超市库存管理系统的设计与实现

冯一鸣 21307346　　韩云昊 21307338

1. 引言

1.1 设计目的：

本设计旨在设计并实现一个简单的超市库存管理系统，能够实现商品信息管理、进货管理、销售管理和库存管理等功能，并体现对数据库的保护（安全性、完整性等）以及较为友好的用户界面。

1.2 系统应具备以下要求：

商品信息管理：能够对商品进行详细的信息记录和管理，实现增删改查。

进货信息管理：负责商品的进货信息的录入、删除和查询。

销售信息管理：负责商品的销售信息的录入、删除和查询。

库存信息管理：负责商品的库存信息的查询和预警；能够查询当前的库存信息，并实现库存预警功能。

安全性：实现不同人员对数据库有不同权限的要求，比如进货员不能修改销售信息，销售员不能修改进货信息，除了管理员外不能修改商品信息等。

完整性：当添加或删除进货信息和销售信息时，库存信息应同步更新。

1.3 系统的设计与实现将基于以下环境：

后端数据库：使用关系型数据库 PostgreSQL

后端编程语言：python（Flask 框架）

前端编程语言：JavaScript 和 Vue.js

1.4 同组人员及分工：

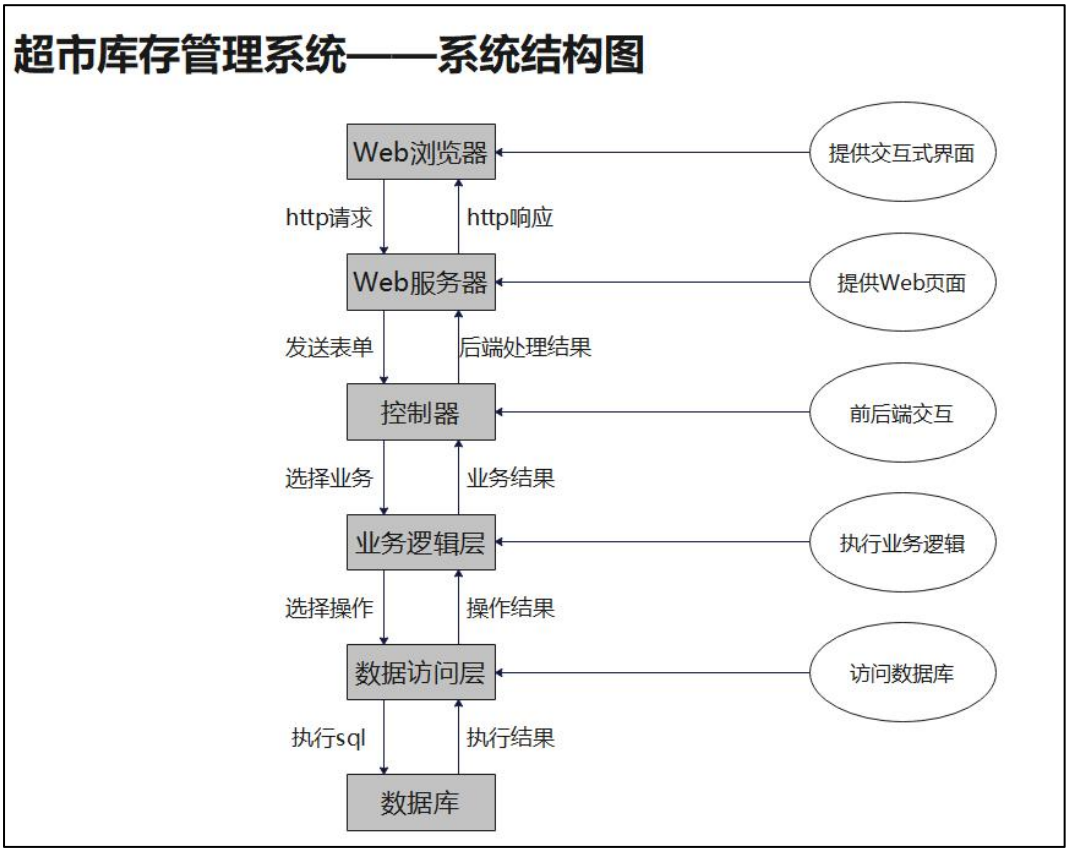
组员 任务	冯一鸣	韩云昊
概要设计	系统功能模块设计	系统结构设计
详细设计	ER 图设计	将 ER 图转为关系模式
系统实现	后端开发/调试	前端开发/调试
实验报告	1 2.1，2.3 3.1.1，3.2，3.3.1 4.1	2.2 3.1.2，3.1.3，3.3.2 4.2，4.3 5 6

2. 概要设计

2.1 系统需求分析：

通过对超市库存管理系统的需求分析，可以得到以下主要功能需求：
商品信息管理，实现对商品信息的增删改查
进货信息管理，实现对进货信息的增删查
销售信息管理，实现对销售信息的增删查
库存信息管理，实现对库存信息的查询和预警功能
安全性体现，设置三个不同权限的角色，包括管理员，进货员，售货员
完整性体现，设置触发器和外键约束，使库存信息与其他表一致

2.2 系统结构设计：



2.3 功能模块设计：

2.3.1 商品信息管理

实现对商品信息的增删改查。

2.3.2 进货信息管理

实现对进货信息的增删查。

2.3.3 销售信息管理

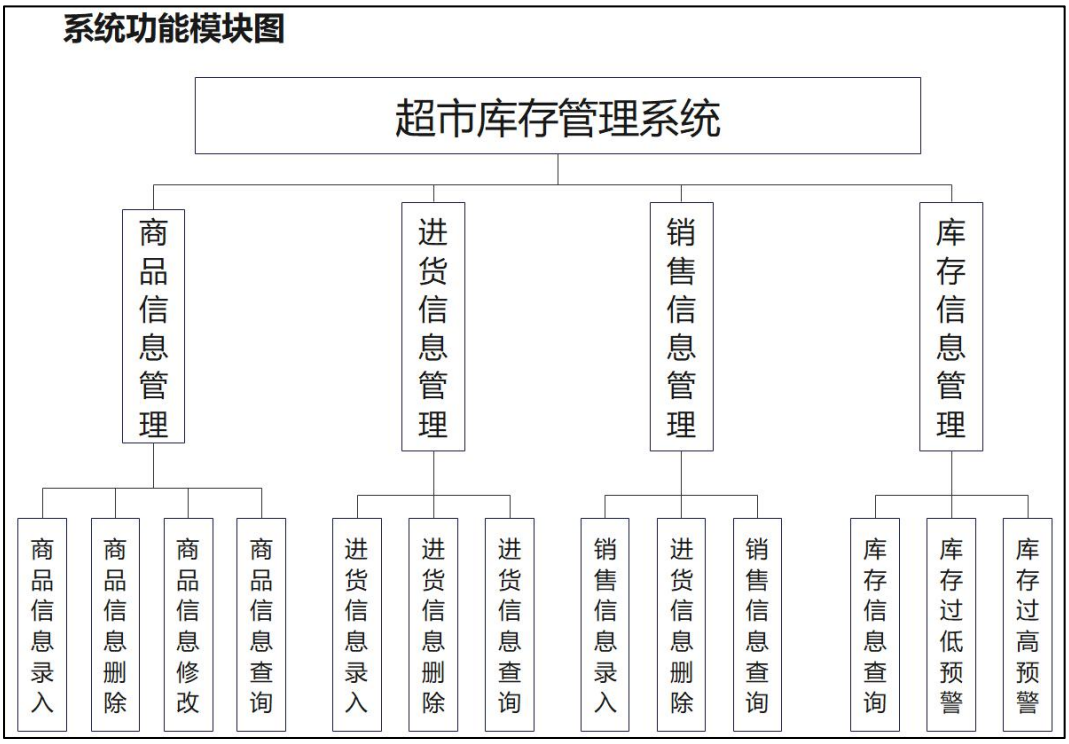
实现对销售信息的增删查。

2.3.4 库存信息管理

实现对库存信息的查询和发送预警，保持库存信息与进货/销售信息的一致性。

2.3.5 权限设置

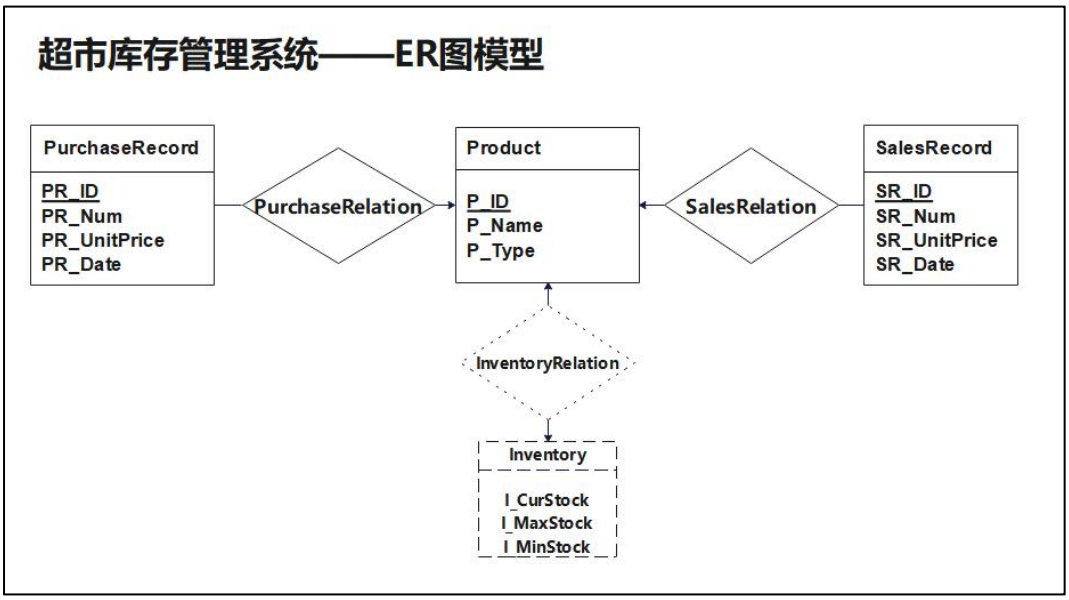
体现安全性，需要避免人员对没有权限的表进行相关操作。



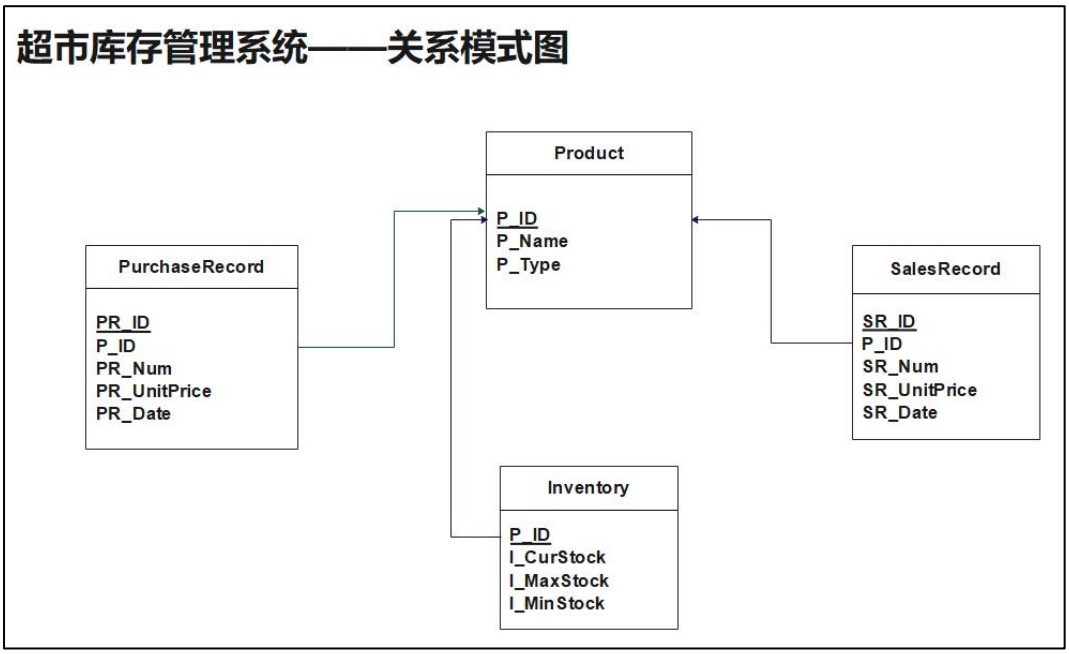
3. 详细设计

3.1 系统数据库设计

3.1.1 ER 图设计



3.1.2 ER 图转换为 3NF 关系模式



3.1.3 各模块所需的基本表的结构、模式和表之间的关系

关系表 **Product**：

属性名	类型	备注（主、外键，解释）
P_ID	char	主键，商品编号
P_Name	char	商品名称
P_Type	char	商品类型

关系表 **PurchaseRecord:**

属性名	类型	备注（主、外键，解释）
PR_ID	char	主键，进货编号
P_ID	char	外键，商品编号
PR_Num	int	进货数量
PR_UnitPrice	int	进货商品单价
PR_Date	date	进货日期

关系表 **SalesRecord:**

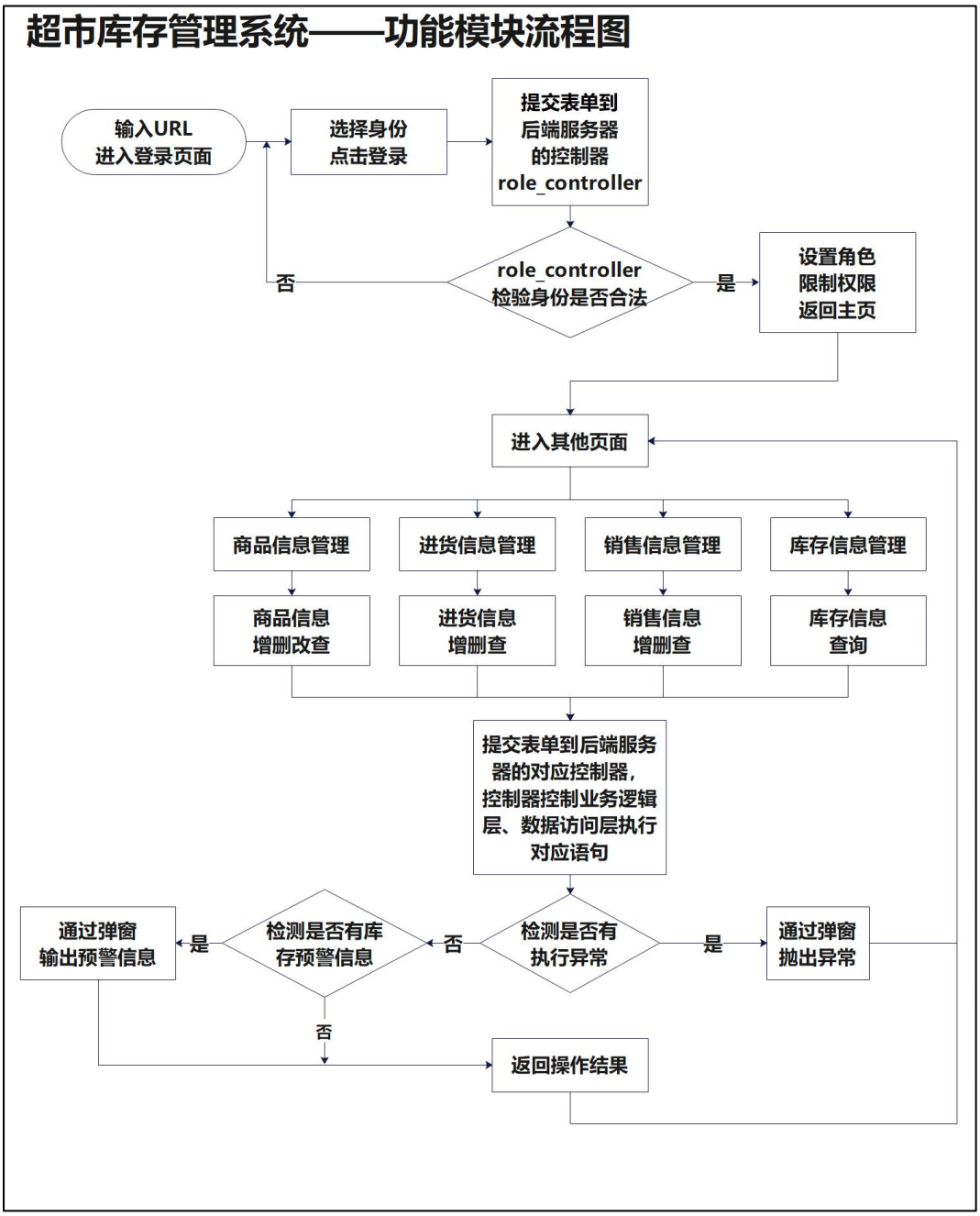
属性名	类型	备注（主、外键，解释）
SR_ID	char	主键，销售编号
P_ID	char	外键，商品编号
SR_Num	int	销售数量
SR_UnitPrice	int	销售商品单价
SR_Date	date	销售日期

关系表 **Inventory:**

属性名	类型	备注（主、外键，解释）
P_ID	char	主键、外键，商品编号

I_CurStock	int	商品当前库存
I_MaxStock	int	商品最大库存
I_MinStock	int	商品最小库存

3.2 系统主要功能模块设计（可用流程图表示）



3.3 各模块的主要算法对应的源代码

3.3.1 后端

3.3.1.1 触发器

添加商品信息时自动追加初始库存信息，当前库存 0，最大库存 200，最小库存 0

```
create or replace function add_product() returns trigger as
    $$
```

```
        begin
            insert into Inventory
            values(new.P_ID, 0, 200, 0);
            return null;
        end;
    $$ language plpgsql;
create trigger T1 after insert on Product
    for each row
    execute function add_product();
```

添加进货信息后自动更新库存信息

```
create or replace function add_purchase() returns trigger as
    $$
        begin
            update Inventory
            set I_Curstock = I_Curstock + new.PR_Num
            where P_ID = new.P_ID;
            return null;
        end
    $$ language plpgsql;
create trigger T2 after insert on PurchaseRecord
    for each row
    execute function add_purchase();
```

删除进货信息后自动更新库存信息

```
create or replace function del_purchase() returns trigger as
    $$
        begin
            update Inventory
            set I_Curstock = I_Curstock - old.PR_Num
            where P_ID = old.P_ID;
            return null;
        end
    $$ language plpgsql;
create trigger T3 after delete on PurchaseRecord
    for each row
    execute function del_purchase();
```

添加销售信息后自动更新库存信息

```
create or replace function add_sales() returns trigger as
    $$
        begin
            update Inventory
            set I_Curstock = I_Curstock - new.SR_Num
            where P_ID = new.P_ID;
            return null;
        end
```

```
    $$ language plpgsql;
create trigger T4 after insert on SalesRecord
    for each row
    execute function add_sales();
```

删除销售信息后自动更新库存信息

```
create or replace function del_sales() returns trigger as
    $$
        begin
            update Inventory
            set I_Curstock = I_Curstock + old.SR_Num
            where P_ID = old.P_ID;
            return null;
        end
    $$ language plpgsql;
create trigger T5 after delete on SalesRecord
    for each row
    execute function del_sales();
```

库存容量限制，不能超出最大容量，不能小于最小库存

```
create or replace function capacity_limitation() returns trigger as
    $$
        begin
            if new.I_Curstock > new.I_Maxstock then
                raise exception '容量不足';
            elseif new.I_Curstock < new.I_Minstock then
                raise exception '库存不足';
            end if;
            return new;
        end
    $$ language plpgsql;
create trigger T6 before update on Inventory
    for each row
    execute function capacity_limitation();
```

3.3.1.2 预警函数

```
# 预警函数，检查数据，查看是否达到预警线
# 如果商品的库存量过多或过少，发出预警信息，否则返回字符串"\n"
def warning():
    rst = dao.select_all()
    warning_info = "\n"
    for tup in rst:
        if tup[1] > tup[2] - 10:
            warning_info += "商品 %s 库存已超出警戒值，该商品
当前库存: %s, 最大库存: %s.\n" % (tup[0], tup[1], tup[2])
        elif tup[1] < tup[3] + 10:
```

```
                warning_info += "商品 %s 库存已低于警戒值，该商品
当前库存: %s，最小库存: %s.\n" % (tup[0], tup[1], tup[3])
        return warning_info
```

3.3.1.3 其他增删改查功能（以商品信息的查询为例）

底层数据库类执行逻辑，并声明一个全局唯一的实例

```
    class Database:
        # 构造函数，传入数据库连接参数
        def __init__(self, db, u, pw, h, p) -> None:
            # 使用 psycopy2.connect()方法连接到 postgresql 数据库
            self.con = psycopg2.connect(database=db,
                                        user=u,
                                        password=pw,
                                        host=h,
                                        port=p)
            # 创建游标，用于执行 SQL 语句，返回查询结果
            self.cur = self.con.cursor()
            print("Successfully connected to database!")

        # 执行函数，输入 SQL 语句执行查询，返回查询结果、异常或 None
        def execute(self, command):
            try:
                # 通过调用 cur.execute()方法对数据库进行操作
                self.cur.execute(command)
            # 如果出现执行异常（如违反约束等），回滚事务后返回异常
            except Exception as e:
                # 使用 con.rollback()回滚事务，保护数据库一致性
                self.con.rollback()
                # 返回异常
                return e
            # 若没有执行异常，取回查询结果后提交事务
            else:
                try:
                    # 如果有查询结果，返回查询结果
                    return self.cur.fetchall()
                # 如果没有返回结果（如增删改），
                # 则捕捉抛出的异常，返回 None
                except Exception as e:
                    return None
                finally:
                    # 使用 con.commit()提交事务，使数据持久化
                    self.con.commit()

        # 析构函数，最后使用 close()关闭数据库
        def __del__(self) -> None:
```

```
            self.con.close()
            print("Successfully disconnected from database!")

    # 设置数据库连接参数，指定数据库名，用户名，密码，主机名，端口号
    database = "超市库存管理系统"
    user = "postgres"
    password = "password"
    host = "localhost"
    port = "5432"
    # 设置参数，建立数据库连接
    db = Database(database, user, password, host, port)
```

数据访问层，直接调用数据库类接口，执行 SQL 语句

```
    # 操作成功，返回查询结果或空结果；失败，返回异常
    def select_all():
        sql = """
        select *
        from Product
        order by P_ID
        """
        return db.execute(sql)
```

业务逻辑层，调用数据访问层接口，实现业务逻辑

```
    def get_all_product():
        # 查询操作成功，返回查询结果或空结果；失败，返回异常
        rst = dao.select_all()
        # 如果查询到元组或失败，返回对应结果，
        # 否则查询结果为空，返回信息
        return rst if rst else "没有商品信息！\n"
```

控制器层，接收前端发送的表单，调用业务逻辑层接口

```
    @app.route("/product/get_all_product", methods=["GET",
"POST"])
    def get_all_product():
        rst = blo.get_all_product()
        return ret_m.return_method_select(rst)
```

返回模块，根据数据库执行结果返回相应信息和错误通知

```
    def return_method_select(rst):
        # 进行预警测试，返回值为预警信息或字符串"\n"
        warning_info = warning()
        # 返回字符串(或异常)表示没有查询结果(或失败)
        if isinstance(rst, str) or isinstance(rst, Exception):
            return jsonify({
                "error": True,
                "message": str(rst)
            })
        # 有结果返回结果
```

```
        else:
            return jsonify({
                "error": False,
                "message": warning_info,
                "result": rst
            })
```

3.3.2 前端

3.3.2.1 网页表格（以商品信息表为例）

```
<el-button type="primary" icon="el-icon-plus" style="margin-bottom: 15px"
@click="addProductDialogVisible = true">新增商品信息</el-button>
  <el-table
  :data="ProductsList"
  border
  style="width: 100%"
  >
  <el-table-column
    prop="P_ID"
    label="商品编号"
    align="center"
    width="150px"
  >
    <template slot-scope="scope">
      <p v-text="scope.row.P_ID"></p>
    </template>
  </el-table-column>
  <el-table-column
    prop="P_Name"
    label="商品名称"
    align="center"
  >
    <template slot-scope="scope">
      <p v-text="scope.row.P_Name"></p>
    </template>
  </el-table-column>
  <el-table-column
    prop="P_Type"
    label="商品类型"
    align="center"
  >
    <template slot-scope="scope">
      <p v-text="scope.row.P_Type"></p>
    </template>
  </el-table-column>
```

```
  <el-table-column
    label="操作"
    width="265px"
    align="center"
  >
    <template slot-scope="scope">
      <el-button type="primary" icon="el-icon-view" size="mini"
@click="getProductBypid(scope.row.P_ID)">编辑</el-button>
      <el-button type="danger" icon="el-icon-delete" size="mini"
@click="deleteProductBypid(scope.row.P_ID)">删除</el-button>
    </template>
  </el-table-column>
  </el-table>
```

3.3.2.2 请求数据库信息的函数（以请求商品信息关系表的函数为例）

```
    getProductList() {
      // 发送请求
      axios.get('http://localhost:8888/product/get_all_product')
        .then(response => {
          let result = response.data;
          if (result.error) {
            return this.$message.warning(result.message);
          } else {
            this.ProductsList = result.result.map(item => {
              return {
                P_ID: item[0],
                P_Name: item[1],
                P_Type: item[2],
              };
            });
          }
        })
        .catch(error => {
          console.error('获取商品列表失败:', error);
          this.$message.error(String(error));
        });
    },
```

3.3.2.3 Vue.js 应用的路由配置

使用 Vue Router，定义多个页面组件，如登录页、主页等，并设置了对应的路由路径。根路径重定向到登录页，主页下有多个子页面路由，通过 Vue Router 实现了在单页面应用中不同页面间的导航，创建了路由实例并导出供应用使用。

```
import Vue from 'vue'
import VueRouter from 'vue-router'
import Login from '../components/Login.vue'
```

```
import Home from '../components/Home.vue'
import Welcome from '../components/Welcome.vue'
import Info from '../components/Info.vue'
import Purchase from '../components/Purchase.vue'
import Product from '../components/Product.vue'
import Sales from '../components/Sales.vue'
import Inventory from '../components/Inventory.vue'


Vue.use(VueRouter)
const routes = [
  { path: '/', redirect: '/login' },
  { path: '/login', component: Login },
  {
    path: '/home',
    component: Home,
    redirect: '/welcome',
    children: [
      { path: '/welcome', component: Welcome },
      { path: '/info', component: Info },
      { path: '/purchase', component: Purchase },
      { path: '/product', component: Product },
      { path: '/sales', component: Sales },
      { path: '/inventory', component: Inventory}
    ]
  }
]

const router = new VueRouter({
  routes
})

export default router
```

4. 调试与运行结果及存在的主要问题

4.1 后端调试与运行结果及存在的主要问题

v1 代码

```
    def execute(self, command):
        self.cur.execute(command)
        return self.cur.fetchall()
```

测试结果

```
Successfully connected to database!
Traceback (most recent call last):
  File "c:\Users\Administrator.DESKTOP-DQ6I26O\Desktop\课件\31课件(大三上)\0数据库系统概念\实验\lab14 课程设计\src\debug.py", line 68, in <module>
    rebuild()
  File "c:\Users\Administrator.DESKTOP-DQ6I26O\Desktop\课件\31课件(大三上)\0数据库系统概念\实验\lab14 课程设计\src\debug.py", line 48, in rebuild
    print(db.execute(sql_rebuild))
          ^^^^^^^^^^^^^^^^^^^^^^^
  File "c:\Users\Administrator.DESKTOP-DQ6I26O\Desktop\课件\31课件(大三上)\0数据库系统概念\实验\lab14 课程设计\src\debug.py", line 19, in execute
    return self.cur.fetchall()
           ^^^^^^^^^^^^^^^^^^^
psycopg2.ProgrammingError: no results to fetch
Successfully disconnected from database!
```

问题分析

为了使用一种通用的方式处理所有 SQL 语句，在执行完 execute()后都同一执行 fetchall()，但是在没有返回结果时，执行该函数会报错。

解决方案，v2 代码

```python
    def execute(self, command):
        self.cur.execute(command)
        try:
            # 如果有查询结果，返回查询结果
            return self.cur.fetchall()
        # 如果没有返回结果（如增删改），
        # 则捕捉抛出的异常，返回 None
        except Exception as e:
            return None
```

测试结果

```
Successfully connected to database!
Traceback (most recent call last):
  File "c:\Users\Administrator.DESKTOP-DQ6I26O\Desktop\课件\31课件(大三上)\0数据库系统概念\实验\lab14 课程设计\src\debug.py", line 75, in <module>
    db.execute(sql)
  File "c:\Users\Administrator.DESKTOP-DQ6I26O\Desktop\课件\31课件(大三上)\0数据库系统概念\实验\lab14 课程设计\src\debug.py", line 18, in execute
    self.cur.execute(command)
psycopg2.errors.UniqueViolation: 错误:  重复键违反唯一约束"product_pkey"
DETAIL:  键值"(p_id)=(1)" 已经存在

Successfully disconnected from database!
```

问题分析

执行 SQL 语句时可能会遇到异常，如违反约束，权限不足等。

应进一步处理，遇到异常应回滚，若成功执行，则应提交。

解决方案，v3 代码，见 3.3.3.1

4.2 前端调试与运行结果及存在的主要问题

在前端运行 npm run serve 命令报错：

```
ERROR  ValidationError: Progress Plugin Invalid Options

    options should NOT have additional properties
    options should NOT have additional properties
    options should NOT have additional properties
    options should pass "instanceof" keyword validation
    options should match exactly one schema in oneOf
```

问题分析：

Webpack 配置对象中存在不被支持的或者不合规的选项，错误信息中“Progress Plugin Invalid Options”指向问题与 Webpack 进度插件（Progress Plugin）的配置相关。

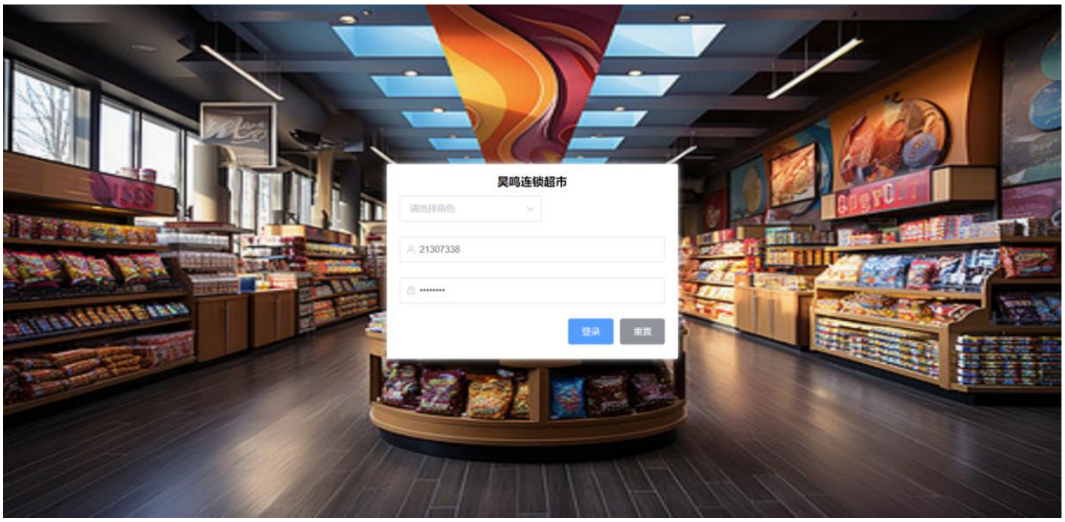
解决方案：

先执行 npm cache clean --force 命令清理 Webpack 缓存，然后执行 npm install webpack@latest --save-dev 命令将 Webpack 升级到最新版本，接着

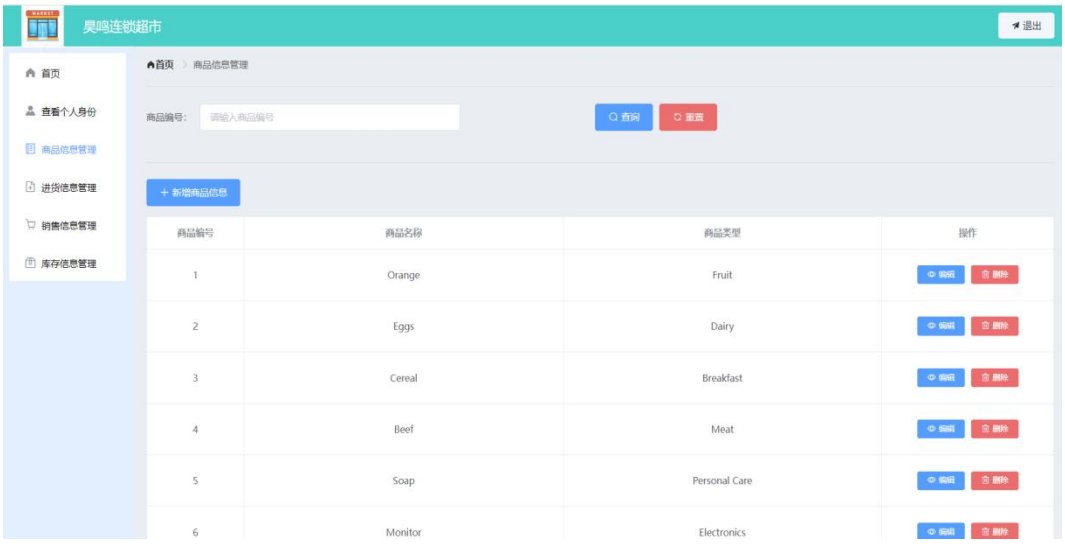
执行 npm install progress-webpack-plugin@latest --save-dev 命令尝试升级 progress-webpack-plugin 插件，最后再重新运行 npm run serve 便可成功执行。

4.3 结果展示：

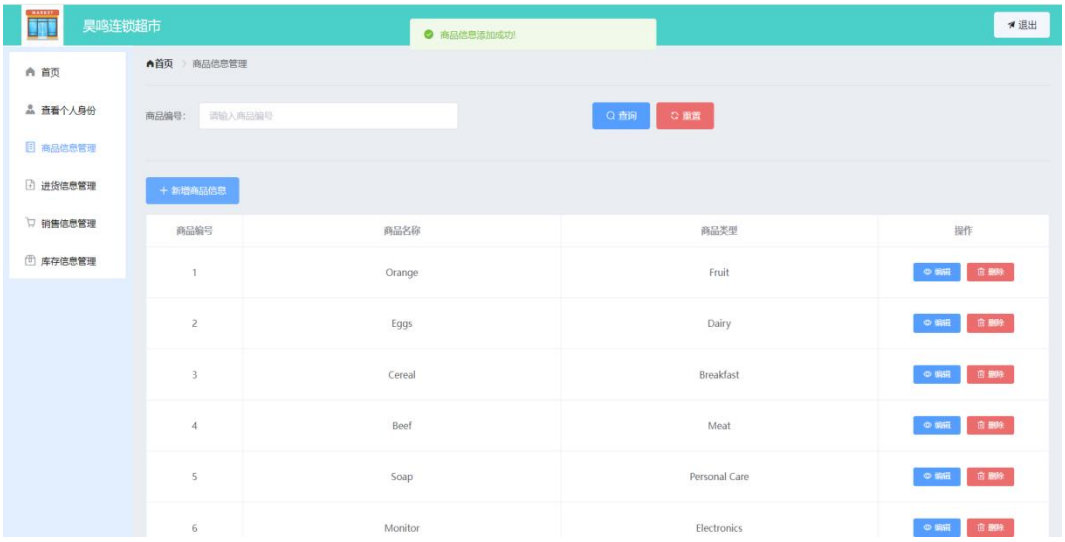
4.3.1 登录界面：



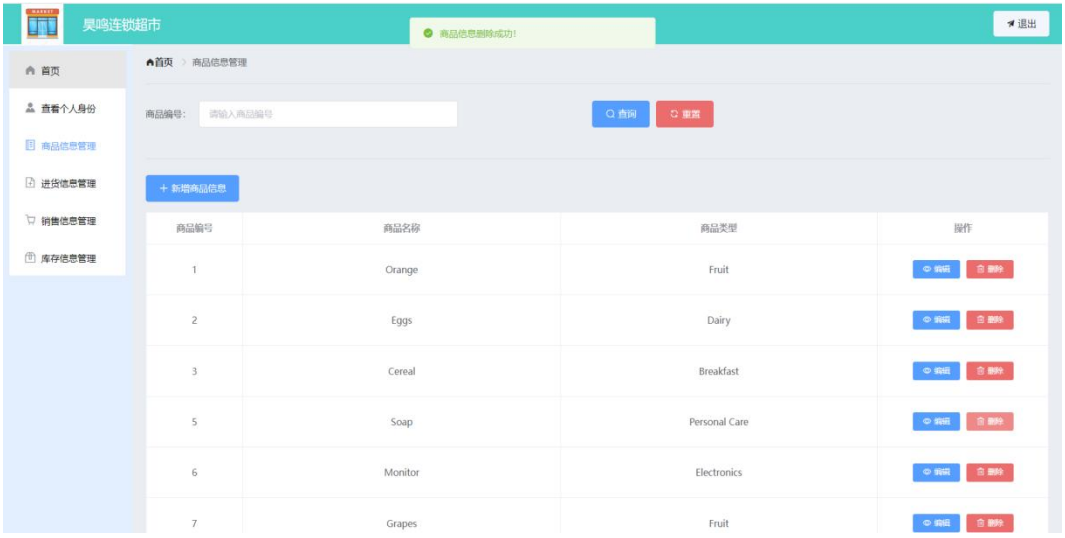
4.3.2 信息管理界面（以商品信息管理为例）：



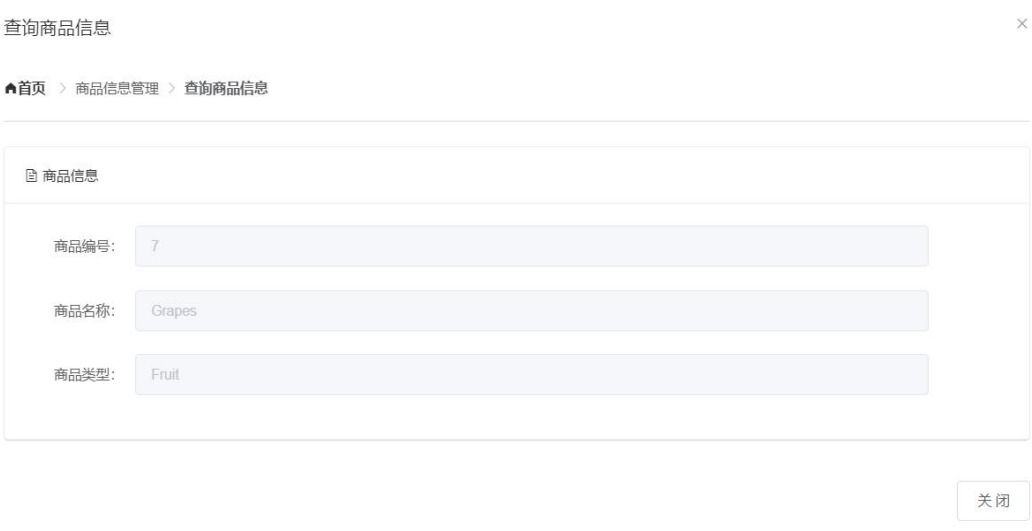
4.3.3 新增信息（以新增商品信息为例）：



4.3.4 删除信息（以删除商品信息为例）：



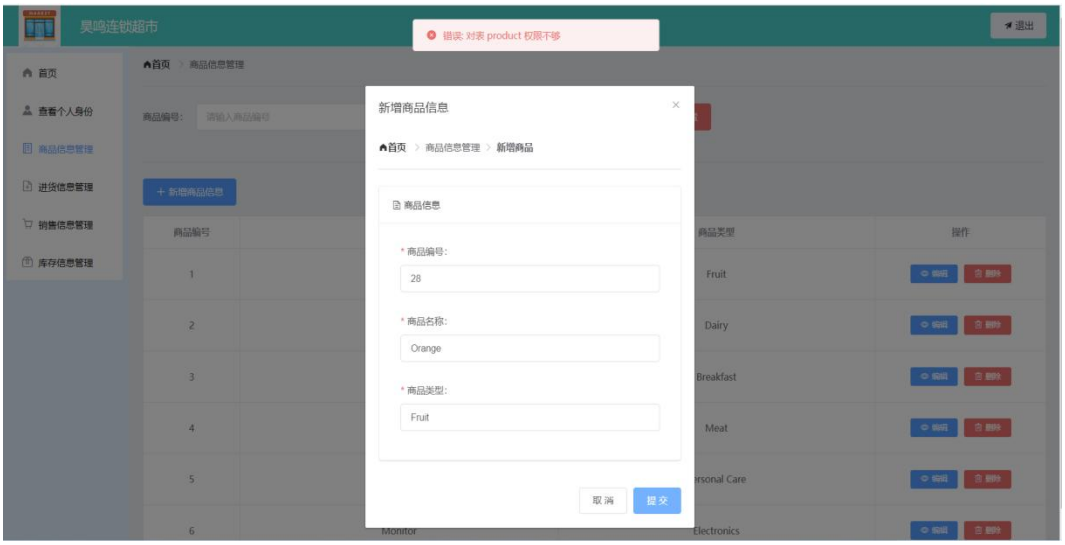
4.3.5 查询信息（以查询商品信息为例）：



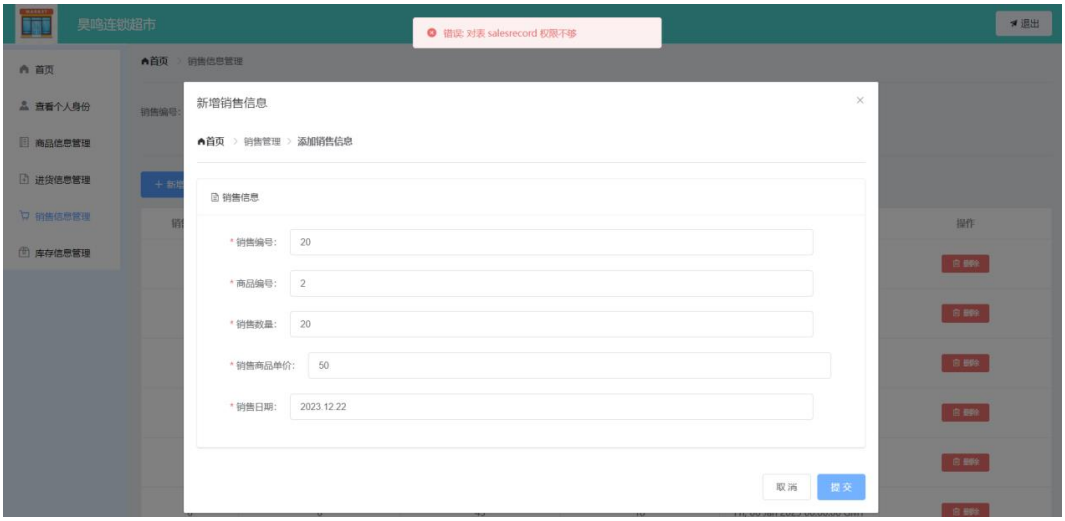
4.3.6 权限

（管理员可以进行所有操作，进货员和售货员不可对商品信息　进行添加、更新和删除操作，进货员不可对销售信息进行操作，售货员不可对进货信息进行操作）以进货员身份为例：

无法新增商品信息：



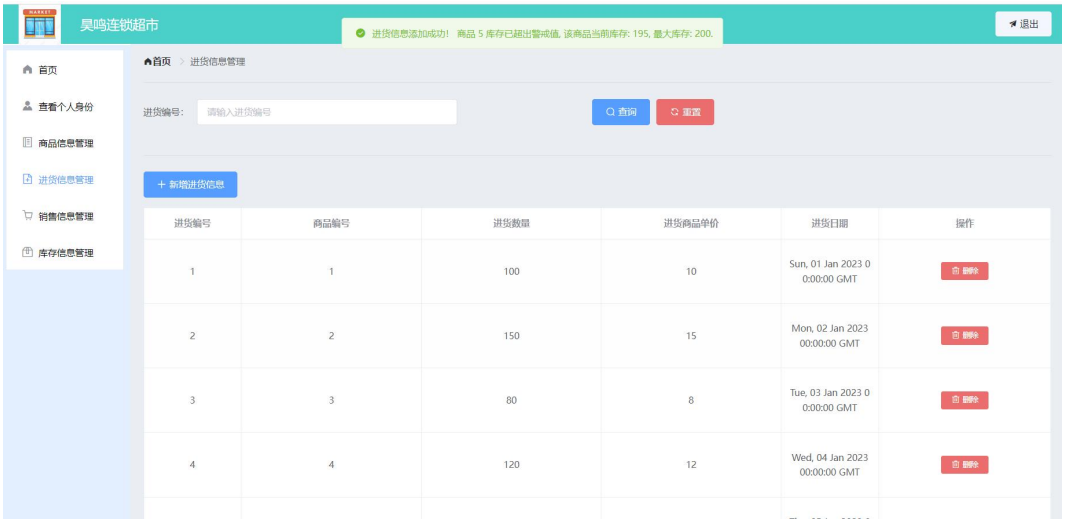
无法新增销售信息：

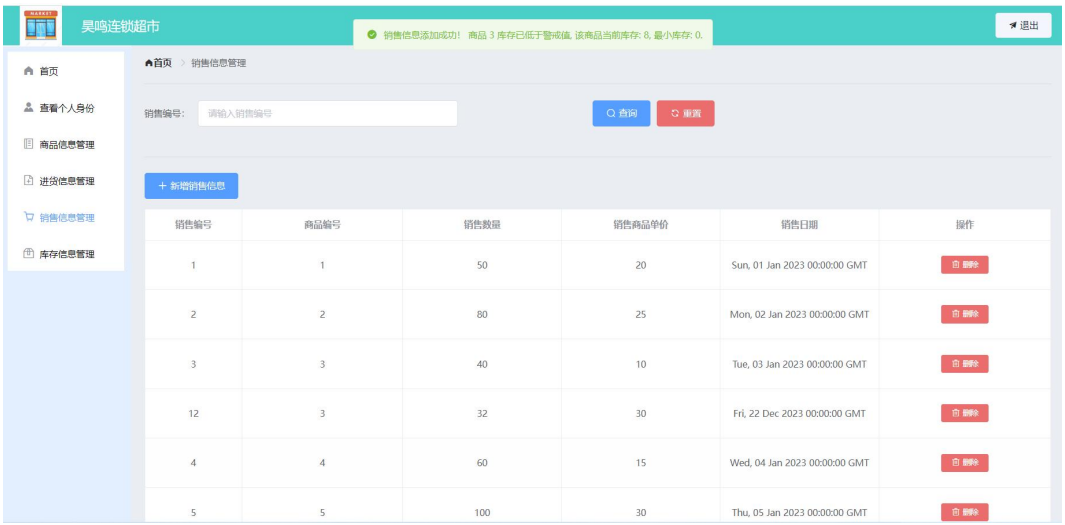


无法删除销售信息：

4.3.7 库存预警

当商品当前库存与最大库存或最小库存的差距在 10 以内时发起预警





5. 课程设计小结

在这个超市库存管理系统的课程设计中，我们深刻体验到了从需求分析到系统实现的全过程，这次项目让我们对数据库和应用程序设计有了更深入的理解，并培养了解决实际业务问题的能力。

首先，我们进行了详细的需求分析，明确了系统所需的功能，包括商品信息管理、进货管理、销售管理和库存管理等。

在系统设计阶段，我们绘制了系统结构图和功能模块图，以直观地呈现系统的架构和各功能之间的关系。采用 E-R 图进行概念建模，有助于理清系统中各实体之间的关联关系。转换概念模型为关系模式，并确保数据库设计满足 3NF 的要求，以保证数据的规范性和一致性。这一阶段的工作帮助我们更好地理清系统整体架构，同时也让我们深入思考了数据库模型的设计。

在前端方面，我们选择了 JavaScript 和 Vue.js 进行开发。通过 Vue.js，我们成功地创建了一个动态、响应迅速的用户界面，使用户能够以直观的方式与系统进行交互。在用户界面设计中，我们注重了布局的清晰性和直观性，以提高用户的使用体验。在后端方面，我们选择了 Python 语言，使用 Flask 框架，搭建了系统的服务端。同时，选择了 PostgreSQL 作为后端数据库，以保证系统数据的可靠性和安全性。在后端编程中，我们注重了系统的安全性和性能。

为了保障系统的安全性，我们设定了不同权限的角色，包括管理员、进货员和售货员，以便不同用户能够执行符合其职责的操作，确保数据的安全和合规。在数据完整性方面，我们使用了触发器，确保库存信息与其他表的一致性，防止数据不一致的情况发生。

总体而言，这个课程设计不仅提升了我们的技术实践和团队合作能力，更让我们深入了解了数据库和应用程序设计的方方面面，从需求分析到系统实现，让我们积累了丰富的经验，同时也提高了我们对系统实用性、安全性和完整性等方面的考虑。这个项目让我们更好地理解了课堂理论知识在实际项目中的应用，是一次很有收获的学习经历。

然而，我们也认识到系统还存在一些可以改进的地方：

- 用户界面优化：尽管已经实现了用户界面，但仍然可以进一步优化，使其更加直观、易用，以便用户更轻松地操作系统，提高用户的操作效率。
- 性能优化：进一步优化后端的数据库查询和操作，确保系统在处理大量数据时能够保持高效性能。
- 系统扩展性：系统的扩展性可以更全面地考虑，以确保系统能够灵活适应变化的业务需求。
- 日志记录与监控：引入更全面的日志记录和系统监控功能，有助于及时发现和解决潜在问题，提高系统的可维护性。
- 数据备份和恢复：添加定期数据备份和恢复机制，以应对数据丢失或系统故障的风险，增强系统的稳定性。

6. 参考文献

6.1 前端开发参考资料：

Vue 安装与配置：

https://blog.csdn.net/m0_57545353/article/details/124366678

HTML、JavaScript 和 Vue.js 学习使用：

https://www.runoob.com/html/html-tutorial.html

https://www.runoob.com/js/js-tutorial.html

https://www.runoob.com/vue2/vue-tutorial.html

6.2 后端开发参考资料：

后端与数据库通信：

https://blog.csdn.net/CaptainTakuya/article/details/113768112

https://www.jianshu.com/p/e57636147791

前端与后端通信：

https://blog.csdn.net/Littleflowers/article/details/113926184

7. 程序源代码

https://github.com/Supremacy-ysyyrps/Database-Project

8. 应用系统链接

http://8.134.218.23:8080/

角色	账号	密码
管理员	manager	manager
进货员	buyer	buyer
售货员	seller	seller