**PHARMACY MANAGEMENT SYSTEM**

**A MINI PROJECT REPORT**

**Submitted by**

**SANJEEV KANTH S 220701250**

**SHANJAY KRISHNAA S 220701260**

# In partial fulfillment for the award of the degree of

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE

RAJALAKSHMI ENGINEERING COLLEGE

(AUTONOMOUS)

THANDALAM

CHENNAI-602105

2023- 24

# BONAFIDE CERTIFICATE

Certified that this project report “**PHARMACY MANAGEMENT SYSTEM**” is the bonafide work of “**SANJEEV KANTH S(220701250),SHANJAY KRISHNAA S(220701260)** ” who carried out the project work under my supervision.

Submitted for the Practical Examination held on\_\_\_\_\_\_\_\_\_\_

**SIGNATURE**

Mrs.k.Mahesmeena

Assistant Professor (SG),

Computer Science and Engineering,

Rajalakshmi Engineering College,

(Autonomous),

Thandalam, Chennai - 602 105

**ABSTRACT**

The Pharmacy Management System is a software application designed to streamline and automate the essential functions of a pharmacy. Developed using Python and SQLite, this system offers a comprehensive solution for managing customer information, medicine inventory, and transaction records through an intuitive graphical user interface (GUI) built with Tkinter. By leveraging Python's simplicity and SQLite's robust data management capabilities, the Pharmacy Management System aims to improve the operational efficiency of pharmacies, reduce manual errors, and provide a reliable tool for managing daily pharmacy activities. This project serves as a practical implementation of software engineering principles, showcasing the integration of modern programming languages and database management systems to solve real-world problems.

**TABLE OF CONTENTS**

|  |  |  |
| --- | --- | --- |
| **S.No** | **CONTENT** | **Pg.No** |
| **1** | **INTRODUCTION** |  |
|  | 1.1 INTRODUCTION |  |
|  | 1.2 OBJECTIVES |  |
|  | 1.3 MODULES |  |
| **2** | **SURVEY OF TECHNOLOGIES** |  |
|  | 2.1 SOFTWARE DESCRIPTION |  |
|  | 2.2 LANGUAGES |  |
|  | 2.2.1 SQL |  |
|  | 2.2.2 PYTHON |  |
| **3** | **REQUIREMENTS AND ANALYSIS** |  |
|  | 3.1 REQUIREMENT SPECIFICATION |  |
|  | 3.2 HARDWARE AND SOFTWARE REQUIREMENTS |  |
|  | 3.3 ER DIAGRAM |  |
|  | 3.4NORMALIZATION |  |
| **4** | **PROGRAM CODE** |  |
| **5** | **RESULTS AND DISCUSSION** |  |
| **6** | **CONCLUSION** |  |
| **7** | **REFERENCES** |  |

# INTRODUCTION

**1.1 Introduction**

The Pharmacy Management System is a comprehensive software solution designed to streamline the operations of a pharmacy. This system integrates various functionalities that support the efficient management of customers, medicines, and transactions. By leveraging a user-friendly graphical interface and robust database management, the system aims to enhance the accuracy, speed, and reliability of pharmacy operations. The project is built using Python and SQLite, providing a cost-effective and scalable solution suitable for pharmacies of varying sizes..

**1.2 Objectives**

The primary objectives of the Pharmacy Management System are as follows:

1. **Automation of Pharmacy Operations**: To automate routine tasks such as adding, updating, and deleting customer and medicine records, reducing manual errors, and saving time.
2. **Efficient Inventory Management**: To track the stock levels of medicines, ensuring that the pharmacy is well-stocked and minimizing the risk of stockouts or overstocking.
3. **Improved Customer Management**: To maintain detailed records of customers, including their contact information and purchase history, facilitating better customer service and personalized care.
4. **Enhanced Data Accessibility**: To provide a user-friendly interface that allows pharmacists and staff to quickly access and manage data, improving workflow efficiency.
5. **Data Security and Integrity**: To ensure the secure storage of sensitive information and maintain the integrity of the data through reliable database management practices.

**1.3 Modules**

The Pharmacy Management System consists of several interconnected modules, each responsible for specific functionalities. The primary modules are:

1. **Customer Management Module**:
   * **Add Customer**: Allows the user to add new customer records, including name, contact information, and address.
   * **Update Customer**: Enables the user to update existing customer details.
   * **Delete Customer**: Provides the functionality to delete customer records from the database.
   * **List Customers**: Displays a list of all customers in the system, showing their details for easy reference.
2. **Medicine Management Module**:
   * **Add Medicine**: Allows the user to add new medicine records, including name, manufacturer, price, and quantity.
   * **Update Medicine**: Enables the user to update existing medicine details.
   * **Delete Medicine**: Provides the functionality to delete medicine records from the database.
   * **List Medicines**: Displays a list of all medicines in the system, showing their details for easy reference.
3. **User Interface Module**:
   * **GUI Design**: Implements a user-friendly graphical interface using Tkinter, enabling easy navigation and interaction with the system.
   * **Form Handling**: Manages the input forms for adding, updating, and deleting customer and medicine records.
   * **Data Display**: Utilizes scrolled text widgets to display lists of customers and medicines dynamically based on user actions.
4. **Database Management Module**:
   * **Database Initialization**: Sets up the SQLite database, creating necessary tables and ensuring the database is ready for operations.
   * **CRUD Operations**: Handles Create, Read, Update, and Delete operations for both customer and medicine records.
   * **Data Validation**: Ensures the data integrity by validating inputs before performing database operations.
5. **Error Handling and Notifications Module**:
   * **Error Handling**: Captures and manages exceptions, providing informative error messages to the user.
   * **User Notifications**: Implements message boxes to notify users of successful operations or errors, enhancing the user experience.

**2. SURVEY OF TECHNOLOGIES**

**2.1 Software Description**

The system is built using Python, a versatile and widely-used programming language, and SQLite, a lightweight, disk-based database that doesn't require a separate server process. Tkinter is used for creating the graphical user interface, providing a simple yet powerful way to develop desktop applications.

**2.2 Languages**

**1.PYTHON:**

**Python** is a high-level, interpreted programming language known for its simplicity, readability, and versatility. It is widely used in various domains, including web development, data science, artificial intelligence, automation, and more. Python's extensive standard library and the availability of numerous third-party packages make it an ideal choice for developing a wide range of applications.

* **Key Features**:
  + **Easy-to-Read Syntax**: Python's syntax is clean and easy to understand, which makes the code more readable and maintainable.
  + **Extensive Standard Library**: Python comes with a rich standard library that provides modules and functions for various tasks, reducing the need to write code from scratch.
  + **Strong Community Support**: Python has a large and active community that contributes to a wealth of resources, libraries, and frameworks.
  + **Cross-Platform Compatibility**: Python is available on multiple platforms, including Windows, macOS, and Linux, making it highly portable.

In this project, Python is used as the primary programming language to develop the core functionality of the Pharmacy Management System, including the user interface, database interactions, and business logic.

2.SQL(STRUCTURED QUERY LANGUAGE)

**SQL** is a domain-specific language used for managing and manipulating relational databases. SQL is essential for performing operations such as querying, updating, and managing data stored in a database. It provides a standard way to interact with databases, ensuring data integrity and consistency.

* **Key Features**:
  + **Data Querying**: SQL allows users to retrieve specific data from one or more tables using various query statements (e.g., SELECT, JOIN).
  + **Data Manipulation**: SQL provides commands to insert, update, delete, and modify data within the database (e.g., INSERT, UPDATE, DELETE).
  + **Data Definition**: SQL includes commands to define and manage the structure of database objects, such as tables and indexes (e.g., CREATE, ALTER, DROP).
  + **Transaction Control**: SQL supports transaction management, ensuring that a series of operations are executed in a reliable and consistent manner (e.g., COMMIT, ROLLBACK).

In this project, SQL is used to define and manipulate the database structure and data within the SQLite database. SQL scripts are executed to create tables, insert records, update existing data, and delete records as needed.

**3. REQUIREMENTS AND ANALYSIS**

**3.1 Requirement Specification**

**Functional Requirements:**

 **Customer Management**:

* **Add Customer**: The system should allow the user to add new customer records, including name, contact information, and address.
* **Update Customer**: The system should allow the user to update existing customer details.
* **Delete Customer**: The system should allow the user to delete customer records from the database.
* **List Customers**: The system should display a list of all customers, showing their details.

 **Medicine Management**:

* **Add Medicine**: The system should allow the user to add new medicine records, including name, manufacturer, price, and quantity.
* **Update Medicine**: The system should allow the user to update existing medicine details.
* **Delete Medicine**: The system should allow the user to delete medicine records from the database.
* **List Medicines**: The system should display a list of all medicines, showing their details.

 **User Interface**:

* The system should provide a graphical user interface (GUI) using Tkinter.
* The interface should include forms for adding, updating, and deleting customer and medicine records.
* The interface should display lists of customers and medicines upon request.

 **Database Management**:

* The system should initialize the database and create necessary tables if they do not exist.
* The system should perform CRUD (Create, Read, Update, Delete) operations on the database.
* The system should ensure data integrity and validation before performing database operations.

 **Error Handling and Notifications**:

* The system should handle exceptions and provide informative error messages to the user.
* The system should use message boxes to notify users of successful operations or errors.

**Non-Functional Requirements:**

 **Usability**:

* The user interface should be intuitive and easy to navigate for users with basic computer skills.
* The system should provide clear instructions and feedback to guide users.

 **Performance**:

* The system should perform all operations (adding, updating, deleting, listing) within an acceptable time frame, typically within a few seconds.
* The system should efficiently handle multiple records without significant degradation in performance.

 **Reliability**:

* The system should ensure data integrity and prevent data loss during operations.
* The system should handle errors gracefully and provide meaningful error messages to the user.

 **Scalability**:

* The system should be able to handle an increasing number of customer and medicine records without requiring major changes to the underlying architecture.

 **Security**:

* The system should protect sensitive customer information and prevent unauthorized access to the database.
* The system should validate user inputs to prevent SQL injection attacks.

 **Maintainability**:

* The system's codebase should be well-documented and modular to facilitate easy maintenance and updates.
* The system should follow coding standards and best practices to ensure readability and maintainability.

**3.2 Hardware and Software Requirement**

**Hardware Requirements:**

A personal computer or laptop with at least 2 GB RAM and 1 GHz processor.

Sufficient storage space for the database and application files.

**Software Requirements:**

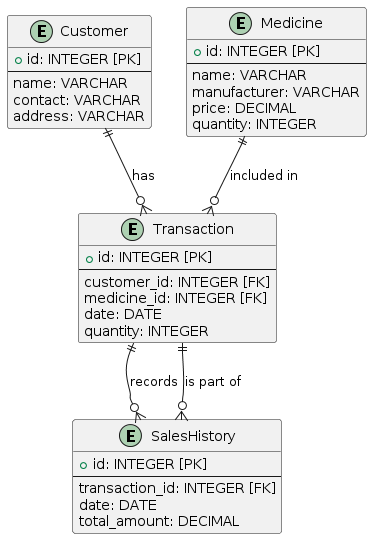
Operating System: Windows, macOS, or Linux.

Python 3.x installed on the system.

Tkinter library for Python (usually included with standard Python installation).

SQLite3 library for Python (also typically included with standard Python installation).

**3.3 ER DIAGRAM**

****

**PROGRAM CODES:**

* **PYTHON WITH SQLITE FOR THE CREATION OF CUSTOMER AND MEDICINE :**

conn = sqlite3.connect('pharmacy.db')

cursor = conn.cursor()

sql\_script = """

-- Create customers table

CREATE TABLE IF NOT EXISTS customers (

customer\_id INTEGER PRIMARY KEY AUTOINCREMENT,

name TEXT NOT NULL,

contact TEXT NOT NULL,

address TEXT

);

-- Create medicines table

CREATE TABLE IF NOT EXISTS medicines (

medicine\_id INTEGER PRIMARY KEY AUTOINCREMENT,

name TEXT NOT NULL,

manufacturer TEXT NOT NULL,

price REAL NOT NULL,

quantity INTEGER NOT NULL

);

"""

cursor.executescript(sql\_script)

conn.commit()

conn.close()

print("Database initialized successfully.")

except Exception as e:

print(f"Error initializing database: {e}")

messagebox.showerror("Database Error", f"Error initializing database: {e}")

* **PYTHON CODE TO HANDLE GUI INTERFACE AND TASKS:**

import sqlite3

import tkinter as tk

from tkinter import messagebox, scrolledtext

def initialize\_db():

try:

conn = sqlite3.connect('pharmacy.db')

cursor = conn.cursor()

sql\_script = """

-- Create customers table

CREATE TABLE IF NOT EXISTS customers (

customer\_id INTEGER PRIMARY KEY AUTOINCREMENT,

name TEXT NOT NULL,

contact TEXT NOT NULL,

address TEXT

);

-- Create medicines table

CREATE TABLE IF NOT EXISTS medicines (

medicine\_id INTEGER PRIMARY KEY AUTOINCREMENT,

name TEXT NOT NULL,

manufacturer TEXT NOT NULL,

price REAL NOT NULL,

quantity INTEGER NOT NULL

);

"""

cursor.executescript(sql\_script)

conn.commit()

conn.close()

print("Database initialized successfully.")

except Exception as e:

print(f"Error initializing database: {e}")

messagebox.showerror("Database Error", f"Error initializing database: {e}")

def add\_customer(name, contact, address):

try:

conn = sqlite3.connect('pharmacy.db')

cursor = conn.cursor()

cursor.execute("INSERT INTO customers (name, contact, address) VALUES (?, ?, ?)", (name, contact, address))

conn.commit()

conn.close()

messagebox.showinfo("Success", "Customer added successfully.")

except Exception as e:

print(f"Error adding customer: {e}")

messagebox.showerror("Error", f"Error adding customer: {e}")

def update\_customer(customer\_id, name, contact, address):

try:

conn = sqlite3.connect('pharmacy.db')

cursor = conn.cursor()

cursor.execute("UPDATE customers SET name = ?, contact = ?, address = ? WHERE customer\_id = ?", (name, contact, address, customer\_id))

conn.commit()

conn.close()

messagebox.showinfo("Success", "Customer updated successfully.")

except Exception as e:

print(f"Error updating customer: {e}")

messagebox.showerror("Error", f"Error updating customer: {e}")

def delete\_customer(customer\_id):

try:

conn = sqlite3.connect('pharmacy.db')

cursor = conn.cursor()

cursor.execute("DELETE FROM customers WHERE customer\_id = ?", (customer\_id,))

conn.commit()

conn.close()

messagebox.showinfo("Success", "Customer deleted successfully.")

except Exception as e:

print(f"Error deleting customer: {e}")

messagebox.showerror("Error", f"Error deleting customer: {e}")

def add\_medicine(name, manufacturer, price, quantity):

try:

conn = sqlite3.connect('pharmacy.db')

cursor = conn.cursor()

cursor.execute("INSERT INTO medicines (name, manufacturer, price, quantity) VALUES (?, ?, ?, ?)", (name, manufacturer, price, quantity))

conn.commit()

conn.close()

messagebox.showinfo("Success", "Medicine added successfully.")

except Exception as e:

print(f"Error adding medicine: {e}")

messagebox.showerror("Error", f"Error adding medicine: {e}")

def update\_medicine(medicine\_id, name, manufacturer, price, quantity):

try:

conn = sqlite3.connect('pharmacy.db')

cursor = conn.cursor()

cursor.execute("UPDATE medicines SET name = ?, manufacturer = ?, price = ?, quantity = ? WHERE medicine\_id = ?", (name, manufacturer, price, quantity, medicine\_id))

conn.commit()

conn.close()

messagebox.showinfo("Success", "Medicine updated successfully.")

except Exception as e:

print(f"Error updating medicine: {e}")

messagebox.showerror("Error", f"Error updating medicine: {e}")

def delete\_medicine(medicine\_id):

try:

conn = sqlite3.connect('pharmacy.db')

cursor = conn.cursor()

cursor.execute("DELETE FROM medicines WHERE medicine\_id = ?", (medicine\_id,))

conn.commit()

conn.close()

messagebox.showinfo("Success", "Medicine deleted successfully.")

except Exception as e:

print(f"Error deleting medicine: {e}")

messagebox.showerror("Error", f"Error deleting medicine: {e}")

def list\_customers():

try:

conn = sqlite3.connect('pharmacy.db')

cursor = conn.cursor()

cursor.execute("SELECT \* FROM customers")

customers = cursor.fetchall()

customer\_list.delete('1.0', tk.END)

for customer in customers:

customer\_list.insert(tk.END, f"ID: {customer[0]}, Name: {customer[1]}, Contact: {customer[2]}, Address: {customer[3]}\n")

customer\_list.pack(pady=10)

conn.close()

except sqlite3.OperationalError as e:

print(f"Error fetching customers: {e}")

messagebox.showerror("Database Error", f"Error fetching customers: {e}")

def list\_medicines():

try:

conn = sqlite3.connect('pharmacy.db')

cursor = conn.cursor()

cursor.execute("SELECT \* FROM medicines")

medicines = cursor.fetchall()

medicine\_list.delete('1.0', tk.END)

for medicine in medicines:

medicine\_list.insert(tk.END, f"ID: {medicine[0]}, Name: {medicine[1]}, Manufacturer: {medicine[2]}, Price: {medicine[3]}, Quantity: {medicine[4]}\n")

medicine\_list.pack(pady=10)

conn.close()

except sqlite3.OperationalError as e:

print(f"Error fetching medicines: {e}")

messagebox.showerror("Database Error", f"Error fetching medicines: {e}")

def add\_customer\_ui():

def submit\_customer():

name = name\_entry.get()

contact = contact\_entry.get()

address = address\_entry.get()

add\_customer(name, contact, address)

customer\_window = tk.Toplevel(root)

customer\_window.title("Add Customer")

tk.Label(customer\_window, text="Name:").grid(row=0, column=0)

tk.Label(customer\_window, text="Contact:").grid(row=1, column=0)

tk.Label(customer\_window, text="Address:").grid(row=2, column=0)

name\_entry = tk.Entry(customer\_window)

contact\_entry = tk.Entry(customer\_window)

address\_entry = tk.Entry(customer\_window)

name\_entry.grid(row=0, column=1)

contact\_entry.grid(row=1, column=1)

address\_entry.grid(row=2, column=1)

tk.Button(customer\_window, text="Submit", command=submit\_customer).grid(row=3, columnspan=2)

def update\_customer\_ui():

def submit\_update\_customer():

customer\_id = int(id\_entry.get())

name = name\_entry.get()

contact = contact\_entry.get()

address = address\_entry.get()

update\_customer(customer\_id, name, contact, address)

customer\_window = tk.Toplevel(root)

customer\_window.title("Update Customer")

tk.Label(customer\_window, text="Customer ID:").grid(row=0, column=0)

tk.Label(customer\_window, text="Name:").grid(row=1, column=0)

tk.Label(customer\_window, text="Contact:").grid(row=2, column=0)

tk.Label(customer\_window, text="Address:").grid(row=3, column=0)

id\_entry = tk.Entry(customer\_window)

name\_entry = tk.Entry(customer\_window)

contact\_entry = tk.Entry(customer\_window)

address\_entry = tk.Entry(customer\_window)

id\_entry.grid(row=0, column=1)

name\_entry.grid(row=1, column=1)

contact\_entry.grid(row=2, column=1)

address\_entry.grid(row=3, column=1)

tk.Button(customer\_window, text="Submit", command=submit\_update\_customer).grid(row=4, columnspan=2)

def delete\_customer\_ui():

def submit\_delete\_customer():

customer\_id = int(id\_entry.get())

delete\_customer(customer\_id)

customer\_window = tk.Toplevel(root)

customer\_window.title("Delete Customer")

tk.Label(customer\_window, text="Customer ID:").grid(row=0, column=0)

id\_entry = tk.Entry(customer\_window)

id\_entry.grid(row=0, column=1)

tk.Button(customer\_window, text="Submit", command=submit\_delete\_customer).grid(row=1, columnspan=2)

def add\_medicine\_ui():

def submit\_medicine():

name = name\_entry.get()

manufacturer = manufacturer\_entry.get()

price = float(price\_entry.get())

quantity = int(quantity\_entry.get())

add\_medicine(name, manufacturer, price, quantity)

medicine\_window = tk.Toplevel(root)

medicine\_window.title("Add Medicine")

tk.Label(medicine\_window, text="Name:").grid(row=0, column=0)

tk.Label(medicine\_window, text="Manufacturer:").grid(row=1, column=0)

tk.Label(medicine\_window, text="Price:").grid(row=2, column=0)

tk.Label(medicine\_window, text="Quantity:").grid(row=3, column=0)

name\_entry = tk.Entry(medicine\_window)

manufacturer\_entry = tk.Entry(medicine\_window)

price\_entry = tk.Entry(medicine\_window)

quantity\_entry = tk.Entry(medicine\_window)

name\_entry.grid(row=0, column=1)

manufacturer\_entry.grid(row=1, column=1)

price\_entry.grid(row=2, column=1)

quantity\_entry.grid(row=3, column=1)

tk.Button(medicine\_window, text="Submit", command=submit\_medicine).grid(row=4, columnspan=2)

def update\_medicine\_ui():

def submit\_update\_medicine():

medicine\_id = int(id\_entry.get())

name = name\_entry.get()

manufacturer = manufacturer\_entry.get()

price = float(price\_entry.get())

quantity = int(quantity\_entry.get())

update\_medicine(medicine\_id, name, manufacturer, price, quantity)

medicine\_window = tk.Toplevel(root)

medicine\_window.title("Update Medicine")

tk.Label(medicine\_window, text="Medicine ID:").grid(row=0, column=0)

tk.Label(medicine\_window, text="Name:").grid(row=1, column=0)

tk.Label(medicine\_window, text="Manufacturer:").grid(row=2, column=0)

tk.Label(medicine\_window, text="Price:").grid(row=3, column=0)

tk.Label(medicine\_window, text="Quantity:").grid(row=4, column=0)

id\_entry = tk.Entry(medicine\_window)

name\_entry = tk.Entry(medicine\_window)

manufacturer\_entry = tk.Entry(medicine\_window)

price\_entry = tk.Entry(medicine\_window)

quantity\_entry = tk.Entry(medicine\_window)

id\_entry.grid(row=0, column=1)

name\_entry.grid(row=1, column=1)

manufacturer\_entry.grid(row=2, column=1)

price\_entry.grid(row=3, column=1)

quantity\_entry.grid(row=4, column=1)

tk.Button(medicine\_window, text="Submit", command=submit\_update\_medicine).grid(row=5, columnspan=2)

def delete\_medicine\_ui():

def submit\_delete\_medicine():

medicine\_id = int(id\_entry.get())

delete\_medicine(medicine\_id)

medicine\_window = tk.Toplevel(root)

medicine\_window.title("Delete Medicine")

tk.Label(medicine\_window, text="Medicine ID:").grid(row=0, column=0)

id\_entry = tk.Entry(medicine\_window)

id\_entry.grid(row=0, column=1)

tk.Button(medicine\_window, text="Submit", command=submit\_delete\_medicine).grid(row=1, columnspan=2)

root = tk.Tk()

root.title("Pharmacy Management System")

tk.Button(root, text="Add Customer", command=add\_customer\_ui).pack(pady=5)

tk.Button(root, text="Update Customer", command=update\_customer\_ui).pack(pady=5)

tk.Button(root, text="Delete Customer", command=delete\_customer\_ui).pack(pady=5)

tk.Button(root, text="Add Medicine", command=add\_medicine\_ui).pack(pady=5)

tk.Button(root, text="Update Medicine", command=update\_medicine\_ui).pack(pady=5)

tk.Button(root, text="Delete Medicine", command=delete\_medicine\_ui).pack(pady=5)

tk.Button(root, text="List Customers", command=list\_customers).pack(pady=5)

tk.Button(root, text="List Medicines", command=list\_medicines).pack(pady=5)

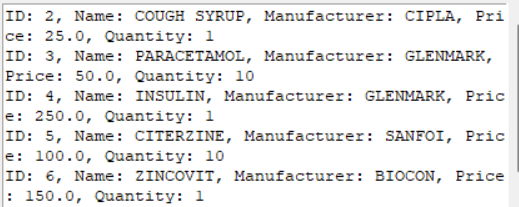
customer\_list = scrolledtext.ScrolledText(root, width=50, height=10)

medicine\_list = scrolledtext.ScrolledText(root, width=50, height=10)

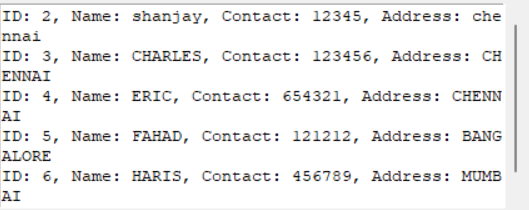
initialize\_db()

root.mainloop()

**TABLE FOR MEDICINE**

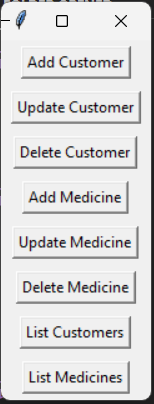
****

**TABLE FOR CUSTOMERS**

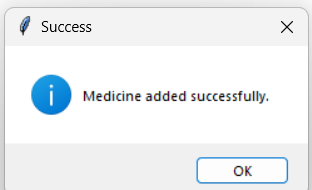
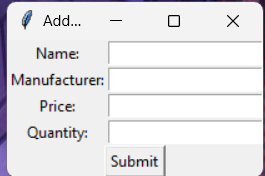
****

**OUTPUT**

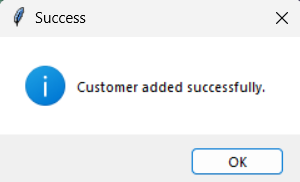
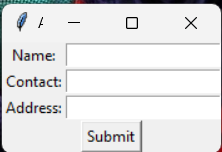
* **STARTING INTERFACE:**

****

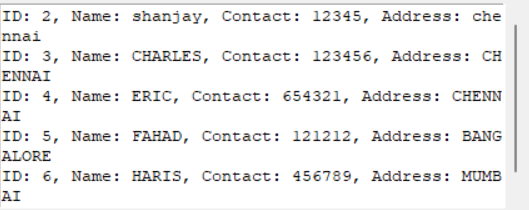
* **ADD MEDICINE INTERFACE:**

****

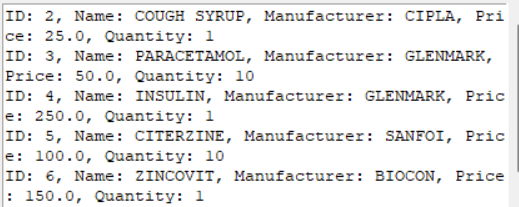
* **ADD CUSTOMER:**

****

* **CUSTOMER INFO INTERFACE:**

****

* **WHEN CLICKING THE LIST MEDICINE :**

****

**RESULTS AND DISCUSSION**

The program above implements a basic supermarket billing system using SQLite for the database and Tkinter for the graphical user interface (GUI). Here are the key features and results of the program:

**Key Features**

1.Database Management:

The program creates four tables in SQLite: Customers, Items, Sales, and Sales\_History.

Functions for adding customers and items to the database.

Functions to retrieve inventory and sales history.

2.GUI Interface:

A main window with buttons for viewing inventory, sales history, adding customers, adding items, and processing sales.

Separate windows for each feature to add items/customers and to view inventory/sales history.

A sale processing window where items can be added to a sale, and the sale can be finalized.

Results

The system successfully manages customer and item data and processes sales, updating the inventory accordingly.

It provides a user-friendly interface to perform different operations related to supermarket management.

The sales processing functionality ensures that the inventory is updated correctly, and sales records are maintained.

**Discussion**

**Advantages:**

The program provides a straightforward solution for small-scale PHARMACY management.

The use of Tkinter ensures that the GUI is simple and easy to use.

**Limitations**:

The program could benefit from additional error handling and validation (e.g., checking for non-numeric input where numeric input is expected).

More complex features like report generation, user authentication, and role-based access control are not included.

**Future Improvements:**

Adding features such as data visualization for sales trends, advanced inventory management, and customer loyalty programs.

Implementing a more sophisticated user interface with modern GUI frameworks.

**6. CONCLUSION**

By leveraging the strengths of Python and SQL, the Pharmacy Management System achieves a robust, scalable, and user-friendly solution for managing pharmacy operations. Python's simplicity and extensive library support, combined with SQL's powerful data management capabilities, provide a solid foundation for building efficient and reliable applications.

**7. REFERENCE**

en.wikipedia.org

www.mysql.com

www.w3schools.com

www.javatpoint