PROJECT NAME: MEDICAL SALES MANAGEMENT SYSTEM

DONE BY:

MEGHANA KUMAR - 3122235002070

NIKHIL S – 3122235002080

IT - B

PROBLEM DESCRIPTION:

Medical Sales Management System

The Medical Sales Management System is designed to streamline operations within a medical sales context, focusing on efficiently managing and tracking sales, inventory, and customer data. The system offers robust features to maintain data integrity, track sales activities, and apply discounts dynamically.

FEATURES

1. Sales Management:

- **Insert, Update, and Delete Sales Records:** Administrators can easily add, modify, or remove sales records, ensuring real-time updates.
- **Track Sales Details:** Each sale record maintains critical details such as sale date, associated customer, items sold, and total amount.

2. Sales Item Management:

- **Link Sales with Items Sold:** Each sale entry links to specific items sold through the `sales_items` table, tracking quantity, unit price, and any applicable discounts for each item.
- **Apply Discount Strategies:** Different discount types can be applied to medicines in sales items, allowing flexible discount options based on policies like:
- **NoDiscount –** Default, no discount applied.
- **SeasonalDiscount –** Percentage discount applied seasonally.
- **FirstTimeBuyerDiscount –** Discount for first-time customers.

3. Medicine Inventory Management:

- **Track and Update Inventory:** Manage inventory records by adding new medicines, updating details for existing ones, and removing outdated or expired items.
- **Stock Verification:** The system prevents sales of out-of-stock items by checking inventory levels during each transaction.
- **Medicine Details:** Maintain records on medicine name, batch number, expiry date, and price to support compliance and ensure product quality.

4. Customer Profile Management:

- **Add, Edit, and Delete Customer Records:** Maintain detailed records of customer information, allowing administrators to quickly retrieve customer profiles.
- **Link Customer with Sales and Prescriptions:** Each customer profile links to their associated sales and prescription records, offering a consolidated view of their purchase history and medical requirements.
- **Discount Eligibility Check:** Track discount eligibility for each customer, especially for first-time buyers or loyal customers eligible for discounts.

5. Supplier Management:

- **Manage Supplier Information:** Add, update, and remove suppliers while keeping track of supplier contact details, product types supplied, and other relevant information.
- **Supplier and Stock Relationship:** Link each supplier with the items they provide, ensuring accurate inventory restocking.

6. Prescription Management:

- **Store and Retrieve Prescriptions:** Maintain records of prescriptions linked to customers, supporting sales staff in confirming which medications are authorized for each customer.
- **Link Prescription and Sales Data:** Track sales items related to each prescription, enabling the seamless validation of purchases against prescriptions where required.

7. Dynamic Discount System for Sales Items:

- **Apply Discounts on Sales Items:** Dynamically calculate discounts based on predefined criteria, allowing custom discounts on specific medicines or sales items.
- **Stack Discounts for Eligible Customers:** Combine multiple discount types (e.g., seasonal, first-time buyer) for qualifying purchases, offering flexibility and promoting customer retention.

8. Enhanced Data Viewing and Management:

- **Consolidated Customer View:** Display comprehensive records showing customer details alongside their linked sales and prescription data.
- **Search and Filter:** Easily search for specific sales, sales items, or customer data and filter records by various criteria, such as date, customer name, or medicine name.
- **Data Overview Panels:** View summaries for different tables (sales, customers, suppliers) on the dashboard, with quick access to detailed views.

9. Undo and Redo Actions:

- **Reversible Actions:** Implement undo and redo capabilities for actions such as adding or deleting records, enabling administrators to quickly correct mistakes.
- **Action History Management:** Maintain a record of recent actions, allowing users to backtrack or reapply modifications as needed.

10. Command Pattern for Operations:

- **Command-based Actions:** Modularize operations (e.g., inserting, updating records) into separate commands for improved structure and maintainability.
- **History Tracking for Command Operations:** Use command history to support undo and redo functionality, tracking each command's state for reliability and error prevention.

11. Strategy Pattern for Discount Calculations:

- **Flexible Discount Application:** Use strategy pattern to apply different discounts (NoDiscount, SeasonalDiscount, FirstTimeBuyerDiscount) to sales items dynamically, allowing adjustments without modifying core code.

12. User-Friendly GUI:

- **Main Dashboard:** A central hub displaying sales, inventory, and customer information, with quick links to detailed views and operations.
- **Login Options:** Secure login for admin users to access management functions.
- **Easy Navigation**: Organized layout for browsing different sections, such as sales, customers, suppliers, and medicines.
- **Scrollable Views for Large Datasets:** Use scrollable frames for viewing long lists of customers, medicines, or sales records, enhancing user experience.

13. Persistent Data Storage:

- **Automatic Data Loading on Startup:** Automatically load data from files when the system starts, allowing administrators to resume work seamlessly.
- **Data Backup and Recovery:** Periodically back up data files to prevent loss and facilitate recovery if needed.

LIST OF DESIGN PATTERNS USED:

1. Singleton Pattern

- **Classes:** DatabaseManager.
- **Purpose:** Ensures a single, shared database connection instance throughout the application. The DatabaseManager class is designed to restrict instantiation to only one instance, providing a global point of access to the database. This prevents redundant connections, improves resource management, and enhances efficiency, as multiple parts of the application can share the same connection.

2. Command Pattern

- **Classes:** Command (Interface), InsertCommand, CommandInvoker, InsertTemplate and specific subclasses for each entity: SupplierInsert, MedicineInsert, CustomerInsert, SalesInsert, PrescriptionInsert, SalesItemInsert.
- **Purpose:** This pattern encapsulates database actions (like adding records for Supplier, Medicine, Customer, Sales, Prescription, and SalesItem) as command objects, each adhering to the Command interface. This modular approach enables the CommandInvoker class to handle the execution of commands and allows each insert operation to be invoked independently, enabling future enhancements such as logging, undoing operations, or managing history.

3. Factory Pattern

- **Classes:** InsertFactory.
- **Purpose:** The factory pattern simplifies the creation of command objects by centralizing the logic for determining which insert command is needed for each entity type (Supplier, Medicine, Customer, Sales, Prescription, and SalesItem). Instead of directly instantiating each command in the client code, InsertFactory generates the appropriate command object based on input parameters. This setup makes the code more extensible and simplifies the addition of new entity types.

4. Template Method Pattern

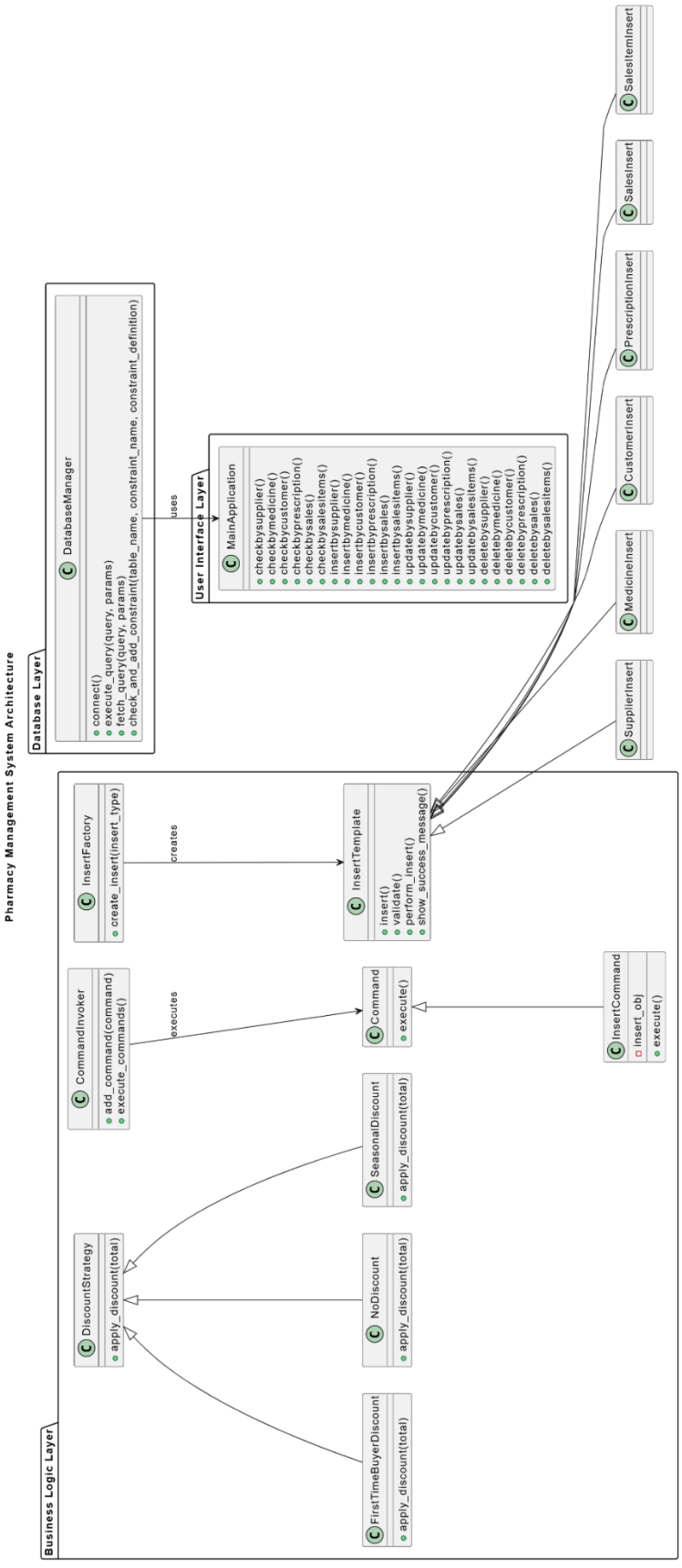
- **Classes:** InsertTemplate (Abstract Class), with subclasses like SupplierInsert, MedicineInsert, CustomerInsert, SalesInsert, PrescriptionInsert, SalesItemInsert.
- **Purpose:** The Template Method pattern provides a structured and reusable process for inserting records into different tables. The abstract InsertTemplate class

defines the common steps for an insert operation, such as validating data and executing the database query. Each subclass (e.g., SupplierInsert, MedicineInsert, etc.) implements the entity-specific parts, like validation logic and entity-specific query details, while inheriting the core insert process. This approach reduces code duplication and keeps the insert operations consistent.

5. Strategy Pattern

- **Classes:** DiscountStrategy (Interface), NoDiscount, SeasonalDiscount, FirstTimeBuyerDiscount.
- **Purpose:** This pattern allows interchangeable discount strategies to be applied to sales based on various criteria (such as seasonal promotions or first-time customer discounts). The DiscountStrategy interface defines the standard approach for calculating discounts, while specific classes (NoDiscount, SeasonalDiscount, FirstTimeBuyerDiscount) implement the different discount calculations. This setup enables the system to apply discounts dynamically, enhancing flexibility and making it easier to add or modify discount options in the future.

UML DIAGRAM:


Pharmacy Management System Architecture
Database Layer
DatabaseManager
connect()
execute_query(query, params)
fetch_query(query, params)
check_and_add_constraint(table_name, constraint_name, constraint_definition)
uses
User Interface Layer
MainApplication
checkbysupplier()
checkbymedicine()
checkbycustomer()
checkbyprescription()
checkbysales()
checkbysaleslitems()
insertbysupplier()
insertbymedicine()
insertbycustomer()
insertbyprescription()
insertbysales()
insertbysaleslitems()
updatebysupplier()
updatebymedicine()
updatebycustomer()
updatebyprescription()
updatebysales()
updatebysaleslitems()
deletebysupplier()
deletebymedicine()
deletebycustomer()
deletebyprescription()
deletebysales()
deletebysaleslitems()
SalesItemInsert
SalesInsert
PrescriptionInsert
CustomerInsert
MedicineInsert
SupplierInsert
Business Logic Layer
InsertFactory
create_insert(insert_type)
creates
InsertTemplate
insert()
validate()
perform_insert()
show_success_message()
CommandInvoker
add_command(command)
execute_commands()
executes
Command
execute()
InsertCommand
insert_obj
execute()
DiscountStrategy
apply_discount(total)
SeasonalDiscount
apply_discount(total)
NoDiscount
apply_discount(total)
FirstTimeBuyerDiscount
apply_discount(total)

CODE:

```
from tkinter import *
from tkinter import messagebox,ttk,Tk, Frame, Label, ttk, Scrollbar, VERTICAL,
HORIZONTAL
from PIL import Image, ImageTk
from customtkinter import *
import cx_Oracle
import re
from datetime import datetime, timedelta

# Establishing Oracle SQL Connectivity using Singleton class
class DatabaseManager:
    _instance = None

    def __new__(cls, *args, **kwargs):
        if not cls._instance:
            cls._instance = super(DatabaseManager, cls).__new__(cls)
        return cls._instance

    def __init__(self, username, password):
        if not hasattr(self, 'initialized'):  # Avoid re-initialization
            try:
                dsn = "localhost:1521"
                self.connection = cx_Oracle.connect(user=username, password=password,
dsn=dsn)
                self.cursor = self.connection.cursor()
                print("Database connection established successfully.")

                # Check for existing constraint before adding it
                self.check_and_add_constraint(
                    'SALES_ITEMS',
                    'check_sales_item_quantity_positive',
                    'CHECK (quantity > 0)'
                )

                # Drop existing trigger if it exists
                try:
                    self.execute_query("DROP TRIGGER update_medicine_stock",
show_success=False)
                except cx_Oracle.DatabaseError:
                    pass  # Ignore error if trigger does not exist

                # Define the trigger to update medicine stock after sale item insert
                trigger_code = """
                CREATE OR REPLACE TRIGGER update_medicine_stock
                AFTER INSERT ON SALES_ITEMS
                FOR EACH ROW
                BEGIN
```

```
                UPDATE MEDICINE
                SET quantity = quantity - :NEW.quantity
                WHERE medicine_id = :NEW.medicine_id
                AND quantity >= :NEW.quantity;
                IF SQL%ROWCOUNT = 0 THEN
                    RAISE_APPLICATION_ERROR(-20001, 'Insufficient stock in MEDICINE
table.');
                END IF;
            END;
            """
            self.execute_query(trigger_code, show_success=False)

            self.initialized = True  # Set flag to indicate initialization
        except cx_Oracle.DatabaseError as e:
            messagebox.showerror("Database Error", str(e))

    def execute_query(self, query, params=(), success_message="Operation completed
successfully", show_success=True):
        try:
            self.cursor.execute(query, params)
            self.connection.commit()
        except cx_Oracle.DatabaseError as e:
            error_message = str(e)
            if "ORA-20001" in error_message:  # Custom trigger error
                messagebox.showerror("Trigger Error", "Trigger prevented the operation:
Insufficient stock in MEDICINE table.")
            else:
                messagebox.showerror("Database Error", error_message)

    def fetch_query(self, query, params=()):
        try:
            self.cursor.execute(query, params)
            return self.cursor.fetchall()
        except cx_Oracle.DatabaseError as e:
            messagebox.showerror("Database Error", str(e))
            return []

    def close(self):
        self.cursor.close()
        self.connection.close()

    def check_and_add_constraint(self, table_name, constraint_name,
constraint_definition):
        # Check if the constraint already exists
        check_query = """
        SELECT COUNT(*)
        FROM all_constraints
        WHERE table_name = :table_name
```

```
        AND constraint_name = :constraint_name
        """
        result = self.fetch_query(check_query, {'table_name': table_name.upper(),
'constraint_name': constraint_name.upper()})

        # If constraint does not exist, add it
        if result and result[0][0] == 0:
            add_constraint_query = f"ALTER TABLE {table_name} ADD CONSTRAINT
{constraint_name} {constraint_definition}"
            self.execute_query(add_constraint_query, show_success=False)
            print(f"Constraint {constraint_name} added successfully.")
        else:
            print(f"Constraint {constraint_name} already exists.")

# Strategy Pattern - Different discount strategies for sales
class DiscountStrategy:
    def apply_discount(self, total):
        return total

class NoDiscount(DiscountStrategy):
    def apply_discount(self, total):
        return total

class SeasonalDiscount(DiscountStrategy):
    def apply_discount(self, total):
        return total * 0.9  # 10% discount

class FirstTimeBuyerDiscount(DiscountStrategy):
    def apply_discount(self, total):
        return total * 0.8

# Template Pattern for Insert Operations
class InsertTemplate:
    def insert(self):
        if not self.validate():
            messagebox.showerror('Error!', 'Validation failed.')
            return
        self.perform_insert()

    def validate(self):
        """Override this method for custom validation logic."""
        raise NotImplementedError

    def perform_insert(self):
        """Override this method to insert the entity into the database."""
        raise NotImplementedError

#Factory Pattern for Insert Operations
```

```
class InsertFactory:
    @staticmethod
    def create_insert(insert_type, supplier_id=None, s_name=None,
contact_number=None, email=None, address=None):
        if insert_type == 'Supplier':
            return SupplierInsert()
        # Add other types as needeed
        elif insert_type == 'Medicine':
            return MedicineInsert()
        elif insert_type == 'Customer':
            return CustomerInsert()
        elif insert_type == 'Prescription':
            return PrescriptionInsert()
        elif insert_type == 'Sales':
            return SalesInsert()
        elif insert_type == 'SalesItem':
            return SalesItemInsert()
        else:
            raise ValueError("Invalid entity type")

# Command Pattern - Queue database operations for batch execution
class Command:
    def execute(self):
        raise NotImplementedError

class InsertCommand(Command):
    def __init__(self, insert_obj):
        self.insert_obj = insert_obj

    def execute(self):
        self.insert_obj.insert()

class CommandInvoker:
    def __init__(self):
        self._commands = []

    def add_command(self, command):
        self._commands.append(command)

    def execute_commands(self):
        for command in self._commands:
            command.execute()
        self._commands.clear()  # Clear the list after executing

# Implementing Specific Insert Classes
class SupplierInsert(InsertTemplate):
    def __init__(self, supplier_id, s_name, contact_number, email, address):
        self.supplier_id = supplier_id
```

```
        self.s_name = s_name
        self.contact_number = contact_number
        self.email = email
        self.address = address

    def validate(self):
        # Implement validation logic for Supplier entity
        return bool(self.s_name and self.contact_number and self.email and self.address)

    def perform_insert(self):
        query = """
        INSERT INTO SUPPLIER (supplier_id, s_name, contact_number, email, address)
        VALUES (:supplier_id, :s_name, :contact_number, :email, :address)
        """
        params = {
            'supplier_id': self.supplier_id,
            's_name': self.s_name,
            'contact_number': self.contact_number,
            'email': self.email,
            'address': self.address
        }
        db_manager = DatabaseManager(username="your_username",
password="your_password")
        db_manager.execute_query(query, params)

class MedicineInsert(InsertTemplate):
    def __init__(self, medicine_id, m_name, brand, batch_number, expiry_date, quantity,
price, supplier_id):
        self.medicine_id = medicine_id
        self.m_name = m_name
        self.brand = brand
        self.batch_number = batch_number
        self.expiry_date = expiry_date
        self.quantity = quantity
        self.price = price
        self.supplier_id = supplier_id

    def validate(self):
        if not self.medicine_id or not self.m_name or not self.brand or not self.batch_number
or not self.expiry_date or not self.quantity or not self.price or not self.supplier_id:
            messagebox.showerror('Error!', 'All fields are required.')
            return False
        if int(self.quantity) <= 0:
            messagebox.showerror('Error!', 'Quantity must be positive.')
            return False
        if float(self.price) <= 0:
            messagebox.showerror('Error!', 'Price must be positive.')
            return False
```

```
        if not re.match(r"^M\d{3}$", self.medicine_id):
            messagebox.showerror('Error!', 'Invalid Medicine ID format. It should start with "M" followed by three digits.')
            return False
        if not re.match(r"^BATCH\d{3}$", self.batch_number):
            messagebox.showerror('Error!', 'Invalid Batch Number format. It should start with "BATCH" followed by three digits.')
            return False
        return True

    def perform_insert(self):
        query = """INSERT INTO MEDICINE (medicine_id, m_name, brand, batch_number, expiry_date, quantity, price, supplier_id)
                   VALUES (:medicine_id, :m_name, :brand, :batch_number, :expiry_date, :quantity, :price, :supplier_id)"""
        params = {
            'medicine_id': self.medicine_id,
            'm_name': self.m_name,
            'brand': self.brand,
            'batch_number': self.batch_number,
            'expiry_date': self.expiry_date,
            'quantity': self.quantity,
            'price': self.price,
            'supplier_id': self.supplier_id
        }
        dbms.execute_query(query, params)
        dbms.execute_query("COMMIT")

class CustomerInsert(InsertTemplate):
    def __init__(self, customer_id, customer_name, contact_number, email, address):
        self.customer_id = customer_id
        self.customer_name = customer_name
        self.contact_number = contact_number
        self.email = email
        self.address = address

    def validate(self):
        if not self.customer_id or not self.customer_name or not self.contact_number or not self.email or not self.address:
            messagebox.showerror('Error!', 'All fields are required.')
            return False
        if not re.match(r"^C\d{3}$", self.customer_id):
            messagebox.showerror('Error!', 'Invalid Customer ID format. It should start with "C" followed by three digits.')
            return False
        return True

    def perform_insert(self):
```

```
        query = """INSERT INTO CUSTOMER (customer_id, c_name, contact_number,
email, address)
                 VALUES (:customer_id, :customer_name, :contact_number, :email,
:address)"""
        params = {
            'customer_id': self.customer_id,
            'customer_name': self.customer_name,
            'contact_number': self.contact_number,
            'email': self.email,
            'address': self.address
        }
        dbms.execute_query(query, params)
        dbms.execute_query("COMMIT")

class PrescriptionInsert(InsertTemplate):
    def __init__(self, prescription_id, customer_id, doctor_name, prescription_date, dosage,
frequency, duration, additional_instructions):
        self.prescription_id = prescription_id
        self.customer_id = customer_id
        self.doctor_name = doctor_name
        self.prescription_date = prescription_date
        self.dosage = dosage
        self.frequency = frequency
        self.duration = duration
        self.additional_instructions = additional_instructions

    def validate(self):
        if not self.prescription_id or not self.customer_id or not self.doctor_name or not
self.prescription_date or not self.dosage or not self.frequency or not self.duration or not
self.additional_instructions:
            messagebox.showerror('Error!', 'All fields are required.')
            return False

        # Validate Prescription ID format (should start with "P" followed by three digits)
        if not re.match(r"^P\d{3}$", self.prescription_id):
            messagebox.showerror('Error!', 'Invalid Prescription ID format. It should start with
"P" followed by three digits.')
            return False

        # Check if the prescription date is in the correct format (DD-MON-YY)
        try:
            datetime.strptime(self.prescription_date, "%d-%b-%y")
        except ValueError:
            messagebox.showerror('Error!', 'Invalid date format. Please enter the prescription
date in DD-MON-YY format.')
            return False

        # Assuming check_customer_id is defined to validate customer ID
```

```
        if not check_customer_id(self.customer_id):
            messagebox.showerror('Error!', 'Customer ID does not exist.')
            return False

        return True

    def perform_insert(self):
        query = """INSERT INTO PRESCRIPTION (prescription_id, customer_id,
doctor_name, prescription_date, dosage, frequency, duration, additional_instructions)
                 VALUES (:prescription_id, :customer_id, :doctor_name, :prescription_date,
:dosage, :frequency, :duration, :additional_instructions)"""
        params = {
            'prescription_id': self.prescription_id,
            'customer_id': self.customer_id,
            'doctor_name': self.doctor_name,
            'prescription_date': self.prescription_date,
            'dosage': self.dosage,
            'frequency': self.frequency,
            'duration': self.duration,
            'additional_instructions': self.additional_instructions
        }
        dbms.execute_query(query, params)
        dbms.execute_query("COMMIT")

class SalesInsert(InsertTemplate):
    def __init__(self, sales_id, customer_id, sales_date, total_amount, payment_method):
        self.sales_id = sales_id
        self.customer_id = customer_id
        self.sales_date = sales_date
        self.total_amount = total_amount
        self.payment_method = payment_method

    def validate(self):
        if not self.sales_id or not self.customer_id or not self.sales_date or not
self.total_amount or not self.payment_method:
            messagebox.showerror('Error!', 'All fields are required.')
            return False
        if self.payment_method not in ['Cash', 'Credit Card', 'Debit Card', 'Online','UPI']:
            messagebox.showerror('Error!', 'Invalid payment method. Choose from: Cash,
Credit Card, Debit Card, Online, or UPI.')
            return False
        return True

    def perform_insert(self):
        query = """INSERT INTO SALES (sale_id, customer_id, sale_date, total_amount,
payment_method)
                 VALUES (:sales_id, :customer_id, :sales_date, :total_amount,
:payment_method)"""
```

```
        params = {
            'sales_id': self.sales_id,
            'customer_id': self.customer_id,
            'sales_date': self.sales_date,
            'total_amount': self.total_amount,
            'payment_method': self.payment_method
        }
        dbms.execute_query(query, params)
        dbms.execute_query("COMMIT")

class SalesItemInsert(InsertTemplate):
    def __init__(self, item_id, sales_id, medicine_id, quantity, price):
        self.item_id = item_id
        self.sales_id = sales_id
        self.medicine_id = medicine_id
        self.quantity = quantity
        self.price = price
        self.subtotal = self.calculate_subtotal()

    def calculate_subtotal(self):
        return self.quantity * self.price

    def validate(self):
        # Validate that quantity and price are positive
        if self.quantity <= 0:
            messagebox.showerror('Error!', 'Quantity must be greater than 0.')
            return False
        if self.price <= 0:
            messagebox.showerror('Error!', 'Price must be greater than 0.')
            return False
        return True

    def perform_insert(self):
        query = """INSERT INTO SALES_ITEMS (sale_item_id, sale_id, medicine_id,
quantity, price_per_unit, subtotal)
                VALUES (:item_id, :sales_id, :medicine_id, :quantity, :price, :subtotal)"""
        params = {
            'item_id': self.item_id,
            'sales_id': self.sales_id,
            'medicine_id': self.medicine_id,
            'quantity': self.quantity,
            'price': self.price,
            'subtotal': self.subtotal
        }
        dbms.execute_query(query, params)
        dbms.execute_query("COMMIT")

# Creating an object for Database to Python link
```

```
dbms = DatabaseManager(username='system', password='Rajini')


#Inserting in sales items
def insertbysalesitems():
    # Create a tkinter page for the insert in sales items page
    root11 = Tk()
    root11.geometry('1000x700+250+50')
    root11.title('Insert in Sales Items Page')
    root11.resizable(0, 0)
    root11.config(bg='gray')

    # Background image setup (make sure the path is correct)
    bgimg = Image.open(R"C:\Meghana\SSN\sem 3\Database Lab\Mini Project\bgpic.jpg")
    bgtk = ImageTk.PhotoImage(bgimg)
    bglabel = Label(root11, image=bgtk, height=750, width=1000)
    bglabel.place(x=0, y=0)

    # Create top frame for displaying title
    Topframe = Frame(root11, bg='black', width=1000, height=100)
    Topframe.place(x=0, y=0)

    # Title text for the introduction
    Introtext = Label(Topframe, text='Insert into Sales Items Table', font=('Georgia', 23,
'bold'), bg='black', fg='white', activebackground='black')
    Introtext.place(x=50, y=25, width=900)

    DetailsFrame = Frame(root11, bg='black', width=820, height=520)
    DetailsFrame.place(x=90, y=150)

    def resetfield():
        saidentry.delete(0, END)
        sidentry.delete(0, END)
        midentry.delete(0, END)
        qtyentry.delete(0, END)
        priceentry.delete(0, END)

    def insertdetails():
        sie = saidentry.get()  # Sale Item ID
        se = sidentry.get()    # Sale ID
        me = midentry.get()    # Medicine ID
        qe = qtyentry.get()    # Quantity
        pe = priceentry.get()  # Price per unit

        # Check if all fields are filled
        if sie == '' or se == '' or me == '' or qe == '' or pe == '':
            messagebox.showerror('Error!', 'Enter all the required values.')
            resetfield()
```

```
            return

        # Validate Sale Item ID format
        def pattern():
            pattern_regex = r"^SI\d{3}$"
            if not re.match(pattern_regex, sie):
                messagebox.showerror('Error!', 'Invalid Sale Item ID format. It should start with
"SI" followed by three digits.')
                return False
            return True

        if not pattern():
            resetfield()
            return

        # Check if Quantity and Price per Unit are valid positive values
        try:
            qe = int(qe)
            pe = float(pe)
            if qe <= 0 or pe <= 0:
                messagebox.showerror('Error!', 'Quantity and Price must be greater than 0.')
                resetfield()
                return
        except ValueError:
            messagebox.showerror('Error!', 'Quantity must be an integer and Price per unit a
number.')
            resetfield()
            return

        # Determine discount strategy based on selection
        discount_strategy = discount_choice.get()
        if discount_strategy == "NoDiscount":
            discount = NoDiscount()
        elif discount_strategy == "SeasonalDiscount":
            discount = SeasonalDiscount()
        elif discount_strategy == "FirstTimeBuyerDiscount":
            discount = FirstTimeBuyerDiscount()

        # Create SalesItemInsert instance and apply discount
        sales_item = SalesItemInsert(sie, se, me, qe, pe)
        sales_item.subtotal = discount.apply_discount(sales_item.subtotal)

        # Validate and perform insert if valid
        if not sales_item.validate():
            resetfield()
            return

        # Check if Sale ID and Medicine ID exist in their respective tables
```

```
        if not check_sale_id(se):  # Assuming check_sale_id is a defined function
            resetfield()
            return
        if not check_medicine_id(me):  # Assuming check_medicine_id is a defined function
            resetfield()
            return

        # Check if the sale_item_id already exists
        query = "SELECT sale_item_id FROM SALES_ITEMS WHERE sale_item_id = :1"
        params = (sie,)
        c = dbms.fetch_query(query, params)

        if c:
            messagebox.showerror('Insert Error!', f'Sale Item ID {sie} already exists!')
            resetfield()
            return
        else:
            # Create InsertSalesItemCommand
            insert_command = InsertCommand(sales_item)

            # Create CommandInvoker
            invoker = CommandInvoker()
            invoker.add_command(insert_command)

            # Execute the command via the invoker
            try:
                invoker.execute_commands()
                messagebox.showinfo('Success!', 'Record inserted successfully into Sales Items
Table!')
            except cx_Oracle.DatabaseError as e:
                error_message = str(e)
                if "ORA-20001" in error_message:
                    messagebox.showerror("Trigger Error", "Trigger prevented the operation:
Insufficient stock in MEDICINE table.")
                else:
                    messagebox.showerror("Database Error", error_message)
            finally:
                resetfield()
            return

    # Labels and entry fields for inserting data by sales items
    saidtext = Label(DetailsFrame, text='Sale Item ID', font=('Georgia', 18, 'bold', 'italic'),
bg='black', fg='white', activebackground='black')
    saidtext.place(x=20, y=50)
    sidtext = Label(DetailsFrame, text='Sale ID', font=('Georgia', 18, 'bold', 'italic'),
bg='black', fg='white', activebackground='black')
    sidtext.place(x=20, y=130)
```

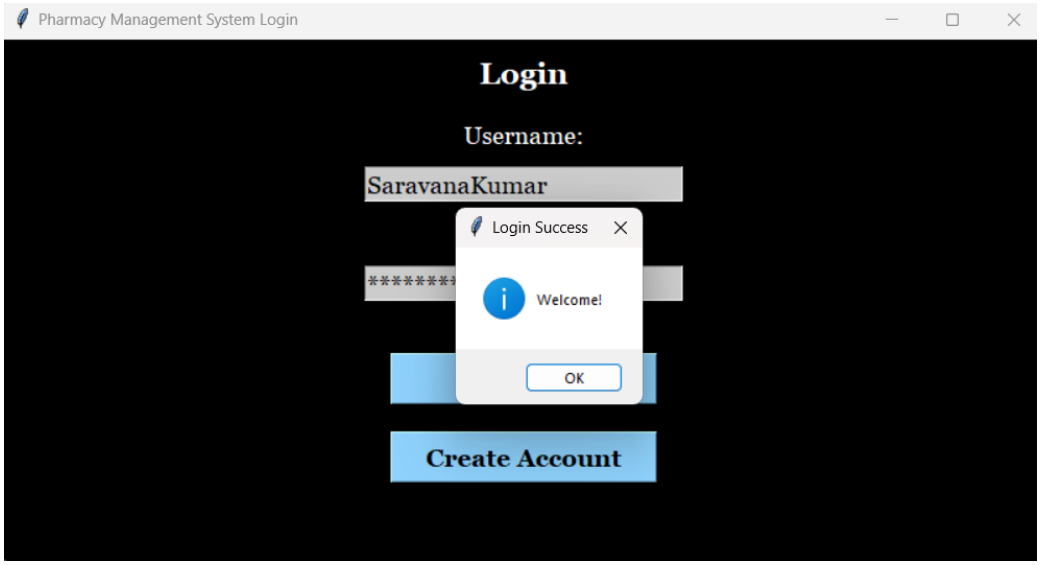
```
    midtext = Label(DetailsFrame, text='Medicine ID', font=('Georgia', 18, 'bold', 'italic'),
bg='black', fg='white', activebackground='black')
    midtext.place(x=20, y=210)
    qtytext = Label(DetailsFrame, text='Quantity', font=('Georgia', 18, 'bold', 'italic'),
bg='black', fg='white', activebackground='black')
    qtytext.place(x=450, y=50)
    pricetext = Label(DetailsFrame, text='Price Per Unit', font=('Georgia', 18, 'bold', 'italic'),
bg='black', fg='white', activebackground='black')
    pricetext.place(x=450, y=130)
    Label(DetailsFrame, text='Discount Type', font=('Georgia', 18, 'bold', 'italic'), bg='black',
fg='white').place(x=450, y=210)

    # Input fields
    saidentry = Entry(DetailsFrame, font=("Georgia", 16, 'bold'), bg='white', fg='black')
    saidentry.place(x=20, y=80, width=350)
    sidentry = Entry(DetailsFrame, font=("Georgia", 16, 'bold'), bg='white', fg='black')
    sidentry.place(x=20, y=160, width=350)
    midentry = Entry(DetailsFrame, font=("Georgia", 16, 'bold'), bg='white', fg='black')
    midentry.place(x=20, y=240, width=350)
    qtyentry = Entry(DetailsFrame, font=("Georgia", 16, 'bold'), bg='white', fg='black')
    qtyentry.place(x=450, y=80, width=350)
    priceentry = Entry(DetailsFrame, font=("Georgia", 16, 'bold'), bg='white', fg='black')
    priceentry.place(x=450, y=160, width=350)

    discount_choice = StringVar(DetailsFrame)
    discount_choice.set("NoDiscount")  # Default value
    discount_menu = ttk.Combobox(DetailsFrame, textvariable=discount_choice,
font=("Georgia", 16, 'bold'), values=["NoDiscount", "SeasonalDiscount",
"FirstTimeBuyerDiscount"])
    discount_menu.place(x=450, y=240, width=350)

    EnterButton = Button(DetailsFrame, text='Enter', command=insertdetails,
font=('Georgia', 18, 'bold'), cursor='hand2', bd=0, bg='light blue', fg='black',
activebackground='light blue')
    EnterButton.place(x=150, y=440, width=220)

    # Function when back button is pressed
    def backpage():
        root11.destroy()
        insert_medicine()

    # Button for going back to previous page
    backbutton = Button(root11, text='Back', command=backpage, font=('Georgia', 18,
'bold'), cursor='hand2', bd=0, bg='light blue', fg='black', activebackground='light blue')
    backbutton.place(x=20, y=25, width=220)

    # Function when back home button is pressed
    def backtohome():
```

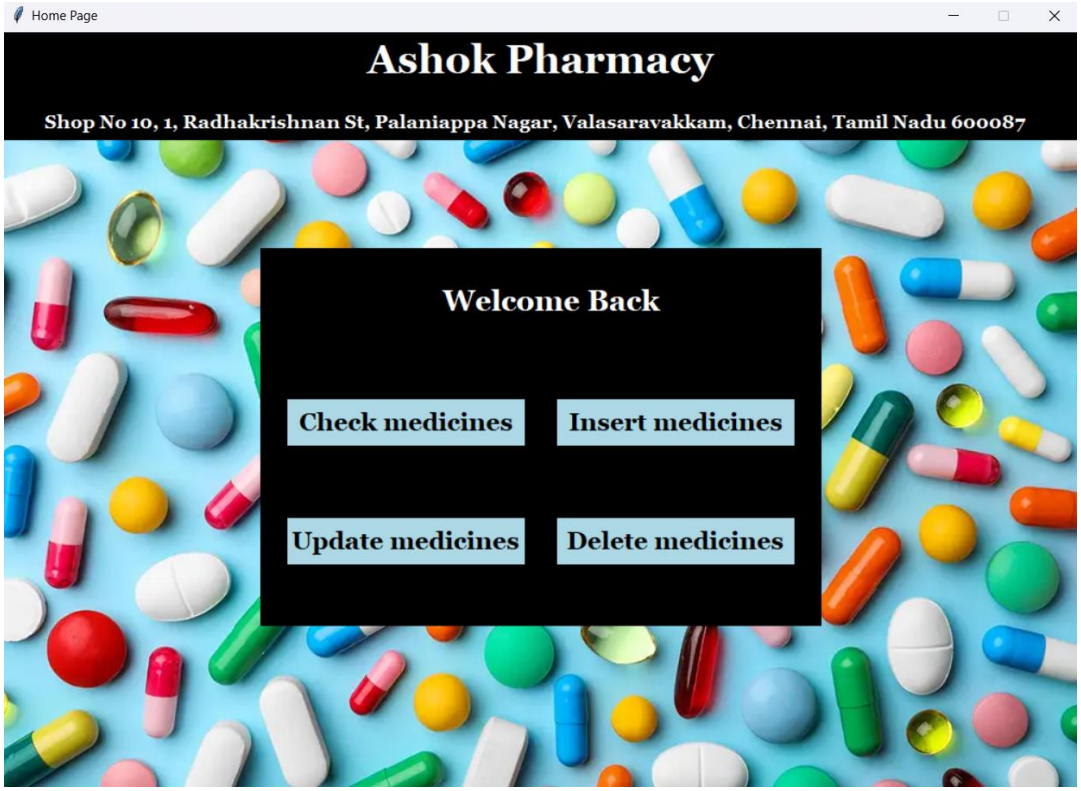
```
        root11.destroy()
        introscreen()

    # Button for going back to home page
    home_page_button = Button(root11, text='Back to home', command=backtohome,
font=('Georgia', 18, 'bold'), cursor='hand2', bd=0, bg='light blue', fg='black',
activebackground='light blue')
    home_page_button.place(x=750, y=25, width=220)

    root11.mainloop()

#Inserting a medicine
def insert_medicine():
    #Create a tkinter page for the insert medicine page
    root3 = Tk()
    root3.geometry('1000x700+250+50') #Set window size
    root3.title('Insert Medicine Page') #Set window title
    root3.resizable(0,0) # Disable window resizing
    root3.config(bg='gray') #Set background colour

    bgimg = Image.open(R"C:\Meghana\SSN\sem 3\Database Lab\Mini Project\bgpic.jpg")
    bgtk = ImageTk.PhotoImage(bgimg)
    bglabel = Label(root3, image=bgtk, height=750, width=1000)
    bglabel.place(x=0, y=0)

    #Function when back home button is pressed
    def backtohome():
        root3.destroy()
        introscreen()

    #Create top frame for displaying title
    Topframe = Frame(root3, bg='black',width=1000, height=100)
    Topframe.place(x=0,y=0)

    #Title text for the introduction
    Introtext = Label(Topframe, text='Insert by',font=('Georgia',
23,'bold'),bg='black',fg='white',activebackground='black')
    Introtext.place(x=50,y=25,width=900)

    ButtonsFrame = Frame(root3,bg='black',width=520,height=500)
    ButtonsFrame.place(x=240,y=150)

    #Function when buttons are pressed for inserting into specific tables
    def insert_supplier():
        root3.destroy()
        insertbysupplier()

    def insert_medicine():
```

```
        root3.destroy()
        insertbymedicine()

    def insert_customer():
        root3.destroy()
        insertbycustomer()

    def insert_prescription():
        root3.destroy()
        insertbyprescription()

    def insert_sales():
        root3.destroy()
        insertbysales()

    def insert_sales_items():
        root3.destroy()
        insertbysalesitems()

    #Button for inserting medicine by supplier
    supplier_button =
Button(ButtonsFrame,text='Supplier',command=insert_supplier,font=('Georgia',18,'bold'),
                           cursor='hand2',bd=0,bg='light blue',fg='black',activebackground='light
blue')
    supplier_button.place(x=150,y=40,width=220)

    #Button for inserting medicine by medicine
    medicine_button =
Button(ButtonsFrame,text='Medicine',command=insert_medicine,font=('Georgia',18,'bold'),
                           cursor='hand2',bd=0,bg='light blue',fg='black',activebackground='light
blue')
    medicine_button.place(x=150,y=115,width=220)

    #Button for inserting medicine by customer
    customer_button =
Button(ButtonsFrame,text='Customer',command=insert_customer,font=('Georgia',18,'bold')
,
                           cursor='hand2',bd=0,bg='light blue',fg='black',activebackground='light
blue')
    customer_button.place(x=150,y=190,width=220)

    #Button for inserting medicine by prescription
    prescription_button =
Button(ButtonsFrame,text='Prescription',command=insert_prescription,font=('Georgia',18,'
bold'),
                           cursor='hand2',bd=0,bg='light blue',fg='black',activebackground='light
blue')
    prescription_button.place(x=150,y=265,width=220)
```

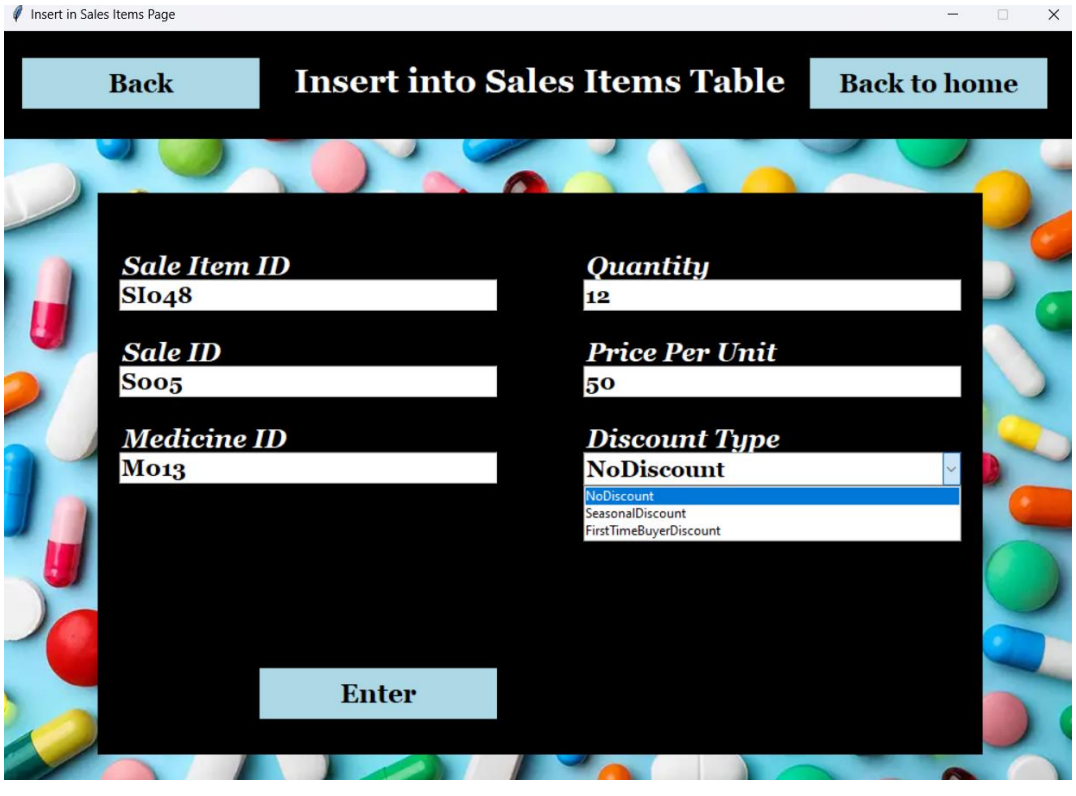
```
    #Button for inserting medicine by sales
    sales_button =
Button(ButtonsFrame,text='Sales',command=insert_sales,font=('Georgia',18,'bold'),
                        cursor='hand2',bd=0,bg='light blue',fg='black',activebackground='light
blue')
    sales_button.place(x=150,y=340,width=220)

    #Button for inserting medicine by sales
    salesitems_button = Button(ButtonsFrame,text='Sales
Items',command=insert_sales_items,font=('Georgia',18,'bold'),
                        cursor='hand2',bd=0,bg='light blue',fg='black',activebackground='light
blue')
    salesitems_button.place(x=150,y=415,width=220)

    #Button for going back to home page
    home_page_button = Button(root3,text='Back to
home',command=backtohome,font=('Georgia',18,'bold'),
                        cursor='hand2',bd=0,bg='light blue',fg='black',activebackground='light
blue')
    home_page_button.place(x=760,y=25,width=220)

    root3.mainloop()
```

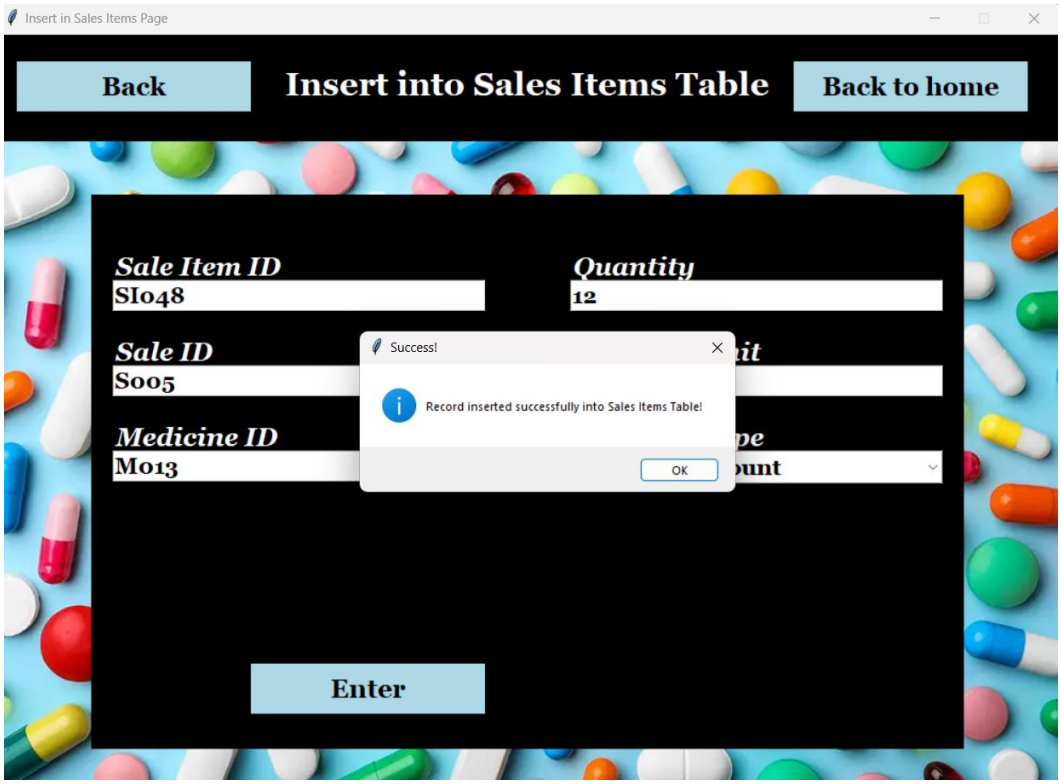
OUTPUT:

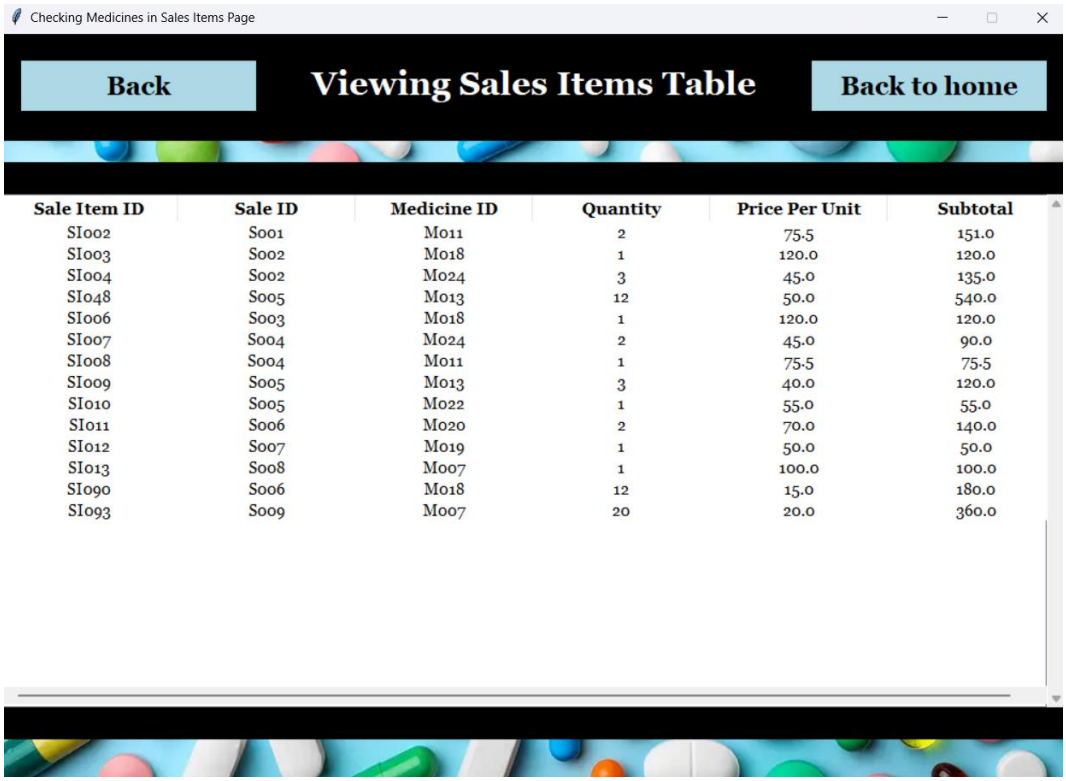

Pharmacy Management System Login
Login
Username:
SaravanaKumar
Login Success
Welcome!
OK
Create Account


Home Page
Ashok Pharmacy
Shop No 10, 1, Radhakrishnan St, Palaniappa Nagar, Valasaravakkam, Chennai, Tamil Nadu 600087
Welcome Back
Check medicines
Insert medicines
Update medicines
Delete medicines


Insert Medicine Page
Insert by
Back to home
Supplier
Medicine
Customer
Prescription
Sales
Sales Items


Insert in Sales Items Page
Back
Insert into Sales Items Table
Back to home
Sale Item ID
SI048
Quantity
12
Sale ID
S005
Price Per Unit
50
Medicine ID
M013
Discount Type
NoDiscount
NoDiscount
SeasonalDiscount
FirstTimeBuyerDiscount
Enter





Sale Item ID	Sale ID	Medicine ID	Quantity	Price Per Unit	Subtotal
SI002	S001	M011	2	75.5	151.0
SI003	S002	M018	1	120.0	120.0
SI004	S002	M024	3	45.0	135.0
SI048	S005	M013	12	50.0	540.0
SI006	S003	M018	1	120.0	120.0
SI007	S004	M024	2	45.0	90.0
SI008	S004	M011	1	75.5	75.5
SI009	S005	M013	3	40.0	120.0
SI010	S005	M022	1	55.0	55.0
SI011	S006	M020	2	70.0	140.0
SI012	S007	M019	1	50.0	50.0
SI013	S008	M007	1	100.0	100.0
SI090	S006	M018	12	15.0	180.0
SI093	S009	M007	20	20.0	360.0