

# **SUPERMARKET MANAGEMENT SYSTEM**

## **A MINI PROJECT REPORT**

**Submitted by**

**SANTHOSH KUMAR R      220701253**

**SHARAN KUMAR D      220701264**

**PRRANAV P      220701207**

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 “**SUPERMARKET MANAGEMENT SYSTEM**” is the bonafide work of “**SANTHOSH KUMAR R(220701253), SHARAN KUMAR D(220701264),PRRANAV P(220701207)**”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 Supermarket Management System is a desktop application developed using Python and SQLite with a graphical user interface built using Tkinter. This application facilitates efficient management of supermarket operations, including inventory management, customer management, sales processing, and report generation. Users can perform essential operations such as adding, updating, and deleting inventory items, recording customer details, processing sales transactions, and viewing detailed reports of inventory, customers, and sales history. Each item in the inventory has a unique identification number, and customers can purchase items by entering the item ID and quantity. The system automatically updates the inventory quantities based on sales and ensures that accurate and up-to-date records are maintained. The application also provides functionality to view the purchase history of individual customers. This system aims to streamline supermarket management processes, improve data accuracy, and enhance overall operational efficiency.



## **TABLE OF CONTENTS**

<b>S.No</b>	<b>CONTENT</b>	<b>Pg.No</b>
<b>1</b>	<b>INTRODUCTION</b>	7
	1.1 INTRODUCTION	7
	1.2 OBJECTIVES	7
	1.3 MODULES	9
<b>2</b>	<b>SURVEY OF TECHNOLOGIES</b>	12
	2.1 SOFTWARE DESCRIPTION	12
	2.2 LANGUAGES	12
	2.2.1 SQL	
	2.2.2 PYTHON	
<b>3</b>	<b>REQUIREMENTS AND ANALYSIS</b>	14
	3.1 REQUIREMENT SPECIFICATION	14
	3.2 HARDWARE AND SOFTWARE REQUIREMENTS	15
	3.3 ARCHITECTURE DIAGRAM	17
	3.4 ER DIAGRAM	18
<b>4</b>	<b>PROGRAM CODE</b>	18
<b>5</b>	<b>RESULTS AND DISCUSSION</b>	35
<b>6</b>	<b>CONCLUSION</b>	38
<b>7</b>	<b>REFERENCES</b>	38

# **INTRODUCTION**

## **1.1 Introduction**

The Supermarket Management System is designed to streamline the management processes within a supermarket. It integrates functionalities such as inventory management, customer management, sales processing, and report generation into a single, user-friendly application. The system utilizes a graphical user interface (GUI) built with Tkinter and leverages SQLite for data storage and retrieval.

## **1.2 Objectives**

### **1. To provide a user-friendly interface for managing supermarket operations:**

- **Ease of Use:** The system should be intuitive and easy to navigate, allowing users to quickly learn how to perform necessary tasks without extensive training.
- **Graphical User Interface (GUI):** A well-designed GUI built with Tkinter ensures that users can interact with the system through simple point-and-click actions, minimizing the need for manual input and reducing errors.

### **2. To maintain accurate and up-to-date inventory records:**

- **Real-time Updates:** The system should immediately reflect any changes in the inventory, such as the addition of new items, updates to existing items, or removal of out-of-stock items.

- **Data Integrity:** Implement mechanisms to prevent data inconsistencies and ensure that inventory records accurately represent the current stock levels at all times.
3. **To facilitate efficient customer management and sales processing:**
- **Customer Information:** Maintain comprehensive records of customers, including their contact details, purchase history, and preferences, to enhance customer service and loyalty programs.
  - **Sales Transactions:** Streamline the sales process to quickly and accurately process customer purchases, update inventory levels, and generate receipts or invoices.
4. **To generate detailed reports on inventory, customers, and sales:**
- **Inventory Reports:** Provide insights into stock levels, fast-moving and slow-moving items, and inventory valuation.
  - **Customer Reports:** Generate reports on customer demographics, purchase patterns, and loyalty program participation.
  - **Sales Reports:** Offer detailed sales analysis, including daily, weekly, monthly sales figures, and sales by category or item.
5. **To ensure data integrity and ease of data retrieval:**
- **Database Management:** Utilize SQLite to store data in a structured format, ensuring data is correctly indexed and easily retrievable.
  - **Backup and Recovery:** Implement regular data backups and provide easy recovery options to protect against data loss and ensure business continuity.



## **1.3 Modules**

### **1. Inventory Management:**

- **Adding Items:**
  - Users can add new products to the inventory, specifying details such as product name, description, category, price, quantity, and supplier information.
  - The system validates the input to ensure all required fields are filled correctly and that there are no duplicate entries.
- **Updating Items:**
  - Users can update the details of existing products, such as adjusting prices, changing descriptions, or updating stock levels.
  - The system tracks these changes to provide a history of updates for auditing purposes.
- **Deleting Items:**
  - Users can remove products from the inventory if they are discontinued or no longer in stock.
  - The system ensures that deleting an item will not disrupt any ongoing processes or historical data integrity.

### **2. Customer Management:**

- **Adding New Customers:**
  - Users can add new customers to the system, capturing essential details such as name, contact information, and any preferences or notes.
  - This information helps in providing personalized service and marketing efforts.
- **Viewing Customer Information:**

- Users can view and search for customer records to quickly access relevant information during interactions.
- The system allows viewing a customer's purchase history and preferences, enabling better service and targeted promotions.

### **3. Sales Processing:**

- **Recording Sales Transactions:**

- The system allows users to record sales transactions, capturing details such as items sold, quantities, prices, and payment methods.
- Each transaction updates the inventory levels in real-time to reflect the current stock.

- **Updating Inventory Accordingly:**

- Upon completing a sale, the inventory is automatically updated to subtract the sold quantities from the current stock levels.
- This ensures accurate inventory tracking and helps in avoiding stockouts or overstocking situations.

### **4. Reporting:**

- **Viewing Inventory:**

- Users can generate and view reports on current inventory levels, including detailed information on each item and stock status.
- Reports can be filtered by categories, suppliers, or stock levels to provide specific insights.

- **Customer Lists:**

- The system can generate lists of customers, including contact details, purchase history, and segmentation based on various criteria.
- These lists are useful for targeted marketing campaigns and customer relationship management.

- **Sales History:**
  - Detailed sales reports are available, showing transaction history, sales trends, and performance metrics over specified periods.
  - Sales history can be analyzed to identify popular products, peak sales times, and sales by category or item.
- **Customer Purchase History:**
  - Users can view individual customer purchase histories to understand buying patterns and preferences.
  - This information supports personalized marketing, loyalty programs, and better customer service.

## **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**

#### **2.2.1 SQL**

**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.

## 2.2.2 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.

## **3. REQUIREMENTS AND ANALYSIS**

### **3.1 Requirement Specification**

#### **Functional Requirements:**

The system shall allow the user to add, update, and delete inventory items.

The system shall allow the user to add new customers.

The system shall allow the user to record sales transactions.

The system shall update inventory quantities based on sales.

The system shall provide views for inventory, customers, and sales history.

The system shall provide a view for the purchase history of a specific customer.

## **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.

## **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.

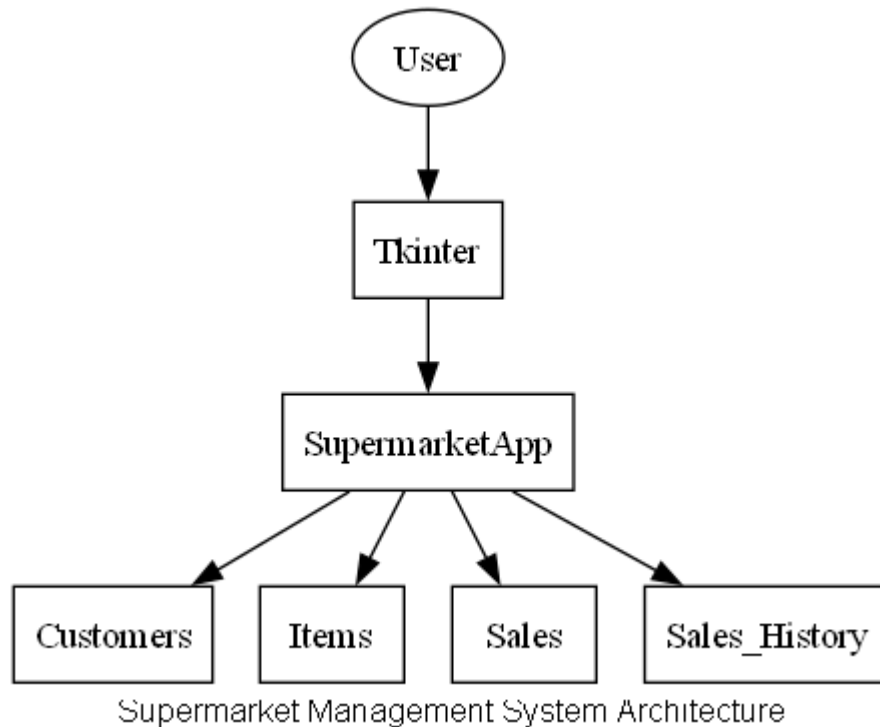
Tkinter library for Python (usually included with standard Python installation).

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

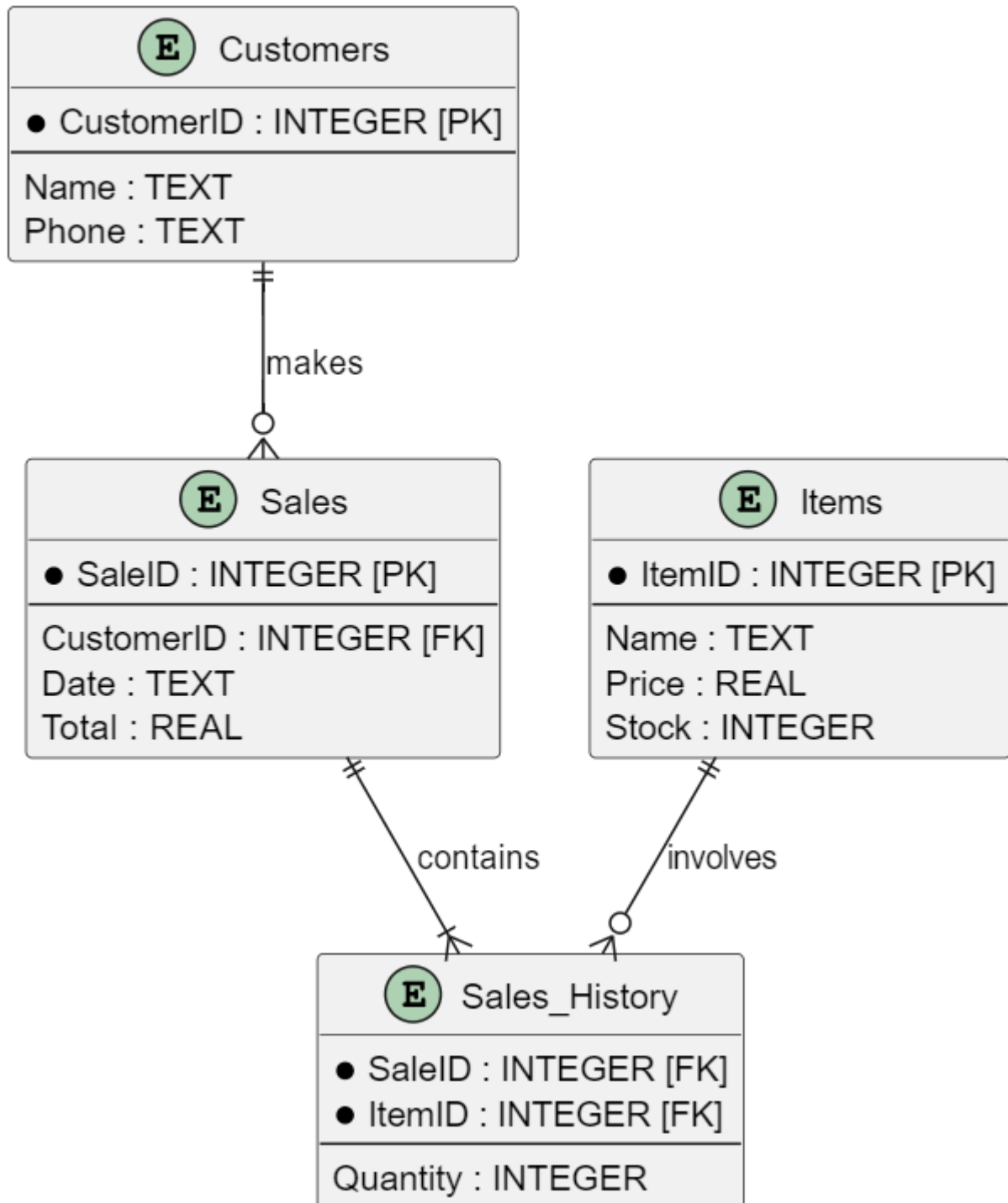
By addressing these requirements and utilizing the specified technologies, the Supermarket Management System aims to enhance the efficiency and accuracy of supermarket operations, providing a comprehensive solution for inventory, customer, and sales management.



### **3.3ARCHITECTURE DIAGRAM**



### 3.4 ER DIAGRAM



## **PROGRAM CODES:**

- **PYTHON WITH SQLINTE FOR THE CREATION OF CUSTOMER,SALES AND INVENTORY TABLES:**

```
import sqlite3
from datetime import datetime

def initialize_db():
    conn = sqlite3.connect('supermarket.db')
    cursor = conn.cursor()

    # Drop existing tables (for testing purposes)
    cursor.execute('DROP TABLE IF EXISTS inventory')
    cursor.execute('DROP TABLE IF EXISTS customers')
    cursor.execute('DROP TABLE IF EXISTS sales')

    # Create tables
    cursor.execute("""
CREATE TABLE IF NOT EXISTS inventory (
    item_id INTEGER PRIMARY KEY AUTOINCREMENT,
    item_name TEXT NOT NULL,
```

```
quantity INTEGER NOT NULL,  
price REAL NOT NULL  
)  
""
```

```
cursor.execute("""  
CREATE TABLE IF NOT EXISTS customers (  
    customer_id INTEGER PRIMARY KEY  
AUTOINCREMENT,  
    customer_name TEXT NOT NULL,  
    customer_phone TEXT NOT NULL  
)  
""
```

```
cursor.execute("""  
CREATE TABLE IF NOT EXISTS sales (  
    sale_id INTEGER PRIMARY KEY AUTOINCREMENT,  
    customer_id INTEGER,  
    item_id INTEGER,  
    quantity INTEGER NOT NULL,  
    sale_date TEXT NOT NULL,  
    FOREIGN KEY(customer_id) REFERENCES  
customers(customer_id),  
    FOREIGN KEY(item_id) REFERENCES inventory(item_id)
```

```
)  
"""
```

```
conn.commit()  
conn.close()
```

```
def execute_db_query(query, params=()):  
    conn = sqlite3.connect('supermarket.db')  
    cursor = conn.cursor()  
    cursor.execute(query, params)  
    conn.commit()  
    conn.close()
```

```
def fetch_db_query(query, params=()):  
    conn = sqlite3.connect('supermarket.db')  
    cursor = conn.cursor()  
    cursor.execute(query, params)  
    records = cursor.fetchall()  
    conn.close()  
    return records
```

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

```
import tkinter as tk
```

```
from tkinter import messagebox, simpledialog
```

```
from database import initialize_db, execute_db_query,  
fetch_db_query
```

```
class SupermarketApp:
```

```
    def __init__(self, root):
```

```
        self.root = root
```

```
        self.root.title("Supermarket Management System")
```

```
        self.root.geometry("600x400")
```

```
        self.create_widgets()
```

```
    def create_widgets(self):
```

```
        # Create Frames
```

```
        self.frame_title = tk.Frame(self.root, pady=20)
```

```
        self.frame_title.pack()
```

```
        self.frame_buttons_top = tk.Frame(self.root, pady=10)
```

```
        self.frame_buttons_top.pack()
```

```
        self.frame_buttons_middle = tk.Frame(self.root, pady=10)
```

```
self.frame_buttons_middle.pack()
```

```
self.frame_buttons_bottom = tk.Frame(self.root, pady=10)
```

```
self.frame_buttons_bottom.pack()
```

```
# Title Label
```

```
self.label_title = tk.Label(self.frame_title, text="Supermarket  
Management System", font=("Helvetica", 16, "bold"))
```

```
self.label_title.pack()
```

```
# Create Buttons with Styling
```

```
button_font = ("Helvetica", 12)
```

```
button_bg = "#4CAF50"
```

```
button_fg = "#FFFFFF"
```

```
button_padx = 15
```

```
button_pady = 10
```

```
self.btn_add_item = tk.Button(self.frame_buttons_top,  
text="Add Item", font=button_font, bg=button_bg, fg=button_fg,  
command=self.add_item, padx=button_padx, pady=button_pady)
```

```
self.btn_update_item = tk.Button(self.frame_buttons_top,  
text="Update Item", font=button_font, bg=button_bg, fg=button_fg,  
command=self.update_item, padx=button_padx, pady=button_pady)
```

```
self.btn_delete_item = tk.Button(self.frame_buttons_top,  
text="Delete Item", font=button_font, bg=button_bg, fg=button_fg,  
command=self.delete_item, padx=button_padx, pady=button_pady)
```

```
self.btn_add_customer = tk.Button(self.frame_buttons_middle,  
text="Add Customer", font=button_font, bg=button_bg,  
fg=button_fg, command=self.add_customer, padx=button_padx,  
pady=button_pady)
```

```
self.btn_make_sale = tk.Button(self.frame_buttons_middle,  
text="Make Sale", font=button_font, bg=button_bg, fg=button_fg,  
command=self.make_sale, padx=button_padx, pady=button_pady)
```

```
self.btn_view_inventory =  
tk.Button(self.frame_buttons_bottom, text="View Inventory",  
font=button_font, bg=button_bg, fg=button_fg,  
command=self.view_inventory, padx=button_padx,  
pady=button_pady)
```

```
self.btn_view_customers =  
tk.Button(self.frame_buttons_bottom, text="View Customers",  
font=button_font, bg=button_bg, fg=button_fg,  
command=self.view_customers, padx=button_padx,  
pady=button_pady)
```

```
self.btn_view_sales = tk.Button(self.frame_buttons_bottom,  
text="View Sales History", font=button_font, bg=button_bg,  
fg=button_fg, command=self.view_sales, padx=button_padx,  
pady=button_pady)
```

```
self.btn_customer_purchase_history =  
tk.Button(self.frame_buttons_bottom, text="Customer Purchase  
History", font=button_font, bg=button_bg, fg=button_fg,  
command=self.view_customer_purchase_history,  
padx=button_padx, pady=button_pady)
```



```

# Layout the buttons
self.btn_add_item.grid(row=0, column=0, padx=10, pady=10)
self.btn_update_item.grid(row=0, column=1, padx=10,
pady=10)
self.btn_delete_item.grid(row=0, column=2, padx=10,
pady=10)
self.btn_add_customer.grid(row=1, column=0, padx=10,
pady=10)
self.btn_make_sale.grid(row=1, column=1, padx=10, pady=10)
self.btn_view_inventory.grid(row=2, column=0, padx=10,
pady=10)
self.btn_view_customers.grid(row=2, column=1, padx=10,
pady=10)
self.btn_view_sales.grid(row=2, column=2, padx=10, pady=10)
self.btn_customer_purchase_history.grid(row=3, column=1,
padx=10, pady=10)

def add_item(self):
    name = simpledialog.askstring("Input", "Enter item name:")
    quantity = simpledialog.askinteger("Input", "Enter item
quantity:")
    price = simpledialog.askfloat("Input", "Enter item price:")
    if name and quantity is not None and price is not None:

```

```
execute_db_query("INSERT INTO inventory (item_name,  
quantity, price) VALUES (?, ?, ?)", (name, quantity, price))  
messagebox.showinfo("Success", "Item added successfully")
```

```
def update_item(self):  
    item_id = simpledialog.askinteger("Input", "Enter item ID to  
update:")  
    name = simpledialog.askstring("Input", "Enter new item  
name:")  
    quantity = simpledialog.askinteger("Input", "Enter new item  
quantity:")  
    price = simpledialog.askfloat("Input", "Enter new item price:")  
    if item_id and name and quantity is not None and price is not  
None:  
        execute_db_query("UPDATE inventory SET item_name = ?,  
quantity = ?, price = ? WHERE item_id = ?", (name, quantity, price,  
item_id))  
        messagebox.showinfo("Success", "Item updated  
successfully")
```

```
def delete_item(self):  
    item_id = simpledialog.askinteger("Input", "Enter item ID to  
delete:")  
    if item_id:  
        execute_db_query("DELETE FROM inventory WHERE  
item_id = ?", (item_id,))
```

```
        messagebox.showinfo("Success", "Item deleted  
successfully")
```

```
def add_customer(self):
```

```
    name = simpledialog.askstring("Input", "Enter customer  
name:")
```

```
    phone = simpledialog.askstring("Input", "Enter customer  
phone:")
```

```
    if name and phone:
```

```
        execute_db_query("INSERT INTO customers  
(customer_name, customer_phone) VALUES (?, ?)", (name, phone))
```

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

```
def make_sale(self):
```

```
    customer_id = simpledialog.askinteger("Input", "Enter  
customer ID:")
```

```
    item_id = simpledialog.askinteger("Input", "Enter item ID:")
```

```
    quantity = simpledialog.askinteger("Input", "Enter quantity  
sold:")
```

```
    if customer_id and item_id and quantity:
```

```
        execute_db_query("INSERT INTO sales (customer_id,  
item_id, quantity, sale_date) VALUES (?, ?, ?, ?)", (customer_id,  
item_id, quantity, datetime.now().strftime("%Y-%m-%d  
%H:%M:%S")))
```

```
execute_db_query("UPDATE inventory SET quantity =  
quantity - ? WHERE item_id = ?", (quantity, item_id))
```

```
messagebox.showinfo("Success", "Sale recorded  
successfully")
```

```
def view_inventory(self):
```

```
    records = fetch_db_query("SELECT * FROM inventory")
```

```
    self.display_records(records, ["Item ID", "Item Name",  
"Quantity", "Price"])
```

```
def view_customers(self):
```

```
    records = fetch_db_query("SELECT * FROM customers")
```

```
    self.display_records(records, ["Customer ID", "Customer  
Name", "Customer Phone"])
```

```
def view_sales(self):
```

```
    records = fetch_db_query("""
```

```
        SELECT sales.sale_id, customers.customer_name,  
inventory.item_name, sales.quantity, sales.sale_date
```

```
        FROM sales
```

```
        JOIN customers ON sales.customer_id =  
customers.customer_id
```

```
        JOIN inventory ON sales.item_id = inventory.item_id
```

```
    """)
```

```
self.display_records(records, ["Sale ID", "Customer Name",  
"Item Name", "Quantity", "Sale Date"])
```

```
def view_customer_purchase_history(self):  
    customer_id = simpledialog.askinteger("Input", "Enter  
customer
```

## **TABLE FOR INVENTORY**

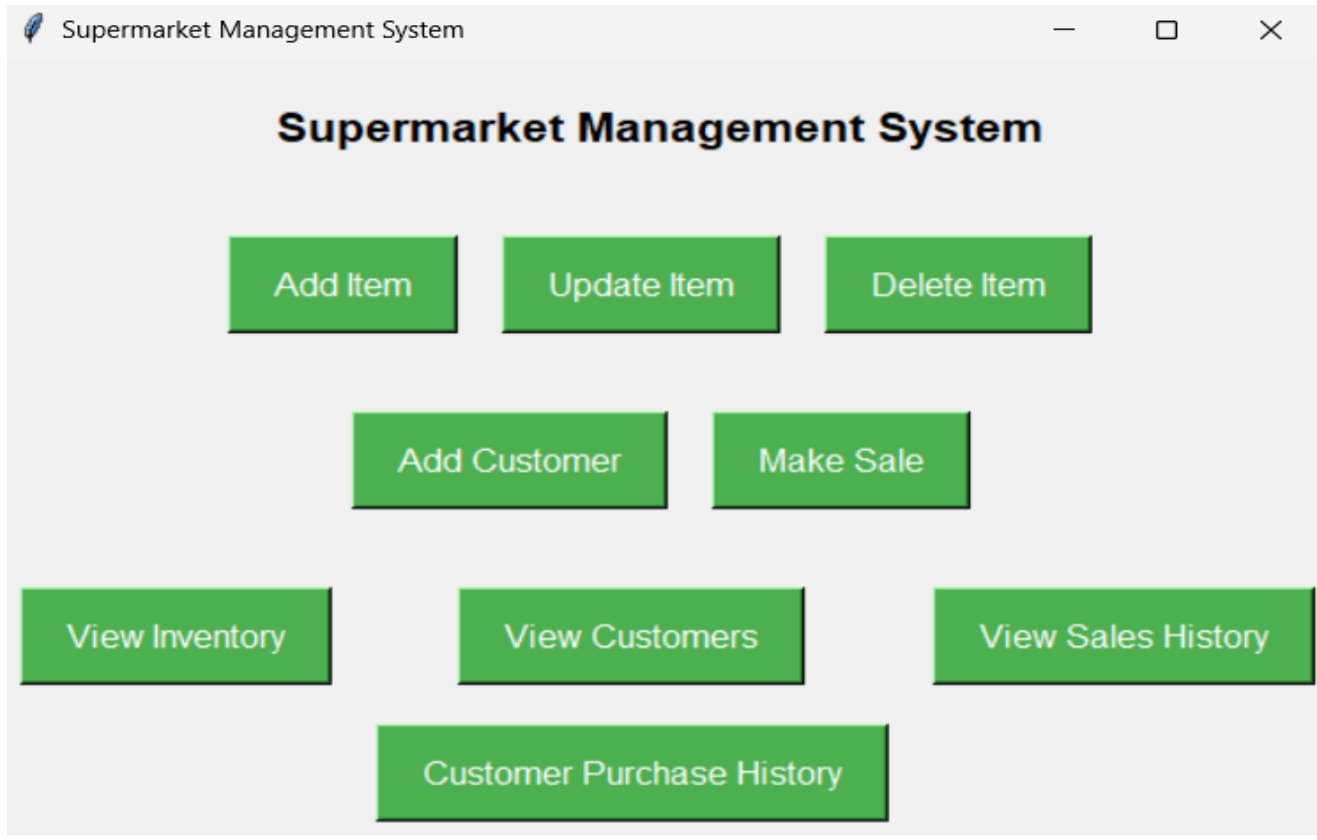
Records			
Item ID	Item Name	Quantity	Price
1	Milk	10	25.0
2	Cheese	5	450.0
3	Yogurt	5	25.0
4	Paneer	5	290.0
5	Butter	3	350.0
6	Egg	50	5.0
7	Chicken	10	200.0
8	Fish	16	200.0
9	Watermelon	4	75.0
10	noodle	10	50.0
11	pasta	9	250.0
12	Rice	10	450.0
13	Lays	20	5.0

## TABLE FOR CUSTOMERS

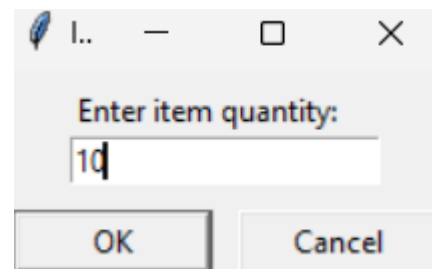
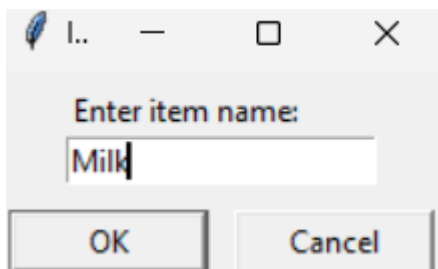
Records		
Customer ID	Customer Name	Customer Phone
1	Ajith	123456
2	Bala	111222
3	Charles	333444
4	Dinesh	121212
5	Eric	232323
6	Fahad	243123
7	Ganesh	834837
8	Haris	147643
9	Ilish	136923
10	Jagan	862456

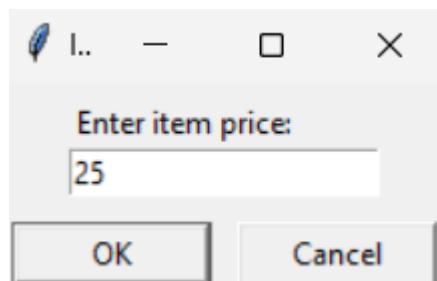
# OUTPUT

- **STARTING INTERFACE:**

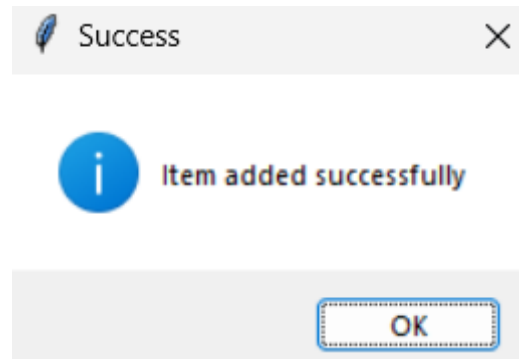


- **ADD ITEM INTERFACE:**



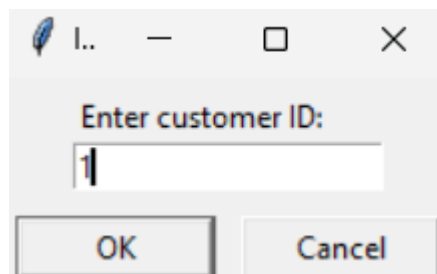


A standard Windows-style dialog box with a title bar containing a feather icon, the text 'I..', and standard window controls (minimize, maximize, close). The main area contains the text 'Enter item price:' followed by a text input field containing the number '25'. At the bottom are two buttons: 'OK' and 'Cancel'.

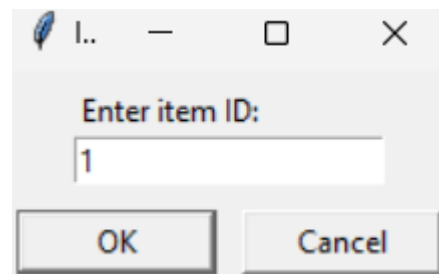


A success message dialog box with a title bar containing a feather icon, the text 'Success', and a close button. The main area features a blue circular information icon followed by the text 'Item added successfully'. At the bottom is a single 'OK' button.

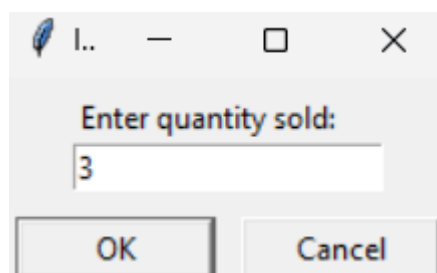
- **MAKE SALE:**



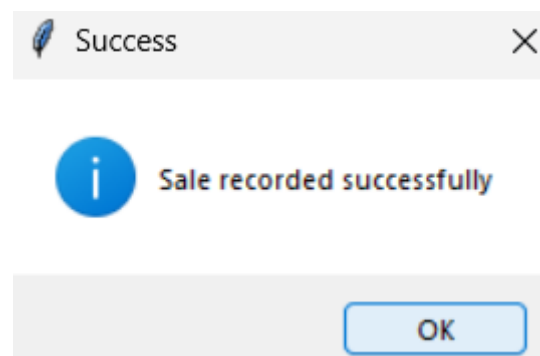
A standard Windows-style dialog box with a title bar containing a feather icon, the text 'I..', and standard window controls. The main area contains the text 'Enter customer ID:' followed by a text input field containing the number '1'. At the bottom are two buttons: 'OK' and 'Cancel'.



A standard Windows-style dialog box with a title bar containing a feather icon, the text 'I..', and standard window controls. The main area contains the text 'Enter item ID:' followed by a text input field containing the number '1'. At the bottom are two buttons: 'OK' and 'Cancel'.



A standard Windows-style dialog box with a title bar containing a feather icon, the text 'I..', and standard window controls. The main area contains the text 'Enter quantity sold:' followed by a text input field containing the number '3'. At the bottom are two buttons: 'OK' and 'Cancel'.



A success message dialog box with a title bar containing a feather icon, the text 'Success', and a close button. The main area features a blue circular information icon followed by the text 'Sale recorded successfully'. At the bottom is a single 'OK' button.

- **SALES HISTORY INTERFACE:**

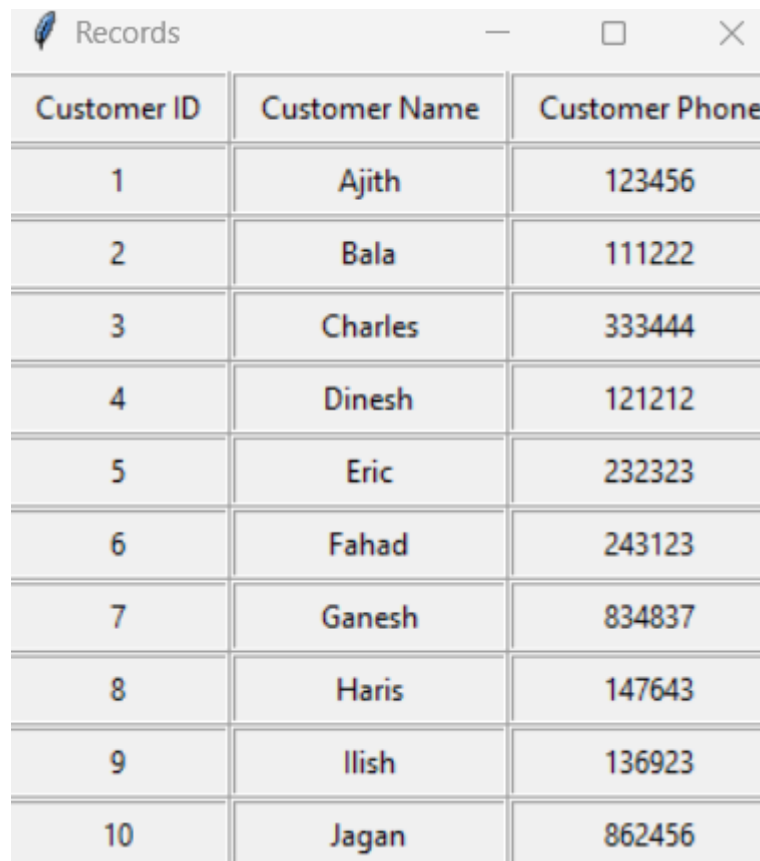


Records				
Sale ID	Customer Name	Item Name	Quantity	Sale Date
1	ALBERT	Milk	1	2024-05-20 18:37:03
2	ALBERT	Milk	3	2024-05-20 18:45:39

- **WHEN CLICKING THE VIEW INVENTORY :**

Records			
Item ID	Item Name	Quantity	Price
1	Milk	10	25.0
2	Cheese	5	450.0
3	Yogurt	5	25.0
4	Paneer	5	290.0
5	Butter	3	350.0
6	Egg	50	5.0
7	Chicken	10	200.0
8	Fish	16	200.0
9	Watermelon	4	75.0
10	noodle	10	50.0
11	pasta	9	250.0
12	Rice	10	450.0
13	Lays	20	5.0

- **WHEN CLICKING THE VIEW CUSTOMERS:**



A screenshot of a Tkinter window titled "Records". The window contains a table with three columns: "Customer ID", "Customer Name", and "Customer Phone". The table lists 10 customers with their respective IDs, names, and phone numbers.

Customer ID	Customer Name	Customer Phone
1	Ajith	123456
2	Bala	111222
3	Charles	333444
4	Dinesh	121212
5	Eric	232323
6	Fahad	243123
7	Ganesh	834837
8	Haris	147643
9	Ilish	136923
10	Jagan	862456

## **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 supermarket 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**

The supermarket billing system developed in this project demonstrates how SQLite and Tkinter can be combined to create a functional application for managing supermarket operations. The program successfully handles customer and item data, processes sales, and provides a user-friendly interface for interacting with the system. Future improvements can further enhance its functionality and usability, making it suitable for more extensive applications.

## **7. REFERENCE**

- <https://www.google.com/amp/s/www.geeksforgeeks.org/python-sqlite/amp/>
- <https://en.m.wikipedia.org/wiki/SQLite>
- <https://www.javatpoint.com/python-sqlite>