SRINATH PUBLIC SCHOOL

COMPUTER SCIENCE (083) PROJECT
SESSION 2025–26

TOPIC:

BANK MANAGEMENT SYSTEM

Submitted by:
1. Name: Aakash Deep
Class: XII – PCM
Roll No.: 01

2. Name: Antariksh Bera
Class: XII – PCM
Roll No.: 06

3. Name: Jay Deep Mahato
Class: XII – PCM
Roll No.: 13

Under the guidance of:
Mr. Ashwajeet Singh

ACKNOWLEDGEMENT

I would like to express my sincere gratitude to the Principal and the Management of Srinath Public School for providing an excellent learning environment and the facilities required to complete this project on "Bank Management System".

I am extremely thankful to my Computer Science teacher, Mr. Ashwajeet Singh, for his valuable guidance, motivation and continuous support throughout the development of this project. Their suggestions helped me to understand the concepts of Python programming and MySQL database connectivity in a better way.

I am also thankful to my parents and friends for their encouragement and help during the preparation of this project.

Lastly, I thank the Almighty for giving me the strength and determination to complete this work successfully.

CERTIFICATE

This is to certify that **Aakash Deep, Antariksh Bera, Jay Deep Mahato** of Class XII – PCM have satisfactorily completed the Computer Science (083) project titled **“BANK MANAGEMENT SYSTEM”** during the academic session 2025-26 under my guidance and supervision. This project is in accordance with the guidelines issued by the Central Board of Secondary Education (CBSE) for AISSCE Practical Examination.

To the best of my knowledge, this work is the original effort of the students.

Signature of Subject Teacher

Signature of External Examiner

Signature of Principal

TABLE OF CONTENTS

INTRODUCTION

The Bank Management System is a menu-driven software application developed using Python and MySQL. It is designed to perform basic banking operations such as creating new accounts, depositing and withdrawing money, checking account balance, viewing all existing accounts and deleting accounts when required.

In the modern world, banks play a very important role in managing the financial activities of individuals and organisations. Manual handling of records often leads to problems like data redundancy, calculation errors and difficulty in searching customer details. By using computers and database management systems, banking operations become faster, more accurate and more secure.

This project demonstrates how Python can be connected to a MySQL database using the mysql-connector module to build a simple but practical banking application.

HARDWARE AND SOFTWARE REQUIREMENTS

- Hardware Requirements:
  - • Desktop / Laptop computer
  - • Processor: Intel/AMD compatible
  - • Minimum 4 GB RAM
  - • 20 GB free Hard Disk space
- Software Requirements:
  - • Operating System: Windows 10 or above
  - • Python 3.x (IDLE / VS Code as IDE)
  - • MySQL Server (via XAMPP)
  - • MySQL Connector for Python
  - • XAMPP Control Panel (Apache and MySQL)

PROBLEM SCOPING

Traditional banking systems often depend on manual records maintained in registers or simple spreadsheets. This leads to several problems such as:

1. Time-consuming process to search and update customer details.
2. Higher chances of calculation errors in deposits and withdrawals.
3. Difficulty in maintaining up-to-date account balances.
4. Lack of data security and risk of data loss.

The Bank Management System developed in this project aims to solve these problems by:

- • Storing all customer and account details in a MySQL database.
- • Automatically updating balances after deposit and withdrawal operations.
- • Allowing quick retrieval of account details through search queries.
- • Providing a structured, menu-driven interface for the user.

ABOUT THE TOPIC

A bank is a financial institution that accepts deposits and provides loans and other financial services to customers. To manage thousands of customers efficiently, banks rely heavily on computerised systems. A Bank Management System helps in organising customer data, tracking account balances and generating reports whenever required.

Using a programming language like Python along with a relational database such as MySQL, we can build applications that are both user-friendly and powerful. Python provides simplicity and readability, while MySQL ensures secure and consistent storage of data. This project is a small-scale implementation of how such a system can work in real life.

SYSTEM DESIGN & DEVELOPMENT

The system is designed using a modular approach. The major components are:

- • Database Design: Two tables – CUSTOMER and ACCOUNT – are created in the MySQL database bank_db.
- • Python Functions: Each banking operation is implemented as a separate function.
- • Menu-Driven Interface: A loop displays the main menu and takes user input to perform the required task.
- • Error Handling: try–except blocks are used to handle runtime errors during database connectivity and user input.

Overall, the project shows how a simple but effective banking application can be built using Python and MySQL.

PYTHON CODING (SOURCE CODE)

The following Python program implements the Bank Management System with options to create account, deposit, withdraw, check balance, view all accounts and delete an account.

```
import mysql.connector as msc


# ---------- DATABASE CONNECTION ----------


def get_connection():

    con = msc.connect(

        host="localhost",

        user="root",

        password="",

        database="bank_db"

    )

    return con




# ---------- FUNCTIONS FOR BANK OPERATIONS ----------


def create_account():
    print("\n--- Create New Account ---")
    try:
name = input("Enter customer name: ")
        address = input("Enter address: ")
        phone = input("Enter phone: ")
        acc_type = input("Enter account type (Saving/Current): ")
        opening_balance = float(input("Enter opening balance: "))

        con = get_connection()
        cur = con.cursor()

        cur.execute(
            "INSERT INTO customer (name, address, phone) VALUES (%s,
%s, %s)",
            (name, address, phone)
        )
        cust_id = cur.lastrowid

        cur.execute(
            "INSERT INTO account (cust_id, acc_type, balance) VALUES (%s,
%s, %s)",
            (cust_id, acc_type, opening_balance)
        )
        acc_no = cur.lastrowid

        con.commit()

        print("\nAccount created successfully!")
        print("Customer ID :", cust_id)
        print("Account No. :", acc_no)

    except Exception as e:
        print("Error while creating account:", e)
    finally:
        try:
            cur.close()
            con.close()
        except:
            pass


def deposit():
    print("\n--- Deposit Amount ---")
    try:
        acc_no = int(input("Enter account number: "))
        amount = float(input("Enter amount to deposit: "))

        con = get_connection()
        cur = con.cursor()

        cur.execute("SELECT balance FROM account WHERE acc_no = %s",
(acc_no,))
        row = cur.fetchone()

        if row is None:
            print("Account not found.")
            return

        current_balance = float(row[0])
        new_balance = current_balance + amount

        cur.execute(
            "UPDATE account SET balance = %s WHERE acc_no = %s",
            (new_balance, acc_no)
        )
        con.commit()

        print("\nDeposit successful.")
```

```
        print("Previous balance:", current_balance)
        print("Deposited       :", amount)
        print("New balance     :", new_balance)

    except Exception as e:
        print("Error while depositing:", e)
    finally:
        try:
            cur.close()
            con.close()
        except:
            pass


def withdraw():
    print("\n--- Withdraw Amount ---")
    try:
        acc_no = int(input("Enter account number: "))
        amount = float(input("Enter amount to withdraw: "))

        con = get_connection()
        cur = con.cursor()
        cur.execute("SELECT balance FROM account WHERE acc_no = %s",
(acc_no,))
        row = cur.fetchone()

        if row is None:
            print("Account not found.")
            return

        current_balance = float(row[0])

        if amount > current_balance:
            print("Insufficient balance.")
            return

        new_balance = current_balance - amount

        cur.execute(
            "UPDATE account SET balance = %s WHERE acc_no = %s",
            (new_balance, acc_no)
        )
        con.commit()
```

```
        print("\nWithdrawal successful.")
        print("Previous balance:", current_balance)
        print("Withdrawn       :", amount)
        print("New balance     :", new_balance)

    except Exception as e:
        print("Error while withdrawing:", e)
    finally:
        try:
            cur.close()
            con.close()
        except:
            pass


def check_balance():
    print("\n--- Check Balance ---")
    try:
        acc_no = int(input("Enter account number: "))

        con = get_connection()
        cur = con.cursor()
        cur.execute("""
            SELECT account.acc_no, customer.name, account.acc_type,
account.balance
            FROM account
            JOIN customer ON account.cust_id = customer.cust_id
            WHERE account.acc_no = %s
        """, (acc_no,))

        row = cur.fetchone()

        if row is None:
            print("Account not found.")
        else:
            print("\nAccount Details:")
            print("Account No.   :", row[0])
            print("Customer Name:", row[1])
            print("Account Type :", row[2])
            print("Balance       :", row[3])

    except Exception as e:
        print("Error while checking balance:", e)
    finally:
        try:
            cur.close()
            con.close()
        except:
            pass



def view_all_accounts():
    print("\n--- View All Accounts ---")
    try:
        con = get_connection()
        cur = con.cursor()

        cur.execute("""
            SELECT account.acc_no, customer.name, account.acc_type,
account.balance
            FROM account
            JOIN customer ON account.cust_id = customer.cust_id
            ORDER BY account.acc_no
        """)
        rows = cur.fetchall()

        if not rows:
            print("No accounts found.")
            return

        print("\nAcc_No | Customer Name           | Type      | Balance")
        print("------------------------------------------------------")
        for r in rows:
            print(f"{r[0]:<6} | {r[1]:<24} | {r[2]:<9} | {r[3]:>8}")

    except Exception as e:
        print("Error while viewing accounts:", e)
    finally:
        try:
            cur.close()
            con.close()
        except:
            pass


def delete_account():
    print("\n--- Delete Account ---")
    try:
        acc_no = int(input("Enter account number to delete: "))
        con = get_connection()
        cur = con.cursor()

        cur.execute("SELECT cust_id FROM account WHERE acc_no = %s", (acc_no,))
        row = cur.fetchone()

        if row is None:
            print("Account not found.")
            return

        cust_id = row[0]

        confirm = input("Are you sure you want to delete this account? (y/n): ")
        if confirm.lower() != 'y':
            print("Deletion cancelled.")
            return

        cur.execute("DELETE FROM account WHERE acc_no = %s", (acc_no,))
        cur.execute("DELETE FROM customer WHERE cust_id = %s", (cust_id,))
        con.commit()

        print("\nAccount deleted successfully.")

    except Exception as e:
        print("Error while deleting account:", e)
    finally:
        try:
            cur.close()
            con.close()
        except:
            pass



# ---------- MAIN MENU ----------

def menu():
    while True:
        print("==========================================")
        print("        BANK MANAGEMENT SYSTEM       ")
        print("==========================================")
        print("1. Create new account")
        print("2. Deposit")
        print("3. Withdraw")
        print("4. Check balance")
        print("5. View all accounts")
        print("6. Delete account")
        print("7. Exit")
        print("======================================")

        choice = int(input("Enter your choice (1-7): "))

        if choice == 1:
            create_account()
        elif choice == 2:
            deposit()
        elif choice == 3:
            withdraw()
        elif choice == 4:
            check_balance()
        elif choice == 5:
            view_all_accounts()
        elif choice == 6:
            delete_account()
```

```
        elif choice == 7:
            print("\nThank you for using Bank Management System.")
            break
        else:
            print("Invalid choice. Please try again.")


if __name__ == "__main__":
    menu()
```

DATABASE (SQL PART)

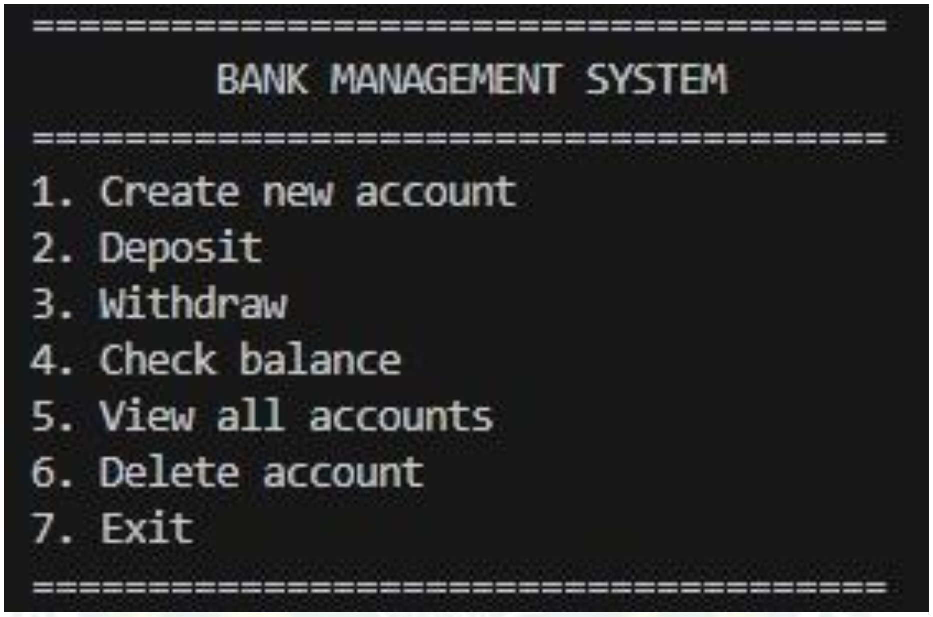
The backend database is created in MySQL using phpMyAdmin in XAMPP. The following SQL statements are used to create the database and the required tables.

```
CREATE DATABASE bank_db;

USE bank_db;


CREATE TABLE customer (

   cust_id INT AUTO_INCREMENT PRIMARY KEY,

   name VARCHAR(100),

   address VARCHAR(200),

   phone VARCHAR(15)

);


CREATE TABLE account (

   acc_no INT AUTO_INCREMENT PRIMARY KEY,

   cust_id INT,

   acc_type VARCHAR(20),

   balance DECIMAL(10,2),

   FOREIGN KEY (cust_id) REFERENCES customer(cust_id));
```

OUTPUT SCREENSHOTS

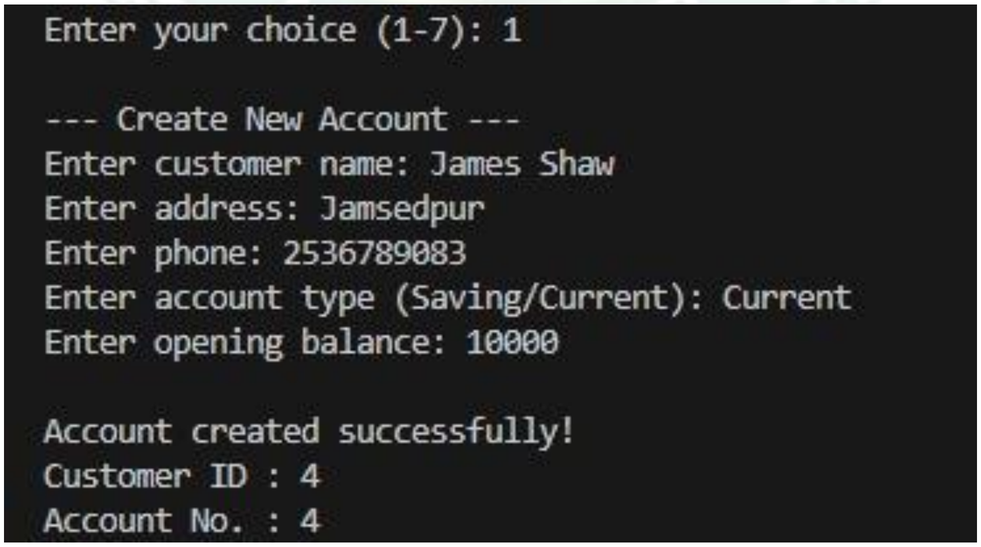
In this section, screenshots of the program execution are to be attached. The screenshots should include:

- Main menu of the Bank Management System.



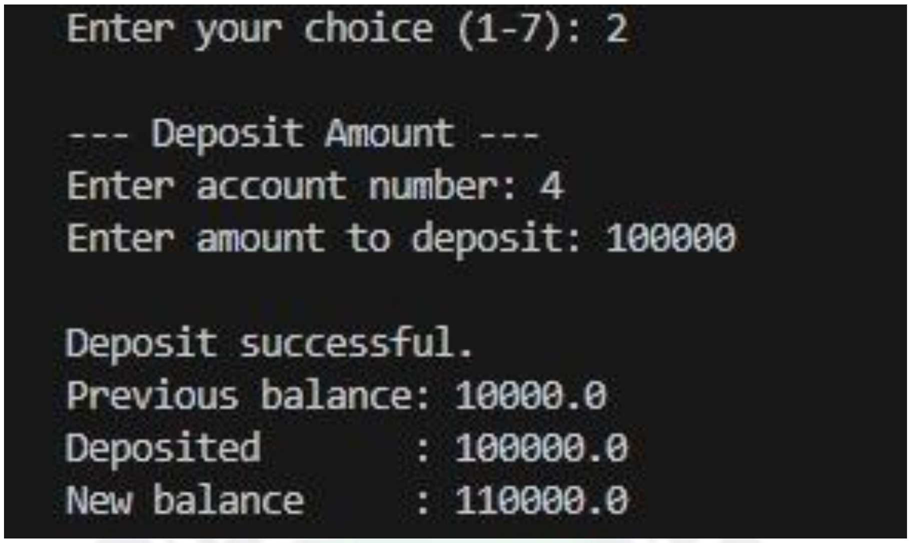
```
========================================
        BANK MANAGEMENT SYSTEM
========================================
1. Create new account
2. Deposit
3. Withdraw
4. Check balance
5. View all accounts
6. Delete account
7. Exit
========================================
```

- Creating a new account.



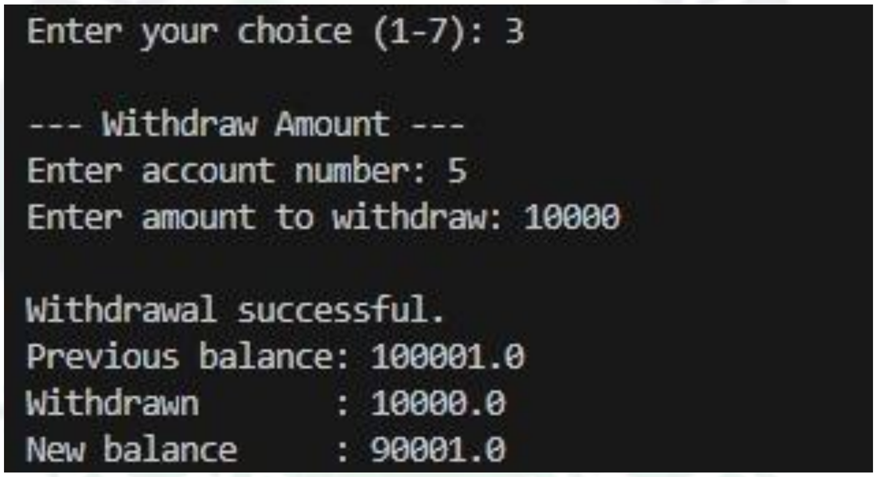
```
Enter your choice (1-7): 1

--- Create New Account ---
Enter customer name: James Shaw
Enter address: Jamsedpur
Enter phone: 2536789083
Enter account type (Saving/Current): Current
Enter opening balance: 10000

Account created successfully!
Customer ID : 4
Account No. : 4
```

- Depositing amount into an account.



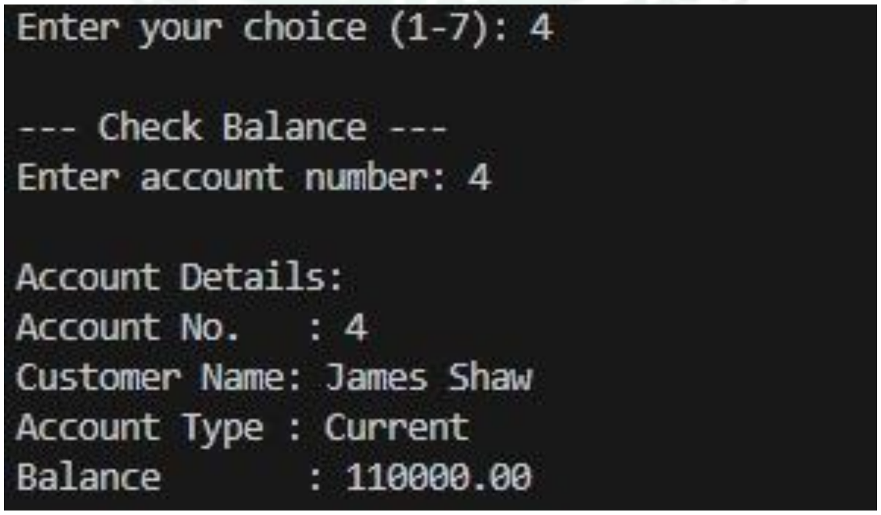
```
Enter your choice (1-7): 2

--- Deposit Amount ---
Enter account number: 4
Enter amount to deposit: 100000

Deposit successful.
Previous balance: 10000.0
Deposited       : 100000.0
New balance     : 110000.0
```

- Withdrawing amount from an account.



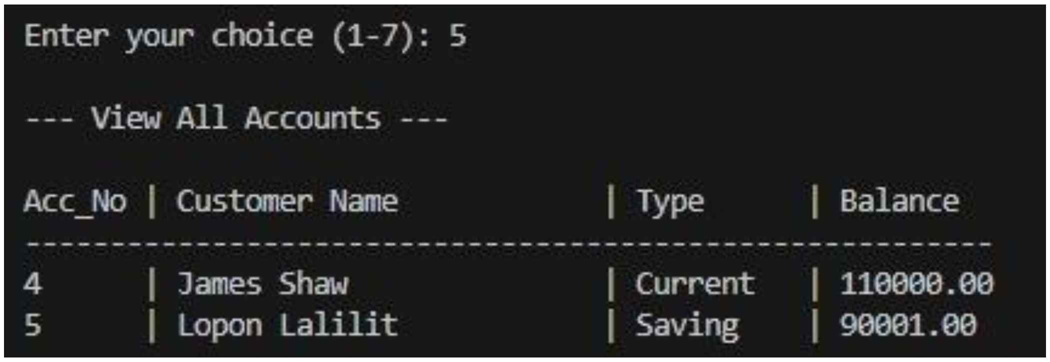
```
Enter your choice (1-7): 3

--- Withdraw Amount ---
Enter account number: 5
Enter amount to withdraw: 10000

Withdrawal successful.
Previous balance: 100001.0
Withdrawn       : 10000.0
New balance     : 90001.0
```

- Checking balance of an account.



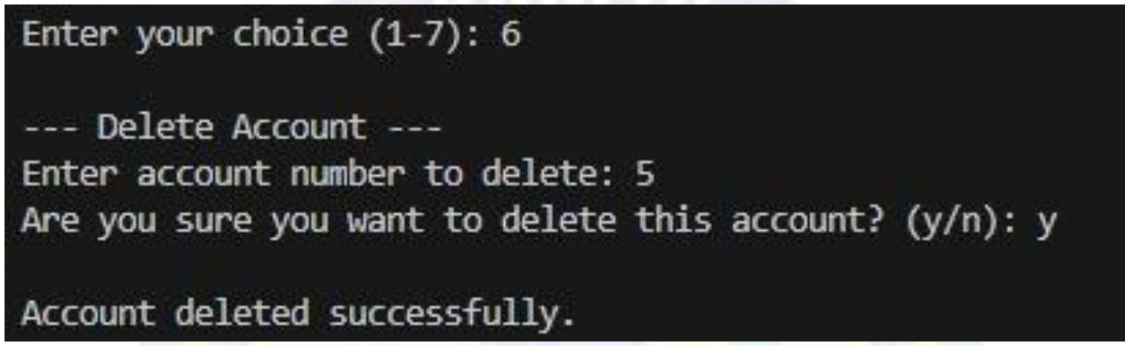
```
Enter your choice (1-7): 4

--- Check Balance ---
Enter account number: 4

Account Details:
Account No.   : 4
Customer Name: James Shaw
Account Type : Current
Balance       : 110000.00
```

- Viewing all accounts.



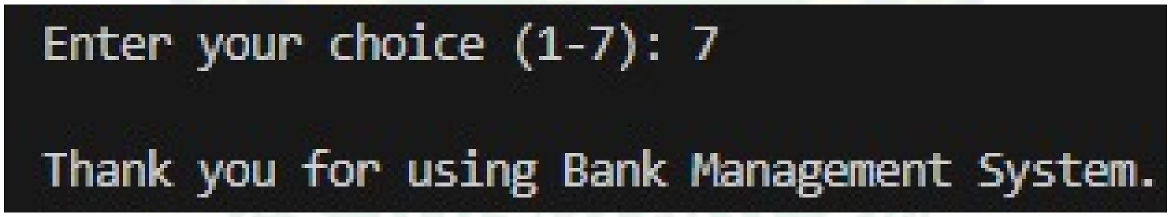
```
Enter your choice (1-7): 5

--- View All Accounts ---

Acc_No | Customer Name             | Type      | Balance
-----------------------------------------------------------
4      | James Shaw                | Current   | 110000.00
5      | Lopon Lalilit             | Saving    | 90001.00
```

-
- Delete account



```
Enter your choice (1-7): 6

--- Delete Account ---
Enter account number to delete: 5
Are you sure you want to delete this account? (y/n): y

Account deleted successfully.
```

- Delete



```
Enter your choice (1-7): 7

Thank you for using Bank Management System.
```

FUTURE SCOPE

The current version of the Bank Management System is limited to basic banking operations. In future, the project can be extended in the following ways:

1. 1. Adding a login system with different roles such as Admin and Cashier.
2. 2. Implementing transaction history and mini statement feature.
3. 3. Adding interest calculation on savings accounts.
4. 4. Providing a graphical user interface (GUI) using Tkinter or a web interface using Django/Flask.
5. 5. Integrating SMS or email alerts for important transactions.

CONCLUSION

The Bank Management System project has helped in understanding how Python can be used with MySQL to create real-life database applications. The project successfully performs basic banking functions like creating accounts, depositing and withdrawing money, checking balance and displaying all accounts.

Through this project, I learnt how to design tables in MySQL, connect them with Python using mysql-connector and write menu-driven programs. The experience gained from this project will be very useful for developing more complex applications in the future.

BIBLIOGRAPHY

1. CBSE Computer Science Textbook for Class XI and XII
2. Official Python Documentation – https://docs.python.org/
3. MySQL Documentation – https://dev.mysql.com/doc/
4. XAMPP Documentation – https://www.apachefriends.org/
5. Online resources and tutorials on Python and MySQL
6. ChatGPT by OpenAI – for explanation and guidance