

超市进销存管理系统

一、项目架构介绍

1、后端技术栈：golang+gin框架+rpc远程调用服务+go-micro微服务框架+swagger在线接口文档+goland集成开发环境+mysql+redis+etcd注册中心+docker容器化部署+docker-compose容器管理+nginx反向代理

Golang

Golang (Go语言) 是一种现代的编程语言，具有简洁、高效、并发支持强等特点。Go语言天生支持并发编程，提供了轻量级的goroutine（协程）机制，可以在不同的goroutine之间高效地并发执行任务，而无需显式地管理线程。Go语言具有高效的垃圾回收（GC）机制，开发者无需手动管理内存，GC会自动处理不再使用的内存对象，确保内存的安全和高效利用。Go语言有强大的包管理工具（`go mod`），可以方便地管理项目依赖。Go标准库提供了丰富的功能模块，涵盖网络编程、文件操作、加密解密、并发控制等各个方面。

Gin框架

Gin 是一个用 Go 编写的高性能 Web 框架。它简洁、易用，适合用于开发高效的 RESTful API 和 Web 应用。Gin 是一个非常轻量级的框架，具有很高的性能，其速度是由基于 httprouter 的高效路由引擎保证的。Gin 提供了丰富的中间件支持，可以方便地实现日志记录、错误恢复、验证、跨域资源共享 (CORS) 等功能。内置了强大的 JSON 处理功能，能够方便地解析和生成 JSON 数据。

Goland集成开发环境

GoLand 是 JetBrains 公司推出的一款专门用于 Go 语言开发的集成开发环境 (IDE)。它基于 IntelliJ 平台，提供了丰富的功能来提高 Go 开发的效率和质量。内置对 Docker 和 Kubernetes 的支持，方便进行容器化应用开发和部署。

Mysql

MySQL 是一个开源的关系型数据库管理系统 (RDBMS)，广泛用于各种应用场景，包括Web应用、数据仓库和分布式系统。它支持多种存储引擎，如 InnoDB、MyISAM 等，用户可以根据需求选择合适的引擎。支持多个用户同时访问数据库。提供强大的访问控制和权限管理机制，保护数据的安全性。

Redis

Redis 是一个开源的内存数据结构存储系统，用作数据库、缓存和消息代理。它支持多种数据结构，如字符串、哈希、列表、集合和有序集合。由于 Redis 是内存数据库，读写操作非常快，适用于需要快速响应的应用。支持 RDB（快照）和 AOF（Append Only File）两种持久化机制，确保数据不会因为服务器宕机而丢失。Redis Cluster 提供了分片功能，使得数据可以分布在多个节点上，提供了水平扩展能力。

Docker容器化部署

Docker 容器化部署通过将应用程序及其依赖打包到一个独立的容器中，使得应用能够在不同的环境中一致地运行。通过 Dockerfile 自动化构建镜像过程，可以大大简化应用的部署和管理。结合 Docker Hub，可以方便地分发和共享容器镜像，进一步提升开发和运维效率。

Docker-compose容器管理

Docker-compose 是一个强大的工具，适用于管理和编排多个 Docker 容器应用的场景。它通过简单的 YAML 文件定义了容器化应用的组件和配置，使得开发、测试和部署容器化应用变得更加高效和可控。

2、前端技术栈：Vue3+Vite+element-plus+axios+node.js

Vue

Vue3 是一款现代化、高性能的前端框架，具备了响应式、组件化、虚拟 DOM、TypeScript 支持等特点，能够有效提升开发效率和应用性能，适用于各种规模的前端项目开发和维护。

Axios

Axios 是一个基于 Promise 的 HTTP 客户端，用于浏览器和 Node.js 环境中进行 HTTP 请求。Axios 使用 Promise 封装了 HTTP 请求过程，支持异步操作，可以更方便地处理请求和响应。Axios 提供了拦截器（interceptors）功能，可以在请求或响应被处理前拦截它们。Axios 默认情况下会自动将响应数据转换为 JSON 格式，这样可以方便地处理和操作返回的数据。Axios 可以通过设置请求头 `X-CSRF-TOKEN` 来防止 CSRF 攻击，这对于安全性要求较高的应用程序是一个重要的功能。

二、Gitee项目地址

<https://gitee.com/freeman7728/database-course-design>







easymoneysniper

@igxy0929

easymoneysniper 暂无简介

Java

Python

个人设置

0 Stars

3 Watches

0 Followers

0 Following

<https://github.com/Igxy0929>

概览

仓库

星选集

热门项目

自定义精选项目

微信小程序周报

Forked from freeman7728 / 微信小程序周报

这是一个微信小程序寒假项目周报

</> CSS

1 ☆ 0 23

数据库课设

这是我的数据库课程设计项目

</> Go

2 ☆ 0 0

贡献度

2024

	七月	八月	九月	十月	十一月	十二月	一月	二月	三月	四月	五月	六月
周一												
周四												
周日												

少多

最近一年贡献: 48 次

最长连续贡献: 8 日

最近连续贡献: 8 日

贡献度的统计数据包括代码提交、创建任务 / Pull Request、合并 Pull Request，其中代码提交的次数需本地配置的 git 邮箱是 Gitee 帐号已确认绑定的才会被统计。

三、软件设计

1.需求分析

本项目旨在实现一个实际生产环境下的超市进货与员工管理系统

权限管理:

管理员除了不能增删管理员和超级管理员之外，拥有其他所有权限

普通员工拥有订单以及订单详情的增删改查权限，拥有供应商的查看权限

员工管理：导入员工，管理员工权限，管理员工信息，验证员工邮箱等

供应商管理：导入供应商，管理供应商信息等

商品管理：导入商品，管理商品信息等

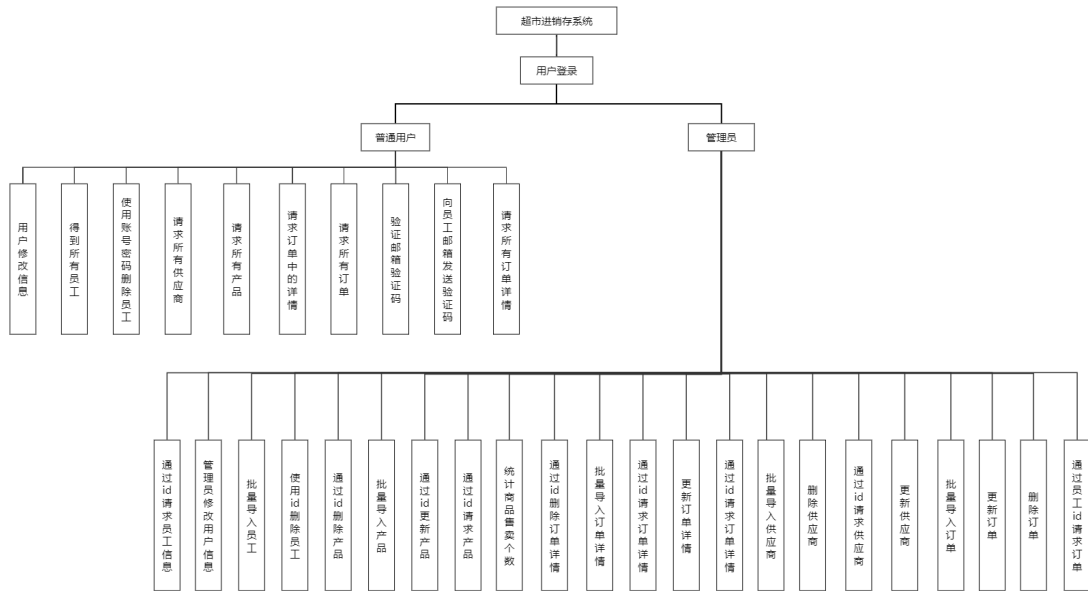
订单管理：导入订单，管理订单信息，自动通过订单详情生成订单信息等

订单详情管理：导入订单详情，管理订单详情等

2.软件功能设计

普通用户的功能收到限制，只能修改自己的信息和查询信息

管理员用户的功能在普通用户的基础之上，可以对员工，商品，订单等实体进行信息的增加，删除，修改



四、数据库设计

1.需求分析

员工 (employee 表):

- 属性:
 - id (主键)：每个员工的唯一标识符。
 - name：员工姓名。
 - tel：员工电话号码。
 - salary：员工薪水。
 - note：员工相关的附加注释。
 - deleted_at：员工被删除的时间戳（软删除）。
 - level：员工级别或职位。
 - email：员工电子邮件地址。
 - email_is_auth：二进制标志，指示员工的电子邮件是否已验证。

账户 (account 表):

- 属性:
 - id (主键)：与员工关联的唯一标识符 (employee.id)。
 - account：唯一的账户标识符。
 - password：账户关联的密码。

清单 (list 表):

- 属性:
 - id (主键)：每个清单的唯一标识符。
 - quantity：清单中所有物品的总数量。
 - total_price：清单中所有物品的总价。
 - time：清单创建的时间戳。
 - note：清单的附加注释。
 - employee_id：创建该清单的员工的外键 (employee.id)。

明细 (detail 表):

- 属性:
 - `id` (主键) : 每个明细条目的唯一标识符。
 - `product_id` : 与产品关联的外键 (`product.id`)。
 - `quantity` : 该明细条目中产品的数量。
 - `total_price` : 计算出的该明细条目的总价。
 - `note` : 该明细条目的附加注释。
 - `list_id` : 关联到所属清单的外键 (`list.id`)。

生产商 (producer 表):

- 属性:
 - `id` (主键) : 每个生产商的唯一标识符。
 - `name` : 生产商名称。
 - `short_name` : 生产商的简称或缩写。
 - `address` : 生产商地址。
 - `tel` : 生产商电话号码。
 - `email` : 生产商电子邮件地址。
 - `contact_name` : 生产商联系人姓名。
 - `contact_tel` : 生产商联系人电话号码。
 - `note` : 生产商的附加注释。

产品 (product 表):

- 属性:
 - `id` (主键) : 每个产品的唯一标识符。
 - `name` : 产品名称。
 - `price` : 产品价格。
 - `introduction` : 产品介绍或描述。
 - `note` : 产品的附加注释。
 - `producer_id` : 生产该产品的生产商的外键 (`producer.id`)。

1. 触发器 `before_insert_detail` (在插入明细前执行)

- **功能:** 在向 `detail` 表插入新记录之前, 计算新记录的 `total_price`。
- 详细描述:
 - 使用 `BEFORE INSERT` 触发器类型, 每次插入新行时都会执行。
 - 从 `product` 表中查询对应产品的单价 (`price`)。
 - 计算新行的 `total_price`, 即 `quantity * product_price`。
 - 将计算得到的 `total_price` 设置给 `NEW.total_price`, 其中 `NEW` 表示即将插入的新行数据。

2. 触发器 `after_insert_detail` (在插入明细后执行)

- **功能:** 在向 `detail` 表成功插入新记录后, 更新相应的 `list` 表中的 `quantity` 和 `total_price`。
- 详细描述:
 - 使用 `AFTER INSERT` 触发器类型, 确保在新行插入完成后执行。
 - 更新与新行关联的 `list` 表记录。

- 计算更新后的 `list` 表中特定 `list_id` 的 `quantity` 和 `total_price`，分别为该列表中所有明细的总数量和总价。

3. 触发器 `after_update_detail` (在更新明细后执行)

- **功能:** 在 `detail` 表中的记录更新后，同步更新相应的 `list` 表中的 `quantity` 和 `total_price`。
- 详细描述:
 - 使用 `AFTER UPDATE` 触发器类型，确保在更新操作完成后执行。
 - 更新与更新行关联的 `list` 表记录。
 - 计算更新后的 `list` 表中特定 `list_id` 的 `quantity` 和 `total_price`，以反映所有明细的最新总数量和总价。

4. 触发器 `after_delete_detail` (在删除明细后执行)

- **功能:** 在 `detail` 表中的记录删除后，同步更新相应的 `list` 表中的 `quantity` 和 `total_price`。
- 详细描述:
 - 使用 `AFTER DELETE` 触发器类型，确保在删除操作完成后执行。
 - 更新与被删除行关联的 `list` 表记录。
 - 计算更新后的 `list` 表中特定 `list_id` 的 `quantity` 和 `total_price`，以反映所有明细的最新总数量和总价。

5. 触发器 `check_level_before_insert` (在插入员工前检查级别)

- **功能:** 在向 `employee` 表插入新员工记录之前，检查并调整员工的级别。
- 详细描述:
 - 使用 `BEFORE INSERT` 触发器类型，每次插入新员工记录时执行。
 - 如果新员工的级别 (`NEW.level`) 大于等于 3，则将其级别设为 2。

6. 触发器 `set_email_auth` (在插入员工前设置邮箱验证状态)

- **功能:** 在向 `employee` 表插入新员工记录之前，设置员工的邮箱验证状态为未验证 (`FALSE`)。
- 详细描述:
 - 使用 `BEFORE INSERT` 触发器类型，每次插入新员工记录时执行。
 - 将新员工的 `email_is_auth` 字段设置为 `FALSE`，表示邮箱未经验证。

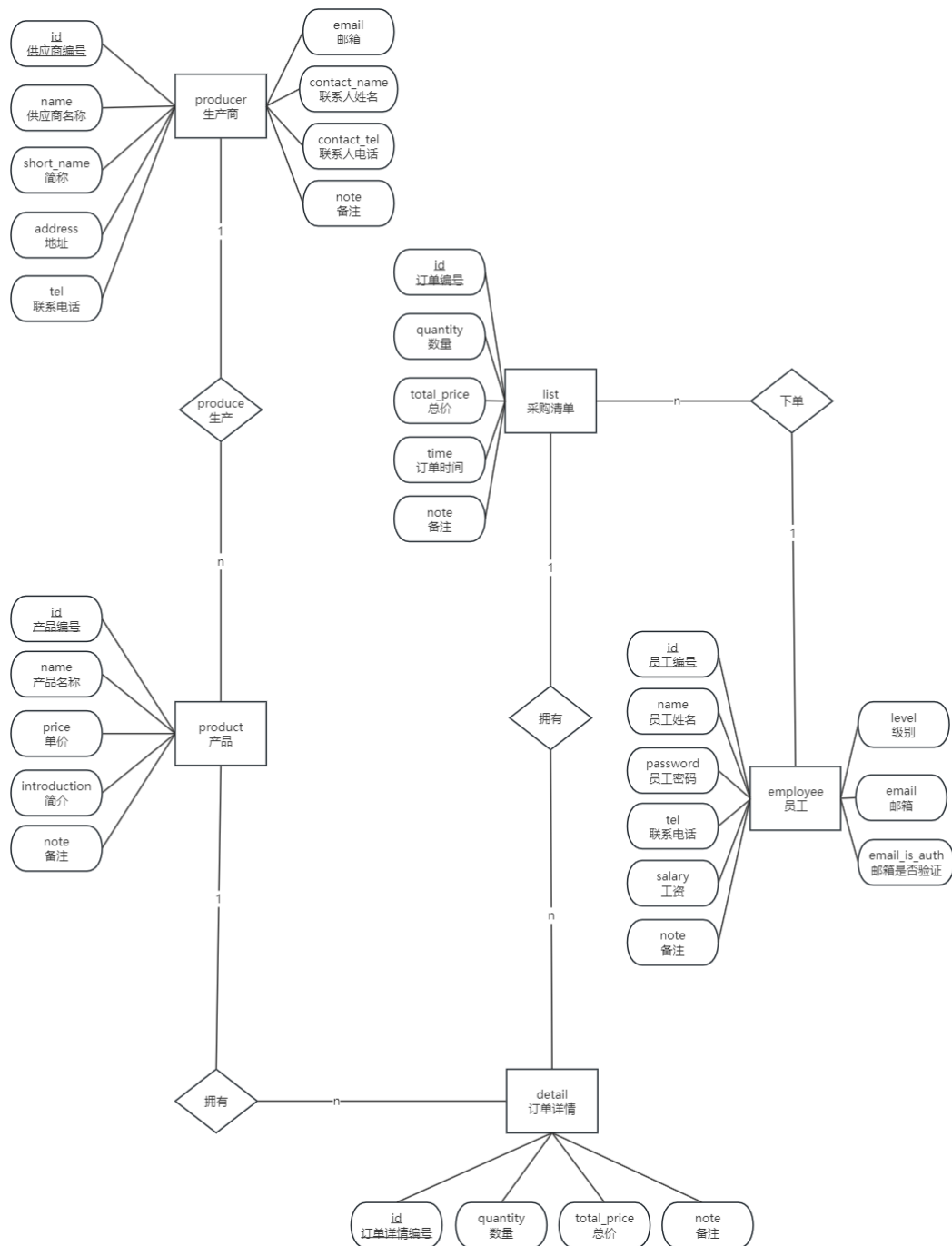
7. 触发器 `check_level_before_update` (在更新员工前检查级别)

- **功能:** 在更新 `employee` 表中的员工记录之前，检查并调整员工的级别。
- 详细描述:
 - 使用 `BEFORE UPDATE` 触发器类型，每次更新员工记录时执行。
 - 如果更新后员工的级别 (`NEW.level`) 大于等于 3，则将其级别设为 2。

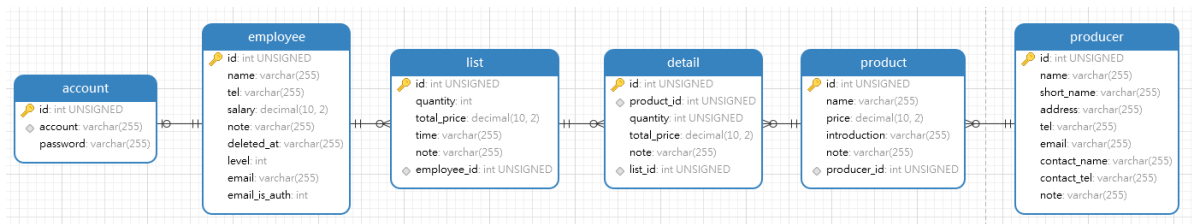
8. 触发器 `reset_auth` (在更新员工前重置邮箱验证状态)

- **功能:** 在更新 `employee` 表中的员工记录之前, 如果员工的邮箱地址发生更改, 则重置员工的邮箱验证状态。
- 详细描述:
 - 使用 `BEFORE UPDATE` 触发器类型, 每次更新员工记录时执行。
 - 如果更新前后员工的 `email` 字段不同 (`NEW.email != OLD.email`), 则将员工的 `email_is_auth` 字段设置为 `FALSE`, 表示需要重新验证新的邮箱地址。

2. 概念结构设计



3.逻辑结构设计



1. product 表

- **字段:** id, name, price
- **分析:**
 - **第一范式 (1NF):** 已满足。每个字段具有原子性，没有重复组。
 - **第二范式 (2NF):** 已满足。每个非主键字段完全依赖于候选键 (id)，不存在部分依赖。
 - **第三范式 (3NF):** 已满足。不存在传递依赖，即非主键字段之间不存在依赖关系。

结论: product 表符合第一、第二和第三范式。

2. list 表

- **字段:** id, name, quantity, total_price
- **分析:**
 - **第一范式 (1NF):** 已满足。每个字段具有原子性。
 - **第二范式 (2NF):** 已满足。每个非主键字段完全依赖于候选键 (id)。
 - **第三范式 (3NF):** 已满足。不存在传递依赖。

结论: list 表符合第一、第二和第三范式。

3. detail 表

- **字段:** id, list_id, product_id, quantity, total_price
- **分析:**
 - **第一范式 (1NF):** 已满足。每个字段具有原子性。
 - **第二范式 (2NF):** 已满足。每个非主键字段完全依赖于候选键 (id, list_id, product_id)。
 - **第三范式 (3NF):** 部分满足。存在 quantity 和 total_price 到 list 表的传递依赖，但通过触发器保证了数据一致性。

结论: detail 表在设计上基本符合第一和第二范式，但在第三范式上存在一定程度的违反，需要依赖触发器来维护数据一致性。

4. employee 表

- **字段:** id, name, email, level, email_is_auth
- **分析:**
 - **第一范式 (1NF):** 已满足。每个字段具有原子性。
 - **第二范式 (2NF):** 已满足。每个非主键字段完全依赖于候选键 (id, email)。
 - **第三范式 (3NF):** 部分满足。email_is_auth 字段依赖于 email 字段，如果 email 变化，email_is_auth 可能需要更新。

结论: `employee` 表在设计上基本符合第一和第二范式，但在第三范式上也存在一定程度的违反，需要依赖触发器来维护 `email_is_auth` 字段的一致性。

5. `producer` 表

- **字段:** `id`, `name`, `country`
- 分析:
 - **第一范式 (1NF):** 已满足。每个字段具有原子性。
 - **第二范式 (2NF):** 已满足。每个非主键字段完全依赖于候选键 (`id`)。
 - **第三范式 (3NF):** 已满足。不存在非主键字段对主键的传递依赖。

结论: `producer` 表符合第一、第二和第三范式。

6. `account` 表

- **字段:** `id`, `username`, `password`, `email`, `created_at`, `updated_at`
- 分析:
 - **第一范式 (1NF):** 已满足。每个字段具有原子性。
 - **第二范式 (2NF):** 已满足。表中不存在部分依赖，每个非主键字段完全依赖于候选键 (`id`)。
 - **第三范式 (3NF):** 已满足。不存在非主键字段对主键的传递依赖。

结论: `account` 表符合第一、第二和第三范式。

4.详细表设计

Table: account

字段名	数据类型	是否为空	默认值	备注
id	int UNSIGNED	NOT NULL		员工id, 主键, 外键
account	varchar(255)	NOT NULL		员工账号
password	varchar(255)	NOT NULL		员工密码

Table: detail

字段名	数据类型	是否为空	默认值	备注
id	int UNSIGNED	NOT NULL		订单详情ID, 主键, 自增
product_id	int UNSIGNED	NOT NULL		产品ID, 外键
quantity	int UNSIGNED	NULL	NULL	数量
total_price	decimal(10, 2)	NULL	NULL	总价格
note	varchar(255)	NULL	NULL	备注
list_id	int UNSIGNED	NOT NULL		列表ID, 外键

Table: employee

字段名	数据类型	是否为空	默认值	备注
id	int UNSIGNED	NOT NULL		员工ID，主键，自增
name	varchar(255)	NOT NULL		姓名
tel	varchar(255)	NOT NULL		手机号
salary	decimal(10, 2)	NULL	NULL	薪资
note	varchar(255)	NULL	NULL	备注
deleted_at	varchar(255)	NULL	NULL	删除时间
level	int	NOT NULL		级别
email	varchar(255)	NULL	NULL	电子邮件
email_is_auth	binary(1)	NULL	NULL	电子邮件认证

Table: list

字段名	数据类型	是否为空	默认值	备注
id	int UNSIGNED	NOT NULL		列表ID，主键，自增
quantity	int	NULL	NULL	数量
total_price	decimal(10, 2)	NULL	NULL	总价格
time	varchar(255)	NULL	NULL	时间
note	varchar(255)	NULL	NULL	备注
employee_id	int UNSIGNED	NOT NULL		员工ID，外键

Table: producer

字段名	数据类型	是否为空	默认值	备注
id	int UNSIGNED	NOT NULL		生产商ID, 主键, 自增
name	varchar(255)	NOT NULL		名称
short_name	varchar(255)	NULL	NULL	简称
address	varchar(255)	NULL	NULL	地址
tel	varchar(255)	NULL	NULL	手机号
email	varchar(255)	NULL	NULL	电子邮件
contact_name	varchar(255)	NULL	NULL	联系人姓名
contact_tel	varchar(255)	NULL	NULL	联系人电话
note	varchar(255)	NULL	NULL	备注

Table: product

字段名	数据类型	是否为空	默认值	备注
id	int UNSIGNED	NOT NULL		产品ID, 主键, 自增
name	varchar(255)	NOT NULL		名称
price	decimal(10, 2)	NOT NULL		价格
introduction	varchar(255)	NULL	NULL	介绍
note	varchar(255)	NULL	NULL	备注
producer_id	int UNSIGNED	NOT NULL		生产商ID, 外键

5.SQL脚本

```
/*
Navicat Premium Data Transfer

Source Server        : local
Source Server Type   : MySQL
Source Server Version : 80033 (8.0.33)
Source Host          : localhost:3306
Source Schema        : database_lesson

Target Server Type    : MySQL
```

Target Server Version : 80033 (8.0.33)
File Encoding : 65001

Date: 23/06/2024 20:37:09

*/

SET NAMES utf8mb4;

SET FOREIGN_KEY_CHECKS = 0;

-- Table structure for account

```
DROP TABLE IF EXISTS `account`;
CREATE TABLE `account` (
  `id` int UNSIGNED NOT NULL,
  `account` varchar(255) CHARACTER SET utf8mb4 COLLATE utf8mb4_0900_ai_ci NOT NULL,
  `password` varchar(255) CHARACTER SET utf8mb4 COLLATE utf8mb4_0900_ai_ci NOT NULL,
  PRIMARY KEY (`id`) USING BTREE,
  UNIQUE INDEX `unique_account` (`account` ASC) USING BTREE,
  CONSTRAINT `fk_account_employee_1` FOREIGN KEY (`id`) REFERENCES `employee` (`id`) ON DELETE CASCADE ON UPDATE RESTRICT
) ENGINE = InnoDB CHARACTER SET = utf8mb4 COLLATE = utf8mb4_0900_ai_ci
ROW_FORMAT = Dynamic;
```

-- Table structure for detail

```
DROP TABLE IF EXISTS `detail`;
CREATE TABLE `detail` (
  `id` int UNSIGNED NOT NULL AUTO_INCREMENT,
  `product_id` int UNSIGNED NOT NULL,
  `quantity` int UNSIGNED NULL DEFAULT NULL,
  `total_price` decimal(10, 2) NULL DEFAULT NULL,
  `note` varchar(255) CHARACTER SET utf8mb4 COLLATE utf8mb4_0900_ai_ci NULL DEFAULT NULL,
  `list_id` int UNSIGNED NOT NULL,
  PRIMARY KEY (`id`) USING BTREE,
  INDEX `fk_detail_product_1` (`product_id` ASC) USING BTREE,
  INDEX `fk_detail_list_1` (`list_id` ASC) USING BTREE,
  CONSTRAINT `fk_detail_list_1` FOREIGN KEY (`list_id`) REFERENCES `list` (`id`) ON DELETE RESTRICT ON UPDATE RESTRICT,
  CONSTRAINT `fk_detail_product_1` FOREIGN KEY (`product_id`) REFERENCES `product` (`id`) ON DELETE RESTRICT ON UPDATE RESTRICT
) ENGINE = InnoDB AUTO_INCREMENT = 26 CHARACTER SET = utf8mb4 COLLATE = utf8mb4_0900_ai_ci
ROW_FORMAT = Dynamic;
```

-- Table structure for employee

```
DROP TABLE IF EXISTS `employee`;
CREATE TABLE `employee` (
  `id` int UNSIGNED NOT NULL AUTO_INCREMENT,
  `name` varchar(255) CHARACTER SET utf8mb4 COLLATE utf8mb4_0900_ai_ci NOT NULL,
```

```

`tel` varchar(255) CHARACTER SET utf8mb4 COLLATE utf8mb4_0900_ai_ci NOT NULL,
`salary` decimal(10, 2) NULL DEFAULT NULL,
`note` varchar(255) CHARACTER SET utf8mb4 COLLATE utf8mb4_0900_ai_ci NULL
DEFAULT NULL,
`deleted_at` varchar(255) CHARACTER SET utf8mb4 COLLATE utf8mb4_0900_ai_ci
NULL DEFAULT NULL,
`level` int NOT NULL,
`email` varchar(255) CHARACTER SET utf8mb4 COLLATE utf8mb4_0900_ai_ci NULL
DEFAULT NULL,
`email_is_auth` int NULL DEFAULT NULL,
PRIMARY KEY (`id`) USING BTREE
) ENGINE = InnoDB AUTO_INCREMENT = 37 CHARACTER SET = utf8mb4 COLLATE =
utf8mb4_0900_ai_ci ROW_FORMAT = Dynamic;

```

```

-- -----
-- Table structure for list
-- -----

```

```

DROP TABLE IF EXISTS `list`;
CREATE TABLE `list` (
  `id` int UNSIGNED NOT NULL AUTO_INCREMENT,
  `quantity` int NULL DEFAULT NULL,
  `total_price` decimal(10, 2) NULL DEFAULT NULL,
  `time` varchar(255) CHARACTER SET utf8mb4 COLLATE utf8mb4_0900_ai_ci NULL
  DEFAULT NULL,
  `note` varchar(255) CHARACTER SET utf8mb4 COLLATE utf8mb4_0900_ai_ci NULL
  DEFAULT NULL,
  `employee_id` int UNSIGNED NOT NULL,
  PRIMARY KEY (`id`) USING BTREE,
  INDEX `fk_list_employee_1` (`employee_id` ASC) USING BTREE,
  CONSTRAINT `fk_list_employee_1` FOREIGN KEY (`employee_id`) REFERENCES
  `employee` (`id`) ON DELETE RESTRICT ON UPDATE RESTRICT
) ENGINE = InnoDB AUTO_INCREMENT = 15 CHARACTER SET = utf8mb4 COLLATE =
utf8mb4_0900_ai_ci ROW_FORMAT = Dynamic;

```

```

-- -----
-- Table structure for producer
-- -----

```

```

DROP TABLE IF EXISTS `producer`;
CREATE TABLE `producer` (
  `id` int UNSIGNED NOT NULL AUTO_INCREMENT,
  `name` varchar(255) CHARACTER SET utf8mb4 COLLATE utf8mb4_0900_ai_ci NOT NULL,
  `short_name` varchar(255) CHARACTER SET utf8mb4 COLLATE utf8mb4_0900_ai_ci
  NULL DEFAULT NULL,
  `address` varchar(255) CHARACTER SET utf8mb4 COLLATE utf8mb4_0900_ai_ci NULL
  DEFAULT NULL,
  `tel` varchar(255) CHARACTER SET utf8mb4 COLLATE utf8mb4_0900_ai_ci NULL
  DEFAULT NULL,
  `email` varchar(255) CHARACTER SET utf8mb4 COLLATE utf8mb4_0900_ai_ci NULL
  DEFAULT NULL,
  `contact_name` varchar(255) CHARACTER SET utf8mb4 COLLATE utf8mb4_0900_ai_ci
  NULL DEFAULT NULL,
  `contact_tel` varchar(255) CHARACTER SET utf8mb4 COLLATE utf8mb4_0900_ai_ci
  NULL DEFAULT NULL,
  `note` varchar(255) CHARACTER SET utf8mb4 COLLATE utf8mb4_0900_ai_ci NULL
  DEFAULT NULL,

```

```

PRIMARY KEY (`id`) USING BTREE
) ENGINE = InnoDB AUTO_INCREMENT = 19 CHARACTER SET = utf8mb4 COLLATE =
utf8mb4_0900_ai_ci ROW_FORMAT = Dynamic;

-- -----
-- Table structure for product
-- -----
DROP TABLE IF EXISTS `product`;
CREATE TABLE `product` (
  `id` int UNSIGNED NOT NULL AUTO_INCREMENT,
  `name` varchar(255) CHARACTER SET utf8mb4 COLLATE utf8mb4_0900_ai_ci NOT NULL,
  `price` decimal(10, 2) NOT NULL,
  `introduction` varchar(255) CHARACTER SET utf8mb4 COLLATE utf8mb4_0900_ai_ci
  NULL DEFAULT NULL,
  `note` varchar(255) CHARACTER SET utf8mb4 COLLATE utf8mb4_0900_ai_ci NULL
  DEFAULT NULL,
  `producer_id` int UNSIGNED NOT NULL,
  PRIMARY KEY (`id`) USING BTREE,
  INDEX `fk_product_producer_1` (`producer_id` ASC) USING BTREE,
  CONSTRAINT `fk_product_producer_1` FOREIGN KEY (`producer_id`) REFERENCES
  `producer` (`id`) ON DELETE RESTRICT ON UPDATE RESTRICT
) ENGINE = InnoDB AUTO_INCREMENT = 26 CHARACTER SET = utf8mb4 COLLATE =
utf8mb4_0900_ai_ci ROW_FORMAT = Dynamic;

-- -----
-- Procedure structure for CreateEmployeeWithAccount
-- -----
DROP PROCEDURE IF EXISTS `CreateEmployeeWithAccount`;
delimiter ;;
CREATE PROCEDURE `CreateEmployeeWithAccount`(IN p_name VARCHAR(255),
  IN p_tel VARCHAR(255),
  IN p_password VARCHAR(255),
  IN p_salary DECIMAL,
  IN p_note VARCHAR(255),
  IN p_email VARCHAR(255),
  IN p_level INT,
  OUT p_account VARCHAR(255))
BEGIN
  DECLARE last_id INT;
  DECLARE current_year VARCHAR(4);
  DECLARE current_month VARCHAR(2);
  DECLARE id_suffix VARCHAR(4);

  -- 插入新员工记录
  INSERT INTO employee (name, tel, salary, note, level, email)
  VALUES (p_name, p_tel, p_salary, p_note, p_level, p_email);

  -- 获取最后插入的ID
  SET last_id = LAST_INSERT_ID();

  -- 获取当前年份和月份
  SET current_year = YEAR(CURDATE());
  SET current_month = LPAD(MONTH(CURDATE()), 2, '0');

  -- 获取ID后四位

```

```

SET id_suffix = LPAD(last_id % 10000, 4, '0');

-- 生成员工账号
SET p_account = CONCAT(current_year, current_month, id_suffix);

-- 插入账号记录到account表
INSERT INTO account (account, password, id)
VALUES (p_account, p_password, last_id);
END
;;
delimiter ;

-- -----
-- Triggers structure for table detail
-- -----

DROP TRIGGER IF EXISTS `before_insert_detail`;
delimiter ;;
CREATE TRIGGER `before_insert_detail` BEFORE INSERT ON `detail` FOR EACH ROW
BEGIN
    DECLARE product_price DECIMAL(10, 2);

    -- 查询产品的单价
    SELECT price INTO product_price FROM `product` WHERE id = NEW.product_id;

    -- 计算总价
    SET NEW.total_price = NEW.quantity * product_price;
END
;;
delimiter ;

-- -----
-- Triggers structure for table detail
-- -----

DROP TRIGGER IF EXISTS `after_insert_detail`;
delimiter ;;
CREATE TRIGGER `after_insert_detail` AFTER INSERT ON `detail` FOR EACH ROW BEGIN
    UPDATE `list`
    SET `quantity` = (SELECT SUM(`quantity`) FROM `detail` WHERE `list_id` =
NEW.list_id),
        `total_price` = (SELECT SUM(`total_price`) FROM `detail` WHERE `list_id`
= NEW.list_id)
    WHERE `id` = NEW.list_id;
END
;;
delimiter ;

-- -----
-- Triggers structure for table detail
-- -----

DROP TRIGGER IF EXISTS `after_update_detail`;
delimiter ;;
CREATE TRIGGER `after_update_detail` AFTER UPDATE ON `detail` FOR EACH ROW BEGIN
    UPDATE `list`
    SET `quantity` = (SELECT SUM(`quantity`) FROM `detail` WHERE `list_id` =
NEW.list_id),

```

```

        `total_price` = (SELECT SUM(`total_price`) FROM `detail` WHERE `list_id`
= NEW.list_id)
        WHERE `id` = NEW.list_id;
END
;;
delimiter ;

-- -----
-- Triggers structure for table detail
-- -----

DROP TRIGGER IF EXISTS `after_delete_detail`;
delimiter ;;
CREATE TRIGGER `after_delete_detail` AFTER DELETE ON `detail` FOR EACH ROW BEGIN
    UPDATE `list`
    SET `quantity` = (SELECT SUM(`quantity`) FROM `detail` WHERE `list_id` =
OLD.list_id),
        `total_price` = (SELECT SUM(`total_price`) FROM `detail` WHERE `list_id`
= OLD.list_id)
        WHERE `id` = OLD.list_id;
END
;;
delimiter ;

-- -----
-- Triggers structure for table employee
-- -----

DROP TRIGGER IF EXISTS `check_level_before_insert`;
delimiter ;;
CREATE TRIGGER `check_level_before_insert` BEFORE INSERT ON `employee` FOR EACH
ROW BEGIN
    IF NEW.level >= 3 THEN
        SET NEW.level = 2;
    END IF;
END
;;
delimiter ;

-- -----
-- Triggers structure for table employee
-- -----

DROP TRIGGER IF EXISTS `set_email_auth`;
delimiter ;;
CREATE TRIGGER `set_email_auth` BEFORE INSERT ON `employee` FOR EACH ROW BEGIN
    SET NEW.email_is_auth = FALSE;
END
;;
delimiter ;

-- -----
-- Triggers structure for table employee
-- -----

DROP TRIGGER IF EXISTS `check_level_before_update`;
delimiter ;;
CREATE TRIGGER `check_level_before_update` BEFORE UPDATE ON `employee` FOR EACH
ROW BEGIN

```



```

IF NEW.level >= 3 THEN
    SET NEW.level = 2;
END IF;
END
;;
delimiter ;

-----
-- Triggers structure for table employee
-----

DROP TRIGGER IF EXISTS `reset_auth`;
delimiter ;;
CREATE TRIGGER `reset_auth` BEFORE UPDATE ON `employee` FOR EACH ROW BEGIN
    IF NEW.email != OLD.email THEN
        SET NEW.email_is_auth = 0;
    END IF;
END
;;
delimiter ;

SET FOREIGN_KEY_CHECKS = 1;

```

五、软件实现

1.界面设计与实现

个人详情页面



该界面可以使用用户修改，注销账户，验证邮箱功能

用户修改功能：通过验证用户的账号密码，修改用户的账号信息

注销账户功能：通过验证用户的账号密码，注销用户的账号

验证邮箱功能：向用户的邮箱发送验证码，用户输入验证码后激活邮箱

员工管理页面



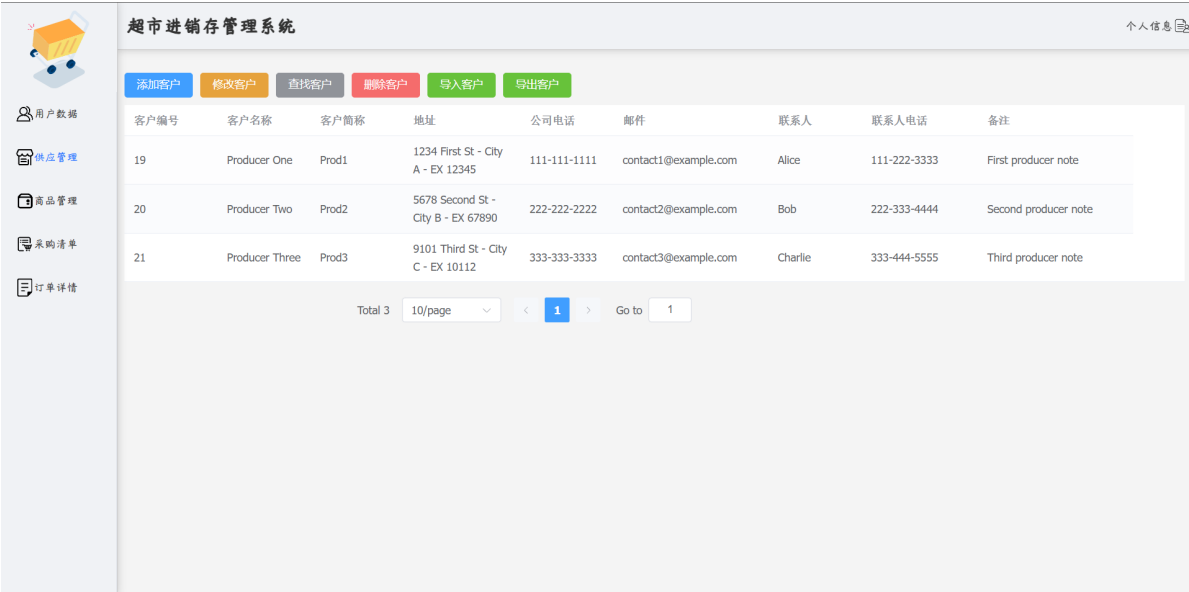
该界面实现了用户的增删改查，以及csv文件的导入导出

插入用户功能：输入用户信息，批量插入用户

删查改用户功能：输入用户id，对应的用户进行操作

csv导入导出功能：使用csv文件导出用户，或者导入用户

供应商管理页面



该界面实现了供应商的增删改查，以及csv文件的导入导出

插入供应商功能：输入供应商信息，批量插入供应商

删查改供应商功能：输入供应商id，对应的供应商进行操作

csv导入导出功能：使用csv文件导出供应商，或者导入供应商

商品管理页面

用户数据

供应管理

商品管理

采购清单

订单详情

超市进销存管理系统

个人信息

添加商品

修改商品

查找商品

删除商品

导入商品

导出商品

统计商品数量

商品编号	商品名称	商品单价	供应商编号	商品简介	备注
26	Product 1	99.99	19	Introduction for Product 1	Note for Product 1
27	Product 2	49.99	19	Introduction for Product 2	Note for Product 2
28	Product 3	199.99	19	Introduction for Product 3	Note for Product 3
29	Product 4	79.99	19	Introduction for Product 4	Note for Product 4
30	Product 5	129.99	21	Introduction for Product 5	Note for Product 5
31	Product 6	149.99	21	Introduction for Product 6	Note for Product 6
32	Product 7	199.99	21	Introduction for Product 7	Note for Product 7
33	Product 8	39.99	20	Introduction for Product 8	Note for Product 8
34	Product 9	299.99	20	Introduction for Product 9	Note for Product 9
35	Product 10	89.99	20	Introduction for Product 10	Note for Product 10

Total 1010/page1Go to1

该界面实现了商品的增删改查，以及csv文件的导入导出

插入商品功能：输入商品信息，批量插入商品

删查改商品功能：输入商品id，对应的商品进行操作

csv导入导出功能：使用csv文件导出商品，或者导入商品

统计商品数量：统计各个商品总共卖出去的件数

采购管理页面

用户数据

供应管理

清单管理

采购清单

订单详情

超市进销存管理系统

个人信息

添加清单

修改清单

查找清单

删除清单

导入清单

导出清单

清单号	员工编号	采购数量	采购总价	采购时间	备注	操作
15	49	85	7749.15	2024-06-23 10:00:00	First list item	查看清单详情
16	49	100	9999	2024-06-23 11:00:00	Second list item	查看清单详情
17	49	0	0	2024-06-23 12:00:00	Third list item	查看清单详情
18	50	0	0	2024-06-23 13:00:00	Fourth list item	查看清单详情
19	51	75	9749.25	2024-06-23 14:00:00	Fifth list item	查看清单详情

Total 510/page1Go to1

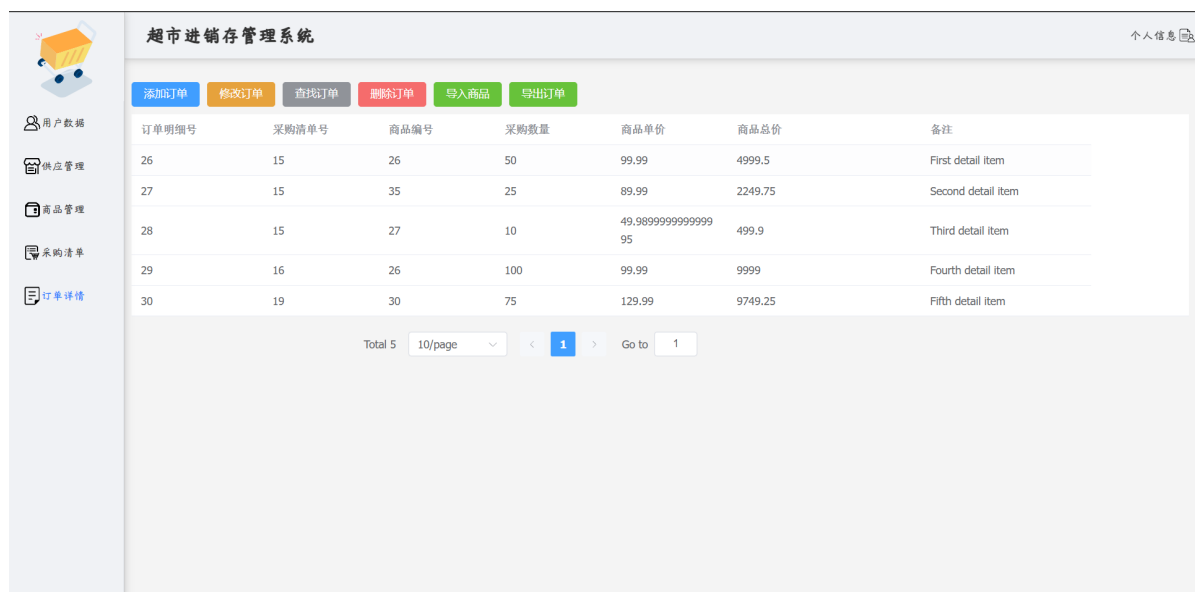
该界面实现了采购清单的增删改查，以及csv文件的导入导出

插入采购清单功能：输入采购清单信息，批量插入采购清单

删查改采购清单功能：输入采购清单id，对应的采购清单进行操作

csv导入导出功能：使用csv文件导出采购清单，或者导入采购清单

订单详情管理页面



订单明细号	采购清单号	商品编号	采购数量	商品单价	商品总价	备注
26	15	26	50	99.99	4999.5	First detail item
27	15	35	25	89.99	2249.75	Second detail item
28	15	27	10	49.989999999999995	499.9	Third detail item
29	16	26	100	99.99	9999	Fourth detail item
30	19	30	75	129.99	9749.25	Fifth detail item

该界面实现了订单详情的增删改查，以及csv文件的导入导出

插入订单详情功能：输入订单详情信息，批量插入订单详情

删查改订单详情功能：输入订单详情id，对应的订单详情进行操作

csv导入导出功能：使用csv文件导出订单详情，或者导入订单详情

2.解决的技术问题

后端与前端进行数据交互时，应使用对象封装数据，方便未来数据扩展

Gorm查询数据库时，用来接受返回值的实体对象的属性名称应与数据库字段名保持一致，忽略大小写

Cors跨域中间件不能使用"*"通配符来允许所有的网络请求，应指定具体ip与端口

服务器使用smtp协议发送邮件时，需要忽略SSL认证

使用docker-compose部署应用时，应使用自定义网桥实现容器间的网络通信

六、总结

经过这次数据库课程设计，学习了Go语言的Gin后端框架，Go-micro微服务框架，RPC远程服务调用等后端技术栈，对Mysql,Etcd,Redis数据库有了更深入的认识与理解，掌握了Docker,Docker-compose,Nginx等运维技术栈，对面向对象编程，面向接口编程有了更全面的理解与实践，深刻体会到软件设计对高内聚，低耦合的要求。

参考文献：

[1] 《Go语言实战》 - 作者William Kennedy

[2] 《Go语言高级编程》 - 作者Noah Snook

[3] <https://go.dev/> -GoLang官方文档

[4] <https://blog.csdn.net/> -CSDN在线博客

[5] <https://github.com/> -Github开源社区

