

Hillwoods School

A Project Report

On

Bank Management System

For

AISSCE 2024-25 Examination

[as a part of the computer science

course new (083)]

Submitted By -

Student Name – Pratham Kariya

Roll No. -

Under the Guidance of:

Ms. Ritu Dembla

PGT C.S



Hillwoods School

CERTIFICATE

This is to certify that the Project of Grade XII Computer Science titled Bank Management System is developed by the following students of Grade 12 Science named, Mr. Pratham Kariya , Roll No. _____ has satisfactorily completed their project in partial fulfilment of CBSE's AISSCE Examination 2024-25.

This is the original work and carried out under our direct supervision and guidance. This report or a similar report on the topic has not been submitted for any other examination and does not form a part of any other course undergone by the candidate. To the best of our knowledge and belief, we further certify that the matter presented in this project report is bonafide.

Date: _____

Subject Teacher

External Examiner

Principal

INDEX

<u>SR. NO.</u>	<u>TITLE</u>	<u>PAGE NO.</u>
<u>1.</u>	<u>Acknowledgment</u>	<u>4</u>
<u>2.</u>	<u>Preface</u>	<u>5</u>
<u>3.</u>	<u>Definition</u>	<u>6</u>
<u>4.</u>	<u>Problem Statement</u>	<u>8</u>
<u>5.</u>	<u>Objective</u>	<u>9</u>
<u>6.</u>	<u>Hardware and Software Used</u>	<u>11</u>
<u>7.</u>	<u>Source Code</u>	<u>13</u>
<u>8.</u>	<u>Output</u>	<u>24</u>
<u>9.</u>	<u>Contribution to Real World</u>	<u>30</u>
<u>10.</u>	<u>Bibliography</u>	<u>32</u>
<u>11.</u>	<u>Thank You</u>	<u>33</u>

ACKNOWLEDGEMENT

I undertook this Project work, as the part of my XII-Computer Science course. I

had tried to apply my best of knowledge and experience, gained during the study and class work experience. However, developing software system is generally a quite complex and time-consuming process. It requires a systematic study, insight vision and professional approach during the design and development. Moreover, the developer always feels the need, the help and good wishes of the people near you, who have considerable experience and idea.

I would like to extend my sincere thanks and gratitude to my teacher **Ms. Ritu Dembla**. I am very much thankful to our Principal **Dr. Smita Laur** for giving valuable time and moral support to develop this software.

I also feel indebted to my friends for their valuable suggestions during the project work.

(Signature of student)

Preface

This project aims to develop a comprehensive **Bank Management System** that streamlines banking operations, ensuring efficient account management, enhanced security, and improved customer service. With the increasing complexity of financial transactions and the growing need for automation, this system addresses the challenges of manual processes, making banking operations faster, more accurate, and secure.

The **Bank Management System** is designed to manage customer accounts, track balances, perform transactions, and provide employees with real-time access to critical information. It incorporates features such as employee login authentication, account creation, balance checking, and transaction processing. These features not only improve internal workflows but also enhance the overall customer experience by providing immediate access to account information.

In an era where digital banking is becoming increasingly important, the system ensures the security and integrity of sensitive data while supporting scalability for future growth. By automating routine tasks and improving access to real-time data, this system provides a foundation for efficient banking operations, contributing to the bank's overall success and customer satisfaction.

This program is a significant step towards modernizing banking systems, offering a robust solution to the challenges faced by financial institutions today.

Definition

A Bank Management System is a software application that helps manage the various banking operations such as account creation, balance checking, deposits, withdrawals, and transfers between accounts. It can be developed using Python, SQL, and Tkinter for the graphical user interface (GUI).

Features of a Bank Management System:

A typical Bank Management System includes several essential features, such as:

1. **Account Creation:** Allows the bank to register new customers by storing personal details and initial deposit information.
2. **Login/Authentication:** Provides a secure login system to ensure that only authorized users (e.g., bank staff or customers) can access the system.
3. **Deposit and Withdrawal:** Customers can deposit funds into their account or withdraw money, with checks in place to prevent overdrawning.
4. **Account Overview:** Displays the current balance, personal details, and transaction history of an account holder.
5. **Transaction History:** Logs and displays the history of deposits, withdrawals, and transfers for each user.
6. **Funds Transfer:** Customers can transfer money between accounts within the same bank.
7. **Simple Reports:** Basic reports showing the total balance and transaction history of customers.

Architecture and Design:

The system can be divided into three primary layers:

1. **Database Layer (SQL):** The database layer manages data storage and ensures that user information and transactions are securely saved. We can use SQLite to store customer data, transaction records, and balance information.
2. **Business Logic Layer (Python):** This layer contains the core functionality of the Bank Management System, such as performing transactions, handling account balance updates, and logging activities.
3. **Presentation Layer (Tkinter):** The user interface (UI) is the front end that allows users (bank employees or customers) to interact with the system. Tkinter is used to create forms, buttons, labels, and other interactive elements.

Database Design:

To store customer data, we can design two primary tables in SQL:

1. **Customers Table:** Stores basic information about customers, including their ID, name, address, and balance.
2. **Transactions Table:** Logs every transaction made, such as deposits, withdrawals, and transfers. It stores the transaction type, amount, date, and the customer ID associated with the transaction.

Tkinter User Interface:

The Tkinter library is used to build the graphical user interface (GUI) for the Bank Management System. The interface consists of windows for login, account management, and transaction processing. The GUI can be extended to include other functionalities such as balance checking, deposit/withdrawal forms, and transaction history displays.

Problem Statement

In traditional banking systems, manual processes for managing customer accounts, processing transactions, and maintaining customer data are time-consuming, prone to errors, and lack scalability. These inefficiencies lead to delays in customer service, increased human errors, and difficulties in handling large volumes of data, especially as the bank grows. Furthermore, the lack of secure and automated systems for accessing sensitive information creates vulnerabilities in data security, exposing both the bank and its customers to potential risks.

The challenge lies in developing an automated, secure, and scalable system that can streamline banking operations, enhance data security, and improve the efficiency of both employees and customers.

A solution is needed that can:

1. Automate account creation, modification, and deletion.
2. Provide real-time access to account balances and transaction histories.
3. Ensure secure access to sensitive customer data through employee login authentication.
4. Handle an increasing volume of transactions and customer data as the bank expands.

This project aims to address these challenges by creating a Bank Management System that automates routine tasks, improves operational efficiency, ensures data security, and supports the bank's growth, all while delivering a seamless and transparent experience to customers.

Objective

The **Bank Management System** project using **Python**, **SQL**, and **Tkinter** provides a robust platform for developing essential skills in software development and problem-solving. This project bridges the gap between theoretical learning and practical application, giving students the opportunity to work on a real-world simulation of a banking system. By tackling challenges in GUI design, database management, and secure data handling, students will gain a well-rounded understanding of programming principles, enhancing their preparation for more complex software development tasks.

Key Objectives and Scope:

1. Graphical User Interface (GUI):

- Designing an intuitive, interactive, and user-friendly interface using **Tkinter**.
- The GUI facilitates seamless interactions for both customers and employees, providing a visually appealing and organized layout.
- Ensuring the interface is responsive and professional, enhancing user experience and accessibility.

2. Database Integration:

- Leveraging SQL (SQLite/MySQL) to manage banking records securely.
- Storing critical data such as customer information, account details, transaction logs, and employee credentials.
- Ensuring data integrity and efficient retrieval through optimized queries and well-structured database schemas.

3. Core Banking Operations:

- Implementing fundamental banking functions, including:
 - Account creation and management.
 - Processing deposits, withdrawals, and fund transfers.
- Ensuring proper validation, error handling, and accurate updates to the database for all operations.

4. Security and Authentication:

- Introducing an employee login system to restrict unauthorized access and ensure secure operation.
- Implementing encrypted storage for passwords and sensitive information.
- Providing role-based access, differentiating between customer and employee functionalities.

5. Learning Outcomes:

- Hands-on experience with GUI development using Tkinter.
- Practical knowledge of integrating SQL databases with Python applications.
- Understanding of software design principles, including modularity, maintainability, and scalability.
- Insights into implementing real-world applications of authentication and security measures.

This project not only focuses on the technical aspects but also emphasizes critical thinking, problem-solving, and collaboration, providing a holistic approach to understanding and implementing software systems.

Hardware and Software Used

Hardware:

1. Computer/Server:

- A computer or server is required to run the banking system application, host the database, and manage user requests. It should have sufficient processing power, memory (RAM), and storage to support the software, handle concurrent user requests, and ensure smooth operation of the system.

2. Input Devices:

- **Keyboard and Mouse:** For inputting data such as customer information, transaction details, and performing administrative tasks within the system.

3. Printer (Optional):

- A printer can be used to generate receipts or transaction reports for customers, especially for printed statements or account summaries.

4. Network Connectivity:

- Reliable internet or intranet connection to ensure real-time synchronization between different components of the system and facilitate remote access or updates.

Software:

1. Operating System:

- **Windows/Linux:** The banking management system can run on either Windows or Linux-based operating systems, providing a stable environment for hosting the software.

2. Database Management System (DBMS):

- **MySQL:** MySQL is used for managing the bank's database, storing customer data, transaction histories, account details, and other necessary records in an organized manner.

3. Programming Language:

- **Python:** Python is used to build the bank management system application due to its simplicity, readability, and extensive libraries, allowing easy integration with the MySQL database for seamless data operations.

4. GUI Development:

- **Tkinter:** Tkinter, a Python library, is used for developing the graphical user interface (GUI) of the system, allowing users to interact with the banking system in an intuitive manner.

5. Database Management Tools:

- **MySQL Workbench:** Used for designing, querying, and managing the database, including setting up tables, relationships, and executing queries for customer and transaction data management.

6. Security Tools:

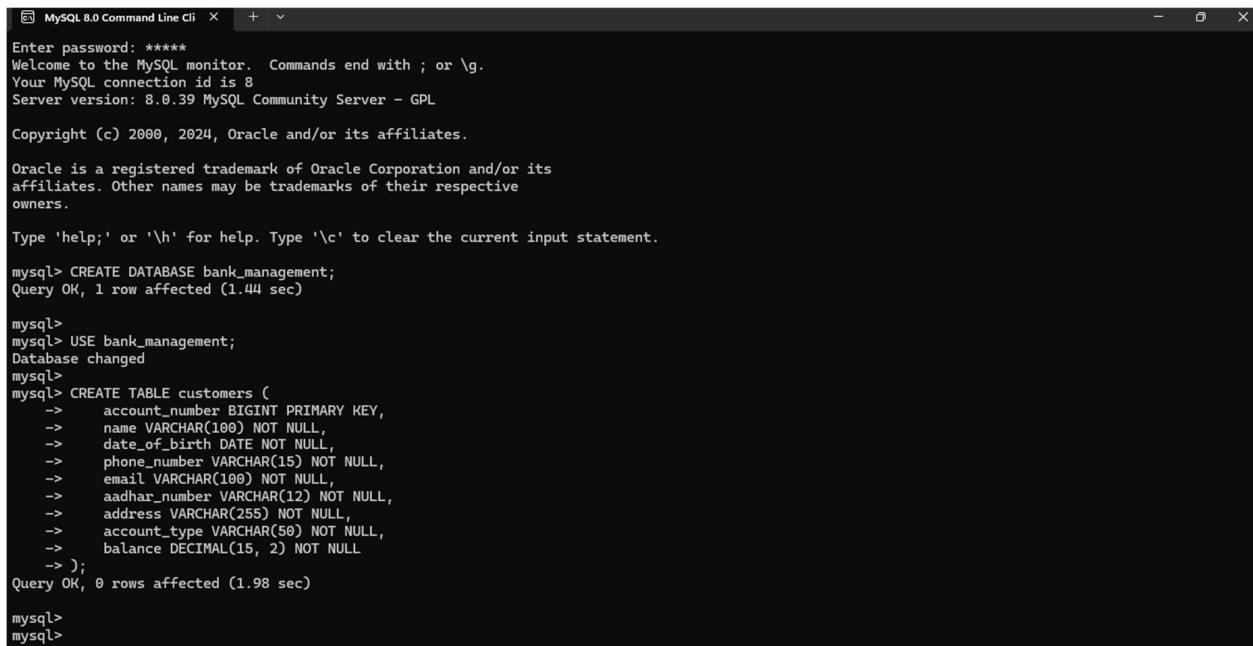
- **SSL/TLS Encryption (Optional):** Secure Sockets Layer (SSL) or Transport Layer Security (TLS) can be used to encrypt sensitive data during transmission, ensuring data security during online transactions and remote access.

These hardware and software components work together to create a seamless, efficient, and secure banking management system, automating core tasks and providing real-time access to customer and transaction information

Source Code

SQL:

- Creating Database “bank_management” and Table “customers”



```

MySQL 8.0 Command Line Cli  X  +  v

Enter password: *****
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 8
Server version: 8.0.39 MySQL Community Server - GPL

Copyright (c) 2000, 2024, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

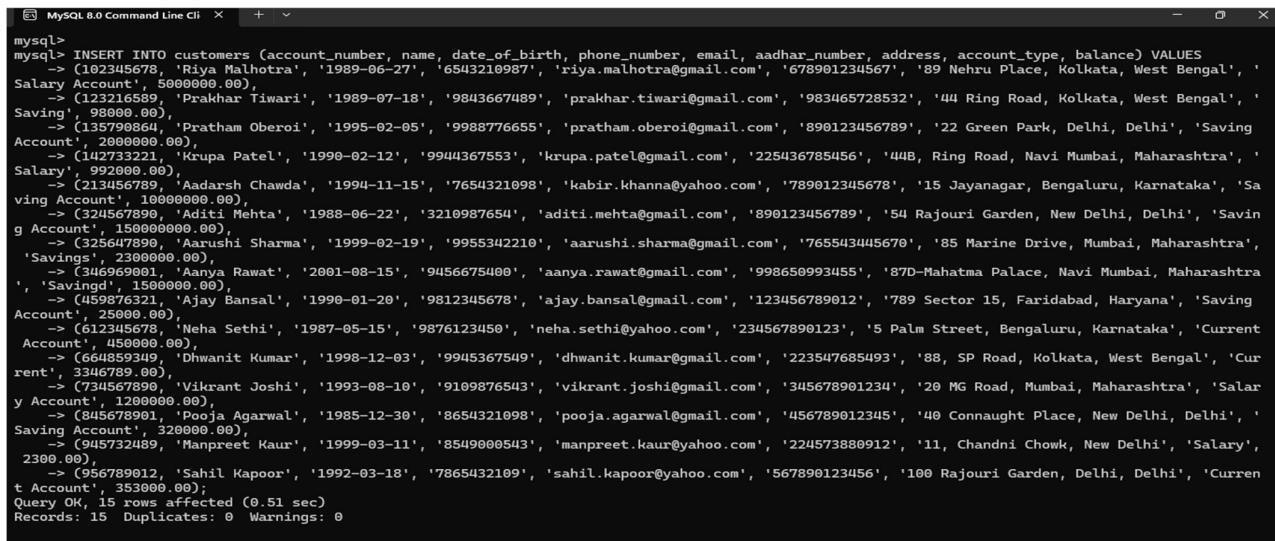
Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> CREATE DATABASE bank_management;
Query OK, 1 row affected (1.44 sec)

mysql>
mysql> USE bank_management;
Database changed
mysql>
mysql> CREATE TABLE customers (
    -> account_number BIGINT PRIMARY KEY,
    -> name VARCHAR(100) NOT NULL,
    -> date_of_birth DATE NOT NULL,
    -> phone_number VARCHAR(15) NOT NULL,
    -> email VARCHAR(100) NOT NULL,
    -> aadhar_number VARCHAR(12) NOT NULL,
    -> address VARCHAR(255) NOT NULL,
    -> account_type VARCHAR(50) NOT NULL,
    -> balance DECIMAL(15, 2) NOT NULL,
    -> );
Query OK, 0 rows affected (1.98 sec)

mysql>
mysql>
```

- Inserting values into table “customers”



```

MySQL>
mysql> INSERT INTO customers (account_number, name, date_of_birth, phone_number, email, aadhar_number, address, account_type, balance) VALUES
    -> (102345678, 'Riya Malhotra', '1989-06-27', '6543210987', 'riya.malhotra@gmail.com', '678901234567', '89 Nehru Place, Kolkata, West Bengal', 'Salary Account', 5000000.00),
    -> (123216589, 'Prakhar Tiwari', '1989-07-18', '9843667489', 'prakhar.tiwari@gmail.com', '983465728532', '44 Ring Road, Kolkata, West Bengal', 'Saving Account', 98000.00),
    -> (135790864, 'Pratham Oberoi', '1995-02-05', '9988776655', 'pratham.oberoi@gmail.com', '890123456789', '22 Green Park, Delhi, Delhi', 'Saving Account', 2000000.00),
    -> (142733221, 'Krupa Patel', '1990-02-12', '9944367553', 'krupa.patel@gmail.com', '225436785456', '44B, Ring Road, Navi Mumbai, Maharashtra', 'Salary Account', 992000.00),
    -> (213456789, 'Aadarsh Chawda', '1994-11-15', '7654321098', 'kabir.khanna@yahoo.com', '789012345678', '15 Jayanagar, Bengaluru, Karnataka', 'Savings Account', 18000000.00),
    -> (324567890, 'Aditi Mehta', '1988-06-22', '3210987654', 'aditi.mehta@gmail.com', '890123456789', '54 Rajouri Garden, New Delhi, Delhi', 'Saving Account', 15000000.00),
    -> (325647890, 'Aarushi Sharma', '1999-02-19', '9955342210', 'aarushi.sharma@gmail.com', '7655434445670', '85 Marine Drive, Mumbai, Maharashtra', 'Savings Account', 2300000.00),
    -> (346969801, 'Aanya Rawat', '2001-08-15', '9456675400', 'aanya.rawat@gmail.com', '998656993455', '87D-Mahatma Palace, Navi Mumbai, Maharashtra', 'Saving Account', 994000.00),
    -> (459876321, 'Ajay Bansal', '1990-01-20', '9812345678', 'ajay.bansal@gmail.com', '123456789012', '789 Sector 15, Faridabad, Haryana', 'Saving Account', 1500000.00),
    -> (612345678, 'Neha Sethi', '1987-05-15', '9876123450', 'neha.sethi@yahoo.com', '234567890123', '5 Palm Street, Bengaluru, Karnataka', 'Current Account', 450000.00),
    -> (664859349, 'Dhwanit Kumar', '1998-12-03', '9945367549', 'dhwanit.kumar@gmail.com', '223547685493', '88, SP Road, Kolkata, West Bengal', 'Current Account', 334678900.00),
    -> (734567890, 'Vikrant Joshi', '1993-08-10', '9109876543', 'vikrant.joshi@gmail.com', '345678901234', '20 MG Road, Mumbai, Maharashtra', 'Salary Account', 1200000.00),
    -> (845678901, 'Pooja Agarwal', '1985-12-30', '8654321098', 'pooja.agarwal@gmail.com', '456789012345', '40 Connaught Place, New Delhi, Delhi', 'Saving Account', 3200000.00),
    -> (945732489, 'Manpreet Kaur', '1999-03-11', '8549000543', 'manpreet.kaur@yahoo.com', '224573880912', '11, Chandni Chowk, New Delhi', 'Salary', 2300.00),
    -> (956789012, 'Sahil Kapoor', '1992-03-18', '7865432109', 'sahil.kapoor@yahoo.com', '567890123456', '100 Rajouri Garden, Delhi, Delhi', 'Current Account', 353000.00);
Query OK, 15 rows affected (0.51 sec)
Records: 15  Duplicates: 0  Warnings: 0
```

Python:

```
import tkinter as tk
from tkinter import messagebox
import mysql.connector
from PIL import Image, ImageTk
import bcrypt

# Employee credentials
employee_id = {
    "username": "emp_007",
    "password": bcrypt.hashpw(b"007", bcrypt.gensalt())
}

# Database Connection
def create_connection():
    """Establish a connection to the MySQL database."""
    try:
        connection = mysql.connector.connect(
            host='localhost',
            user='root',
            password='12345',
            database='bank_management'
        )
        return connection
    except mysql.connector.Error as e:
        messagebox.showerror("Database Error", f"Connection failed: {e}")
        return None

# Employee Login
def employee_login(username, password):
    """Handle employee login."""
    if username == employee_id["username"] and bcrypt.checkpw(password.encode('utf-8'), employee_id["password"]):
        messagebox.showinfo("Login Successful", f"Welcome to DT Bank, {username}!")
        return True
    else:
        messagebox.showerror("Login Failed", "Incorrect username or password. Please try again.")
        return False

# Customer Account Management Functions
#1.Create Customer
```

```

def create_customer(account_number, name, dob, phone, email, aadhar_number, address,
account_type, initial_balance):
    """Create a new customer account in the database."""
    try:
        initial_balance = float(initial_balance)
        if initial_balance < 0:
            raise ValueError("Initial balance must be a non-negative number.")

    except ValueError as e:
        messagebox.showerror("Input Error", str(e))
        return

    connection = create_connection()
    if connection is None:
        return

    cursor = connection.cursor()
    try:
        query = """
        INSERT INTO customers (account_number, name, date_of_birth, phone_number, email,
        aadhar_number, address, account_type, balance)
        VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s)
        """
        cursor.execute(query, (account_number, name, dob, phone, email, aadhar_number,
        address, account_type, initial_balance))
        connection.commit()
        messagebox.showinfo("Success", "Customer created successfully!")
    except mysql.connector.Error as err:
        messagebox.showerror("Error", f"Error: {err}")
    finally:
        cursor.close()
        connection.close()

#View Customer
def view_customer(account_number):
    """View customer details based on account number."""
    connection = create_connection()
    if connection is None:
        return

    cursor = connection.cursor()
    query = "SELECT * FROM customers WHERE account_number = %s"

    try:
        cursor.execute(query, (account_number,))
        customer = cursor.fetchone()

        if customer:
            details = (f"Account Number: {customer[0]}\n"
                      f"Name: {customer[1]}\n"
                      f"DOB: {customer[2]}")
    
```

```

        f"Phone: {customer[3]}\n"
        f"Email: {customer[4]}\n"
        f"Aadhar: {customer[5]}\n"
        f"Address: {customer[6]}\n"
        f"Account Type: {customer[7]}\n"
        f"Balance: ₹{customer[8]:,.2f}")
    messagebox.showinfo("Customer Details", details)
else:
    messagebox.showerror("Error", "Customer not found.")
except mysql.connector.Error as err:
    messagebox.showerror("Database Error", f"Error: {err}")
finally:
    cursor.close()
    connection.close()

#Update Customer
def update_customer(account_number, field, new_value):
    """Update customer information in the database."""
    connection = create_connection()
    if connection is None:
        return

    cursor = connection.cursor()

    if field not in ['name', 'phone_number', 'email', 'aadhar_number', 'address',
'account_type']:
        messagebox.showerror("Error", "Invalid field for update.")
        cursor.close()
        connection.close()
        return

    messagebox.showerror("Input Error", "Invalid email format.")
    cursor.close()
    connection.close()
    return

    query = f"UPDATE customers SET {field} = %s WHERE account_number = %s"
    cursor.execute(query, (new_value, account_number))
    connection.commit()

    if cursor.rowcount > 0:
        messagebox.showinfo("Success", f"Updated {field} successfully!")
    else:
        messagebox.showerror("Error", "Failed to update customer details.")

    cursor.close()
    connection.close()

#Delete Customer
def delete_customer(account_number):
    """Delete a customer from the database."""

```

```

connection = create_connection()
if connection is None:
    return

cursor = connection.cursor()
query = "DELETE FROM customers WHERE account_number = %s"
cursor.execute(query, (account_number,))
connection.commit()

if cursor.rowcount > 0:
    messagebox.showinfo("Success", "Customer deleted successfully!")
else:
    messagebox.showerror("Error", "Customer not found.")

cursor.close()
connection.close()

#Perform Transactions
def perform_transaction(account_number, amount, transaction_type):
    """Perform a deposit or withdrawal transaction."""
    connection = create_connection()
    if connection is None:
        return

    cursor = connection.cursor()

    try:
        amount = float(amount)
        if amount <= 0:
            messagebox.showerror("Input Error", "Amount must be a positive number.")
            return

        if transaction_type == 'Deposit':
            update_query = "UPDATE customers SET balance = balance + %s WHERE account_number = %s"
            cursor.execute(update_query, (amount, account_number))
        elif transaction_type == 'Withdrawal':
            cursor.execute("SELECT balance FROM customers WHERE account_number = %s",
                           (account_number,))
            balance = cursor.fetchone()
            if balance is None:
                messagebox.showerror("Error", "Account not found.")
                return
            if balance[0] < amount:
                messagebox.showerror("Error", "Insufficient funds.")
                return
            update_query = "UPDATE customers SET balance = balance - %s WHERE account_number = %s"
            cursor.execute(update_query, (amount, account_number))
        else:
            messagebox.showerror("Error", "Invalid transaction type.")

    except Exception as e:
        messagebox.showerror("Error", str(e))

```

```

        return

    connection.commit()
    messagebox.showinfo("Success", f"{transaction_type} of ₹{amount:.2f} completed
successfully!")
except ValueError:
    messagebox.showerror("Input Error", "Please enter a valid number for the amount.")
except mysql.connector.Error as err:
    messagebox.showerror("Error", f"Transaction failed: {err}")
finally:
    cursor.close()
    connection.close()

#Check Balance
def check_balance(account_number):
    """Check the balance of a customer account."""
    connection = create_connection()
    if connection is None:
        return

    cursor = connection.cursor()
    query = "SELECT name, balance FROM customers WHERE account_number = %s"
    cursor.execute(query, (account_number,))
    result = cursor.fetchone()

    if result:
        name, balance = result
        messagebox.showinfo("Balance Inquiry", f"Account Holder: {name}\nCurrent Balance:
₹{balance:.2f}")
    else:
        messagebox.showerror("Error", "Account not found.")

    cursor.close()
    connection.close()

# GUI Functions

#1.Window for Create Customer Account

def create_customer_account():
    create_window = tk.Toplevel()
    create_window.title("Create Customer Account")
    create_window.geometry("500x700")
    create_window.configure(bg="#E0F2F1")

    tk.Label(create_window, text="Create Customer Account", font=("Arial", 18, "bold"),
bg="#E0F2F1").pack(pady=10)

    labels = [
        "Account Number:", "Name:", "Date of Birth (YYYY-MM-DD):",
        "Phone Number:", "Email:", "Aadhar Number:",

```

```

        "Address:", "Account Type:", "Initial Balance:"
    ]
entries = []

for label in labels:
    tk.Label(create_window, text=label, bg="#E0F2F1").pack(pady=5)
    entry = tk.Entry(create_window)
    entry.pack(pady=5, padx=20, fill=tk.X)
    entries.append(entry)

tk.Button(create_window, text="Submit", bg="#87CEFA", command=lambda: create_customer(
    entries[0].get(), entries[1].get(), entries[2].get(),
    entries[3].get(), entries[4].get(), entries[5].get(),
    entries[6].get(), entries[7].get(), entries[8].get()
)).pack(pady=20)

#2.Window for View Customer Account
def view_customer_details():
    view_window = tk.Toplevel()
    view_window.title("View Customer Details")
    view_window.geometry("500x300")
    view_window.configure(bg="#E0F2F1")

    tk.Label(view_window, text="View Customer Details", font=("Arial", 18, "bold"),
    bg="#E0F2F1").pack(pady=10)

    tk.Label(view_window, text="Account Number:", bg="#E0F2F1").pack(pady=5)
    acc_num_entry = tk.Entry(view_window)
    acc_num_entry.pack(pady=5, padx=20, fill=tk.X)

    tk.Button(view_window, text="View Details", bg="#87CEFA", command=lambda:
    view_customer(acc_num_entry.get())).pack(pady=10)

#3.Window for Displaying Customer Details
def update_customer_details():
    update_window = tk.Toplevel()
    update_window.title("Update Customer Details")
    update_window.geometry("500x300")
    update_window.configure(bg="#E0F2F1")

    tk.Label(update_window, text="Update Customer Details", font=("Arial", 18, "bold"),
    bg="#E0F2F1").pack(pady=10)

    tk.Label(update_window, text="Account Number:", bg="#E0F2F1").pack(pady=5)
    acc_num_entry = tk.Entry(update_window)
    acc_num_entry.pack(pady=5, padx=20, fill=tk.X)

    tk.Label(update_window, text="Field to Change (e.g., name, phone_number):",
    bg="#E0F2F1").pack(pady=5)
    field_entry = tk.Entry(update_window)

```

```

field_entry.pack(pady=5, padx=20, fill=tk.X)

tk.Label(update_window, text="New Value:", bg="#E0F2F1").pack(pady=5)
new_value_entry = tk.Entry(update_window)
new_value_entry.pack(pady=5, padx=20, fill=tk.X)

tk.Button(update_window, text="Update", bg="#87CEFA", command=lambda:
update_customer(acc_num_entry.get(), field_entry.get(),
new_value_entry.get())).pack(pady=20)

#4.Window for Transaction
def perform_transaction_window():
    transaction_window = tk.Toplevel()
    transaction_window.title("Perform Transaction")
    transaction_window.geometry("500x400")
    transaction_window.configure(bg="#E0F2F1")

    tk.Label(transaction_window, text="Perform Transaction", font=("Arial", 18, "bold"),
bg="#E0F2F1").pack(pady=10)

    tk.Label(transaction_window, text="Account Number:", bg="#E0F2F1").pack(pady=5)
    acc_num_entry = tk.Entry(transaction_window)
    acc_num_entry.pack(pady=5, padx=20, fill=tk.X)

    tk.Label(transaction_window, text="Amount:", bg="#E0F2F1").pack(pady=5)
    amount_entry = tk.Entry(transaction_window)
    amount_entry.pack(pady=5, padx=20, fill=tk.X)

    tk.Label(transaction_window, text="Transaction Type:", bg="#E0F2F1").pack(pady=5)
    transaction_type_var = tk.StringVar(value='Deposit')
    tk.Radiobutton(transaction_window, text="Deposit", variable=transaction_type_var,
value='Deposit', bg="#E0F2F1").pack(pady=5)
    tk.Radiobutton(transaction_window, text="Withdrawal", variable=transaction_type_var,
value='Withdrawal', bg="#E0F2F1").pack(pady=5)

    tk.Button(transaction_window, text="Submit", bg="#87CEFA", command=lambda:
perform_transaction(acc_num_entry.get(), amount_entry.get(),
transaction_type_var.get())).pack(pady=20)

#5.Window for Viewing Balance of Customer
def check_balance_window():
    balance_window = tk.Toplevel()
    balance_window.title("Check Balance")
    balance_window.geometry("500x300")
    balance_window.configure(bg="#E0F2F1")

    tk.Label(balance_window, text="Check Balance", font=("Arial", 18, "bold"),
bg="#E0F2F1").pack(pady=10)

    tk.Label(balance_window, text="Account Number:", bg="#E0F2F1").pack(pady=5)
    acc_num_entry = tk.Entry(balance_window)

```

```

acc_num_entry.pack(pady=5, padx=20, fill=tk.X)

tk.Button(balance_window, text="Check Balance", bg="#87CEFA", command=lambda:
check_balance(acc_num_entry.get())).pack(pady=20)

#6.Window for Deleting Account
def delete_customer_account():
    delete_window = tk.Toplevel()
    delete_window.title("Delete Customer Account")
    delete_window.geometry("500x300")
    delete_window.configure(bg="#E0F2F1")

    tk.Label(delete_window, text="Delete Customer Account", font=("Arial", 18, "bold"),
bg="#E0F2F1").pack(pady=10)

    tk.Label(delete_window, text="Account Number:", bg="#E0F2F1").pack(pady=5)
    acc_num_entry = tk.Entry(delete_window)
    acc_num_entry.pack(pady=5, padx=20, fill=tk.X)

    tk.Button(delete_window, text="Delete Account", bg="#87CEFA", command=lambda:
delete_customer(acc_num_entry.get())).pack(pady=20)

#Showing Menu
def show_menu():
    menu_window = tk.Tk()
    menu_window.title("Menu - Bank Management System")
    menu_window.geometry("850x650")
    menu_window.configure(bg="#FFFFFF")

    logo_path = r"D:\Pratham\python programs\12 proj\logo.png"

    try:
        logo = Image.open(logo_path)
        logo = logo.resize((200, 100), Image.Resampling.LANCZOS)
        logo = ImageTk.PhotoImage(logo)

        header_frame = tk.Frame(menu_window, bg="#00796B", padx=10, pady=10)
        header_frame.pack(fill=tk.X)

        logo_label = tk.Label(header_frame, image=logo, bg="#00796B")
        logo_label.pack(pady=20)

    except Exception as e:
        print(f"Error loading logo: {e}")
        logo = None

    menu_frame = tk.Frame(menu_window, bg="#E0F2F1", padx=10, pady=10)
    menu_frame.pack(expand=True, fill=tk.BOTH)

    tk.Label(menu_frame, text="Select an option to proceed:", font=("Arial", 18, "bold"),
bg="#E0F2F1").pack(pady=10)

```

```

functions = {
    "Create New Account": create_customer_account,
    "View Account Details": view_customer_details,
    "Update Account Details": update_customer_details,
    "Perform Transaction": perform_transaction_window,
    "Check Account Balance": check_balance_window,
    "Delete Account": delete_customer_account,
    "Exit": menu_window.destroy
}

for func_name, func_command in functions.items():
    btn_color = "#B2DFDB" if func_name == "Create New Account" else (
        "#C8E6C9" if func_name == "View Account Details" else (
        "#FFE082" if func_name == "Update Account Details" else (
        "#80DEEA" if func_name == "Perform Transaction" else (
        "#FFCCBC" if func_name == "Check Account Balance" else (
        "#F44336" if func_name == "Delete Account" else "#FFAB91")))))
    btn = tk.Button(menu_frame, text=func_name, font=("Arial", 14), bg=btn_color,
fg="black", command=func_command)
    btn.pack(pady=5, padx=20, fill=tk.X)

menu_window.mainloop()

#For Login Button
def handle_login(username_entry, password_entry, login_window):
    """Handle the login process."""
    username = username_entry.get()
    password = password_entry.get()

    if employee_login(username, password):
        login_window.destroy()
        show_menu()

# Define the login window function
def create_login_window():
    """Display the login screen for the bank management system."""
    login_window = tk.Tk()
    login_window.title("Employee Login - DT Bank")

    window_width = 800
    window_height = 600
    login_window.geometry(f"{window_width}x{window_height}")
    login_window.configure(bg="#FFFFFF")

    logo_path = r"D:\Pratham\python programs\12 proj\logo.png"

    try:
        logo = Image.open(logo_path)

```

```

        logo = logo.resize((200, 100), Image.Resampling.LANCZOS) # Resize the logo as
needed
        logo = ImageTk.PhotoImage(logo)

header_frame = tk.Frame(login_window, bg="#00796B", padx=10, pady=10)
header_frame.pack(fill=tk.X)

logo_label = tk.Label(header_frame, image=logo, bg="#00796B")
logo_label.pack(pady=10)

except Exception as e:
    print(f"Error loading logo: {e}")
    logo = None

login_frame = tk.Frame(login_window, bg="#E0F2F1", padx=10, pady=10)
login_frame.pack(expand=True, fill=tk.BOTH)

tk.Label(login_frame, text="Employee Login", font=("Arial", 18, "bold"),
bg="#E0F2F1").pack(pady=20)

tk.Label(login_frame, text="Username:", font=("Arial", 12), bg="#E0F2F1").pack(pady=5)
username_entry = tk.Entry(login_frame, font=("Arial", 12))
username_entry.pack(pady=5)

tk.Label(login_frame, text="Password:", font=("Arial", 12), bg="#E0F2F1").pack(pady=5)
password_entry = tk.Entry(login_frame, font=("Arial", 12), show="*")
password_entry.pack(pady=5)

login_button = tk.Button(login_frame, text="Login", font=("Arial", 12, "bold"),
bg="#00796B", fg="white", width=20, command=lambda: handle_login(username_entry,
password_entry, login_window))
login_button.pack(pady=20)

login_window.mainloop()

if __name__ == "__main__":
    create_login_window()

```

Output

MySQL Tables

1. Describing Table “customers”:

```
mysql> desc customers;
+-----+-----+-----+-----+-----+-----+
| Field | Type   | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| account_number | bigint | NO  | PRI | NULL    |       |
| name           | varchar(100) | NO |     | NULL    |       |
| date_of_birth  | date    | NO  |     | NULL    |       |
| phone_number   | varchar(15)  | NO |     | NULL    |       |
| email          | varchar(100) | NO |     | NULL    |       |
| aadhar_number  | varchar(12)  | NO |     | NULL    |       |
| address         | varchar(255) | NO |     | NULL    |       |
| account_type   | varchar(50)  | NO |     | NULL    |       |
| balance         | decimal(15,2) | NO |     | NULL    |       |
+-----+-----+-----+-----+-----+-----+
9 rows in set (0.54 sec)
```

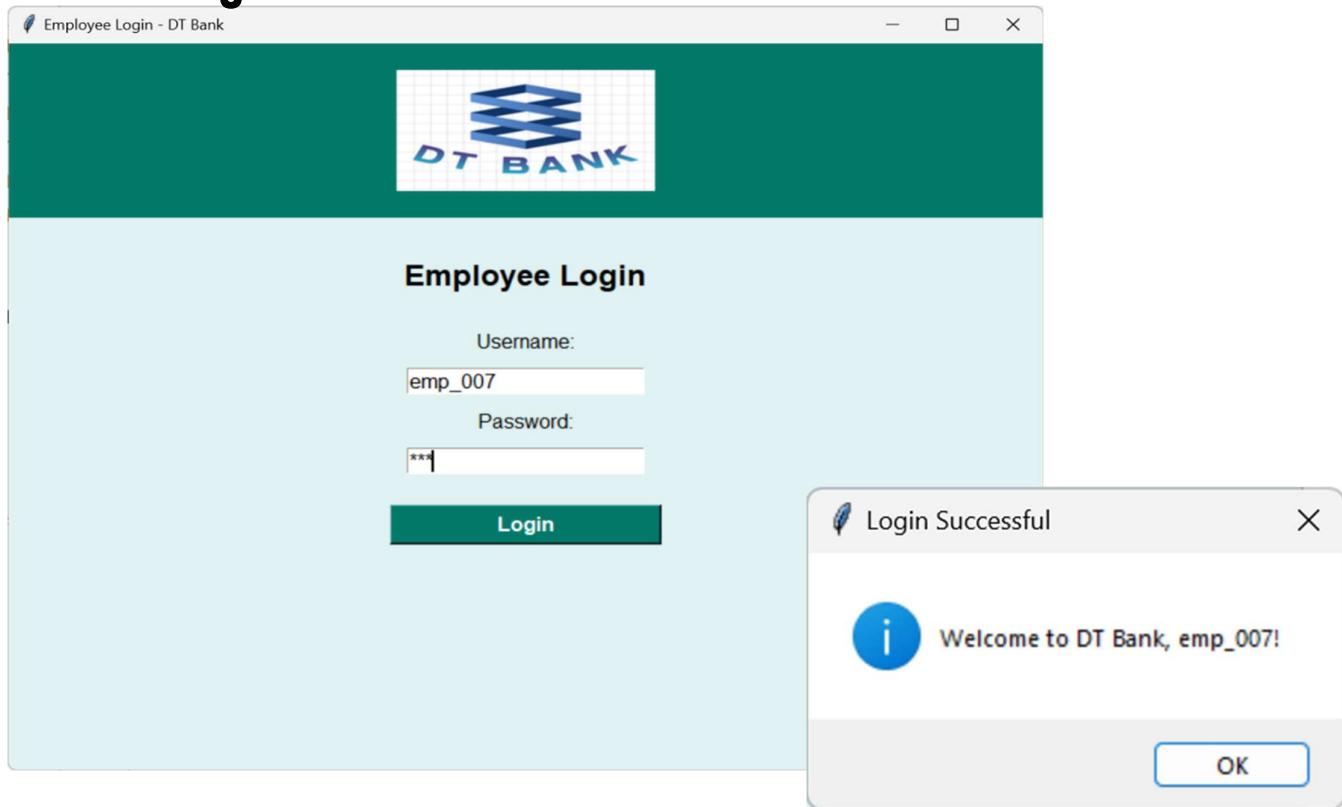
2. Table “customers”:

account_number	name	date_of_birth	email	aadhar_number	address	account_type	balance
102345678	Riya Malhotra	1989-06-27	riya.malhotra@gmail.com	678901234567	89 Nehru Place, Kolkata, West Bengal	Salary	5000000.00
123216589	Prakhar Tiwari	1989-07-18	prakhar.tiwari@gmail.com	983465728532	44 Ring Road, Kolkata, West Bengal	Saving	98000.00
135790864	Pratham Oberoi	1995-02-05	pratham.oberoi@gmail.com	890123456789	22 Green Park, Delhi, Delhi	Saving	2000000.00
142733221	Krupa Patel	1990-02-12	krupa.patel@gmail.com	225436785456	44B, Ring Road, Navi Mumbai, Maharashtra	Salary	992000.00

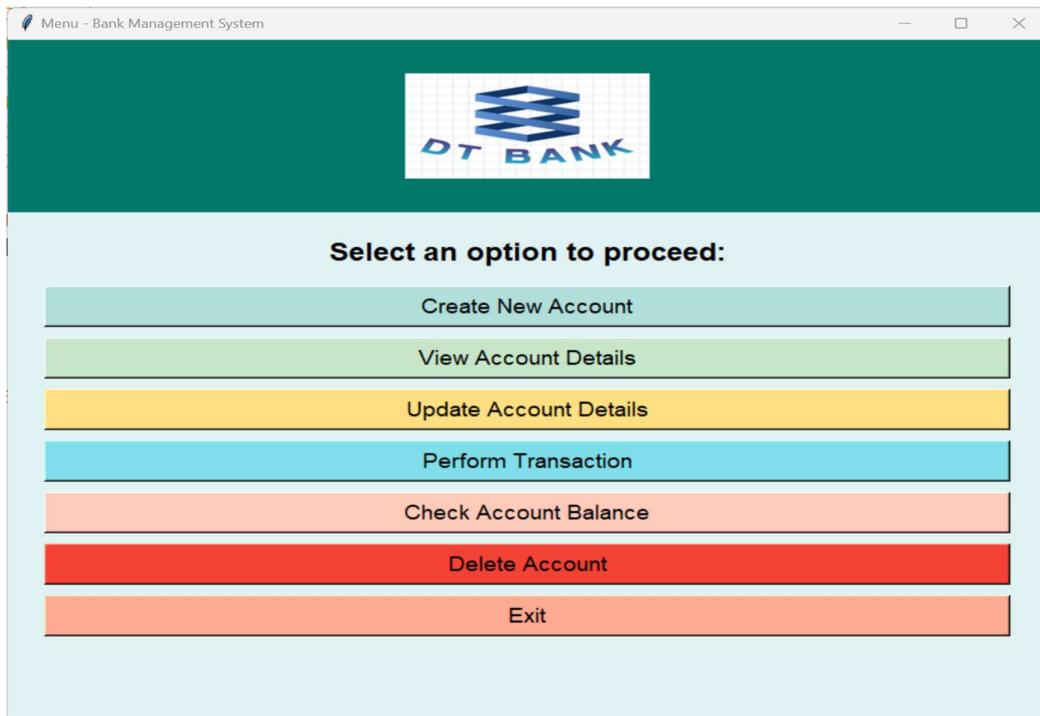
213456789	Aadarsh Chawda	1994-11-15	aadarsh.chawda@yahoo.com	789012345678	15 Jayanagar, Bengaluru, Karnataka	Saving	10000000.00
324567890	Aditi Mehta	1988-06-22	aditi.mehta@gmail.com	890123456789	54 Rajouri Garden, New Delhi, Delhi	Saving	150000000.00
325647890	Aarushi Sharma	1999-02-19	aarushi.sharma@gmail.com	765543445670	85 Marine Drive, Mumbai, Maharashtra	Savings	2300000.00
459876321	Ajay Bansal	1990-01-20	ajay.bansal@gmail.com	123456789012	789 Sector 15, Faridabad, Haryana	Saving	25000.00
612345678	Neha Sethi	1987-05-15	neha.sethi@yahoo.com	234567890123	5 Palm Street, Bengaluru, Karnataka	Current	450000.00
664859349	Dhwanit Kumar	1998-12-03	dhwanit.kumar@gmail.com	223547685493	88, SP Road, Kolkata, West Bengal	Current	3346789.00
734567890	Vikrant Joshi	1993-08-10	vikrant.joshi@gmail.com	345678901234	20 MG Road, Mumbai, Maharashtra	Salary	1200000.00
845678901	Pooja Agarwal	1985-12-30	pooja.agarwal@gmail.com	456789012345	40 Connaught Place, New Delhi, Delhi	Saving	320000.00
945732489	Manpreet Kaur	1999-03-11	manpreet.kaur@yahoo.com	224573880912	11, Chandni Chowk, New Delhi	Salary	2300.00
956789012	Sahil Kapoor	1992-03-18	sahil.kapoor@yahoo.com	567890123456	100 Rajouri Garden, Delhi, Delhi	Current	353000.00

- Performing Code

1. Login



2. Menu



3.Create New Account

The screenshot shows the 'Create Customer Account' window. It contains fields for Account Number (123456789), Name (Aadarsh Chauhan), Date of Birth (1999-03-12), Phone Number (9967004356), Email (aadarsh.chauhan@yahoo.com), Aadhar Number (327658490032), Address (44B, Ring Road, Navi Mumbai, Maharashtra), Account Type (Salary), and Initial Balance (15000). A 'Submit' button is at the bottom. To the right, a 'Success' dialog box displays the message 'Customer created successfully!' with an 'OK' button.

Create Customer Account

Account Number:

123456789

Name:

Aadarsh Chauhan

Date of Birth (YYYY-MM-DD):

1999-03-12

Phone Number:

9967004356

Email:

aadarsh.chauhan@yahoo.com

Aadhar Number:

327658490032

Address:

44B, Ring Road, Navi Mumbai, Maharashtra

Account Type:

Salary

Initial Balance:

15000

Submit

Success

Customer created successfully!

OK

4.View Customer Details

The screenshot shows the 'View Customer Details' window with an 'Account Number' field containing '123456789'. Below it is a 'View Details' button. To the right, a 'Customer Details' dialog box is open, displaying the following information: Account Number: 123456789, Name: Aadarsh Chauhan, DOB: 1999-03-12, Phone: 9967004356, Email: aadarsh.chauhan@yahoo.com, Aadhar: 327658490032, Address: 44B, Ring Road, Navi Mumbai, Maharashtra, Account Type: Salary, and Balance: ₹15,000.00. An 'OK' button is at the bottom of the dialog.

View Customer Details

Account Number:

123456789

View Details

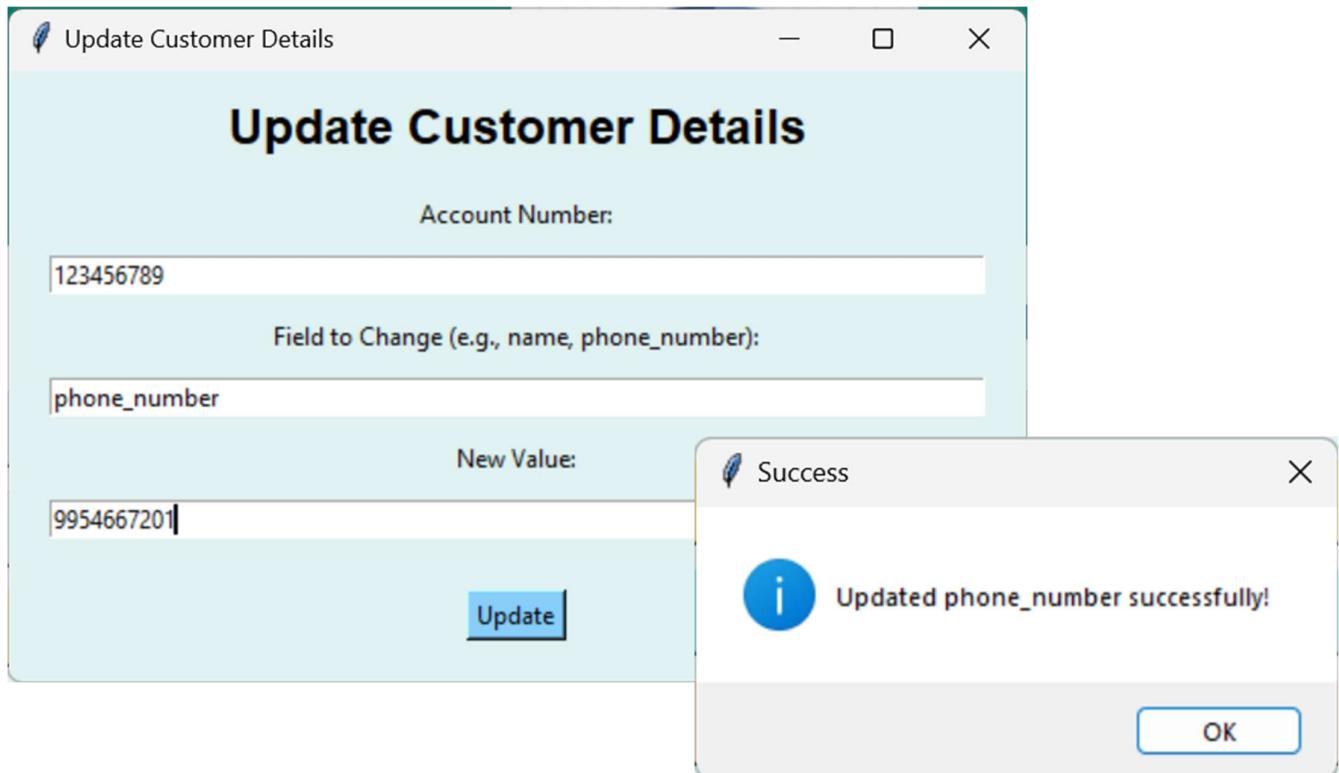
Customer Details

i

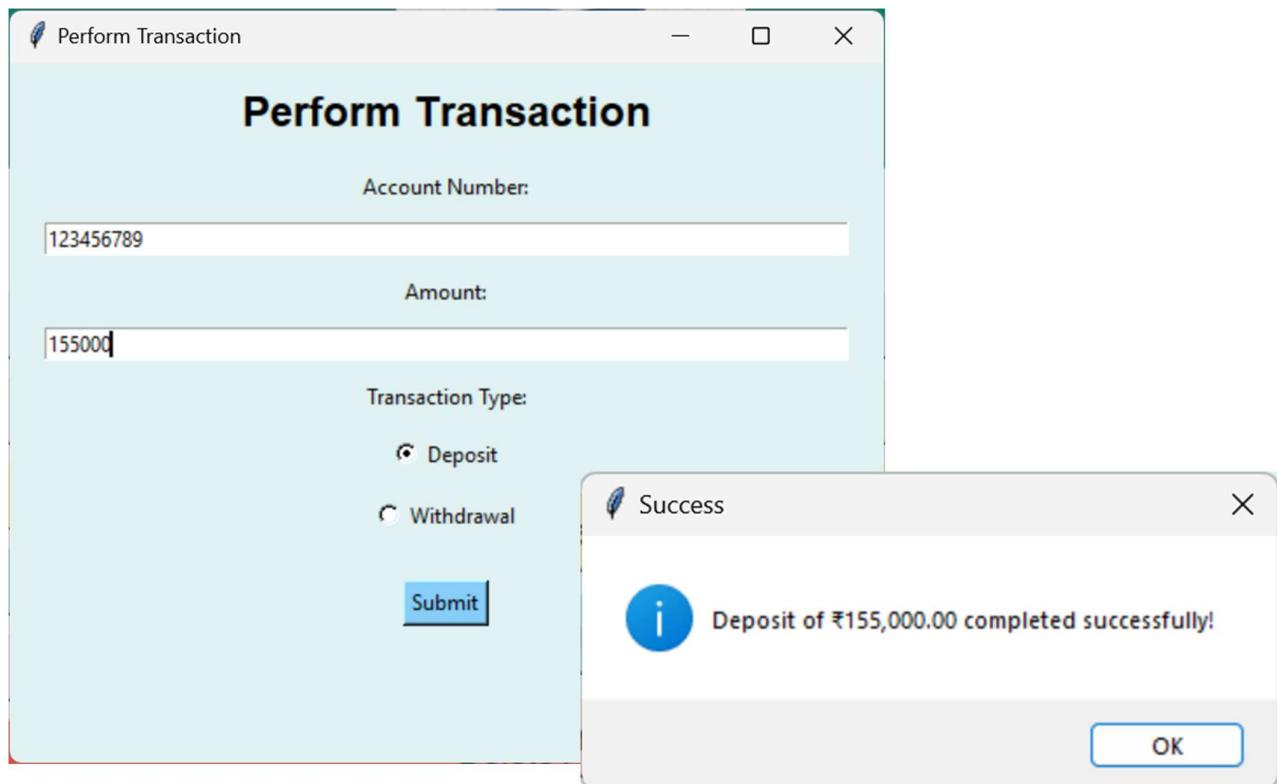
Account Number: 123456789
Name: Aadarsh Chauhan
DOB: 1999-03-12
Phone: 9967004356
Email: aadarsh.chauhan@yahoo.com
Aadhar: 327658490032
Address: 44B, Ring Road, Navi Mumbai, Maharashtra
Account Type: Salary
Balance: ₹15,000.00

OK

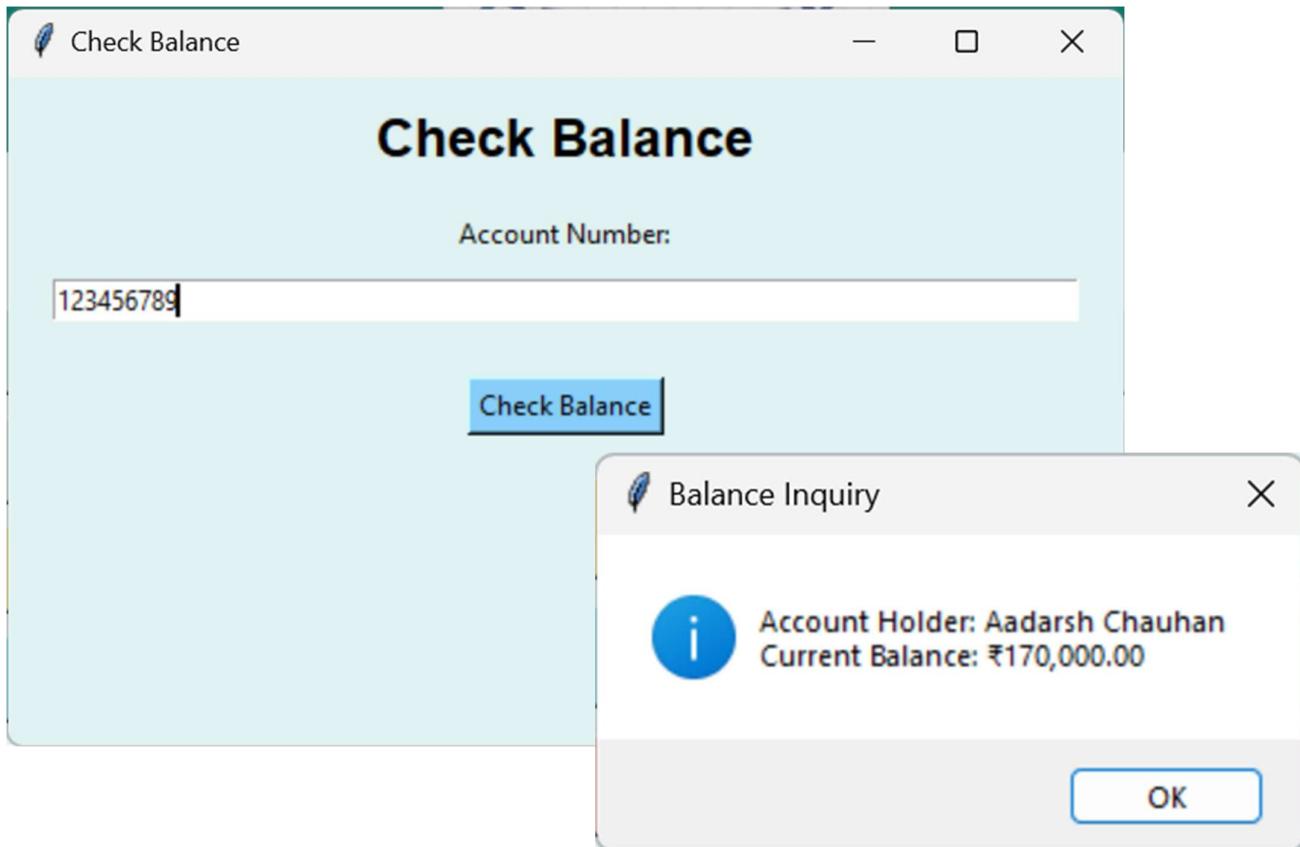
5. Update Customer Details



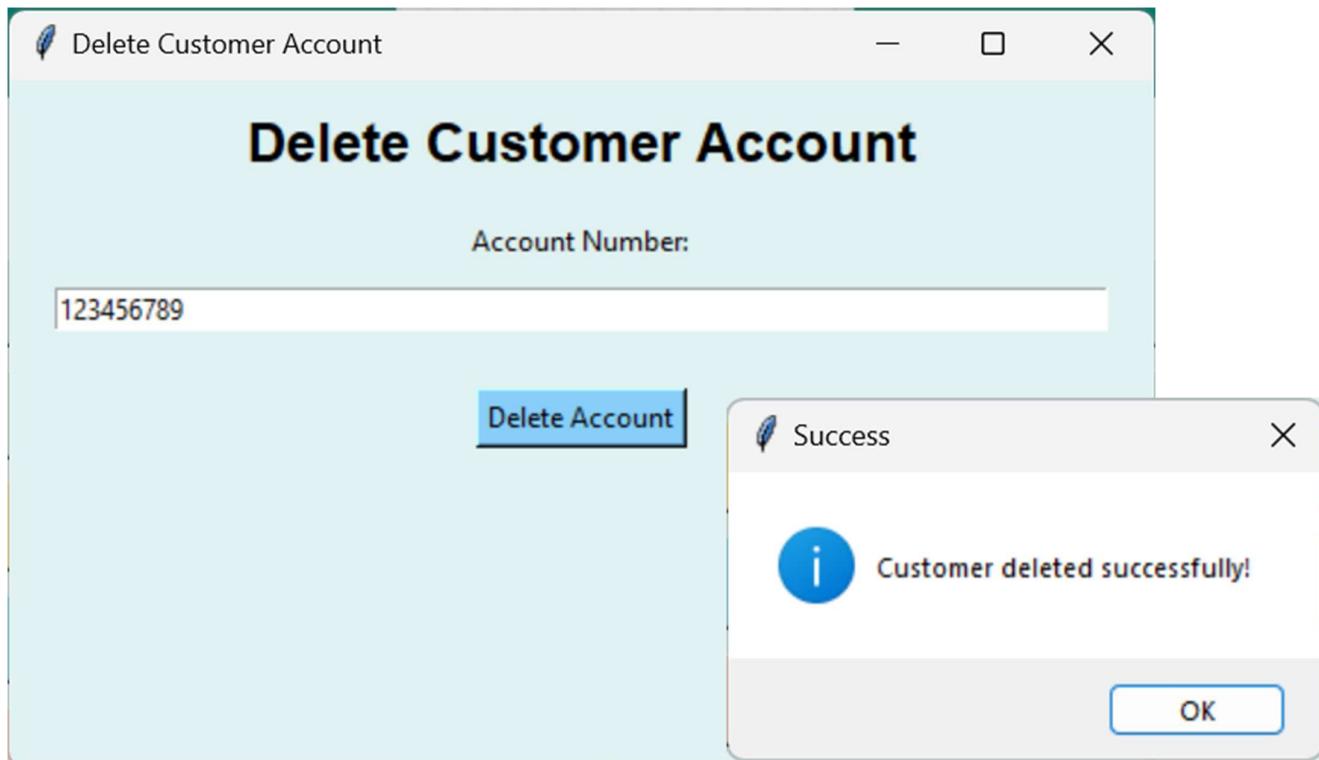
6. Perform Transaction



7. Check Balance



8. Delete Account



Contribution to Real World

The Bank Management System designed for this project offers several valuable contributions to the real world, particularly in terms of efficiency, security, and customer service within the banking sector. Below are some key aspects of how this project contributes to the real-world banking environment:

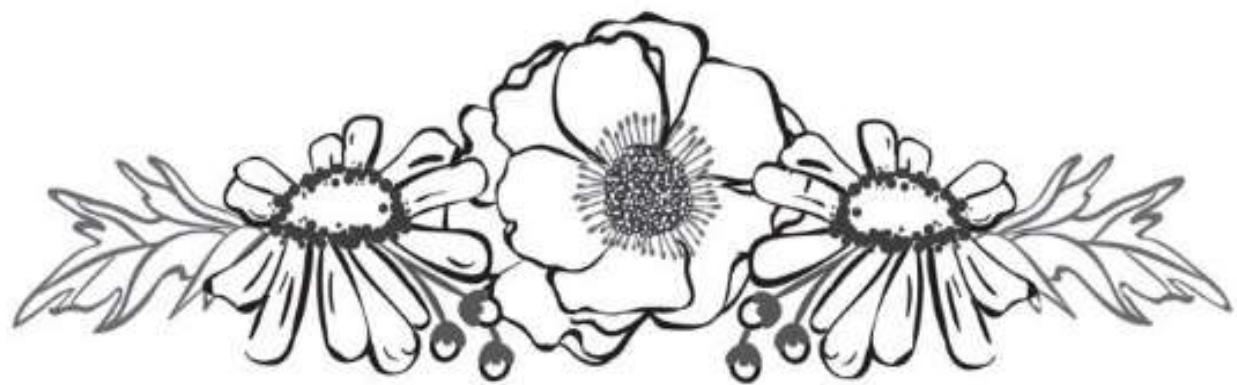
1. **Improved Operational Efficiency:** Automating account management tasks like adding, modifying, or deleting customer accounts reduces manual labour, speeds up operations, and minimizes human errors. This allows employees to focus on more important tasks, such as customer interaction, instead of administrative work.
2. **Enhanced Security:** The employee login feature with proper authentication ensures only authorized individuals can access sensitive data and perform key functions. This enhances security, protecting both the bank's and customers' data from unauthorized access and cyber threats, which is crucial in today's digital environment.
3. **Real-time Data Access:** The system provides real-time access to customer balances and transaction histories, improving transparency and decision-making for employees. It also allows customers to receive immediate updates, boosting trust in the bank's services. This is crucial in fast-paced environments requiring quick, accurate information.
4. **Better Customer Service:** The system's ability to track and update customer accounts in real-time helps employees serve customers more effectively. The efficiency of balance checks, transaction updates, and account creation ensures that customers receive timely and accurate service, leading to higher customer satisfaction. The ability to address customer queries swiftly contributes to building stronger customer relationships.
5. **Scalability:** The system is designed to scale with the bank's growth, efficiently handling an increasing volume of transactions and customer data. Its database structure and backend logic ensure continued performance, maintaining efficiency as the bank expands.

6. **Cost Savings:** The automation of key banking processes and the elimination of manual paperwork reduce operational costs for banks. Employees spend less time on repetitive tasks, and the system's streamlined processes minimize errors that could lead to costly mistakes. Over time, these efficiencies translate into significant cost savings for the bank.
7. **Accuracy in Financial Transactions:** The system ensures that all financial transactions are processed correctly, with automatic updates to customer balances. This reduces the risk of discrepancies or errors that can arise in manual systems, ensuring that financial records are always up-to-date and accurate.
8. **Better Compliance and Record Keeping:** With the system's ability to track every transaction and maintain a comprehensive record of customer accounts, it becomes easier for the bank to comply with regulatory requirements. This is essential in today's financial landscape, where transparency and accountability are critical for maintaining trust and meeting legal obligations.

In conclusion, the Bank Management System contributes significantly to the real-world banking industry by improving operational efficiency, security, and customer service. Its ability to scale with the bank's growth, enhance data accuracy, and streamline transactions makes it a valuable tool for modern banking institutions looking to stay competitive in a rapidly evolving digital landscape.

Bibliography

- **MySQL Documentation.** (n.d.). *MySQL Reference Manual*. Retrieved from <https://dev.mysql.com/doc/>
 - This manual provides a detailed overview of MySQL, a popular relational database management system, and covers its syntax, features, and functionality used for managing databases in the bank management system.
- **Python Software Foundation.** (n.d.). *Python Programming Language*. Retrieved from <https://www.python.org/>
 - The official website of Python, a versatile programming language used in the development of the bank management system for backend logic and database interaction.
- **Tkinter Documentation.** (n.d.). *Tkinter 8.5 reference: a GUI for Python*. Retrieved from <https://docs.python.org/3/library/tkinter.html>
 - The official Python documentation for Tkinter, the library used for building the graphical user interface (GUI) for the banking system.
- **IBM.** (2021). *Cyber Security in Banking: The Importance of Protecting Sensitive Data*. Retrieved from <https://www.ibm.com/security>
 - This article discusses the significance of cybersecurity measures in the banking industry, focusing on how security tools and encryption protocols ensure the protection of customer data in modern banking systems.
- **Google.** (2020). *Best Practices for Building Scalable Systems*. Retrieved from <https://cloud.google.com/architecture/best-practices>
 - This guide by Google outlines best practices for building scalable systems, which were implemented to ensure that the bank management system could handle increased customer and transaction volumes as the bank grows.
- **Oracle.** (2020). *Database Security Best Practices*. Retrieved from <https://www.oracle.com/database/security/>
 - Oracle's guide to best practices for database security, providing insights into how data protection protocols and secure database management practices were applied in the development of the banking system.



THANK YOU

