | 通过 |

表5.6 业务管理测试用例

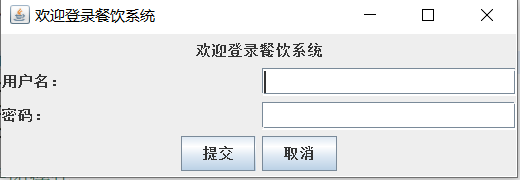
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 功能 | 用例编号 | 操作步骤  （输入条件） | 测试数据 | 预期结果 | 实际结果/判定 |
| 开台 | y6-001 | 输入客户名，餐桌号，就餐人数 | 1，1，2 | 开台成功 | 通过 |
| 生成订单 | y6-002 | 点菜后点击保存 | 无 | 操作成功 | 通过 |

表5.7 结账管理测试用例

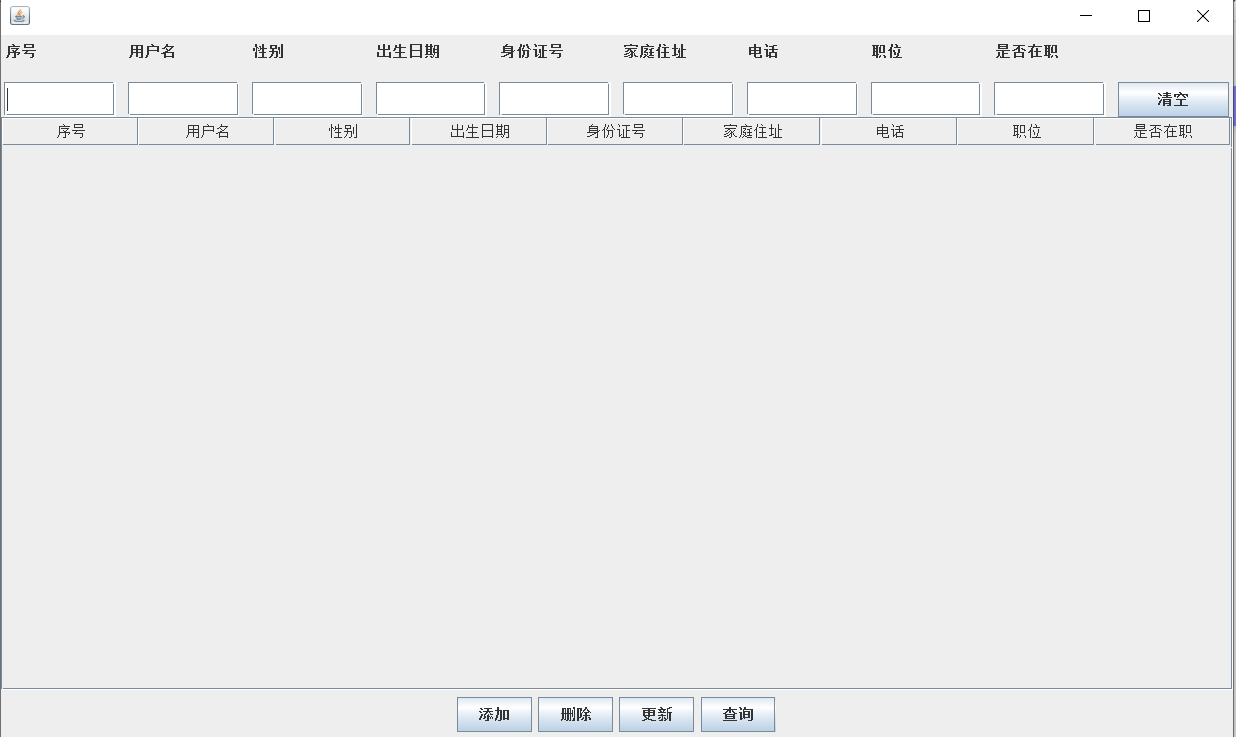
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 功能 | 用例编号 | 操作步骤  （输入条件） | 测试数据 | 预期结果 | 实际结果/判定 |
| 删除订单 | y7-001 | 点击订单后点击删除按钮 | 202306122224223410 | 删除成功 | 通过 |
| 结账操作 | Y7-002 | 点击订单后输入金额 | 15.0 | 支付成功 | 通过 |

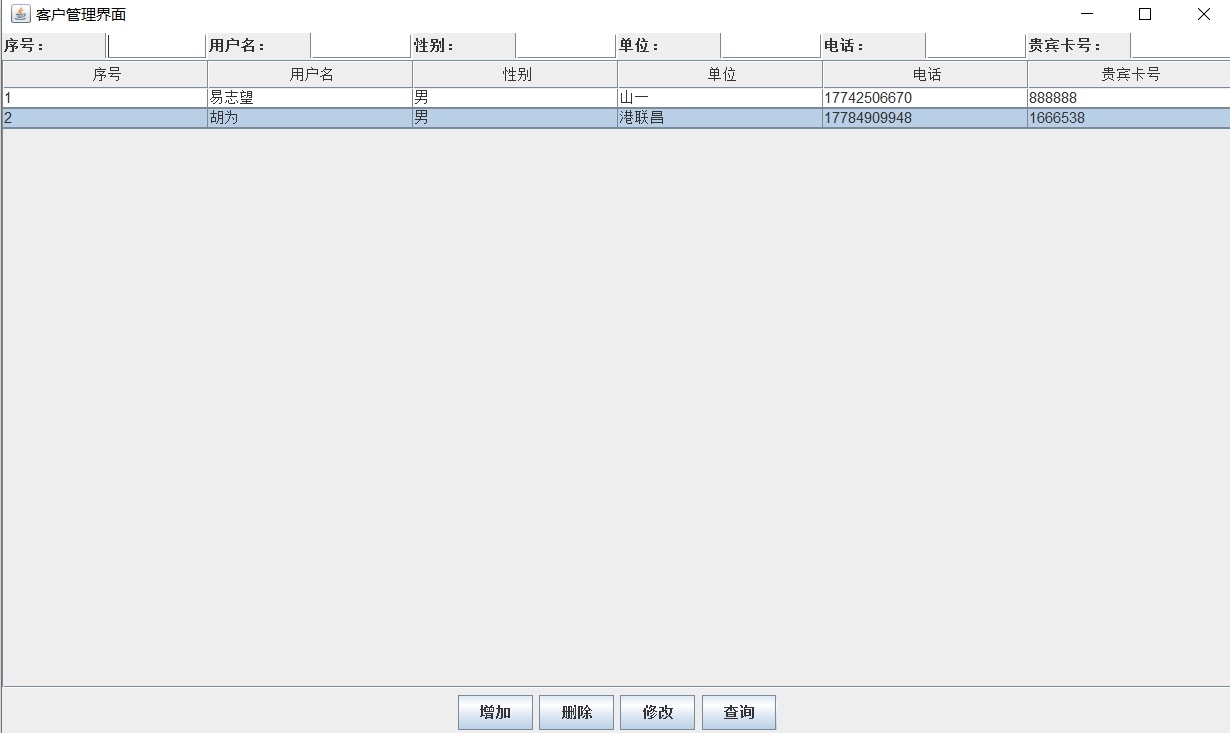
# 附录

## 附录1 程序运行结果

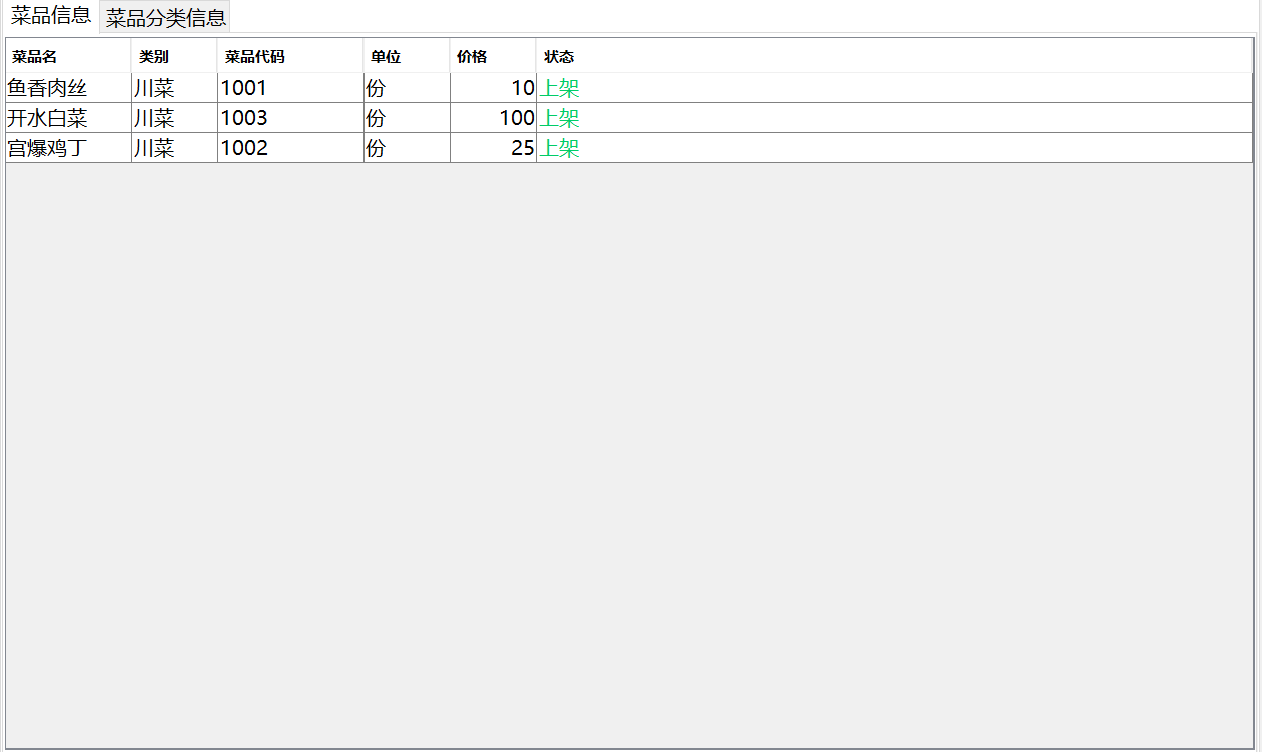


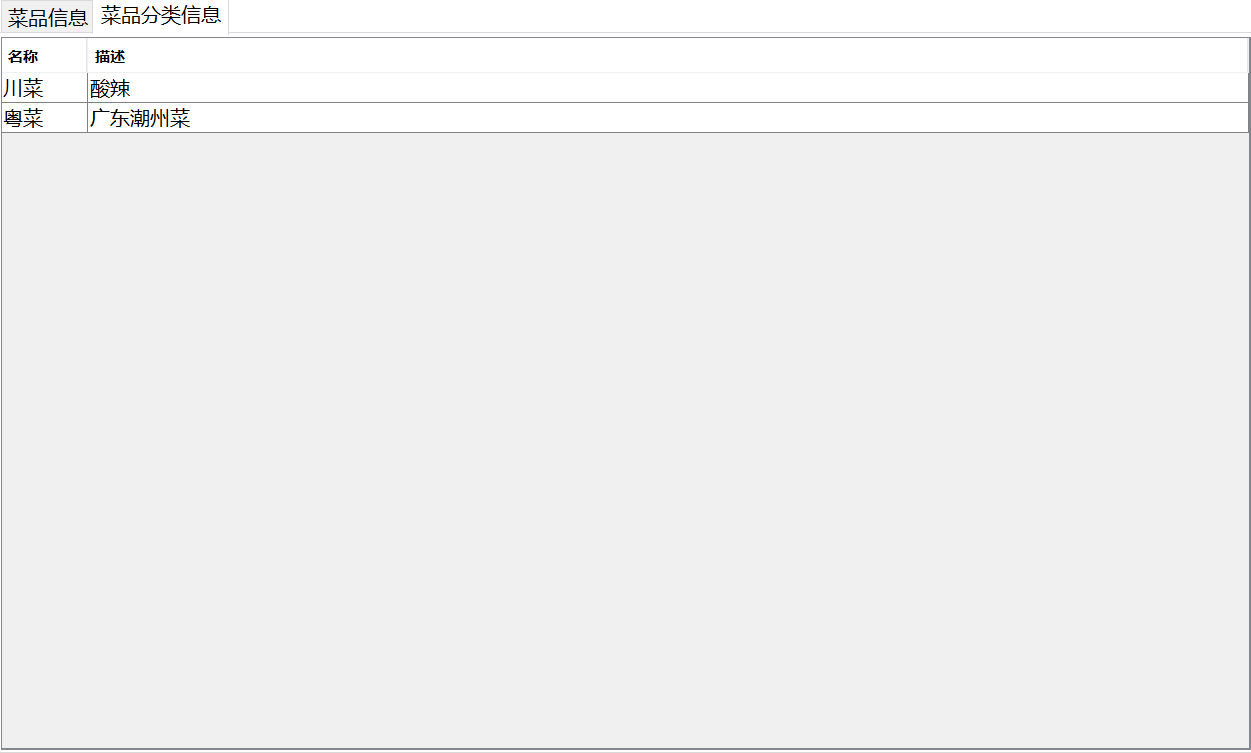


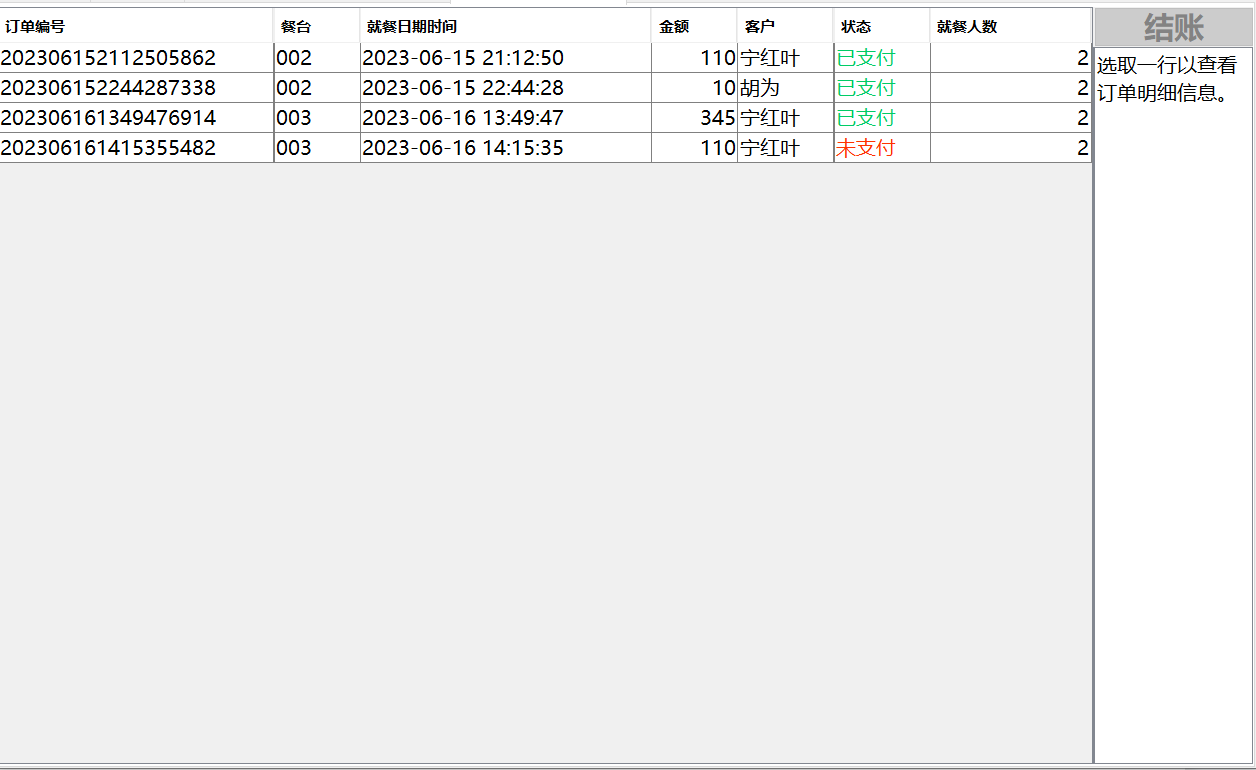




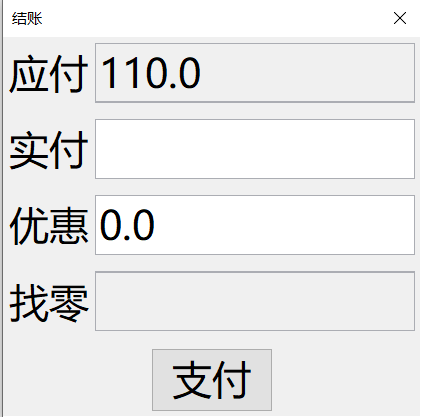












## 附录2 程序源代码

package dao;

import java.sql.PreparedStatement;

import java.sql.ResultSet;

import java.sql.Statement;

import java.util.ArrayList;

import java.util.List;

import entity.Customer;

import tool.DataOperation;

//定义抽象类 CustomerOperation，用于定义与客户相关的数据库操作方法

public abstract class CustomerOperation {

// 从数据库中读取所有客户的信息，并返回一个列表

public static List<Customer> load() {

List<Customer> list = new ArrayList<Customer>();

try {

// 创建 PreparedStatement 对象，执行查询语句，获取结果集

PreparedStatement preparedStatement = DataOperation.connection.prepareStatement("select \* from customer");

ResultSet resultSet = preparedStatement.executeQuery();

// 遍历结果集，将每个客户信息封装为 Customer 对象，添加到列表中

while (resultSet.next()) {

Customer customer = new Customer(

resultSet.getInt(1),

resultSet.getString(2),

resultSet.getString(3),

resultSet.getString(4),

resultSet.getString(5),

resultSet.getString(6));

list.add(customer);

}

} catch (Exception e) {

DataOperation.showException(e);

}

// 返回客户信息列表

return list;

}

// 向数据库中添加一个新客户

public static int add(Customer customer) {

try {

// 创建 PreparedStatement 对象，执行插入语句，获取自动生成的主键

PreparedStatement preparedStatement = DataOperation.connection.prepareStatement(

"insert into customer(name,sex,company,tel,cardId) values(?,?,?,?,?)",

Statement.RETURN\_GENERATED\_KEYS);

preparedStatement.setString(1, customer.getName());

preparedStatement.setString(2, customer.getSex());

preparedStatement.setString(3, customer.getCompany());

preparedStatement.setString(4, customer.getTel());

preparedStatement.setString(5, customer.getCardId());

if (preparedStatement.executeUpdate() != 0) {

ResultSet resultSet = preparedStatement.getGeneratedKeys(); // 获取插入的数据的主键

if (resultSet.next())

return resultSet.getInt(1);

}

} catch (Exception e) {

DataOperation.showException(e);

}

return -1;

}

// 根据给定的客户ID，从数据库中删除对应的客户信息

public static boolean delete(int id) {

try {

// 创建 PreparedStatement 对象，执行删除语句

PreparedStatement preparedStatement = DataOperation.connection.prepareStatement(

"delete from customer where id=?");

preparedStatement.setInt(1, id);

return preparedStatement.executeUpdate() != 0;

} catch (Exception e) {

DataOperation.showException(e);

}

return false;

}

// 根据给定的客户信息，更新数据库中对应客户的信息

public static boolean change(Customer customer) {

try {

// 创建 PreparedStatement 对象，执行更新语句

PreparedStatement preparedStatement = DataOperation.connection.prepareStatement(

"update customer set name=?,sex=?,company=?,tel=?,cardId=? where id=?");

preparedStatement.setString(1, customer.getName());

preparedStatement.setString(2, customer.getSex());

preparedStatement.setString(3, customer.getCompany());

preparedStatement.setString(4, customer.getTel());

preparedStatement.setString(5, customer.getCardId());

preparedStatement.setInt(6, (int) customer.getId());

return preparedStatement.executeUpdate() != 0;

} catch (Exception e) {

DataOperation.showException(e);

}

return false;

}

// 根据给定的卡号，返回对应客户的ID

public static int toCustomerId(String cardId) {

try {

// 创建 PreparedStatement 对象，执行查询语句，获取结果集

PreparedStatement preparedStatement = DataOperation.connection

.prepareStatement("select id from customer where cardId=?");

preparedStatement.setString(1, cardId);

ResultSet resultSet = preparedStatement.executeQuery();

while (resultSet.next()) {

return resultSet.getInt(1);

}

} catch (Exception e) {

DataOperation.showException(e);

}

return -1;

}

// 根据给定的客户ID，返回对应客户的姓名

public static String toCustomerName(int id) {

try {

// 创建 PreparedStatement 对象，执行查询语句，获取结果集

PreparedStatement preparedStatement = DataOperation.connection

.prepareStatement("select name from customer where id=?");

preparedStatement.setInt(1, id);

ResultSet resultSet = preparedStatement.executeQuery();

while (resultSet.next()) {

return resultSet.getString(1);

}

} catch (Exception e) {

DataOperation.showException(e);

}

return "???";

}

// 判断给定的卡号是否唯一，如果唯一则返回 true，否则返回 false

public static boolean isUnique(String cardId, int id) {

try {

// 创建 PreparedStatement 对象，执行查询语句，获取结果集

PreparedStatement preparedStatement = DataOperation.connection

.prepareStatement("select id from customer where cardId=?");

preparedStatement.setString(1, cardId);

ResultSet resultSet = preparedStatement.executeQuery();

while (resultSet.next()) {

if (id != resultSet.getInt(1))

return false;

}

} catch (Exception e) {

DataOperation.showException(e);

}

return true;

}

// 重载的 isUnique 方法，不带客户ID，仅判断给定的卡号在数据库中是否唯一

public static boolean isUnique(String cardId) {

return isUnique(cardId, -1);

}

package dao;

import java.sql.PreparedStatement;

import java.sql.ResultSet;

import java.sql.Statement;

import java.util.ArrayList;

import java.util.List;

import entity.Dish;

import tool.DataOperation;

public abstract class DishOperation {

// 从数据库中读取所有餐品的信息，并以 Dish 对象的形式返回一个餐品信息列表。

public static List<Dish> load() {

List<Dish> list = new ArrayList<Dish>();

try {

// 创建 PreparedStatement 对象，使用 select 语句从数据库中查询数据

PreparedStatement preparedStatement = DataOperation.connection.prepareStatement("select \* from dish");

// 执行查询操作

ResultSet resultSet = preparedStatement.executeQuery();

// 遍历结果集，将每条记录转化为 Dish 对象，并添加到列表中

while (resultSet.next()) {

Dish dish = new Dish(

resultSet.getInt(1),

resultSet.getString(2),

resultSet.getInt(3),

resultSet.getString(4),

resultSet.getString(5),

resultSet.getDouble(6),

resultSet.getString(7));

list.add(dish);

}

} catch (Exception e) {

// 如果出现异常，则打印异常信息

DataOperation.showException(e);

}

// 返回餐品列表

return list;

}

// 向数据库中添加一个新餐品，参数是一个 Dish 对象，返回值为插入的数据的主键。

public static int add(Dish dish) {

try {

// 创建 PreparedStatement 对象，使用 insert 语句向数据库中插入数据，并返回插入的主键

PreparedStatement preparedStatement = DataOperation.connection.prepareStatement(

"insert into dish(name,dishTypeId,code,unit,price,status) values(?,?,?,?,?,?)",

Statement.RETURN\_GENERATED\_KEYS);

preparedStatement.setString(1, dish.getName());

preparedStatement.setInt(2, dish.getDishTypeId());

preparedStatement.setString(3, dish.getCode());

preparedStatement.setString(4, dish.getUnit());

preparedStatement.setDouble(5, dish.getPrice());

preparedStatement.setString(6, dish.getStatus());

// 执行插入操作，并获取插入的主键

if (preparedStatement.executeUpdate() != 0) {

ResultSet resultSet = preparedStatement.getGeneratedKeys(); // 获取插入的数据的主键

if (resultSet.next())

return resultSet.getInt(1);

}

} catch (Exception e) {

// 如果出现异常，则打印异常信息

DataOperation.showException(e);

}

// 如果插入失败，则返回 -1

return -1;

}

// 根据给定的餐品ID，从数据库中删除对应的餐品信息，返回删除操作的结果。

public static boolean delete(int id) {

try {

// 创建 PreparedStatement 对象，使用 delete 语句从数据库中删除数据

PreparedStatement preparedStatement = DataOperation.connection.prepareStatement(

"delete from dish where id=?");

preparedStatement.setInt(1, id);

// 执行删除操作，返回删除结果

return preparedStatement.executeUpdate() != 0;

} catch (Exception e) {

// 如果出现异常，则打印异常信息

DataOperation.showException(e);

}

// 如果删除失败，则返回 false

return false;

}

// 根据给定的餐品信息，更新数据库中对应餐品的信息，返回更新操作的结果。

public static boolean change(Dish dish) {

try {

// 创建 PreparedStatement 对象，使用 update 语句更新数据库中的数据

PreparedStatement preparedStatement = DataOperation.connection.prepareStatement(

"update dish set name=?,dishTypeId=?,code=?,unit=?,price=?,status=? where id=?");

preparedStatement.setString(1, dish.getName());

preparedStatement.setInt(2, dish.getDishTypeId());

preparedStatement.setString(3, dish.getCode());

preparedStatement.setString(4, dish.getUnit());

preparedStatement.setDouble(5, dish.getPrice());

preparedStatement.setString(6, dish.getStatus());

preparedStatement.setInt(7, dish.getId());

// 执行更新操作，返回更新结果

return preparedStatement.executeUpdate() != 0;

} catch (Exception e) {

// 如果出现异常，则打印异常信息

DataOperation.showException(e);

}

// 如果更新失败，则返回 false

return false;

}

// 根据给定的餐品名称，返回对应餐品的ID。

public static int toDishId(String dishName) {

try {

// 创建 PreparedStatement 对象，使用 select 语句查询数据库中的数据

PreparedStatement preparedStatement = DataOperation.connection

.prepareStatement("select id from dish where name=?");

preparedStatement.setString(1, dishName);

// 执行查询操作，并返回查询结果

ResultSet resultSet = preparedStatement.executeQuery();

while (resultSet.next()) {

return resultSet.getInt(1);

}

} catch (Exception e) {

// 如果出现异常，则打印异常信息

DataOperation.showException(e);

}

// 如果查询失败，则返回 -1

return -1;

}

// 判断给定的餐品名称是否唯一，如果唯一则返回 true，否则返回 false。

public static boolean isUnique(String name, int id) {

try {

// 创建 PreparedStatement 对象，使用 select 语句查询数据库中的数据

PreparedStatement preparedStatement = DataOperation.connection

.prepareStatement("select id from dish where name=?");

preparedStatement.setString(1, name);

// 执行查询操作，并返回查询结果

ResultSet resultSet = preparedStatement.executeQuery();

while (resultSet.next()) {

if (id != resultSet.getInt(1))

return false;

}

} catch (Exception e) {

// 如果出现异常，则打印异常信息

DataOperation.showException(e);

}

// 如果查询成功，则说明给定名称不是唯一的，返回 false；否则返回 true

return true;

}

// 重载的 isUnique 方法，不带餐品ID，仅判断给定的餐品名称在数据库中是否唯一。

public static boolean isUnique(String name) {

// 调用带 id 参数的 isUnique 方法，id 传入 -1，表示不需要指定 id，只判断名称唯一性即可。

return isUnique(name, -1);

}

}

package dao;

import java.sql.PreparedStatement;

import java.sql.ResultSet;

import java.sql.Statement;

import java.util.ArrayList;

import java.util.List;

import entity.DishType;

import tool.DataOperation;

//DishTypeOperation 类，包含了一些对菜品类型进行操作的静态方法。

public abstract class DishTypeOperation {

//从数据库中读取所有菜品类型的信息，并以 DishType 对象的形式返回一个菜品类型信息列表。

public static List<DishType> load() {

List<DishType> list = new ArrayList<DishType>();

try {

PreparedStatement preparedStatement = DataOperation.connection.prepareStatement("select \* from dishType");

ResultSet resultSet = preparedStatement.executeQuery();

while (resultSet.next()) {

DishType dishType = new DishType(

resultSet.getInt(1),

resultSet.getString(2),

resultSet.getString(3));

list.add(dishType);

}

} catch (Exception e) {

DataOperation.showException(e);

}

return list;

}

//向数据库中添加一个新的菜品类型。

public static int add(DishType dishType) {

try {

PreparedStatement preparedStatement = DataOperation.connection.prepareStatement(

"insert into dishType(name,`describe`) values(?,?)",

Statement.RETURN\_GENERATED\_KEYS);

preparedStatement.setString(1, dishType.getName());

preparedStatement.setString(2, dishType.getDescribe());

if (preparedStatement.executeUpdate() != 0) {

ResultSet resultSet = preparedStatement.getGeneratedKeys(); // 获取插入的数据的主键

if (resultSet.next())

return resultSet.getInt(1);

}

} catch (Exception e) {

DataOperation.showException(e);

}

return -1;

}

//根据给定的菜品类型ID，从数据库中删除对应的菜品类型信息。

public static boolean delete(int id) {

try {

PreparedStatement preparedStatement = DataOperation.connection.prepareStatement(

"delete from dishType where id=?");

preparedStatement.setInt(1, id);

return preparedStatement.executeUpdate() != 0;

} catch (Exception e) {

DataOperation.showException(e);

return false;

}

}

//根据给定的菜品类型信息，更新数据库中对应菜品类型的信息。

public static boolean change(DishType dishType) {

try {

PreparedStatement preparedStatement = DataOperation.connection.prepareStatement(

"update dishType set name=?,`describe`=? where id=?");

preparedStatement.setString(1, dishType.getName());

preparedStatement.setString(2, dishType.getDescribe());

preparedStatement.setInt(3, (int) dishType.getId());

return preparedStatement.executeUpdate() != 0;

} catch (Exception e) {

DataOperation.showException(e);

}

return false;

}

//从数据库中读取所有菜品类型的名称，并以字符串列表的形式返回。

public static List<String> loadDishTypeName() {

List<String> list = new ArrayList<String>();

try {

PreparedStatement preparedStatement = DataOperation.connection

.prepareStatement("select name from dishType");

ResultSet resultSet = preparedStatement.executeQuery();

while (resultSet.next()) {

list.add(resultSet.getString(1));

}

} catch (Exception e) {

DataOperation.showException(e);

}

return list;

}

//根据给定的菜品类型名称，返回对应菜品类型的ID。

public static int toDishTypeId(String dishTypeName) {

try {

PreparedStatement preparedStatement = DataOperation.connection

.prepareStatement("select id from dishType where name=?");

preparedStatement.setString(1, dishTypeName);

ResultSet resultSet = preparedStatement.executeQuery();

while (resultSet.next()) {

return resultSet.getInt(1);

}

} catch (Exception e) {

DataOperation.showException(e);

}

return -1;

}

//根据给定的菜品类型ID，返回对应菜品类型的名称。

public static String toDishTypeName(int dishTypeId) {

try {

PreparedStatement preparedStatement = DataOperation.connection

.prepareStatement("select name from dishType where id=?");

preparedStatement.setInt(1, dishTypeId);

ResultSet resultSet = preparedStatement.executeQuery();

while (resultSet.next()) {

return resultSet.getString(1);

}

} catch (Exception e) {

DataOperation.showException(e);

}

return "???";

}

//判断给定的菜品类型名称是否唯一，如果唯一则返回 true，否则返回 false。

public static boolean isUnique(String name, int id) {

try {

PreparedStatement preparedStatement = DataOperation.connection

.prepareStatement("select id from dishType where name=?");

preparedStatement.setString(1, name);

ResultSet resultSet = preparedStatement.executeQuery();

while (resultSet.next()) {

if (id != resultSet.getInt(1))

return false;

}

} catch (Exception e) {

DataOperation.showException(e);

}

return true;

}

//判断给定的菜品类型名称在数据库中是否唯一，如果唯一则返回 true，否则返回 false。

public static boolean isUnique(String name) {

return isUnique(name, -1);

}

}

**package** dao;

**import** java.sql.PreparedStatement;

**import** java.sql.ResultSet;

**import** java.sql.Statement;

**import** java.util.ArrayList;

**import** java.util.List;

**import** entity.Employee;

**import** tool.DataOperation;

**public** **abstract** **class** EmployeeOperation {

// 从数据库中读取所有员工信息，并以 Employee 对象的形式返回一个员工信息列表。

**public** **static** List<Employee> load() {

List<Employee> list = **new** ArrayList<Employee>();

**try** {

PreparedStatement preparedStatement = DataOperation.*connection*.prepareStatement("select \* from employee");

ResultSet resultSet = preparedStatement.executeQuery();

**while** (resultSet.next()) {

Employee employee = **new** Employee(

resultSet.getInt(1), // 员工ID

resultSet.getString(2), // 员工姓名

resultSet.getString(3), // 性别

resultSet.getDate(4), // 生日

resultSet.getString(5), // 身份证号码

resultSet.getString(6), // 住址

resultSet.getString(7), // 电话

resultSet.getString(8), // 职位

resultSet.getString(9)); // 冻结状态

list.add(employee);

}

} **catch** (Exception e) {

DataOperation.*showException*(e); // 打印异常信息到控制台

e.printStackTrace();

}

**return** list;

}

// 向数据库中添加一个新的员工信息。

**public** **static** **int** add(Employee employee) {

**try** {

PreparedStatement preparedStatement = DataOperation.*connection*.prepareStatement(

"insert into employee(name,sex,birthday,identityId,address,tel,position,freeze) values(?,?,?,?,?,?,?,?)",

Statement.***RETURN\_GENERATED\_KEYS***);

preparedStatement.setString(1, employee.getName());

preparedStatement.setString(2, employee.getSex());

preparedStatement.setDate(3, employee.getBirthday());

preparedStatement.setString(4, employee.getIdentityId());

preparedStatement.setString(5, employee.getAddress());

preparedStatement.setString(6, employee.getTel());

preparedStatement.setString(7, employee.getPosition());

preparedStatement.setString(8, employee.getFreeze());

**if** (preparedStatement.executeUpdate() != 0) {

ResultSet resultSet = preparedStatement.getGeneratedKeys(); // 获取插入的数据的主键

**if** (resultSet.next())

**return** resultSet.getInt(1); // 返回新插入的员工信息的ID

}

} **catch** (Exception e) {

DataOperation.*showException*(e); // 打印异常信息到控制台

e.printStackTrace();

}

**return** -1;

}

// 根据给定的员工ID，从数据库中删除对应的员工信息。

**public** **static** **boolean** delete(**int** id) {

**try** {

PreparedStatement preparedStatement = DataOperation.*connection*.prepareStatement(

"delete from employee where id=?");

preparedStatement.setInt(1, id);

**return** preparedStatement.executeUpdate() != 0;

} **catch** (Exception e) {

DataOperation.*showException*(e); // 打印异常信息到控制台

e.printStackTrace();

}

**return** **false**;

}

// 根据给定的员工信息，更新数据库中对应员工的信息。

**public** **static** **boolean** change(Employee employee) {

**try** {

PreparedStatement preparedStatement = DataOperation.*connection*.prepareStatement(

"update employee set name=?,sex=?,birthday=?,identityId=?,address=?,tel=?,position=?,freeze=? where id=?");

preparedStatement.setString(1, employee.getName());

preparedStatement.setString(2, employee.getSex());

preparedStatement.setDate(3, employee.getBirthday());

preparedStatement.setString(4, employee.getIdentityId());

preparedStatement.setString(5, employee.getAddress());

preparedStatement.setString(6, employee.getTel());

preparedStatement.setString(7, employee.getPosition());

preparedStatement.setString(8, employee.getFreeze());

preparedStatement.setInt(9, employee.getId());

**return** preparedStatement.executeUpdate() != 0;

} **catch** (Exception e) {

DataOperation.*showException*(e); // 打印异常信息到控制台

e.printStackTrace();

}

**return** **false**;

}

}

package dao;

import java.sql.PreparedStatement;

import java.sql.ResultSet;

import java.sql.Statement;

import java.util.ArrayList;

import java.util.LinkedHashMap;

import java.util.List;

import java.util.Map;

import entity.Order;

import entity.OrderItem;

import tool.DataOperation;

public abstract class OrderOperation {

// 读取所有顾客的身份证号码和姓名，返回一个 Map 对象，其中 key 为身份证号码，value 为顾客姓名。

public static Map<String, String> loadCustomerNameAndCardId() {

Map<String, String> map = new LinkedHashMap<String, String>();

try {

PreparedStatement preparedStatement = DataOperation.connection

.prepareStatement("select cardId,name from customer");

ResultSet resultSet = preparedStatement.executeQuery();

while (resultSet.next()) {

map.put(resultSet.getString(1), resultSet.getString(2));

}

} catch (Exception e) {

DataOperation.showException(e);

}

return map; // 返回结果

}

// 读取所有订单的信息，返回一个 List 对象，其中每个元素是一个 Order 对象，表示一个订单。

public static List<Order> loadOrder() {

List<Order> list = new ArrayList<Order>();

try {

PreparedStatement preparedStatement = DataOperation.connection.prepareStatement("select \* from `order`");

ResultSet resultSet = preparedStatement.executeQuery();

while (resultSet.next()) {

Order order = new Order(

resultSet.getInt(1),

resultSet.getString(2),

resultSet.getInt(3),

resultSet.getTimestamp(4),

resultSet.getDouble(5),

resultSet.getInt(6),

resultSet.getString(7),

resultSet.getInt(8));

list.add(order); // 将每条记录转换为 Order 对象，并添加到 list 中

}

} catch (Exception e) {

DataOperation.showException(e);

}

return list; // 返回结果

}

// 向数据库中添加一条新的订单记录，返回新插入的订单记录的 ID。

public static int addOrder(Order order) {

try {

PreparedStatement preparedStatement = DataOperation.connection.prepareStatement(

"insert into `order`(orderNo,tableId,createtime,money,customerId,status,number) values(?,?,?,?,?,?,?)",

Statement.RETURN\_GENERATED\_KEYS);

preparedStatement.setString(1, order.getOrderNo());

preparedStatement.setInt(2, order.getTableId());

preparedStatement.setTimestamp(3, order.getCreatetime());

preparedStatement.setDouble(4, order.getMoney());

preparedStatement.setInt(5, order.getCustomerId());

preparedStatement.setString(6, order.getStatus());

preparedStatement.setInt(7, order.getNumber());

if (preparedStatement.executeUpdate() != 0) {

ResultSet resultSet = preparedStatement.getGeneratedKeys(); // 获取插入的数据的主键

if (resultSet.next())

return resultSet.getInt(1); // 返回新插入的订单记录的 ID

}

} catch (Exception e) {

DataOperation.showException(e);

}

return -1;

}

// 修改指定 ID 的订单状态为“已支付”，返回操作是否成功。

public static boolean changeOrder(int id) {

try {

PreparedStatement preparedStatement = DataOperation.connection.prepareStatement(

"update `order` set status='已支付' where id=?");

preparedStatement.setInt(1, id);

return preparedStatement.executeUpdate() != 0; // 返回操作结果

} catch (Exception e) {

DataOperation.showException(e);

}

return false;

}

// 向数据库中添加一条新的订单项记录，返回新插入的订单记录的 ID。

public static int addOrderItem(OrderItem orderItem) {

try {

PreparedStatement preparedStatement = DataOperation.connection.prepareStatement(

"insert into orderitem(orderId,dishId,amount) values(?,?,?)",

Statement.RETURN\_GENERATED\_KEYS);

preparedStatement.setInt(1, orderItem.getOrderId());

preparedStatement.setInt(2, orderItem.getDishId());

preparedStatement.setInt(3, orderItem.getAmount());

if (preparedStatement.executeUpdate() != 0) {

ResultSet resultSet = preparedStatement.getGeneratedKeys(); // 获取插入的数据的主键

if (resultSet.next())

return resultSet.getInt(1); // 返回新插入的订单记录的 ID

}

} catch (Exception e) {

DataOperation.showException(e);

}

return -1;

}

// 根据给定的订单 ID，从数据库中读取该订单的详细信息，并将其以字符串的形式返回。

// 返回的字符串包含订单的菜品详情，格式为“菜品名称 \* 数量”（每个菜品占一行）。

public static String getDetails(int id) {

String dishName = new String(), amount = new String(), result = new String();

try {

PreparedStatement preparedStatement = DataOperation.connection.prepareStatement(

"select dish.name,orderitem.amount from dish,orderitem where orderitem.dishId = dish.id and orderitem.orderId = ?");

preparedStatement.setInt(1, id);

ResultSet resultSet = preparedStatement.executeQuery();

while (resultSet.next()) {

dishName = resultSet.getString(1);

amount = resultSet.getInt(2) + "\n";

result += dishName + " \* " + amount; // 将菜品详情拼接为一个字符串

}

} catch (Exception e) {

DataOperation.showException(e);

}

return "订单明细：\n\n" + result; // 返回订单的菜品详情

}

}

package dao;

import java.sql.PreparedStatement;

import java.sql.ResultSet;

import java.sql.Statement;

import java.util.ArrayList;

import java.util.List;

import entity.Table;

import tool.DataOperation;

public abstract class TableOperation {

// 从 `table` 表中读取所有数据并封装成一个 `List<Table>` 返回。

public static List<Table> load() {

List<Table> list = new ArrayList<Table>();

try {

// 按照 SQL 语句创建 PreparedStatement 对象

PreparedStatement preparedStatement = DataOperation.connection.prepareStatement("select \* from `table`");

// 执行查询语句，并获取结果集

ResultSet resultSet = preparedStatement.executeQuery();

while (resultSet.next()) {

// 将每个结果集中的行数据创建为一个 Table 对象并添加到列表中

Table table = new Table(

resultSet.getInt(1),

resultSet.getString(2),

resultSet.getInt(3),

resultSet.getString(4));

list.add(table);

}

} catch (Exception e) {

DataOperation.showException(e);

}

return list;

}

// 向表中插入一个新行，并返回插入行的主键。

public static int add(Table table) {

try {

// 按照 SQL 语句创建 PreparedStatement 对象，同时要求返回插入行的主键

PreparedStatement preparedStatement = DataOperation.connection.prepareStatement(

"insert into `table`(`no`,seating,status) values(?,?,?)",

Statement.RETURN\_GENERATED\_KEYS);

// 设置 SQL 语句中的参数值

preparedStatement.setString(1, table.getNo());

preparedStatement.setInt(2, table.getSeating());

preparedStatement.setString(3, table.getStatus());

// 执行插入操作，并返回影响的行数

if (preparedStatement.executeUpdate() != 0) {

// 获取插入的数据的主键

ResultSet resultSet = preparedStatement.getGeneratedKeys();

if (resultSet.next())

return resultSet.getInt(1);

}

} catch (Exception e) {

DataOperation.showException(e);

}

return -1;

}

// 从表中删除指定的行。

public static boolean delete(int id) {

try {

// 按照 SQL 语句创建 PreparedStatement 对象

PreparedStatement preparedStatement = DataOperation.connection.prepareStatement(

"delete from `table` where id=?");

// 设置 SQL 语句中的参数值

preparedStatement.setInt(1, id);

// 执行删除操作，并返回影响的行数

return preparedStatement.executeUpdate() != 0;

} catch (Exception e) {

DataOperation.showException(e);

}

return false;

}

// 修改表中指定的行。

public static boolean change(Table table) {

try {

// 按照 SQL 语句创建 PreparedStatement 对象

PreparedStatement preparedStatement = DataOperation.connection.prepareStatement(

"update `table` set `no`=?,seating=?,status=? where id=?");

// 设置 SQL 语句中的参数值

preparedStatement.setString(1, table.getNo());

preparedStatement.setInt(2, table.getSeating());

preparedStatement.setString(3, table.getStatus());

preparedStatement.setInt(4, (int) table.getId());

// 执行更新操作，并返回影响的行数

return preparedStatement.executeUpdate() != 0;

} catch (Exception e) {

DataOperation.showException(e);

}

return false;

}

// 根据一个餐桌编号获取该餐桌在表中的主键。

public static int toTableId(String tableNo) {

try {

// 按照 SQL 语句创建 PreparedStatement 对象

PreparedStatement preparedStatement = DataOperation.connection

.prepareStatement("select id from `table` where `no`=?");

// 设置 SQL 语句中的参数值

preparedStatement.setString(1, tableNo);

// 执行查询操作，并获取结果集

ResultSet resultSet = preparedStatement.executeQuery();

while (resultSet.next()) {

// 返回结果集中 id 列的数值

return resultSet.getInt(1);

}

} catch (Exception e) {

DataOperation.showException(e);

}

return -1;

}

// 根据一个餐桌主键获取该餐桌的编号。

public static String toTableNo(int tableId) {

try {

// 按照 SQL 语句创建 PreparedStatement 对象

PreparedStatement preparedStatement = DataOperation.connection

.prepareStatement("select `no` from `table` where id=?");

// 设置 SQL 语句中的参数值

preparedStatement.setInt(1, tableId);

// 执行查询操作，并获取结果集

ResultSet resultSet = preparedStatement.executeQuery();

while (resultSet.next()) {

// 返回结果集中 no 列的字符串值

return resultSet.getString(1);

}

} catch (Exception e) {

DataOperation.showException(e);

}

return "???";

}

// 判断一个餐桌编号是否唯一，可传入一个主键排除自身

public static boolean isUnique(String no, int id) {

try {

// 按照 SQL 语句创建 PreparedStatement 对象

PreparedStatement preparedStatement = DataOperation.connection

.prepareStatement("select id from `table` where `no`=?");

// 设置 SQL 语句中的参数值

preparedStatement.setString(1, no);

// 执行查询操作，并获取结果集

ResultSet resultSet = preparedStatement.executeQuery();

while (resultSet.next()) {

// 如果存在其他记录的编号和该编号相同，则返回 false

if (id != resultSet.getInt(1))

return false;

}

} catch (Exception e) {

DataOperation.showException(e);

}

// 否则返回 true

return true;

}

// 判断一个餐桌编号是否唯一，不传入主键则认为自身是新记录。

public static boolean isUnique(String no) {

return isUnique(no, -1);

}

// 获取该餐桌最近一次客户的姓名和贵宾卡号，并返回字符串形式的结果。

public static String getDetails(int id) {

String customerName = null, customerCardId = null;

try {

// 按照 SQL 语句创建 PreparedStatement 对象

PreparedStatement preparedStatement = DataOperation.connection.prepareStatement(

"select name,cardId from customer where id in (select customerId from `order` where createtime in (select max(createtime) from `order` where tableId = ?) and tableId = ?)");

for (int i = 1; i <= 2; i++)

preparedStatement.setInt(i, id);

// 执行查询操作，并获取结果集

ResultSet resultSet = preparedStatement.executeQuery();

while (resultSet.next()) {

// 获取结果集中 name 列和 cardId 列的值

customerName = resultSet.getString(1);

customerCardId = resultSet.getString(2);

}

} catch (Exception e) {

DataOperation.showException(e);

}

// 如果没有获取到客户信息，返回默认字符串。

if (customerName == null)

return "无";

// 否则返回格式化后的字符串。

return "该餐台最近一次的就餐客户信息：\n\n客户姓名：\n" + customerName + "\n贵宾卡号：\n" + customerCardId;

}

}

**package** entity;

**public** **class** Customer {

**private** **int** id;

**private** String name, sex, company, tel, cardId;

**public** Customer(**int** id, String name, String sex, String company, String tel, String cardId) {

**this**.id = id;

**this**.name = name;

**this**.sex = sex;

**this**.company = company;

**this**.tel = tel;

**this**.cardId = cardId;

**if** (company != **null** && company.equals(""))

**this**.company = **null**;

**if** (tel != **null** && tel.equals(""))

**this**.tel = **null**;

}

**public** **int** getId() {

**return** id;

}

**public** String getName() {

**return** name;

}

**public** String getSex() {

**return** sex;

}

**public** String getCompany() {

**return** company;

}

**public** String getTel() {

**return** tel;

}

**public** String getCardId() {

**return** cardId;

}

}

**package** entity;

**public** **class** Dish {

**private** **int** id, dishTypeId;

**private** String name, code, unit, status;

**private** **double** price;

**public** Dish(**int** id, String name, **int** dishTypeId, String code, String unit, **double** price, String status) {

**this**.id = id;

**this**.dishTypeId = dishTypeId;

**this**.name = name;

**this**.code = code;

**this**.unit = unit;

**this**.status = status;

**this**.price = price;

**if** (unit != **null** && unit.equals(""))

**this**.unit = **null**;

}

**public** **int** getId() {

**return** id;

}

**public** **int** getDishTypeId() {

**return** dishTypeId;

}

**public** String getName() {

**return** name;

}

**public** String getCode() {

**return** code;

}

**public** String getUnit() {

**return** unit;

}

**public** String getStatus() {

**return** status;

}

**public** **double** getPrice() {

**return** price;

}

}

**package** entity;

**public** **class** DishType {

**int** id;

String name, describe;

**public** DishType(**int** id, String name, String describe) {

**this**.id = id;

**this**.name = name;

**this**.describe = describe;

**if** (describe != **null** && describe.equals(""))

**this**.describe = **null**;

}

**public** **int** getId() {

**return** id;

}

**public** String getName() {

**return** name;

}

**public** String getDescribe() {

**return** describe;

}

}

**package** entity;

**import** java.sql.Date;

**public** **class** Employee {

**private** **int** id;

**private** String name, sex, identityId, address, tel, position, freeze;

**private** Date birthday;

**public** Employee(**int** id, String name, String sex, Date birthday, String identityId, String address, String tel,

String position, String freeze) {

**this**.id = id;

**this**.name = name;

**this**.sex = sex;

**this**.identityId = identityId;

**this**.address = address;

**this**.tel = tel;

**this**.position = position;

**this**.freeze = freeze;

**this**.birthday = birthday;

**if** (identityId != **null** && identityId.equals(""))

**this**.identityId = **null**;

**if** (address != **null** && address.equals(""))

**this**.address = **null**;

**if** (tel != **null** && tel.equals(""))

**this**.tel = **null**;

}

**public** **int** getId() {

**return** id;

}

**public** String getName() {

**return** name;

}

**public** String getSex() {

**return** sex;

}

**public** String getIdentityId() {

**return** identityId;

}

**public** String getAddress() {

**return** address;

}

**public** String getTel() {

**return** tel;

}

**public** String getPosition() {

**return** position;

}

**public** String getFreeze() {

**return** freeze;

}

**public** Date getBirthday() {

**return** birthday;

}

}

package tool;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.SQLException;

import java.sql.Timestamp;

import java.text.DateFormat;

import java.text.SimpleDateFormat;

import java.util.Random;

import javax.swing.JOptionPane;

import javax.swing.JTextField;

/\*\*

\* 包含数据转换、数据库连接等各种数据操作方法。

\*/

public abstract class DataOperation {

public static Connection connection;

/\*\*

\* 清除文本域首尾的空格。

\*

\* @param textFields 文本域对象

\*/

public static void trimTextField(JTextField... textFields) {

for (JTextField textField : textFields) {

textField.setText(textField.getText().trim());

}

}

/\*\*

\* 将异常以错误信息框的形式显示出来。

\*

\* @param e 异常

\*/

public static void showException(Exception e) {

JOptionPane.showMessageDialog(null, e.getMessage(), "Exception", JOptionPane.ERROR\_MESSAGE);

e.printStackTrace();

}

/\*\*

\* 将字符串转换为SQL日期。

\*

\* @param str 字符串

\* @return SQL日期

\*/

public static java.sql.Date stringToDate(String str) {

DateFormat df = new SimpleDateFormat("yyyy-M-d");

java.util.Date date = null;

try {

date = df.parse(str);

} catch (Exception e) {

}

if (date == null)

return null;

return new java.sql.Date(date.getTime());

}

/\*\*

\* 将SQL日期转换为字符串数组，分别为年月日。

\*

\* @param date SQL日期

\* @return 日期字符串数组，若SQL日期为null，则为空字符串

\*/

public static String[] dateToString(java.sql.Date date) {

if (date == null)

return new String[] { "", "", "" };

java.util.Date utilDate = new java.util.Date(date.getTime());

return new String[] { new SimpleDateFormat("yyyy").format(utilDate),

new SimpleDateFormat("M").format(utilDate),

new SimpleDateFormat("d").format(utilDate) };

}

/\*\*

\* 将SQL时间戳转换为yyyy-MM-dd HH:mm:ss格式的字符串

\*

\* @param timestamp SQL时间戳

\* @return 日期字符串数组，若SQL日期为null，则为空字符串

\*/

public static String timestampToString(Timestamp timestamp) {

return new SimpleDateFormat("yyyy-MM-dd HH:mm:ss").format(new java.util.Date(timestamp.getTime()));

}

/\*\*

\* 生成订单编号

\*

\* @return 订单编号

\*/

public static String generateOrderNo() {

SimpleDateFormat df = new SimpleDateFormat("yyyyMMddHHmmss");

String timeStr = df.format(new java.util.Date());

Random random = new Random();

return timeStr + random.nextInt(10) + random.nextInt(10) + random.nextInt(10) + random.nextInt(10);

}

/\*\*

\* 获取SQL数据库连接。

\*

\* @param username SQL用户名

\* @param password SQL密码

\* @return SQL连接

\* @throws SQLException SQL异常

\*/

public static Connection getConnection(String username, String password) throws SQLException {

try {

Class.forName("com.mysql.cj.jdbc.Driver");

} catch (ClassNotFoundException e) {

e.printStackTrace();

}

connection = DriverManager.getConnection("jdbc:mysql://localhost:3306/restaurant\_management", username, password);

return connection;

}

}

package gui.panel;

import java.awt.BorderLayout;

import java.awt.Dimension;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.util.List;

import javax.swing.JButton;

import javax.swing.JOptionPane;

import javax.swing.JPanel;

import javax.swing.JScrollPane;

import javax.swing.JTable;

import javax.swing.JTextArea;

import javax.swing.event.ListSelectionEvent;

import javax.swing.event.ListSelectionListener;

import javax.swing.table.DefaultTableModel;

import dao.TableOperation;

import entity.Table;

import gui.dialog.OrderDialog;

import tool.Appearance;

public class TablePanel extends JPanel {

public JButton orderButton = new JButton("点菜");

private JTextArea infoArea = new JTextArea();

private JPanel borderPanel = new JPanel(new BorderLayout());

private DefaultTableModel model = new DefaultTableModel(new Object[][] {},

new String[] { "序号", "餐台编号", "座位数", "状态" }) {

@Override

public Class<?> getColumnClass(int column) { // 使表格模型能正确识别类型以便排序

Class<?> result = Object.class;

if (getRowCount() > 0) {

Object value = getValueAt(0, column);

if (value != null)

result = value.getClass();

}

return result;

}

@Override

public boolean isCellEditable(int row, int column) { // 使表格不可编辑

return false;

}

};

public JTable table = new JTable(model);

public TablePanel() {

setLayout(new BorderLayout());

Appearance.tableInit(table);

table.getSelectionModel().addListSelectionListener(selectAction);

orderButton.addActionListener(orderAction);

infoArea.setEditable(false);

infoArea.setLineWrap(true); // 让文本域自动换行

infoArea.setWrapStyleWord(true);

orderButton.setPreferredSize(new Dimension(128, 32));

List<Table> list = TableOperation.load();

for (int i = 0; i < list.size(); i++) {

Table table\_ = list.get(i);

model.addRow(new Object[] {

table\_.getId(),

table\_.getNo(),

table\_.getSeating(),

table\_.getStatus()

});

}

Appearance.fitTableColumns(table);

Appearance.setCellColor(table, "状态", "空闲");

selectAction.valueChanged(null);

borderPanel.add(orderButton, BorderLayout.EAST); // Change this line to move the button to the bottom

borderPanel.add(new JScrollPane(infoArea), BorderLayout.SOUTH);

add(borderPanel, BorderLayout.SOUTH);

add(new JScrollPane(table), BorderLayout.CENTER);

}

public ListSelectionListener selectAction = new ListSelectionListener() {

@Override

public void valueChanged(ListSelectionEvent e) {

int row = table.getSelectedRow();

if (row != -1) {

orderButton.setEnabled(table.getValueAt(row, 2).toString().equals("空闲"));

infoArea.setText(

TableOperation.getDetails((int) model.getValueAt(table.convertRowIndexToModel(row), 0)));

} else {

orderButton.setEnabled(false);

infoArea.setText("选取餐台以查看最近一次在该餐台就餐的客户。");

}

}

};

private ActionListener orderAction = new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

int row = table.convertRowIndexToModel(table.getSelectedRow());

if (TableOperation.change(new Table((int) model.getValueAt(row, 0), (String) model.getValueAt(row, 1),

(int) model.getValueAt(row, 2), "忙碌"))) {

model.setValueAt("忙碌", row, 3);

selectAction.valueChanged(null);

new OrderDialog(table);

} else

JOptionPane.showMessageDialog(null, "点菜失败！", "错误", JOptionPane.ERROR\_MESSAGE);

}

};

}