|  |  |  |
| --- | --- | --- |
| 南农大  **数据库课程设计报告**  XH2 | | |
|  | 题 目: | 数据库课程设计-销售管理系统6 |
|  | 组 长: | 学号: |
|  | 组 员 | 学号: |
|  |  | 学号: |
|  | 学 院: | 人工智能学院 |
|  | 专 业: | 人工智能、计算机科学与技术 |
|  | 班 级: | 人工智能211、计科212 |
|  | 指导教师: | 职称: 副教授 |
| 2023年5月18日 | | |

**目 录**

1开发技术、开发工具及技术路线2

1.1开发技术及开发工具2

1.1.1开发技术2

1.1.2开发工具2

1.2技术路线3

2功能分析与设计4

2.1需求分析4 2.1.1用户调研4 2.1.2初步设计4

2.2系统设计4

2.2.1系统流程设计4

2.2.2系统功能分析5

2.3数据库概念结构设计5 2.4数据库逻辑结构设计6

2.4.1 E-R图向关系模型的转换6

2.4.2利用数据依赖的理论对关系模式进行分析7

2.4.3最终关系模式设计7

2.5数据库物理结构设计8

2.5.1确定存储结构8

2.6模块划分9

3实现10

3.1前端界面实现10

3.1.1用户界面设计11

3.1.2雇员界面设计12

3.1.3供应商界面设计以及其他界面13

3.2后端代码实现13

3.2.1 Python与SQL Server的连接和交互13

3.2.2通过PySide2进行前端界面和后端数据库的相关操作14

3.2.3 SQL Server建立数据库18

3.2.4 SQL Server索引的设置26

3.2.5 SQL Server触发器的设置27

4系统测试与应用28

4.1系统测试运行环境28

4.2数据库测试与运行29

5总结与展望33

5.1总结33

5.1.1客户中心33

5.1.2雇员中心33

5.1.3供应商中心33

5.2不足34

5.3展望35

**销售管理系统**

**Sales Management System**

**1 开发技术、开发工具及技术路线**

**1.1 开发技术及开发工具**

**1.1.1 开发技术**

本平台是基于Python开发的，后台数据库采用Microsoft SQL server。

（1）Python

Python是一种高级编程语言，一种通用、解释型、面向对象的编程语言。它由[荷兰](https://baike.baidu.com/item/%E8%8D%B7%E5%85%B0/190469?fromModule=lemma_inlink" \t "https://baike.baidu.com/item/Python/_blank)[数学](https://baike.baidu.com/item/%E6%95%B0%E5%AD%A6/107037?fromModule=lemma_inlink" \t "https://baike.baidu.com/item/Python/_blank)和[计算机科学](https://baike.baidu.com/item/%E8%AE%A1%E7%AE%97%E6%9C%BA%E7%A7%91%E5%AD%A6/9132?fromModule=lemma_inlink" \t "https://baike.baidu.com/item/Python/_blank)研究学会的[吉多·范罗苏姆](https://baike.baidu.com/item/%E5%90%89%E5%A4%9A%C2%B7%E8%8C%83%E7%BD%97%E8%8B%8F%E5%A7%86/328361?fromModule=lemma_inlink" \t "https://baike.baidu.com/item/Python/_blank)于1990年代初设计，作为一门[ABC语言](https://baike.baidu.com/item/ABC%E8%AF%AD%E8%A8%80/334996?fromModule=lemma_inlink" \t "https://baike.baidu.com/item/Python/_blank)的替代品。

发展至今，它提供了高效的高级[数据结构](https://baike.baidu.com/item/%E6%95%B0%E6%8D%AE%E7%BB%93%E6%9E%84/1450?fromModule=lemma_inlink" \t "https://baike.baidu.com/item/Python/_blank)，语言简洁、易读易写、简单有效，支持多种编程范式，如面向对象、函数式和过程式编程等。其[语法](https://baike.baidu.com/item/%E8%AF%AD%E6%B3%95/23734941?fromModule=lemma_inlink" \t "https://baike.baidu.com/item/Python/_blank)和动态类型，以及[解释型语言](https://baike.baidu.com/item/%E8%A7%A3%E9%87%8A%E5%9E%8B%E8%AF%AD%E8%A8%80/8888952?fromModule=lemma_inlink" \t "https://baike.baidu.com/item/Python/_blank)的本质（易于扩展，可以使用[C语言](https://baike.baidu.com/item/C%E8%AF%AD%E8%A8%80/105958?fromModule=lemma_inlink" \t "https://baike.baidu.com/item/Python/_blank)或[C++](https://baike.baidu.com/item/C++/99272?fromModule=lemma_inlink" \t "https://baike.baidu.com/item/Python/_blank)或者其他可以通过C调用的语言扩展新的功能和[数据类型](https://baike.baidu.com/item/%E6%95%B0%E6%8D%AE%E7%B1%BB%E5%9E%8B/10997964?fromModule=lemma_inlink" \t "https://baike.baidu.com/item/Python/_blank)），使它成为多数平台上写[脚本](https://baike.baidu.com/item/%E8%84%9A%E6%9C%AC/1697005?fromModule=lemma_inlink" \t "https://baike.baidu.com/item/Python/_blank)和快速开发应用的编程语言， 随着版本的不断更新、标准库的扩充和语言新功能的添加，Python逐渐被用于独立的、[大型项目](https://baike.baidu.com/item/%E5%A4%A7%E5%9E%8B%E9%A1%B9%E7%9B%AE/3986637?fromModule=lemma_inlink" \t "https://baike.baidu.com/item/Python/_blank)的开发，也被广泛应用于人工智能、数据科学、Web 开发、网络编程、游戏开发、科学计算等领域。

（2）SQL Server

SQL Server是一种关系型数据库管理系统，采用了“客户端-服务器”模型，由Microsoft公司开发和维护。它使用结构化查询语言SQL来管理和查询数据，能够存储和检索大量数据，并提供高可用性、安全性和可扩展性。并且，它还支持存储多种数据类型、索引、触发器、存储过程和视图等数据库对象，能够实现大量数据的备份、恢复，提供强大的分析和报告功能。

另外，作为数据库管理系统，SQL Server可以与微软的其他产品（比如NET Framework、VS等）集成，因而适用于各种规模的企业应用和互联网应用，是许多互联网企业的首选数据库管理系统。

1.1.2 开发工具

本系统在Windows 11操作系统上开发，采用PyCharm集成开发环境，利用Python QT Designer可视化界面设计工具进行开发，数据库管理系统采用Microsoft SQL Server Management Studio 19。

（1）PyCharm

PyCharm是由JetBrains开发的Python集成开发环境（IDE）。

集成开发环境 (IDE) 是一个功能丰富的软件应用程序，可用于软件开发的许多方面，使软件开发人员能高效编写、测试、调试代码。常见的IDE有Visual Studio（主要用于C语言系列）、PyCharm（主要用于Python语言）、Eclipse（主要用于JAVA语言）等。

PyCharm除了拥有大多数IDE所具有的标准编辑器和调试器之外，还包含编辑器、调试器、交互式控制台、版本控制集成等，专用于提高Python开发工作的效率、简化软件开发过程。

并且，PyCharm支持多种Python框架和库，如Django、Flask、Pyramid等，也能够与许多外部工具集成，如数据库、Web服务和云平台，功能十分强大。

（2）Python QT Designer

QT全称为Qt Toolkit，是一个跨平台的应用程序开发模块、开源的框架。它提供一系列GUI构件和可重用的部件，支持多种操作系统平台，也支持多种编程语言（包括但不仅限于Java、C++、Python等）。

QT Designer是一款可视化界面设计工具，用于QT应用程序的图形用户界面（GUI）设计。它允许用户通过拖拽和放置组件来创建GUI（比如按钮、图标、下拉列表、文本框等）及为组件添加事件处理程序和信号槽等，并生成可通过Python代码调用的GUI文件，用于开发交互式桌面应用程序。

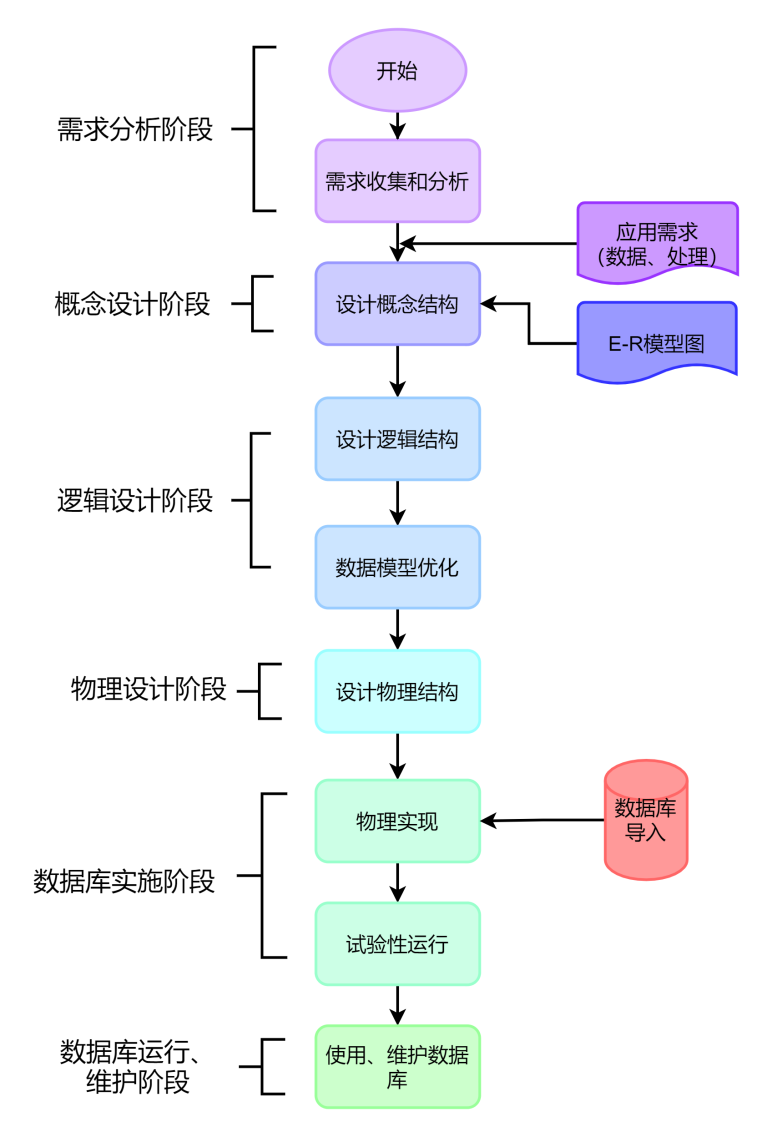
Python QT Designer也称为PyQT Designer，是与Qt Designer集成的PyQt库，它可以利用 Python 编程语言来操作Qt Designer，并在Qt Designer的基础上增加了更多工具和功能，包括界面调试、布局管理、事件处理、自定义控件的创建等。

（3）Microsoft SQL Server Management Studio

SSMS，也就是SQL Server Management Studio，由微软公司提供的一个集成环境，用于访问、配置、管理和开发SQL Server的所有组件，管理从 SQL Server 到 Azure SQL 数据库的任何 SQL 基础结构。SSMS组合了大量图形工具和丰富的脚本编辑器，可以帮助用户创建、修改、备份、还原、调试、查询数据库，进行数据的导入、导出、分析，进行数据库性能监视和优化等。使各种技术水平的用户、开发人员和管理员都能访问SQL Server，是SQL Server数据库管理与监视的主要工具之一。

**1.2 技术路线**

本系统着重分析当前线上购物批发平台的运营流程、销售情况，详细设计、实现了针对商品货物供应商、平台雇员以及用户（客户）的功能，比如：

****

用户可以选择心仪的商品，按照需求选择数量、查看购物车、发订单给雇员、确认收货等等。

平台雇员通过查看用户的商品订单以及供货商的库存量、地址等手动选择下订单数量等，连接平台用户、供货商，提供一种渠道，以满足消费者的购买需求、供货者的出售需求，从而获得更大的利润。

供货商可以从雇员处接受订单、确认发货、更新商品信息、新建商品等

利用数据库设计的思想，结合关系型数据库平台、python面向对象的程序设计、QT Designer等技术，设计实现了本销售管理系统。

本平台的开发技术路线如右图（图1-1）所示：

图 1-1 技术开发路线图

**2 功能分析与设计**

**2.1 需求分析**

**2.1.1 用户调研**

本线上销售管理系统针对的用户是拥有货源的供货商、拥有需求的客户和对订单进行确认的平台雇员，旨在搭建一个货物供应商与客户之间的交易中间平台，满足客户随时随地下订单挑选、购买商品、确认收货的需求，也扩大了供货商的销售渠道、开拓更多消费市场，完成了查询订单、确认发货的功能。满足了雇员对于新增订单的检查、订单查询、错误订单删除、货物商品收件后确认订单完成等的需求，提高了工作效率。

**2.1.2 初步设计**

根据用户调研的需求分析，我们将系统功能大致分为如下几块：

（1）用户登录：本系统每一次初始化后界面为登录界面，同时根据是否拥有系统账号可以选择注册账户或检查登录；密码输入错误则需要重新登陆，密码输入成功时，会各个用户根据自己的需要选择客户用户界面或雇员用户界面或供应商用户界面。

（2）客户用户：跳转至客户首页界面，可在此界面选择查看产品、添加购物车，购物车界面可修改商品数量、进行支付、确认收货等。也可以修改个人信息、查看积分、进行兑换，将密码还原为初始密码等。

（3）雇员（雇户）用户：跳转至雇员信息界面，可在此界面选择，查看或修改（增删改）订单信息，选择供应商，查询产品库存信息，查询销售额情况，也可以在此界面修改个人信息，将密码还原为初始密码等。

（4）供应商用户：跳转至供应商信息界面，可在此界面选择查看订单信息、产品上架（新建、更新）、产品库存修改、确认发货等，也可以选择修改个人信息，将密码还原为初始密码等。

**2.2 系统设计**

**2.2.1 系统流程设计**

根据对需求分析阶段所得到的信息进行分析后，将系统流程细化到如图3-1所示：

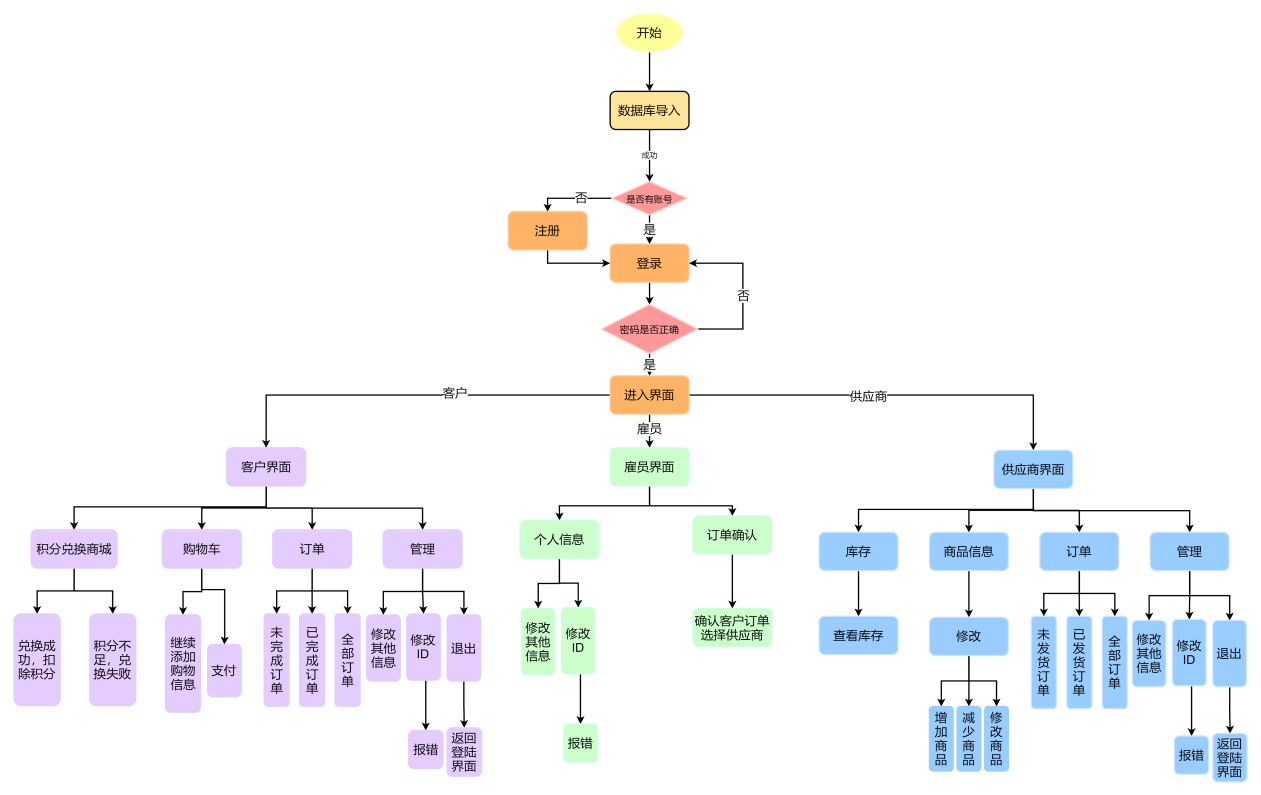
****

图3-1系统流程图

**2.2.2 系统功能分析**

根据上述系统流程图分析，系统功能概括为图3-2：

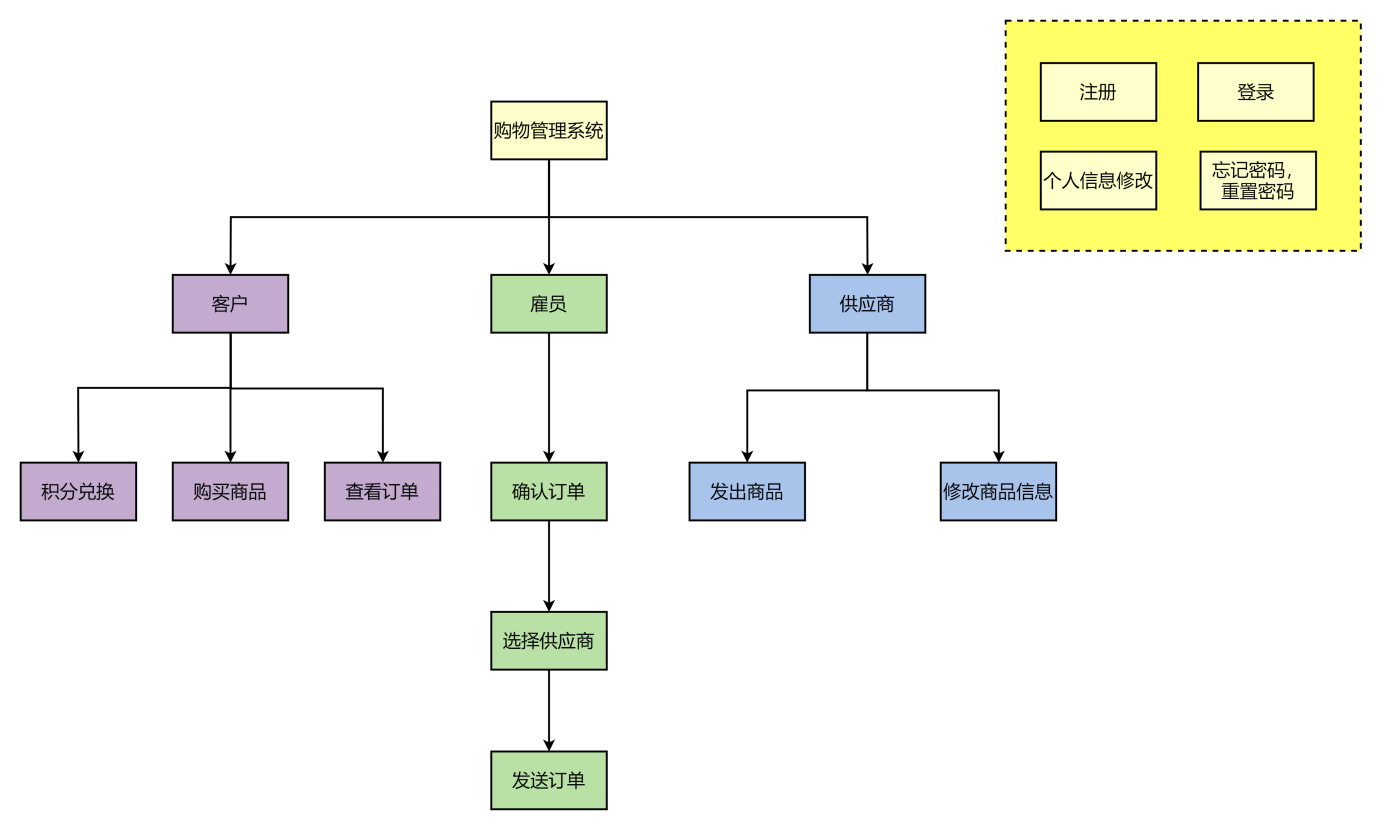


图3-2系统功能图

**2.3 数据库概念结构设计**

数据库概念结构设计即根据对现实世界的调研、通过对用户需求进行综合，归纳与抽象，形成一个独立于具体数据库管理系统的概念模型，能真实、充分地反映现实世界，包括事物、事物自身的属性和事物之间的联系，能满足用户对数据的处理要求。

本课程设计采用E-R图，以实体关系图的形式来完成概念结构设计：通过对收集到的数据进行分类、组织、确定实体、实体的属性、实体之间的联系类型，形成E-R图。

根据需求分析，我们建立了如下的E-R模型，如图3-3、图3-4所示：

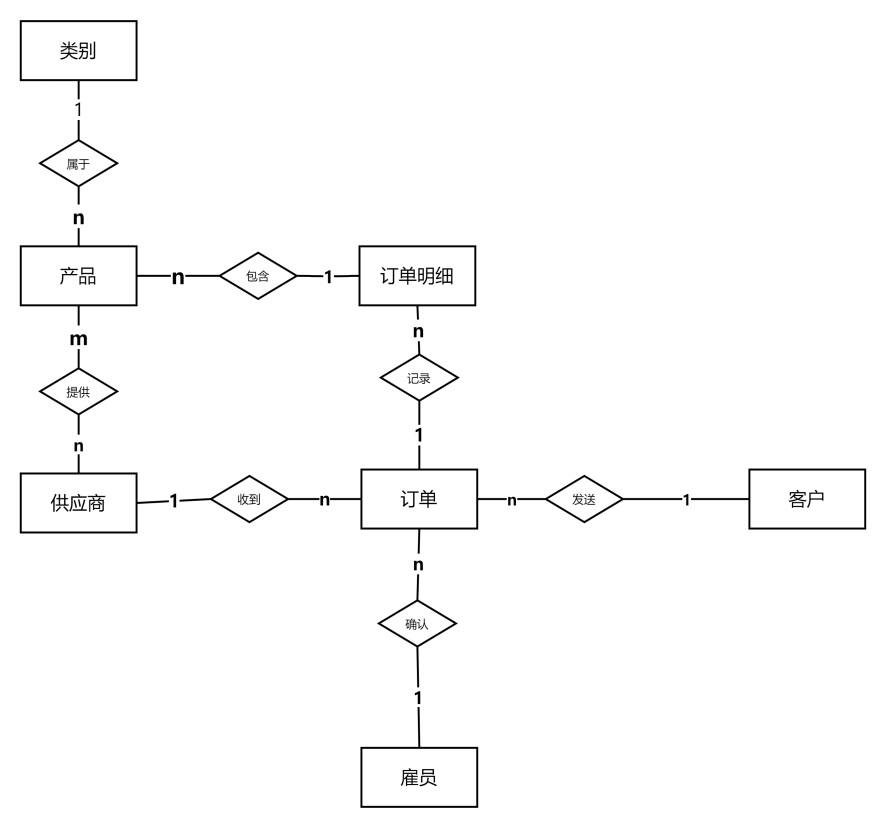


图3-4 E-R模型图（无属性版）

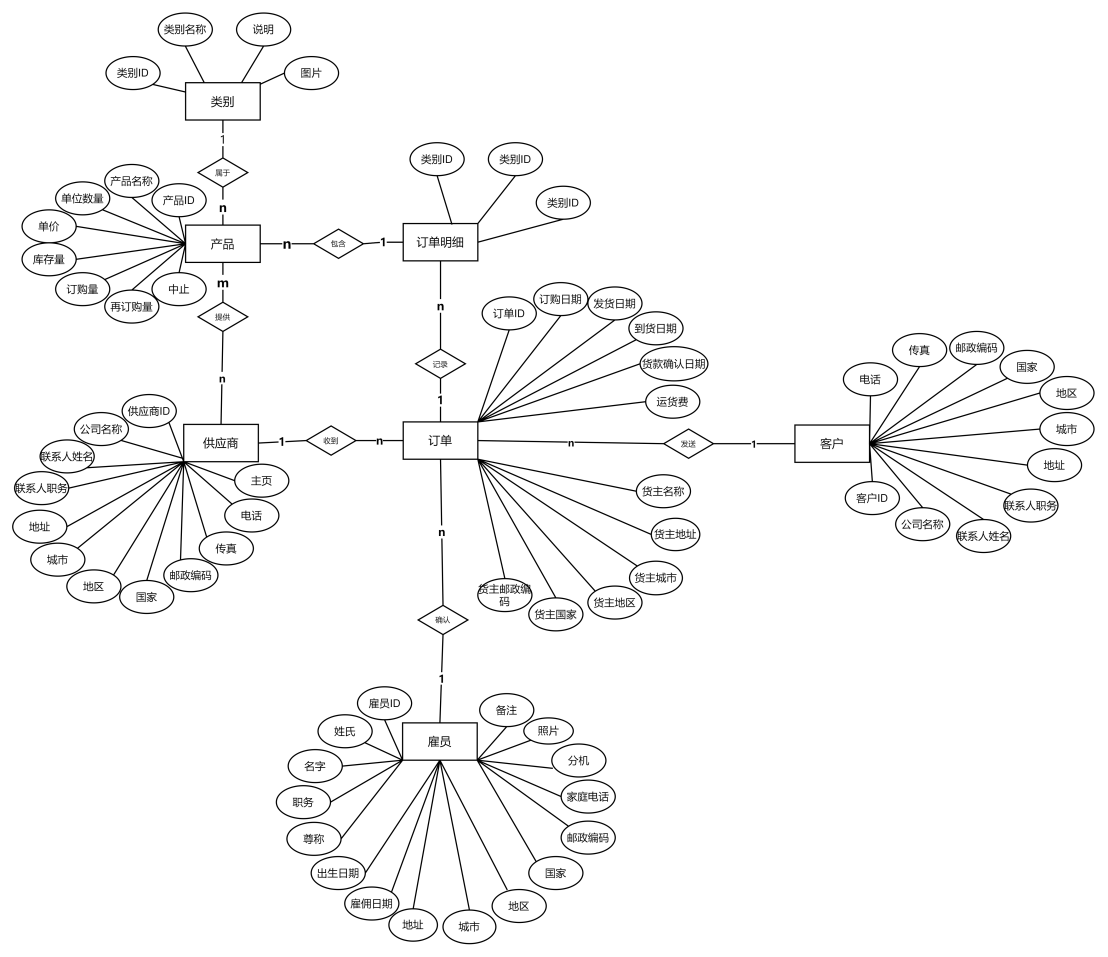
****

图3-3 E-R模型图（含属性版）

**2.4 数据库逻辑结构设计**

数据库逻辑结构设计即将数据库的概念结构模型（E-R图）转换为某个数据库管理系统所支持的数据模型，并对其进行修改与优化。

本课程设计采用关系数据库，需要分析实体名、实体所包含的属性特征、实体与实体、实体与属性之间关系，再根据关系规范化理论对关系进行优化、消除数据冗余，得到相对高级的范式，实现E-R图到关系模型的转换，保证数据的正常处理功能。

**2.4.1 E-R图向关系模型的转换**

分析并消除二义性、消除转换1：1关系、1：n关系、设计新关系存储m：n关系、并依据实际前端设计需要进行修改后，将上图（图3-3）E-R模型图初步转化为:

**（1）产品**（产品ID，产品名称，产品单位数量，产品单价，产品库存量，产品订购量，产品再订购量，产品中止，产品类别ID）

**（2）订单**（订单ID，订单订购日期，订单发货日期，订单到货日期，订单货款确认日期，订单运货费，订单货主名称，订单货主详细地址，订单货主所在国家，订单货主所在城市，订单货主所在地区，订单货主所在地邮政编码，订单折扣，客户ID，雇员ID，供应商ID）

**（3）订单明细**（订单ID，产品ID，数量，单项产品总金额）

**（4）雇员**（雇员ID，雇员姓氏，雇员名字，雇员职务，雇员尊称，雇员出生日期，雇员雇佣日期，雇员详细地址，雇员所在城市，雇员所在地区，雇员所在国家，雇员所在地邮政编码，雇员家庭电话，雇员电话分机，雇员照片，雇员备注，雇员上级，雇员登陆密码）

**（5）供应商**（供应商ID，供应商公司名称，供应商联系人姓名，供应商联系人职务，供应商详细地址，供应商所在国家，供应商所在城市，供应商所在地区，供应商所在地邮政编码，供应商电话，供应商传真，供应商网站主页，供应商登陆密码）

**（6）客户**（客户ID，客户公司名称，联系人姓名，联系人职务，客户详细地址，客户所在国家，客户所在城市，客户所在地区，客户所在地邮政编码，客户电话，客户传真，客户登录密码，客户总积分）

**（7）类别**（类别ID，类别名称，类别说明，类别图片）

**（8）供应商-产品关系**（产品ID，供应商ID，产品当前库存量）

**2.4.2 利用数据依赖的理论对关系模式进行分析**

分析上述关系模式表;

（1）1NF：数据库表中的字段都是单一属性的，不可再分。

显然对于本次表的字段，基本都满足不可再分，则设计的表的关系满足1NF。

（2）2NF：数据库表中每列都与主键有关系，而不能只与主键的某部分有关系.

也就是任意表中，主键与非主键都遵循完全函数依赖关系。本次共8个表，其中有6个表主键为单独的属性，则其他属性都与主键遵循完全函数依赖，而另外的两个表：

“订单明细”表；非主键的属性：数量、单项产品总金额，由主键属性组（订单ID，产品ID）中的两个主属性共同决定，不存在其中只凭借一个主属性唯一标识一个元组的情况。

“供应商-产品关系”表同理，非主属性：产品当前库存量由主键的属性组（产品ID，供应商ID）中的两个主属性共同决定。

因此每个表都满足2NF。

（3）3NF：主属性以外的键之间不存在依赖。

对于雇员、客户、供应商三个表，其中的详细地址属性与所在国家、城市、地区属性，在我们的设计中由用户自行填写，可能出错，且不存在由详细地址自动导出所在国家、城市、地区的过程，由我们判断不存在依赖。

因此，上述设计的关系表达到3NF，满足要求。

**2.4.3最终关系模式设计**

**（1）产品**（产品ID，产品名称，产品单位数量，产品单价，产品库存量，产品订购量，产品再订购量，产品中止，产品类别ID）

主键为产品ID。

外键为产品类别ID（参照类别表的类别ID属性）。

对产品库存量，产品订购量，产品再订购量，产品中止，产品类别ID设置初始值。

设置CHECK约束：产品ID约束、产品库存量约束。

**（2）订单**（订单ID，订单订购日期，订单发货日期，订单到货日期，订单货款确认日期，订单运货费，订单货主名称，订单货主详细地址，订单货主所在国家，订单货主所在城市，订单货主所在地区，订单货主所在地邮政编码，订单折扣，客户ID，雇员ID，供应商ID）

主键为订单ID。

外键为客户ID（参照客户表的客户ID属性）、雇员ID（参照雇员表的雇员ID属性）、供应商ID（参照供应商表的供应商ID属性）。

设置CHECK约束：订单ID约束。

**（3）订单明细**（订单ID，产品ID，数量，单项产品总金额）

主键为（订单ID，产品ID）属性组。

外键为订单ID（参照订单表的订单ID属性）、产品ID（参照产品表的产品ID属性）。

**（4）雇员**（雇员ID，雇员姓氏，雇员名字，雇员职务，雇员尊称，雇员出生日期，雇员雇佣日期，雇员详细地址，雇员所在城市，雇员所在地区，雇员所在国家，雇员所在地邮政编码，雇员家庭电话，雇员电话分机，雇员照片，雇员备注，雇员上级，雇员登陆密码）

主键为雇员ID。

设置初始值：雇员登陆密码。

**（5）供应商**（供应商ID，供应商公司名称，供应商联系人姓名，供应商联系人职务，供应商详细地址，供应商所在国家，供应商所在城市，供应商所在地区，供应商所在地邮政编码，供应商电话，供应商传真，供应商网站主页，供应商登陆密码）

主键为供应商ID。

设置初始值：供应商登陆密码。

设置CHECK约束：供应商详细地址约束，供应商所在地邮政编码约束，供应商电话约束，供应商ID约束。

**（6）客户**（客户ID，客户公司名称，联系人姓名，联系人职务，客户详细地址，客户所在国家，客户所在城市，客户所在地区，客户所在地邮政编码，客户电话，客户传真，客户登录密码，客户总积分）

主键为客户ID。

设置初始值：客户登陆密码、客户总积分。

设置CHECK约束：客户ID约束、客户电话约束。

**（7）类别**（类别ID，类别名称，类别说明，类别图片）

主键为类别ID。

设置CHECK约束：类别ID约束，类别名称约束。

**（8）供应商-产品关系**（产品ID，供应商ID，产品当前库存量）

主键为（产品ID，供应商ID）属性组。

外键为产品ID（参照产品表的产品ID属性）、供应商ID（参照供应商表的供应商ID属性）。

设置初始值：产品当前库存量。

**2.5 数据库物理结构设计**

数据库在物理设备上的存储结构与存取方法称为数据库的物理结构，它依赖于选定的数据库管理系统。在逻辑设计的基础上为一个给定的逻辑数据模型选取一个最适合应用要求，能够高效、稳定、安全地支持业务应用的物理结构的过程，就是数据库的物理设计。

数据库的物理设计通常包括两步：（1）确定数据库的物理结构，在关系数据库中主要指存取方法和存储结构；（2）对物理结构进行评价，重点是时间和空间效率。[1]

其中最重要的就是确定数据库的物理结构（存储结构）：

**2.5.1确定存储结构**

我们采用的数据库平台是SQL Server 2022，数据库以文件形式存储在计算机磁盘中，其中主要的文件有主数据文件，次数据文件，事物日志文件：

（1）主数据文件

每个数据库有且只能有一个主数据文件，扩展名是.mdf，包含了数据库的初始信息、重要的元数据信息，并记录数据库还拥有哪些文件，用于存储表、索引、视图、存储过程等数据库对象的实际数据。

另外，主数据文件负责支持数据库的正常运行，因此必须随时保障主数据文件的完整性、安全性。

（2）次数据文件

数据库中，除了主数据文件以外的所有其他数据文件都是次数据库文件，扩展名是.ndf。次数据文件可以用于储存数据与对象，可以独立管理物理存储空间，使逻辑上连续的对象不一定要存储于连续的物理地址。也可以在创建或使用数据库的途中分配存储空间，可以适应不同自动增长大小，灵活适应不同需求。

此数据不是数据库必需的文件，可以有多个。

（3）事务日记文件（日志文件）

每个数据库至少拥有一个自己的日记文件（也可以拥有多个日记文件）。后缀是是.ldf，用来记录数据库的事务以及每个数据对数据库所做的所有的修改操作，用于数据库恢复阶段（根据日志文件逐项还原已经执行的操作，使数据库回到故障前的状态）。

（4）索引设计

对于数据库的索引，SQL Server会自动给所有表的主键建立聚集索引。因此我们只需要对部分常用于查询且重复值较小的外键建立非聚集索引即可（聚集索引一张表只能建立一个）。

详见本篇3.2.4。

2.6 模块划分

按照上述分析将系统最终划分为如下5个模块:

（1）注册、登录模块：

a. 注册模块：

无论是雇员、客户还是供货商，如果希望使用本平台的功能则必须首先完成注册，获取系统分配的账号，可选择性修改系统初始密码12345678，并且在丰富完整基础信息（包括但不仅限于地址、联系方式、姓名称呼等），即可提交注册申请并使用本平台。

b. 登录模块：

无论是雇员、客户还是供货商，都使用各自的登陆界面：

* 客户填写消费用户账号、密码登录平台，若密码验证正确则进入客户用户界面，不正确则需要重新输入。
* 雇员需要填写雇员用户账号、密码登录平台，若密码验证正确则进入雇员用户界面，不正确则需要重新输入。
* 供应商填写供应商用户账号、密码登录平台，若密码验证正确则进入供应商用户界面，不正确则需要重新输入。

c. 密码重置子模块：

若在登录时忘记密码，点击“忘记密码”，输入ID号恢复为系统初始密码12345678。

（2）客户用户模块：

客户用户模块又分成如下4个子模块：

a. 使用积分兑换礼品子模块。

b. 购买商品子模块。

c. 查看订单信息子模块。

d. 个人信息管理模块。

（3）雇员用户模块：

雇户用户模块又分为如下2个子模块：

a.确认客户订单并选择供应商子模块。

b.个人信息管理子模块。

（4）供应商用户模块：

供应商模块又分成如下4个子模块：

a.查看商品库存子模块。

b.修改商品信息子模块。

c.订单发货子模块。

d.个人信息管理子模块。

（5）数据导入模块：

数据导入只有一个界面与一个模块：

a.通过选择本地Excel文件进行顺序导入。

**3 实现**

**3.1 前端界面实现**

这次数据库课程设计的前端界面开发主要借助PyCharm集成开发环境中的Qt Designer来实现项目前端界面的设计和开发工作。

利用Qt Designer，可以较为方便、快捷地实现所需界面的设计。在实际操作中，只需要将所需的控件拖拽至界面，再设置其属性，如位置、大小等，Qt Designer将自动生成后缀名为.ui的文件，记录当前界面和控件的属性信息。通过这种交互式的界面设计方法，大大减轻了程序员的工作量，使得界面开发更为轻松、便捷。

开发界面如图4-1所示。

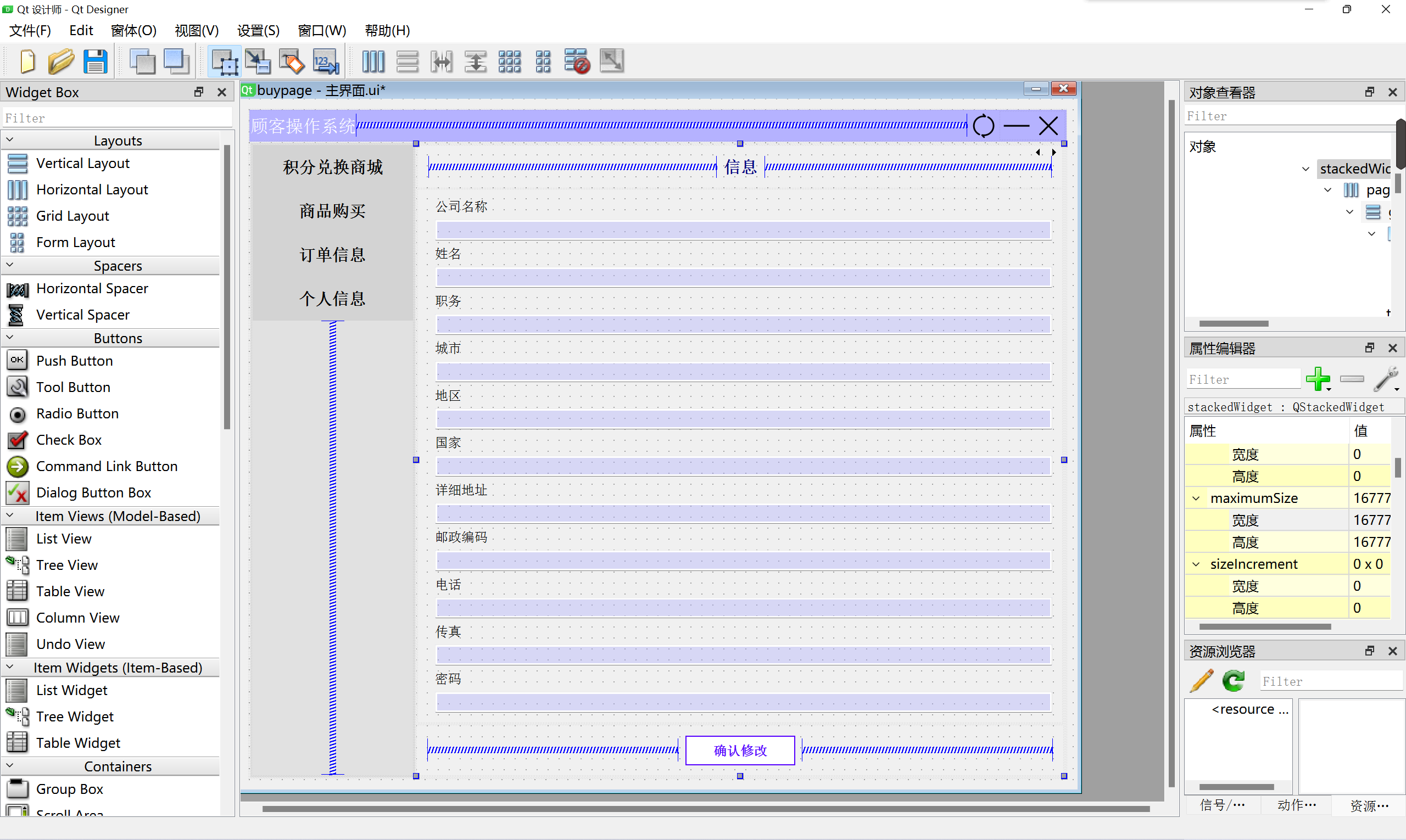
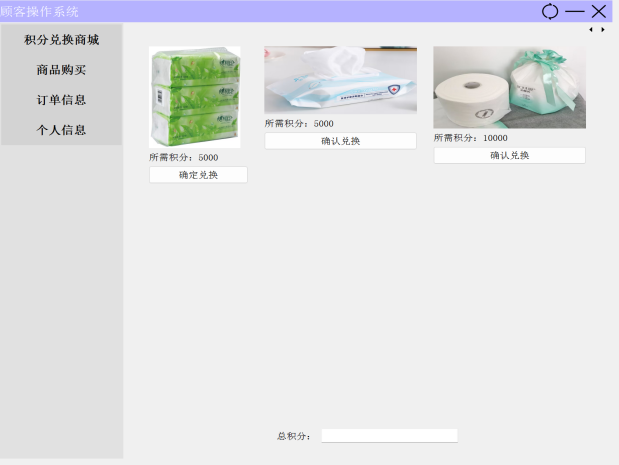
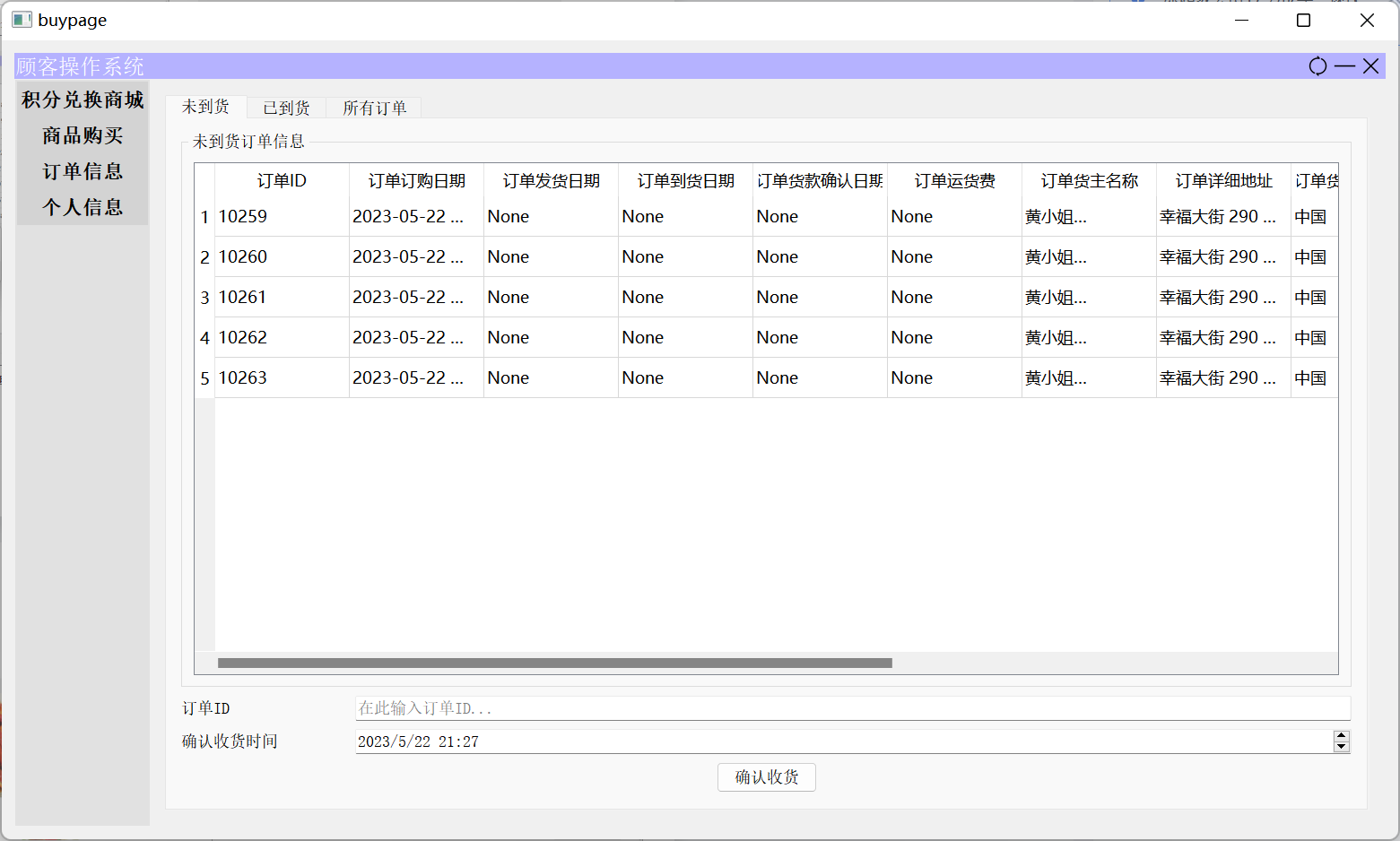


图4-1 PySide2—Qt Designer前端界面开发设计器

**3.1.1 用户界面设计**

用户界面如图4-2-1至图4-2-5所示。其中主要展示了登录界面、积分兑换界面、订单详情界面、个人修改信息功能界面等主要界面设计。

图4-2-1 登录界面 图4-2-2 用户积分兑换商城界面

图4-2-3购物车界面及下单弹窗 图4-2-4订单详情页面（未到货订单）

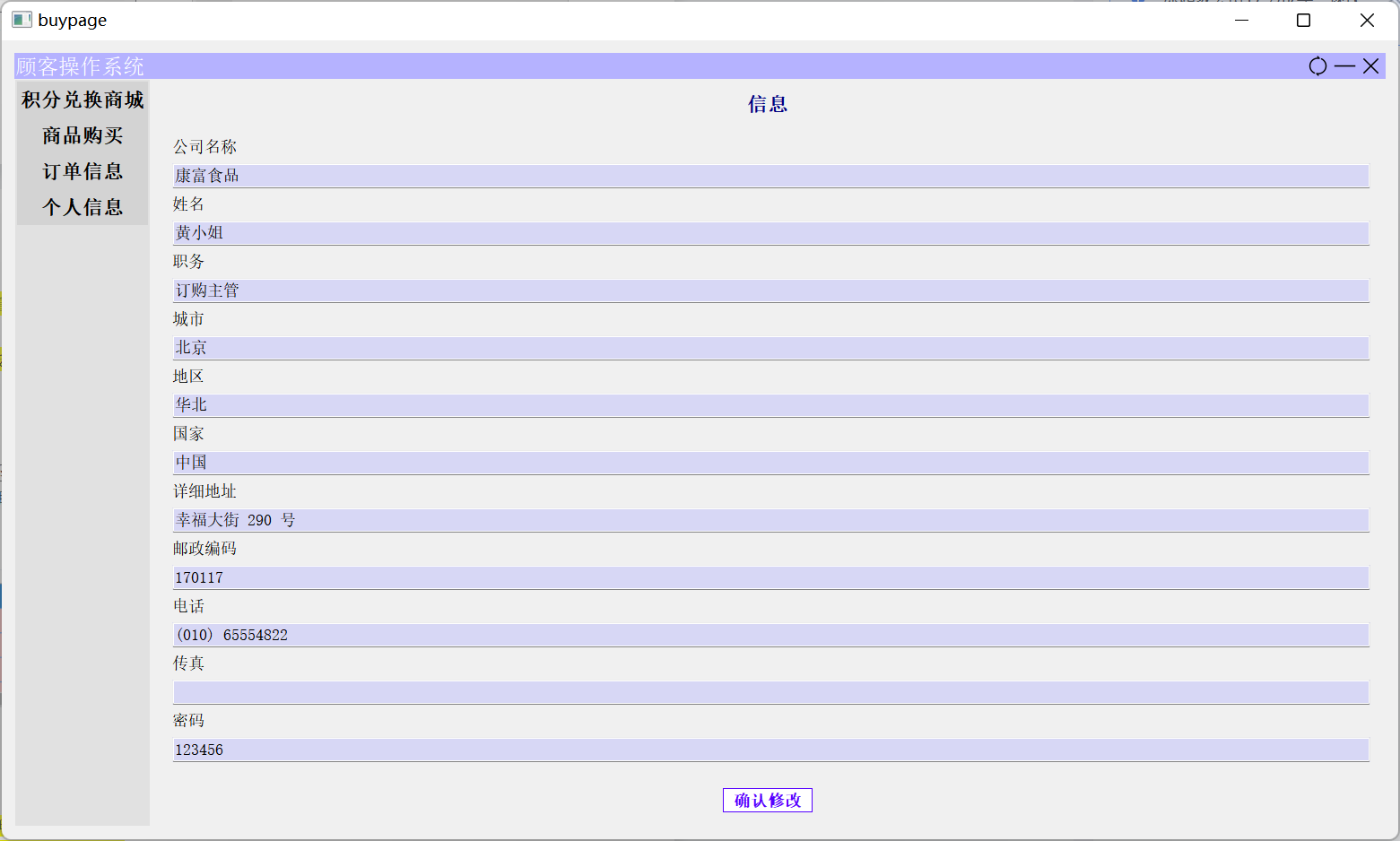


图4-2-5 个人信息修改界面

**3.1.2 雇员界面设计**

雇员界面如图4-2-6至图4-2-8所示。登录界面与菜单界面与用户的界面类似，主要展示雇员的具体功能界面设计，包括订单确认。





图4-2-6 雇员登录界面 图4-2-7 个人信息修改界面



图4-2-8 订单确认界面

**3.1.3 供应商界面设计以及其他界面**

供应商的主要功能是对商品信息管理，包括增加和减少商品，修改商品库存和中止状态等。

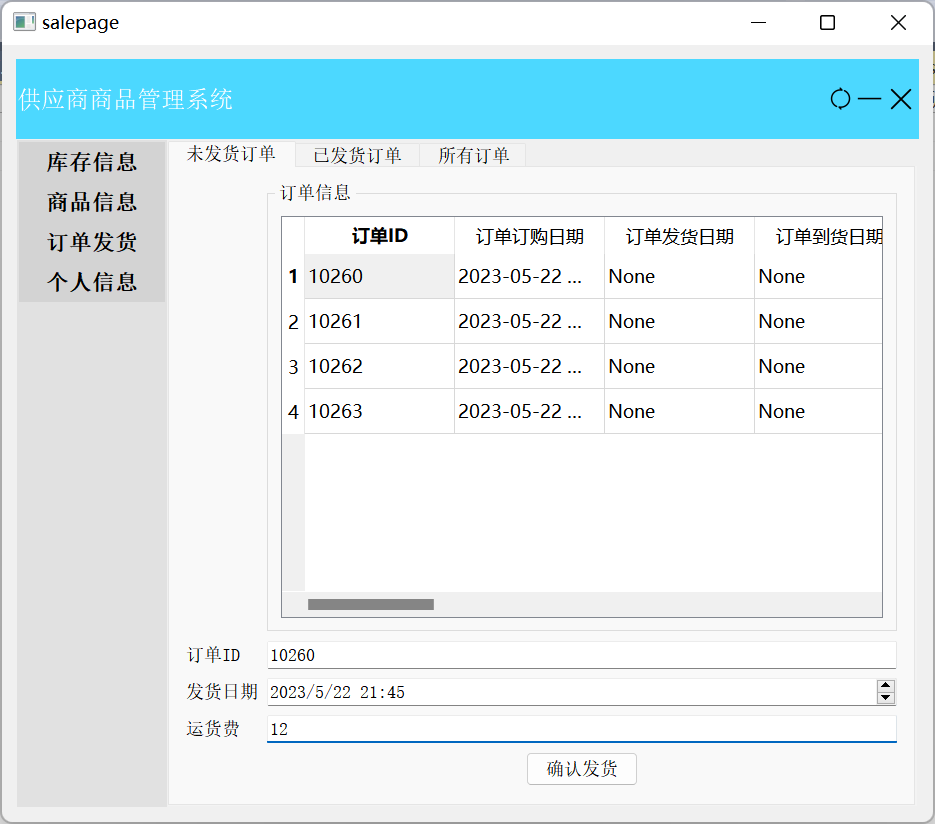


 图4-2-11 供应商发货界面 图4-2-12 库存管理界面

图4-2-13 重置密码界面 图4-2-14 注册界面

**3.2 后端代码实现**

此次数据库课程设计的代码量较多且由于性能相近而重复部分较多，具有冗余性，为此仅选取核心和关键部分的代码进行说明和解释。

**3.2.1 Python与SQL Server的连接与交互**

**（1）**连接到数据库

为了实现连接到数据库的操作，便于进行后续的复杂操作，利用了pymssql（一个Python数据库库），用于连接并进行后续的数据库交互。

连接操作是通过利用pymssql库中提供的connect函数完成，代码示例如下：

**/\*数据库连接代码\*/**

import pymssql # 引用pymssql库

servername = "LAPTOP-07LLA2EL\MSSQLSERVER2022" # IP地址号

port = 1433 # 通信端口号

user = "sa" # 数据库所有者登录名

password = "12345678" # 数据库登录名对应的登录密码

database = "DataBase" # 连接的数据库名

charset = "GBK" # 编码格式”GBK”

# 建立连接

connect = pymssql.connect(servername, user, password, database, charset)

其中，connect是一个连接对象，它是pymssql.connect()函数返回的结果，该对象用于以后续操作中管理与SQL Server数据库的连接，也提供了与数据库交互所需要的相关功能和方法。

**（2）**与数据库进行交互（部分功能代码）

通过利用connect对象创建游标对象，执行SQL语句、获取结果、提交事务等等。

查询操作示例代码如下：

**/\*查询操作代码\*/**

cursor = connect.cursor() # 创建一个游标对象来进行SQL查询语句

query = "SELECT \* FROM 订单 WHERE 订单贷款确认日期 IS NULL" # 编写查询语句

cursor.execute(query) # 执行查询

resluts = cursor.fetchall() # 获取查询结果

cursor.close() # 关闭游标

利用变量存储查询结果，便于后续进行数据在前端界面的显示和处理。

更新操作示例代码如下：

**/\* 更新操作代码 \*/**

cursor = connect.cursor() # 编写更新语句

query = "UPDATE 订单 SET 订单贷款确认日期 = %s WHERE 订单ID = %s"

cursor.execute(query, (qdatetime, text\_from\_line\_edit))

connect.commit() # 提交更新事务

cursor.close()

其中，qdatetime和text\_from\_line\_edit为存储着代替占位符实际值的变量。

**3.2.2 通过PySide2进行前端界面和后端数据库的相关操作**

通过使用PySide2创建丰富界面的应用程序，并利用Python的强大的功能与SQL Server数据库进行交互，实时地将数据库中的数据显示在前端界面上，也能将前端界面中对数据的更改实时更新在数据库中。

在本次课程设计中，通过设计UI类的方法，将各种功能函数封装进了UI类中，在主函数和UI类的初始化函数中进行对应功能函数的调用，实现了基本的后端数据库与前端界面交互操作。

以下节选了一些代码，来说明通过PySide2，前后端是如何进行交互的。

1. 登录函数的设计

**/\* 登录界面代码 \*/**

def Clac2login(self):

self.UserName = self.loginui.textUser.text() # 获取用户名

self.GID = self.UserName

self.KeyWord = self.loginui.textPwd.text() # 获取密码

cursor = connect.cursor()

sql\_select = "SELECT 供应商ID, 供应商登陆密码 FROM 供应商"

cursor.execute(sql\_select)

results = cursor.fetchall()

sql\_select = "SELECT COUNT(\*) FROM 供应商"

cursor.execute(sql\_select)

flag = cursor.fetchall()

flag = int(flag[0][0])

for result in results:

if int(self.UserName) == result[0] and str(self.KeyWord) == result[1]:

QMessageBox.about(self.loginui, "提示", "登陆成功!\n欢迎回来!") # 弹窗

self.loginui.close()

self.mainui.show()

self.table2GInf()

self.table2Sallshow()

self.table2Dallshow()

self.table2Sinfshow()

break

else:

flag = flag - 1

continue

if not flag: # 弹窗

QMessageBox.about(self.loginui, "提示", "供应商ID或密码错误, 请重试")

1. 从数据库中获取产品信息并显示到QTableWidget表格上

**/\* 表格显示代码 \*/**

def table2Sallshow(self):

cursor = connect.cursor()

query = "select 产品ID, 产品名称, 产品库存量, 产品订购量, 产品再订购量, 产品中止 from 产品 "

cursor.execute(query)

resluts = cursor.fetchall()

self.mainui.tableSAll.setRowCount(len(resluts))

for i, row in enumerate(resluts):

for j, cell in enumerate(row):

self.mainui.tableSAll.setItem(i,j,QTableWidgetItem(str(cell)))

1. 根据QLineEdit中输入的订单号检索已发送订单，并显示到QTableWidget表格上

**/\* 检索已发送订单并显示代码 \*/**

def Clac2Dsearchsended(self):

textofsearch = self.mainui.text2Dsearchsended.text()

cursor = connect.cursor()

query = "SELECT \* FROM 订单 where 订单ID = %s and 订单发货日期 is not null"

cursor.execute(query, (textofsearch,))

resluts = cursor.fetchall()

self.mainui.table2Dsended.setRowCount(len(resluts))

for i, row in enumerate(resluts):

for j, cell in enumerate(row):

self.mainui.table2Dsended.setItem(i, j, QTableWidgetItem(str(cell)))

其中，该函数绑定到了前端界面的一个QPushButton对象上，当按钮按下时，会激活该函数，完成相应的动作。

1. 根据客户下单的时间进行订单的生成并入库

**/\*订单生成代码\*/**

def Clac2Dright(self):

result\_list = []

last\_cell\_value = None

first\_cell\_value = None

mid\_cell\_value = None

sum = 0

for row in range(self.mainui.table2Kshopping.rowCount()):

mid\_col = 3

last\_column = 4

if self.mainui.table2Kshopping.item(row, last\_column) is not None:

last\_cell\_value = self.mainui.table2Kshopping.item(row, last\_column).text()

mid\_cell\_value = self.mainui.table2Kshopping.item(row, mid\_col).text()

first\_cell\_value = self.mainui.table2Kshopping.item(row, 0).text()

if last\_cell\_value:

result\_list.append((first\_cell\_value, mid\_cell\_value, last\_cell\_value))

last\_cell\_value = None

first\_cell\_value = None

mid\_cell\_value = None

if result\_list:

text2Dname = self.mainui.text2KName.text()

text2DAdrr = self.mainui.text2KAddr.text()

text2Dbtime = datetime.now()

text2DCountry = self.mainui.text2KCountry.text()

text2DCity = self.mainui.text2KCity.text()

text2DArea = self.mainui.text2KArea.text()

text2Decode = self.mainui.text2Kecode.text()

cursor = connect.cursor()

query = "SELECT MAX(订单ID) FROM 订单"

cursor.execute(query)

results = cursor.fetchall()

new\_ID = int(results[0][0]) + 1

data = (new\_ID, text2Dbtime, text2Dname, text2DAdrr, text2DCountry, text2DCity, text2DArea, text2Decode)

cursor = connect.cursor()

sql\_insert = "INSERT INTO 订单 (订单ID, 订单订购日期, 订单货主名称, 订单货主详细地址, 订单货主所在国家, 订单货主所在城市, 订单货主所在地区, 订单货主所在地邮政编码) VALUES (%d, %s, %s, %s, %s, %s, %s, %s)"

cursor.execute(sql\_insert, data)

connect.commit()

cursor.close()

cursor = connect.cursor()

for result in result\_list:

money = Decimal(float(result[1]) \* int(result[2]))

sum = sum + money

data = (new\_ID, int(result[0]), int(result[2]), money)

sql\_insert = "INSERT INTO 订单明细 (订单ID, 产品ID, 数量, 单项产品总金额) VALUES (%d, %d, %d, %s)"

cursor.execute(sql\_insert, data)

connect.commit()

cursor.close()

QMessageBox.about(self.mainui, "提示", "下单成功, 共花费" + str(sum) + "元")

# 遍历所有行

for row in range(self.mainui.table2Kshopping.rowCount()):

# 删除最后一列中的数据

self.mainui.table2Kshopping.setItem(row,4,QTableWidgetItem(""))

1. 根据QDateTimeEdit中的时间进行订单确认

**/\*订单确认代码\*/**

def Clac2Dget(self):

qdate = self.mainui.datatime2Drig.dateTime()

qdatetime = qdate.toPython()

ID = self.mainui.text2DID.text()

cursor = connect.cursor()

query = "UPDATE 订单 SET 订单到货日期 = %s WHERE 订单ID = %s"

cursor.execute(query, (qdatetime, ID,))

connect.commit()

cursor.close()

QMessageBox.about(self.mainui, "提示", "签收成功") # 弹窗

self.table2Dallshow() # 更新数据

**3.2.3 SQL Server建立数据库**

建立以下8个基本表（表4-1至表4-8）

注：PK，即Primary Key，主码。FK，即Foreign Key，外码。

表4-1 订单（表）

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 序号 | 列名 | 数据类型 | 数据长度 | 码 | 是否可空 |
| 1 | 订单ID | int |  | PK | 否 |
| 2 | 订单订购日期 | datetime |  |  | 是 |
| 3 | 订单发货日期 | datetime |  |  | 是 |
| 4 | 订单到货日期 | datetime |  |  | 是 |
| 5 | 订单货款确认日期 | datetime |  |  | 是 |
| 6 | 订单运货费 | money |  |  | 是 |
| 7 | 订单货主名称 | nvarchar | 80 |  | 是 |
| 8 | 订单货主详细地址 | nvarchar | 120 |  | 否 |
| 9 | 订单货主所在国家 | nvarchar | 30 |  | 是 |
| 10 | 订单货主所在城市 | nvarchar | 30 |  | 是 |
| 11 | 订单货主所在地区 | nvarchar | 30 |  | 是 |
| 12 | 订单货主所在地邮政编码 | nvarchar | 20 |  | 否 |
| 13 | 订单折扣 | money |  |  | 是 |
| 14 | 客户ID | nvarchar | 10 | FK | 是 |
| 15 | 雇员ID | int |  | FK | 是 |
| 16 | 供应商ID | int |  | FK | 是 |

表4-2 雇员（表）

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 序号 | 列名 | 数据类型 | 数据长度 | 码 | 是否可空 |
| 1 | 雇员ID | int |  | PK | 否 |
| 2 | 雇员姓氏 | nvarchar | 40 |  | 否 |
| 3 | 雇员名字 | nvarchar | 20 |  | 否 |
| 4 | 雇员职务 | nvarchar | 60 |  | 否 |
| 5 | 雇员尊称 | nvarchar | 50 |  | 是 |
| 6 | 雇员出生日期 | datetime |  |  | 否 |
| 7 | 雇员雇佣日期 | datetime |  |  | 否 |
| 8 | 雇员详细地址 | nvarchar | 120 |  | 否 |
| 9 | 雇员所在国家 | nvarchar | 30 |  | 是 |
| 10 | 雇员所在城市 | nvarchar | 30 |  | 是 |
| 11 | 雇员所在地区 | nvarchar | 30 |  | 是 |
| 12 | 雇员所在地邮政编码 | nvarchar | 20 |  | 否 |
| 13 | 雇员家庭电话 | nvarchar | 48 |  | 是 |
| 14 | 雇员电话分机 | nvarchar | 8 |  | 是 |
| 15 | 雇员照片 | nvarchar | MAX |  | 是 |
| 16 | 雇员备注 | ntext |  |  | 是 |
| 17 | 雇员上级 | int |  |  | 是 |
| 18 | 雇员登陆密码 | nvarchar | 30 |  | 否 |

表4-3 客户（表）

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 序号 | 列名 | 数据类型 | 数据长度 | 码 | 是否可空 |
| 1 | 客户ID | nvarchar | 10 | PK | 否 |
| 2 | 客户公司名称 | nvarchar | 80 |  | 否 |
| 3 | 联系人姓名 | nvarchar | 60 |  | 否 |
| 4 | 联系人职务 | nvarchar | 60 |  | 是 |
| 5 | 客户详细地址 | nvarchar | 120 |  | 否 |
| 6 | 客户所在国家 | datetime | 30 |  | 是 |
| 7 | 客户所在城市 | datetime | 30 |  | 是 |
| 8 | 客户所在地区 | nvarchar | 30 |  | 是 |
| 9 | 客户所在地邮政编码 | nvarchar | 20 |  | 否 |
| 10 | 客户电话 | nvarchar | 48 |  | 否 |
| 11 | 客户传真 | nvarchar | 48 |  | 是 |
| 12 | 客户登录密码 | nvarchar | 30 |  | 否 |
| 13 | 客户总积分 | money |  |  | 是 |

表4-4 订单明细（表）

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 序号 | 列名 | 数据类型 | 数据长度 | 码 | 是否可空 |
| 1 | 订单ID | int |  | PK | 否 |
| 2 | 产品ID | int |  | PK | 否 |
| 3 | 数量 | smallint |  |  | 否 |
| 4 | 单项产品总金额 | money |  |  | 是 |

表4-5 供应商（表）

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 序号 | 列名 | 数据类型 | 数据长度 | 码 | 是否可空 |
| 1 | 供应商ID | int |  | PK | 否 |
| 2 | 供应商公司名称 | nvarchar | 80 |  | 否 |
| 3 | 供应商联系人姓名 | nvarchar | 60 |  | 否 |
| 4 | 供应商联系人职务 | nvarchar | 60 |  | 是 |
| 5 | 供应商详细地址 | nvarchar | 120 |  | 否 |
| 6 | 供应商所在国家 | datetime | 30 |  | 是 |
| 7 | 供应商所在城市 | datetime | 30 |  | 是 |
| 8 | 供应商所在地区 | nvarchar | 30 |  | 是 |
| 9 | 供应商所在地邮政编码 | nvarchar | 20 |  | 否 |
| 10 | 供应商电话 | nvarchar | 48 |  | 否 |
| 11 | 供应商传真 | nvarchar | 48 |  | 是 |
| 12 | 供应商网站主页 | ntext |  |  | 是 |
| 13 | 供应商登陆密码 | nvarchar | 30 |  | 否 |

表4-6 产品（表）

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 序号 | 列名 | 数据类型 | 数据长度 | 码 | 是否可空 |
| 1 | 产品ID | int |  | PK | 否 |
| 2 | 产品名称 | nvarchar | 80 |  | 否 |
| 3 | 产品单位数量 | nvarchar | 40 |  | 否 |
| 4 | 产品单价 | money |  |  | 否 |
| 5 | 产品库存量 | smallint |  |  | 否 |
| 6 | 产品订购量 | smallint |  |  | 否 |
| 7 | 产品再订购量 | smallint |  |  | 是 |
| 8 | 产品中止 | bit |  |  | 是 |
| 9 | 产品类别ID | int |  | FK | 是 |

表4-7 类别（表）

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 序号 | 列名 | 数据类型 | 数据长度 | 码 | 是否可空 |
| 1 | 类别ID | int |  | PK | 否 |
| 2 | 类别名称 | nvarchar | 30 |  | 否 |
| 3 | 类别说明 | ntext |  |  | 否 |
| 4 | 类别图片 | nvarchar | MAX |  | 是 |

表4-8 供应商-产品关系（表）

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 序号 | 列名 | 数据类型 | 数据长度 | 码 | 是否可空 |
| 1 | 产品ID | int |  | PK | 否 |
| 2 | 供应商ID | int |  | PK | 否 |
| 3 | 产品当前库存量 | smallint |  |  | 否 |

数据库关系图如图4-9所示：



图4-9 数据库关系图

根据数据库逻辑结构设计，编写SQL语句建立数据库。

具体实现代码如下：

**/\* 建立产品表、定义主码 \*/**

CREATE TABLE [dbo].[产品]

(

[产品ID] [int] NOT NULL,

[产品名称] [nvarchar](80) NOT NULL,

[产品单位数量] [nvarchar](40) NOT NULL,

[产品单价] [money] NOT NULL,

[产品库存量] [smallint] NOT NULL,

[产品订购量] [smallint] NOT NULL,

[产品再订购量] [smallint] NULL,

[产品中止] [bit] NULL,

[产品类别ID] [int] NULL,

CONSTRAINT [PK\_产品表] PRIMARY KEY CLUSTERED ([产品ID] ASC)

)

**/\* 添加产品表上完整性约束、设置默认值 \*/**

ALTER TABLE [dbo].[产品]

ADD CONSTRAINT [DF\_产品\_产品库存量] DEFAULT ((0)) FOR [产品库存量];

ALTER TABLE [dbo].[产品]

ADD CONSTRAINT [DF\_产品\_产品订购量] DEFAULT ((0)) FOR [产品订购量];

ALTER TABLE [dbo].[产品]

ADD CONSTRAINT [DF\_产品\_产品再订购量] DEFAULT ((0)) FOR [产品再订购量];

ALTER TABLE [dbo].[产品]

ADD CONSTRAINT [DF\_产品\_产品中止] DEFAULT ((1)) FOR [产品中止];

ALTER TABLE [dbo].[产品]

ADD CONSTRAINT [DF\_产品\_产品类别ID] DEFAULT ((0)) FOR [产品类别ID];

ALTER TABLE [dbo].[产品]

WITH CHECK ADD CONSTRAINT [产品类别ID外键约束] FOREIGN KEY([产品类别ID])

REFERENCES [dbo].[类别] ([类别ID]);

ALTER TABLE [dbo].[产品]

WITH CHECK ADD CONSTRAINT [产品ID约束] CHECK (([产品ID]>(0)));

ALTER TABLE [dbo].[产品]

WITH CHECK ADD CONSTRAINT [产品库存量约束] CHECK (([产品库存量]>=(0)));

**/\* 建立订单表、设置主键 \*/**

CREATE TABLE [dbo].[订单](

[订单ID] [int] NOT NULL,

[订单订购日期] [datetime] NULL,

[订单发货日期] [datetime] NULL,

[订单到货日期] [datetime] NULL,

[订单货款确认日期] [datetime] NULL,

[订单运货费] [money] NULL,

[订单货主名称] [nvarchar](80) NULL,

[订单货主详细地址] [nvarchar](120) NULL,

[订单货主所在国家] [nvarchar](30) NULL,

[订单货主所在城市] [nvarchar](30) NULL,

[订单货主所在地区] [nvarchar](30) NULL,

[订单货主所在地邮政编码] [nvarchar](20) NOT NULL,

[订单折扣] [money] NULL,

[客户ID] [nvarchar](10) NULL,

[雇员ID] [int] NULL,

[供应商ID] [int] NULL,

CONSTRAINT [PK\_订单表] PRIMARY KEY CLUSTERED ([订单ID] ASC)

)

**/\* 添加订单表上完整性约束 \*/**

ALTER TABLE [dbo].[订单]

WITH CHECK ADD CONSTRAINT [供应商ID外键约束] FOREIGN KEY([供应商ID])

REFERENCES [dbo].[供应商] ([供应商ID]);

ALTER TABLE [dbo].[订单]

WITH CHECK ADD CONSTRAINT [雇员ID外键约束] FOREIGN KEY([雇员ID])

REFERENCES [dbo].[雇员] ([雇员ID]);

ALTER TABLE [dbo].[订单]

WITH CHECK ADD CONSTRAINT [客户ID外键约束] FOREIGN KEY([客户ID])

REFERENCES [dbo].[客户] ([客户ID]);

ALTER TABLE [dbo].[订单]

WITH CHECK ADD CONSTRAINT [订单ID约束] CHECK (([订单ID]>=(0)));

**/\* 建立订单明细表、设置主键 \*/**

CREATE TABLE [dbo].[订单明细](

[订单ID] [int] NOT NULL,

[产品ID] [int] NOT NULL,

[数量] [smallint] NOT NULL,

[单项产品总金额] [money] NULL,

CONSTRAINT [PK\_订单详细] PRIMARY KEY CLUSTERED([订单ID] ASC,[产品ID] ASC)

)

**/\* 添加订单明细表上完整性约束、设置默认值 \*/**

ALTER TABLE [dbo].[订单明细]

WITH NOCHECK ADD CONSTRAINT [产品ID外键约束] FOREIGN KEY([产品ID])

REFERENCES [dbo].[产品] ([产品ID]);

ALTER TABLE [dbo].[订单明细]

WITH NOCHECK ADD CONSTRAINT [订单ID外键约束] FOREIGN KEY([订单ID])

REFERENCES [dbo].[订单] ([订单ID]);

**/\* 建立供应商表、设置主键 \*/**

CREATE TABLE [dbo].[供应商](

[供应商ID] [int] NOT NULL,

[供应商公司名称] [nvarchar](80) NOT NULL,

[供应商联系人姓名] [nvarchar](60) NOT NULL,

[供应商联系人职务] [nvarchar](60) NULL,

[供应商详细地址] [nvarchar](120) NOT NULL,

[供应商所在国家] [nvarchar](30) NULL,

[供应商所在城市] [nvarchar](30) NULL,

[供应商所在地区] [nvarchar](30) NULL,

[供应商所在地邮政编码] [nvarchar](20) NOT NULL,

[供应商电话] [nvarchar](48) NOT NULL,

[供应商传真] [nvarchar](48) NULL,

[供应商网站主页] [ntext] NULL,

[供应商登陆密码] [nvarchar](30) NOT NULL,

CONSTRAINT [PK\_供应商表] PRIMARY KEY CLUSTERED ([供应商ID] ASC)

)

**/\* 添加供应商表上完整性约束 \*/**

ALTER TABLE [dbo].[供应商]

ADD CONSTRAINT [DF\_供应商\_供应商登陆密码] DEFAULT ((12345678)) FOR [供应商登陆密码];

ALTER TABLE [dbo].[供应商]

WITH CHECK ADD CONSTRAINT [供应商ID约束]

CHECK (([供应商ID]>=(0)));

ALTER TABLE [dbo].[供应商]

WITH CHECK ADD CONSTRAINT [供应商电话约束]

CHECK ((len([供应商电话])>=(10)));

ALTER TABLE [dbo].[供应商]

WITH CHECK ADD CONSTRAINT [供应商所在地邮政编码约束]

CHECK ((len([供应商所在地邮政编码])>=(4) AND len([供应商所在地邮政编码])<=(20)));

ALTER TABLE [dbo].[供应商]

WITH CHECK ADD CONSTRAINT [供应商详细地址]

CHECK ((len([供应商详细地址])>=(4)));

**/\* 建立客户表、设置主键 \*/**

CREATE TABLE [dbo].[客户](

[客户ID] [nvarchar](10) NOT NULL,

[客户公司名称] [nvarchar](80) NOT NULL,

[联系人姓名] [nvarchar](60) NOT NULL,

[联系人职务] [nvarchar](60) NULL,

[客户详细地址] [nvarchar](120) NOT NULL,

[客户所在国家] [nvarchar](30) NULL,

[客户所在城市] [nvarchar](30) NULL,

[客户所在地区] [nvarchar](30) NULL,

[客户所在地邮政编码] [nvarchar](20) NOT NULL,

[客户电话] [nvarchar](48) NOT NULL,

[客户传真] [nvarchar](48) NULL,

[客户登录密码] [nvarchar](30) NOT NULL,

[客户总积分] [money] NULL,

CONSTRAINT [PK\_客户] PRIMARY KEY CLUSTERED ([客户ID] ASC)

)

**/\* 添加客户表上完整性约束，设置初始值 \*/**

ALTER TABLE [dbo].[客户]

ADD CONSTRAINT [DF\_客户\_客户登录密码\_1]

DEFAULT ((12345678)) FOR [客户登录密码];

ALTER TABLE [dbo].[客户]

ADD CONSTRAINT [DF\_客户\_客户总积分]

DEFAULT ((0)) FOR [客户总积分];

ALTER TABLE [dbo].[客户]

WITH CHECK ADD CONSTRAINT [客户ID约束]

CHECK ((len([客户ID])=(5)));

ALTER TABLE [dbo].[客户]

WITH CHECK ADD CONSTRAINT [客户电话约束]

CHECK ((len([客户电话])>=(10)));

**/\* 建立雇员表、设置主键 \*/**

CREATE TABLE [dbo].[雇员](

[雇员ID] [int] NOT NULL,

[雇员姓氏] [nvarchar](40) NOT NULL,

[雇员名字] [nvarchar](20) NOT NULL,

[雇员职务] [nvarchar](60) NOT NULL,

[雇员尊称] [nvarchar](50) NULL,

[雇员出生日期] [datetime] NOT NULL,

[雇员雇佣日期] [datetime] NOT NULL,

[雇员详细地址] [nvarchar](120) NOT NULL,

[雇员所在国家] [nvarchar](30) NULL,

[雇员所在城市] [nvarchar](30) NULL,

[雇员所在地区] [nvarchar](30) NULL,

[雇员所在地邮政编码] [nvarchar](20) NOT NULL,

[雇员家庭电话] [nvarchar](48) NULL,

[雇员电话分机] [nvarchar](8) NULL,

[雇员照片] [nvarchar](510) NULL,

[雇员备注] [ntext] NULL,

[雇员上级] [int] NULL,

[雇员登陆密码] [nvarchar](30) NOT NULL,

CONSTRAINT [PK\_雇员] PRIMARY KEY CLUSTERED ([雇员ID] ASC)

)

**/\* 在雇员表上，设置默认值 \*/**

ALTER TABLE [dbo].[雇员]

ADD CONSTRAINT [DF\_雇员\_雇员登陆密码]

DEFAULT ((12345678)) FOR [雇员登陆密码];

**/\* 建立类别表、设置主键 \*/**

CREATE TABLE [dbo].[类别]

(

[类别ID] [int] NOT NULL,

[类别名称] [nvarchar](30) NOT NULL,

[类别说明] [ntext] NOT NULL,

[类别图片] [varbinary](max) NULL,

CONSTRAINT [PK\_类别表] PRIMARY KEY CLUSTERED ([类别ID] ASC)

)

**/\* 添加类别表上完整性约束 \*/**

ALTER TABLE [dbo].[类别]

WITH CHECK ADD CONSTRAINT [类别ID约束]

CHECK (([类别ID]>(0)));

ALTER TABLE [dbo].[类别]

WITH CHECK ADD CONSTRAINT [类别名称约束]

CHECK ((len([类别名称])>=(1)));

**/\* 建立供应商-产品关系表、设置主键 \*/**

CREATE TABLE [dbo].[供应商-产品关系](

[产品ID] [int] NOT NULL,

[供应商ID] [int] NOT NULL,

[产品当前库存量] [smallint] NOT NULL,

CONSTRAINT [PK\_供应商-产品关系] PRIMARY KEY CLUSTERED

([产品ID] ASC,[供应商ID] ASC)

)

**/\* 添加供应商-产品关系表上完整性约束、设置默认值 \*/**

ALTER TABLE [dbo].[供应商-产品关系]

ADD CONSTRAINT [DF\_供应商-产品关系\_产品当前库存量]

DEFAULT ((0)) FOR [产品当前库存量];

ALTER TABLE [dbo].[供应商-产品关系]

WITH CHECK ADD CONSTRAINT [产品ID外键约束\_供应商-产品关系]

FOREIGN KEY([产品ID])

REFERENCES [dbo].[产品] ([产品ID]);

ALTER TABLE [dbo].[供应商-产品关系]

WITH CHECK ADD CONSTRAINT [供应商ID外键约束\_供应商-产品关系]

FOREIGN KEY([供应商ID])

REFERENCES [dbo].[供应商] ([供应商ID]);

针对上述SQL语句的的说明如下：

1. 语句Primary Key（PK）：被设置为主码的属性值唯一且非空，用以唯一标识一个元组，体现了实体完整性。
2. 语句Foreign Key（FK）：设定了外码，被设定的属性取值只能为被参照表列中的值或空值，体现了参照完整性。
3. 语句NOT NULL：体现了用户定义的完整性，保证列值非空。
4. 语句CHECK：保证列值满足一个用户自定义的条件表达式，可以通过这个语句对列值的取值、长度等进行限制。
5. 语句CONSTRAINT：对完整性约束条件命名，便于后期对约束的修改，删除等操作。
6. 语句CLUSTERED(ASC)：在主键上建立聚集索引、定义顺序排列，定义表中数据的存储方式，提高按照主键检索等操作的速率。

**3.2.4 SQL Server索引的设置**

SQL Server 的索引是一种数据结构，可以以物理存储空间为代价，通过对表中的数据进行排序、聚集，避免进行全表扫描，提升数据的SELECT、UPDATAE、DELETE语句的执行速度与检索查询效率。

对外键建立索引：比如在订单表中除了主键订单ID常被用于检索，还会对于外键客户ID、雇员ID、供应商ID进行检索。因此有必要对订单表中的外键建立非聚集索引（一般聚集索引建立于主键上，使数据行按照排列顺序存储），以一种与存储在表中的数据相分离的索引结构，可对一个或多个选定列重新排序，使用户可以通过比搜索基础表更快的速度查找数据。

**/\*建立客户ID非聚集索引\*/**

USE [DataBase]

GO

CREATE NONCLUSTERED INDEX 客户ID索引

ON dbo.订单(客户ID)

WITH FILLFACTOR = 80

GO

**/\*建立雇员ID非聚集索引\*/**

USE [DataBase]

GO

CREATE NONCLUSTERED INDEX 雇员ID索引

ON dbo.订单(雇员ID)

WITH FILLFACTOR = 80

GO

**/\*建立供应商ID非聚集索引\*/**

USE [DataBase]

GO

CREATE NONCLUSTERED INDEX 供应商ID索引

ON dbo.订单(供应商ID)

WITH FILLFACTOR = 80

GO

针对上述SQL语句的的说明如下：

FILLFACTOR表示填充因子，可以根据读写操作的比例进行灵活调整，以便更大地发挥出索引的特性：

对于低更改的表（读写比例100：1左右）——FILLFACTOR = 100

对于高更改的表（写超过读操作）——FILLFACTOR = 50~70

对于读写操作差不多的表——FILLFACTOR = 80~90

**3.2.5 SQL Server触发器的设置**

触发器是存储在SQL Server中的一种特殊类型的存储过程，它们是与表相关联的代码，通过监视和响应激活事件，当指定表中的数据发生修改（一般是执行INSERT、UPDATE或DELETE语句）时自动生效，作为一个独立的单元执行预先设定好的代码，来自动执行操作。

通常可以通过设定触发器实现比CHECK约束更复杂的完整性约束、可以通过引用其他关系（表）的数据项执行更为复杂的操作，用于管理数据完整性、执行与业务规则相关的操作，总体上更加灵活。

在本数据库中我们设定了两个AFTER类（事件发生之后生效的）的DML触发器（发生数据操作语言DML后调用的触发器），触发器名为one和two：

1. 触发器one

one触发器创建于订单明细表下，当对订单明细表执行插入（INSERT）操作后，触发器one自动生效，通过产品ID查找产品表，获取产品单价，计算单项产品总金额并填入订单明细表。

**/\* 创建触发器one \*/**

USE [DataBase]

GO

SET ANSI\_NULLS ON

GO

SET QUOTED\_IDENTIFIER ON

GO

CREATE trigger one on [dbo].[订单明细]

after insert

as

begin

declare @产品ID int;

declare @订单ID int;

declare @数量 smallint;

declare @单价 money;

set @产品ID=(SELECT 产品ID FROM inserted);

set @订单ID=(SELECT 订单ID FROM inserted);

set @数量=(SELECT 数量 FROM inserted);

set @单价=(select 产品单价 from dbo.产品 where dbo.产品.产品ID=@产品ID );

update dbo.订单明细

set 单项产品总金额=@单价\*@数量 where 订单ID=@订单ID and 产品ID=@产品ID;

end;

1. 触发器two

two触发器创建于订单表下，当对订单表执行插入（INSERT）操作后，触发器two自动生效，通过订单ID查询订单明细表，获取没想产品的总金额，计算当前订单的总金额后通过公式，计算折扣（返利）。

公式如下：

|  |  |
| --- | --- |
| 每单总金额范围 | 返还金额占本单总金额百分比（%） |
| [ 10000 , ∞ ) | 3 |
| [ 5000 , 10000 ) | 2 |
| [ 1000 , 5000 ) | 1 |
| [ 0 , 1000 ) | 0 |

**/\* 创建触发器two \*/**

USE [DataBase]

GO

SET ANSI\_NULLS ON

GO

SET QUOTED\_IDENTIFIER ON

GO

CREATE trigger two

on [dbo].[订单]

after insert

as

begin

declare @订单ID int;

declare @总价 money;

declare @折扣 money;

declare @邮政编码 nvarchar(20);

set @邮政编码=(SELECT 订单货主所在地邮政编码 FROM inserted);

set @订单ID=(SELECT 订单ID FROM inserted);

set @总价=(

select sum(单项产品总金额)

from dbo.订单明细

where dbo.订单明细.订单ID=@订单ID);

if(@总价<1000) set @折扣=0;

if(@总价>=1000 and @总价<5000)set @折扣=@总价\*0.01;

if(@总价>=5000 and @总价<10000)set @折扣=@总价\*0.02;

if(@总价>=10000)set @折扣=@总价\*0.03;

update dbo.订单

set 订单折扣=@折扣;

end;

**4 系统测试与应用**

**4.1 系统测试运行环境**

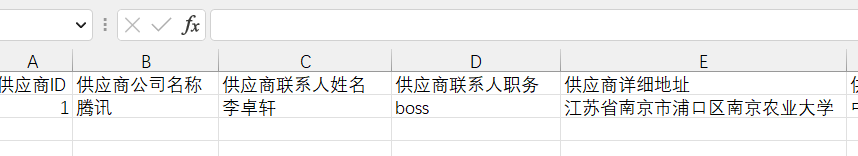
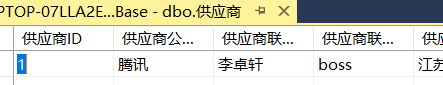
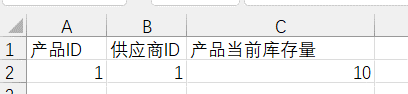
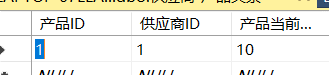
测试系统软件环境：操作系统为Windows 10并且安装了python310虚拟环境，并安装了相关的库（PySide2、pymssql等），同时安装了SQL Server Management Studio v19.2。

测试系统硬件配置：处理器为Intel Core i5-10300H,内存16G.

**4.2 数据库测试与运行**

由于此次数据库课程设计中的功能有类似或重复的情况，比如修改个人信息、增添商品信息等，在这里展示主要功能的测试。

4**.2.1 数据入库**

图形用户界面

中度可信度描述已自动生成

图5-1 数据入库界面、传入数据库的数据

4**.2.2 用户登录与注册**



图5-2 用户注册、用户登录成功

4**.2.3 客户进行购买**表格

描述已自动生成图形用户界面, 应用程序

描述已自动生成

图5-3 用户下单、扣费

图形用户界面, 应用程序

描述已自动生成图形用户界面, 应用程序

描述已自动生成4**.2.4 客户进行确认收货**

图5-4 客户确认收货、更新数据库

4**.2.5 客户积分商城**

图形用户界面, 应用程序

描述已自动生成

图5-5 积分商城兑换情况其一：积分不足

**4.2.6雇员对用户进行订单确认**

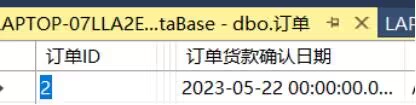


图5-6 雇员确认订单

**4.2.7供应商发货确认日期**

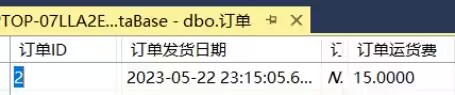


图5-7 供应商确认发货

**4.2.8供应商商品查询查询**



图5-8 供应商商品查询界面

**5 总结与展望**

**5.1 总结**

本系统实现了如下核心功能：

**5.1.1 客户中心**

（1）客户注册系统

客户按照我们界面的提示通过输入客户ID、公司信息、国家、地址、密码等信息即可注册。

（2）客户登录系统

客户通过自己的账户ID和密码登陆该系统。

注意：只有在系统内存在的客户才可进行登录操作。

客户登录之后可以完成的功能有：

1. 积分兑换：

客户可以选择要兑换的礼品，随后扣除总积分中相应的积分进行兑换，兑换成功后发出礼品。

1. 商品购买：

在所需要的商品后输入数量，不需要的不填或者填0，点击确定购买，由系统自动计算折扣并生成订单发送给雇员。

1. 订单信息查看：

分为未到货订单、已到货以及所有订单查询。可供客户查看商品是否发货等信息，并且可以通过输入想要查询的订单号进行查询。

1. 个人信息修改：

可以修改自己的个人信息，同时也可以修改密码。

**5.1.2 雇员中心**

（1）雇户登录系统

经理通过自己的账户ID和密码登陆该系统。

注意：只有在系统内存在的经理才可进行登录操作。

经理登录之后可以完成的功能有：

a. 订单管理：

雇户可确认客户提交订单，通过订单详情信息选择供货商并发送订单。

b. 个人信息修改：

雇户可以修改自己的个人信息，同时也可以修改密码

**5.1.3 供应商中心**

（1）供应商注册系统

供应商按照我们界面的提示通过输入公司信息、国家、地址、密码等信息即可注册。

（2）供应商登录系统

供应商通过自己的账户ID和密码登陆该系统。

注意：只有在系统内存在的用户才可进行登录操作。

供应商登录之后可以完成的功能有：

a. 查看商品库存：

对想查看的商品ID进行查询，查看库存量，以便及时进行补货。

b. 修改商品信息：

增删改商品信息，可以增加商品也可以减少商品，或者修改商品的信息。

c. 订单发货：

收到雇员发来的订单信息后，比对商品库存量，若满足要求则进行发货，输入订单ID号进行发货，时间自动获取为点击发货按钮时间，若不满足则进行补货，暂不发货。

d. 个人信息修改：

供应商可以修改自己的个人信息，同时也可以修改密码。

通过对本课题的研究以及该管理系统的设计与开发，我们对数据库的理论知识有了更进一步的理解，也能更加熟练地使用SQL语句，同时也学习了python的基本语法并可以使用python完成基础的软件开发。

在整个数据库课设中，本小组成员分工明确，相互配合，积极讨论，团结互助，在整个过程中我们小组成员一起成长一起进步。在我们小组的团结合作下，通过查阅资料、网上搜索等方式共同完成了本次课程设计，最终我们都收获很多。

**5.2 不足**

目前，本系统的大体核心功能都基本能够实现，但是整个系统的应用上来说还存在一些不足：

1. 用户（客户）注册时账号自行填写，除了长度以外，无法限制为后续ID检索工作造成不便。n用户安全性：在客户中心和供应商中心中，账户登录仅基于账户ID和密码，缺乏其他身份验证机制（如邮箱验证），导致系统易受到攻击。
2. 对于附加功能的积分系统，进行积分商城兑换实体积分小礼物后，对于是否寄到、是否需要修改寄件地址的界面不够完整。
3. 数据可靠性：在数据库设计方面未考虑日志文件，以防止数据库发生故障后能够及时对数据库进行恢复。数据库系统的可靠性问题，如系统故障、断电等问题，可能导致数据损坏或丢失，需要进行备份和恢复等工作。
4. 购物车未实时记录：客户端在购物车采购时，只能通过手动输入数量进行采购，并且具有时效性，在退出登录后，购物车内有商品不能进行保存而是忽略，只能等下次登录时重新输入，没有扩展更新与记录购物车的功能。客户对商品信息查看只通过表单信息，没有图片，没有详情页面的信息界面。
5. 数据库系统安全性：如黑客攻击、病毒感染等，一旦发生安全问题，可能导致数据丢失甚至泄露。
6. 整个界面可以设计排版得更加美观、简洁。

**5.3 展望**

设计一个商品购物数据库可以涉及到很多方面，比如商品信息的存储、订单的处理、库存的管理等等。本管理系统面向客户、雇员和供应商等多个对象，从不同视角看待商品销售的各个环节。

本管理系统是对于网上商品购物管理的一次尝试，我们仅仅只做出一个小小的雏形。希望以后我们能以本次课程设计为基础，在已有的基础之上优化并扩展新的功能：

（1）监控和分析销售趋势。

根据销售数据的分析，供应商或个人可以更好地了解哪些产品最受欢迎，哪些顾客群体最有可能购买他们的产品，以及哪些时间段销售最为活跃等等。这些信息可以帮助供应商或个人作出更为有根据的决策，例如制定更加有效的促销策略。

（2）跟踪订单和库存。

一个好的销售数据库可以帮助人更好地管理订单和库存。当客户下单时，系统可以自动更新库存数量，避免产品无法满足需求的情况出现。此外，还可以利用销售数据库的报告功能来监控库存水平，并根据需求进行补货或调整存货；

（3）管理顾客数据。

企业或个人可以将客户的购买历史、联系方式、偏好等数据存储在销售数据库中，并根据这些信息制定更加针对客户的促销策略和提高客户满意度的措施；设计优化的搜索算法，帮助用户快速找到自己所需要的商品，并提供个性化的搜索结果推荐。

（4）提高销售效率和准确性。

使用销售数据管理系统可以提高销售的效率和准确性。当销售人员进行销售时，可以通过系统随时了解产品信息、价格、库存、客户信息等其他需要的信息，减少销售人员的出错率，提高销售额和客户满意度。

（5）增加社交化功能，为用户提供商品晒单、打卡签到等互动形式，提升用户黏性和社区参与度。

参考文献：

[1] 王珊，萨师煊.《数据库系统概论》[M]. 第五版，北京：高等教育出版社，2014.9：234-235.