तत् त्वं पूषन् अपावृणु
केन्द्रीय विद्यालय संगठन

PROJECT ON

POST OFFICE ACCOUNT SYSTEM

MADE BY

Name: Saumya Mishra,Anushka

Roll Number: 17,24

Class: 12th

School: Kendriya Vidyalaya No.1 Cantt, Shahjahanpur

Under: Kendriya Vidyalaya Sangathan (KVS)

Subject Teacher: Vaibhav Sir

� INDEX

CERTIFICATE

This is to certify that **Saumya Mishra**,**Anushka** student of Class XII, Kendriya Vidyalaya No.1 Cantt, Shahjahanpur, has successfully completed the Computer Science project titled **“Post Office Account System”** under my guidance.

Teacher’s Signature: __________

Date: __________

ACKNOWLEDGEMENT

I would like to express my sincere gratitude to my Computer Science teacher **Vaibhav Sir** for his constant guidance, encouragement, and valuable suggestions throughout the development of this project. His support and motivation helped me to understand the concepts of Python programming and MySQL database clearly.

I am also thankful to the authorities of **Kendriya Vidyalaya No.1 Cantt, Shahjahanpur** for providing the necessary facilities and resources required for the successful completion of this project.

I would like to extend my appreciation to my parents and friends for their support and encouragement during the course of this work.

Lastly, I thank everyone who directly or indirectly contributed to the completion of this project.

1. Introduction

The Post Office Account System is a database-based application developed using Python and MySQL. The objective of this project is to computerize the account management system of a post office, making it efficient, accurate, and easy to use.

2. Objectives

- To create new post office accounts
- To deposit money into accounts
- To withdraw money from accounts
- To check account balance
- To maintain transaction history
- To reduce manual work and errors

3. Hardware and Software Requirements

Hardware Requirements:

- Computer or Laptop
- Minimum 4 GB RAM

Software Requirements:

- Windows Operating System
- Python 3.x
- MySQL Server
- MySQL Connector for Python

4. Database Design

Accounts Table:

- acc_no (Primary Key)
- name
- address
- mobile
- acc_type
- balance

Transactions Table:

- txn_id (Primary Key)
- acc_no (Foreign Key)
- txn_type
- amount
- txn_date

5. Tools and Technologies Used

- Python Programming Language
- MySQL Database
- mysql-connector-python Library
- IDLE / VS Code

6. Advantages of the System

- Fast and accurate account management
- Secure data storage
- Easy to use
- Reduces paperwork

7. Interest Calculation Feature

Interest is calculated using **Simple Interest Formula**:

Interest = (Principal × Rate × Time) / 100

Example:
If balance = ₹10,000, Rate = 4%, Time = 1 year
Interest = ₹400

? SOURCE CODE

? SQL CODE (MySQL Database Code)

```
-- New database

CREATE DATABASE post_office;

USE post_office;



-- Accounts table

CREATE TABLE accounts (

    acc_no VARCHAR(10) PRIMARY KEY,

    name VARCHAR(50),

    address VARCHAR(100),

    mobile VARCHAR(15),

    acc_type VARCHAR(5),

    balance DECIMAL(10,2)

);



-- Transactions table

CREATE TABLE transactions (

    txn_id INT AUTO_INCREMENT PRIMARY KEY,

    acc_no VARCHAR(10),

    txn_type VARCHAR(20),

    amount DECIMAL(10,2),

    txn_date DATETIME,

    FOREIGN KEY (acc_no) REFERENCES accounts(acc_no) );
```

▯ PYTHON CODE (post_office.py)

```
import mysql.connector

import random

import math



# ================= DATABASE CONNECTION =================

print("Program started")



try:

    con = mysql.connector.connect(

        host="localhost",

        user="root",

        password="1234",   # <-- change this

        database="post_office",

        port=3307,

        connection_timeout=5

    )
    cur = con.cursor()

    print("Database connected")

except:

    print("ERROR: Database not connected")

    input("Press Enter to exit")

    exit()



# ================= UTILITY FUNCTIONS =================



def generate_account_number(acc_type):

    prefix = {"SB": "010", "RD": "020", "TD": "030"}

    return prefix[acc_type] + str(random.randint(1000000, 9999999))



def is_valid_mobile(mobile):

    return mobile.isdigit() and len(mobile) == 10



# ================= BASIC OPERATIONS =================



def create_account():
```

```
name = input("Name: ")

address = input("Address: ")



mobile = input("Mobile Number: ")

if not is_valid_mobile(mobile):

    print("Invalid phone number")

    return



acc_type = input("Account Type (SB/RD/TD): ").upper()

acc_no = generate_account_number(acc_type)



if acc_type == "SB":

    balance = float(input("Opening Balance (Min 500): "))

    if balance < 500:

        print("Minimum opening balance for SB is 500")

        return



elif acc_type == "RD":

    monthly = float(input("Monthly RD Amount: "))
    balance = monthly        # first installment

    months = 1



elif acc_type == "TD":

    balance = float(input("TD Lump Sum Amount: "))



else:

    print("Invalid account type")

    return



cur.execute(

    "INSERT INTO accounts VALUES (%s,%s,%s,%s,%s,%s,'Active')",

    (acc_no, name, address, mobile, acc_type, balance)

)



if acc_type == "RD":

    cur.execute(

        "INSERT INTO rd_details VALUES (%s,%s,%s)",

        (acc_no, monthly, months)
        )

    con.commit()
    print("Account Created Successfully")
    print("Account Number:", acc_no)


# ----------------------------------------------------


def deposit():
    acc = input("Account Number: ")
    amt = float(input("Amount: "))

    cur.execute("SELECT status FROM accounts WHERE acc_no=%s", (acc,))
    d = cur.fetchone()

    if not d or d[0] == "Closed":
        print("Invalid / Closed Account")
        return
    cur.execute("UPDATE accounts SET balance=balance+%s WHERE acc_no=%s", (amt, acc))

    con.commit()

    print("Deposit Successful")



# ----------------------------------------------------



def withdraw():

    acc = input("Account Number: ")

    amt = float(input("Amount: "))



    cur.execute("SELECT balance,acc_type,status FROM accounts WHERE acc_no=%s", (acc,))

    d = cur.fetchone()



    if not d or d[2] == "Closed":

        print("Invalid / Closed Account")

        return



    balance = float(d[0])

    acc_type = d[1]
    min_bal = 500 if acc_type == "SB" else 0

    if balance - amt < min_bal:
        print("Minimum balance rule violated")
        return

    cur.execute("UPDATE accounts SET balance=balance-%s WHERE acc_no=%s", (amt, acc))
    con.commit()
    print("Withdrawal Successful")

# -----------------------------------------------------

def balance_enquiry():
    acc = input("Account Number: ")
    cur.execute("SELECT name,balance,status FROM accounts WHERE acc_no=%s", (acc,))
    d = cur.fetchone()

    if d:
        print("Name:", d[0])
        print("Balance:", d[1])

        print("Status:", d[2])

    else:

        print("Account not found")



# ----------------------------------------------------



def calculate_interest():

    acc = input("Account Number: ")

    rate = 5



    cur.execute("SELECT balance,status FROM accounts WHERE acc_no=%s", (acc,))

    d = cur.fetchone()



    if not d or d[1] == "Closed":

        print("Invalid / Closed Account")

        return



    interest = (float(d[0]) * rate) / 100
    cur.execute("UPDATE accounts SET balance=balance+%s WHERE acc_no=%s", (interest,
acc))

    con.commit()

    print("Interest Added:", interest)


# ----------------------------------------------------


def search_account():

    mobile = input("Mobile Number: ")

    cur.execute("SELECT acc_no,name,acc_type,balance FROM accounts WHERE mobile=%s",
(mobile,))

    rows = cur.fetchall()


    if not rows:

        print("No accounts found")

        return


    for r in rows:

        print(r)
```

```
# ----------------------------------------------------

def close_account():
    acc = input("Account Number: ")
    cur.execute("UPDATE accounts SET status='Closed' WHERE acc_no=%s", (acc,))
    con.commit()
    print("Account Closed")

# ================= RD SCHEME =================

def rd_monthly_deposit():
    acc = input("RD Account Number: ")

    cur.execute("""
        SELECT r.monthly_amount,a.status
        FROM rd_details r JOIN accounts a ON r.acc_no=a.acc_no
        WHERE r.acc_no=%s
    """, (acc,))
    d = cur.fetchone()
    if not d or d[1] == "Closed":

        print("Invalid / Closed RD Account")

        return



    cur.execute("UPDATE accounts SET balance=balance+%s WHERE acc_no=%s", (d[0], acc))

    cur.execute("UPDATE rd_details SET months_completed=months_completed+1 WHERE
acc_no=%s", (acc,))

    con.commit()

    print("RD Monthly Deposit Successful")



# ----------------------------------------------------




def rd_compound_interest():

    acc = input("RD Account Number: ")

    rate = 5.8



    cur.execute("""

        SELECT a.balance,r.months_completed,a.status

        FROM accounts a JOIN rd_details r ON a.acc_no=r.acc_no
        WHERE a.acc_no=%s

    """, (acc,))

    d = cur.fetchone()



    if not d or d[2] == "Closed":

        print("Invalid / Closed RD Account")

        return



    balance = float(d[0])

    months = d[1]

    t = months / 12



    maturity = balance * ((1 + rate/100) ** t)

    cur.execute("UPDATE accounts SET balance=%s WHERE acc_no=%s", (maturity, acc))

    con.commit()



    print("RD Compound Interest Applied")

    print("Maturity Amount:", round(maturity,2))
```

```
# ----------------------------------------------------

def rd_schedule():

    acc = input("RD Account Number: ")

    rate = 5.8



    cur.execute("""

        SELECT a.balance,r.months_completed,r.monthly_amount,a.status

        FROM accounts a JOIN rd_details r ON a.acc_no=r.acc_no

        WHERE a.acc_no=%s

    """, (acc,))

    d = cur.fetchone()



    if not d or d[3] == "Closed":

        print("Invalid / Closed RD Account")

        return



    balance = float(d[0])

    months = d[1]
    monthly = float(d[2])



    print("\nMONTH | DEPOSIT | INTEREST | BALANCE")

    print("------------------------------------")



    for m in range(1, months + 1):

        interest = (balance * rate) / (12 * 100)

        balance = balance + interest + monthly

        print(m, "|", monthly, "|", round(interest,2), "|", round(balance,2))




# ----------------------------------------------------





def rd_to_sb_transfer():

    rd = input("RD Account Number: ")

    sb = input("SB Account Number: ")




    cur.execute("SELECT balance,status FROM accounts WHERE acc_no=%s AND acc_type='RD'",
(rd,))

    r = cur.fetchone()
    if not r or r[1] == "Closed":

        print("Invalid RD Account")

        return



    cur.execute("UPDATE accounts SET balance=balance+%s WHERE acc_no=%s", (r[0], sb))

    cur.execute("UPDATE accounts SET balance=0,status='Closed' WHERE acc_no=%s", (rd,))

    con.commit()

    print("RD Maturity Amount Transferred to SB")



# ================= SCHEME MENUS =================



def rd_menu():

    while True:

        print("""

--- RD SCHEME ---

1. Monthly RD Deposit

2. RD Compound Interest

3. RD Schedule

4. RD Maturity Transfer
0. Back
""")
        ch = input("Choice: ")
        if ch == '1': rd_monthly_deposit()
        elif ch == '2': rd_compound_interest()
        elif ch == '3': rd_schedule()
        elif ch == '4': rd_to_sb_transfer()
        elif ch == '0': break


def sb_menu():
    while True:
        print("""
--- SB SCHEME ---
1. Deposit
2. Withdraw
3. Interest
4. Account Closure
0. Back
""")
```

```
        ch = input("Choice: ")
        if ch == '1': deposit()
        elif ch == '2': withdraw()
        elif ch == '3': calculate_interest()
        elif ch == '4': close_account()
        elif ch == '0': break


def td_menu():
    while True:
        print("""
--- TD SCHEME ---
1. Interest Calculation
2. Maturity Closure
0. Back
""")
        ch = input("Choice: ")
        if ch == '1': calculate_interest()
        elif ch == '2': close_account()
        elif ch == '0': break
```

```
def schemes_menu():

    while True:

        print("""

--- SCHEMES ---

1. RD Scheme

2. SB Scheme

3. TD Scheme

0. Back

""")

        ch = input("Choice: ")

        if ch == '1': rd_menu()

        elif ch == '2': sb_menu()

        elif ch == '3': td_menu()

        elif ch == '0': break



# ================= MAIN MENU =================



while True:
    print("""

=== POST OFFICE ACCOUNT SYSTEM ===

1. Open Account

2. Deposit

3. Withdraw

4. Balance Enquiry

5. Interest Calculation

6. Search Account

7. Close Account

8. Schemes Menu

0. Exit

""")



    ch = input("Enter Choice: ")



    if ch == '1': create_account()

    elif ch == '2': deposit()

    elif ch == '3': withdraw()

    elif ch == '4': balance_enquiry()
elif ch == '5': calculate_interest()

elif ch == '6': search_account()

elif ch == '7': close_account()

elif ch == '8': schemes_menu()

elif ch == '0':

    print("Thank You")

    break

else:

    print("Invalid Choice")
```

CODE EXPLANATION

---

1. Importing Required Modules

In this project, the `mysql.connector` module is used to connect the Python program with the MySQL database. The `random` module is used to generate unique account numbers, and the `math` module is used for interest calculations, especially in Recurring Deposit (RD) accounts.

---

2. Database Connection

This section establishes a connection between Python and the MySQL database using host name, user name, password, database name, and port number. A `try-except` block is used to handle connection errors. If the connection is successful, the program can perform database operations like insert, update, and select.

---

3. Cursor Object

A cursor object is created after establishing the database connection. The cursor is used to execute SQL queries from the Python program. It acts as an interface between Python and the MySQL database.

---

4. Utility Functions

Utility functions are created to improve code reusability:

- A function to generate a 10-digit account number based on account type (SB, RD, TD).
- A mobile number validation function that ensures only valid 10-digit numeric mobile numbers are accepted.

---

5. Account Creation Function

This function is used to create new post office accounts. It takes user input such as name, address, mobile number, and account type.

- For SB accounts, a minimum opening balance of ₹500 is required.
- For RD accounts, the first monthly installment is credited at the time of account opening.
- For TD accounts, a lump sum amount is deposited.
  All account details are stored in the `accounts` table, and RD-specific details are stored in the `rd_details` table.

---

6. Deposit Function

The deposit function allows users to deposit money into an active account. Before depositing, the function checks whether the account is valid and not closed. After validation, the deposited amount is added to the account balance in the database.

---

7. Withdraw Function

This function is used to withdraw money from an account. It checks whether the account is active and ensures that minimum balance rules are followed (minimum ₹500 for SB accounts). If all conditions are satisfied, the withdrawal amount is deducted from the balance.

---

8. Balance Enquiry Function

The balance enquiry function displays the account holder's name, current balance, and account status using the account number. This helps the user to easily check account details.

---

9. Interest Calculation Function

This function calculates simple interest on Savings Bank (SB) and Time Deposit (TD) accounts using a fixed rate of interest. The calculated interest is added directly to the account balance in the database.

---

10. RD Monthly Deposit Function

This function allows monthly deposits in RD accounts. Before depositing, it verifies whether the RD account is active. After successful validation, the monthly deposit amount is added to the RD balance and the number of completed months is increased.

---

11. RD Compound Interest Function

This function calculates compound interest for RD accounts based on the number of months completed. The maturity amount is calculated using a compound interest formula and updated in the account balance.

---

12. RD Schedule Function

The RD schedule function displays a month-wise table showing deposit amount, interest earned, and balance for each month. This helps in understanding the growth of the RD account clearly.

---

13. RD Maturity Transfer Function

This function transfers the maturity amount of an RD account to a linked SB account. After successful transfer, the RD account is marked as closed to prevent further transactions.

---

14. Scheme-wise Menu System

The program follows a hierarchical menu system:

- A main menu for basic banking operations.
- A schemes menu that separates RD, SB, and TD operations.
- Individual sub-menus for each scheme to keep the program organized and user-friendly.

---

15. Menu-Driven Program

The entire project is menu-driven. Users select operations by entering choices from the menu. Based on the selected option, the corresponding function is executed, making the system easy to use and interactive.

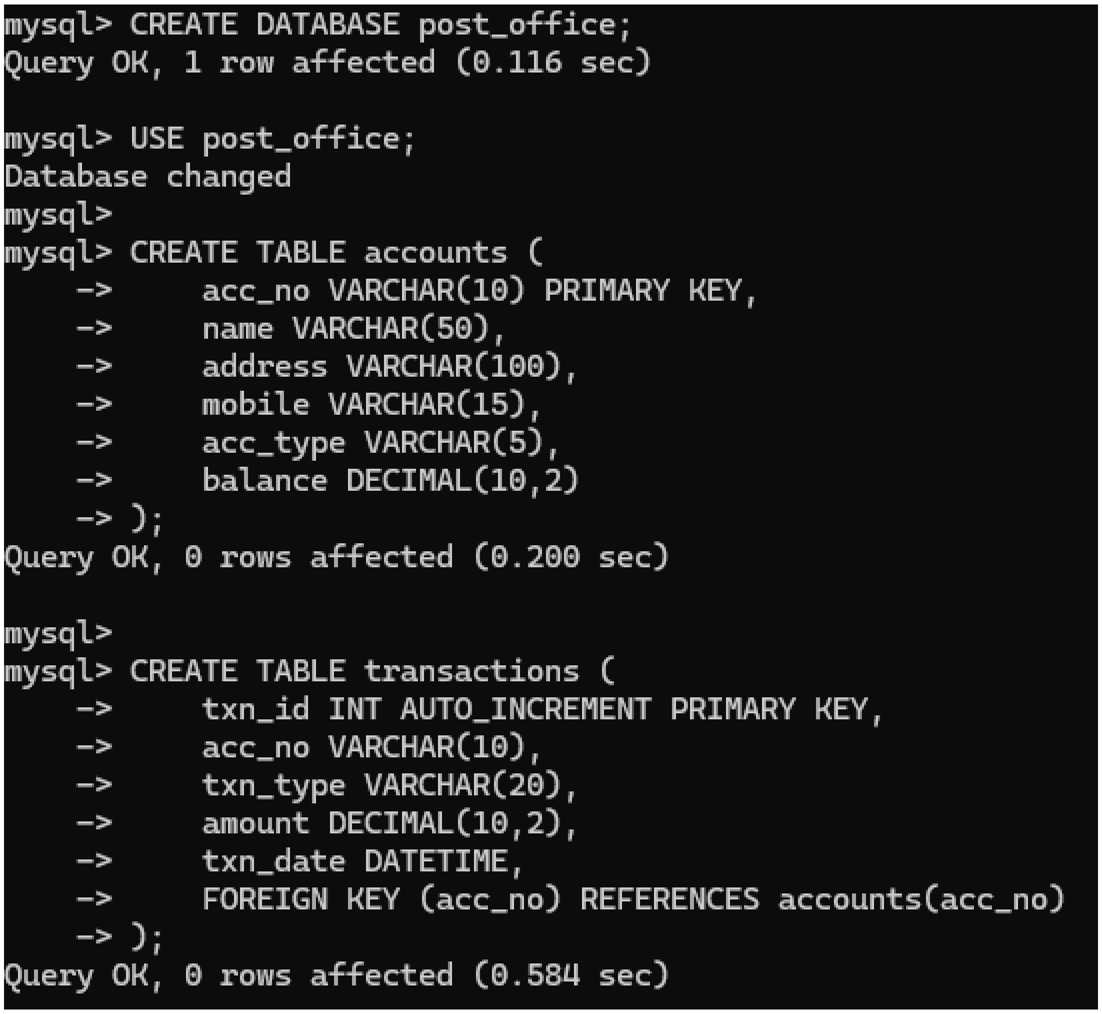
---

16. Database Commit Operation

After every insert, update, or delete operation, the `commit()` function is used to save changes permanently in the MySQL database. This ensures data consistency and reliability.

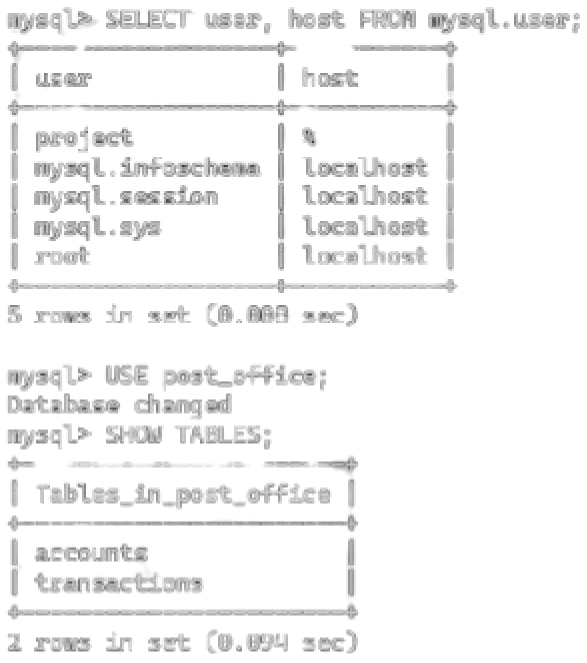
📷 OUTPUT SCREENSHOTS (HEADINGS)

Output Screen 1: MySQL Database and Tables Creation



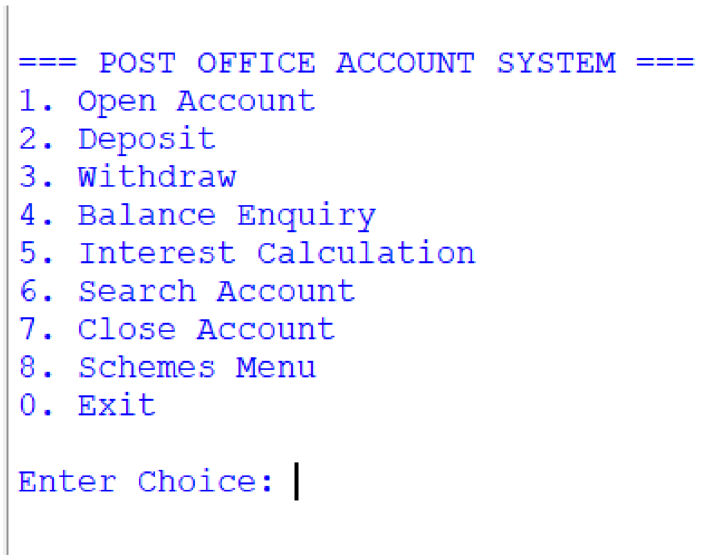
```
mysql> CREATE DATABASE post_office;
Query OK, 1 row affected (0.116 sec)

mysql> USE post_office;
Database changed
mysql>
mysql> CREATE TABLE accounts (
    ->     acc_no VARCHAR(10) PRIMARY KEY,
    ->     name VARCHAR(50),
    ->     address VARCHAR(100),
    ->     mobile VARCHAR(15),
    ->     acc_type VARCHAR(5),
    ->     balance DECIMAL(10,2)
    -> );
Query OK, 0 rows affected (0.200 sec)

mysql>
mysql> CREATE TABLE transactions (
    ->     txn_id INT AUTO_INCREMENT PRIMARY KEY,
    ->     acc_no VARCHAR(10),
    ->     txn_type VARCHAR(20),
    ->     amount DECIMAL(10,2),
    ->     txn_date DATETIME,
    ->     FOREIGN KEY (acc_no) REFERENCES accounts(acc_no)
    -> );
Query OK, 0 rows affected (0.584 sec)
```

Output Screen 2: Successful Database Connection



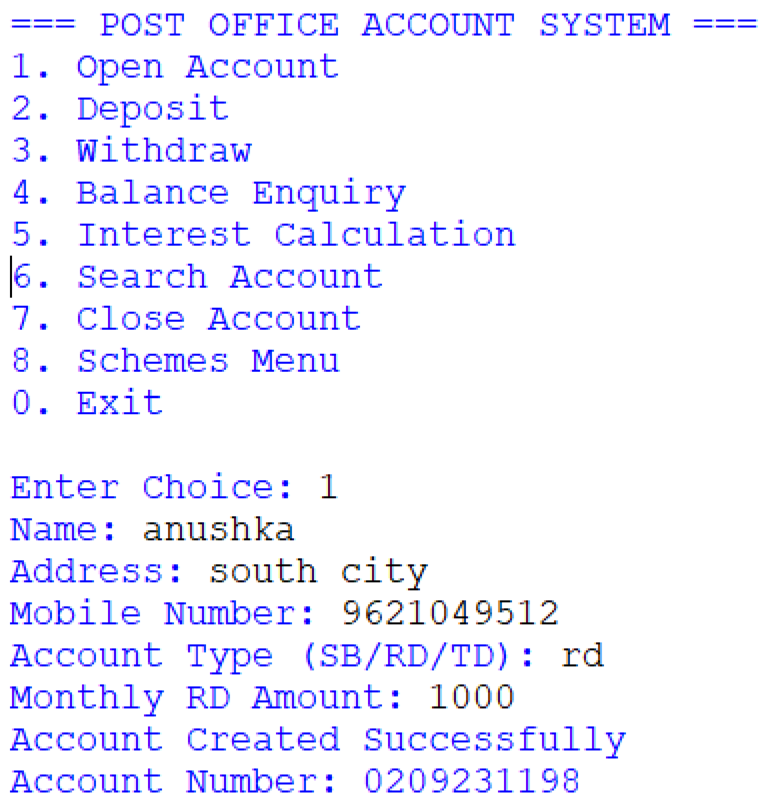
```
mysql> SELECT user, host FROM mysql.user;
+------------------+-----------+
| user             | host      |
+------------------+-----------+
| project          | %         |
| mysql.infoschema | localhost |
| mysql.session    | localhost |
| mysql.sys        | localhost |
| root             | localhost |
+------------------+-----------+
5 rows in set (0.008 sec)

mysql> USE post_office;
Database changed
mysql> SHOW TABLES;
+-----------------------+
| Tables_in_post_office |
+-----------------------+
| accounts              |
| transactions          |
+-----------------------+
2 rows in set (0.094 sec)
```

Output Screen 3: Main Menu of Post Office Account System



```
=== POST OFFICE ACCOUNT SYSTEM ===
1. Open Account
2. Deposit
3. Withdraw
4. Balance Enquiry
5. Interest Calculation
6. Search Account
7. Close Account
8. Schemes Menu
0. Exit

Enter Choice:
```

Output Screen 4: Account Creation Output



```
=== POST OFFICE ACCOUNT SYSTEM ===
1. Open Account
2. Deposit
3. Withdraw
4. Balance Enquiry
5. Interest Calculation
6. Search Account
7. Close Account
8. Schemes Menu
0. Exit

Enter Choice: 1
Name: anushka
Address: south city
Mobile Number: 9621049512
Account Type (SB/RD/TD): rd
Monthly RD Amount: 1000
Account Created Successfully
Account Number: 0209231198
```

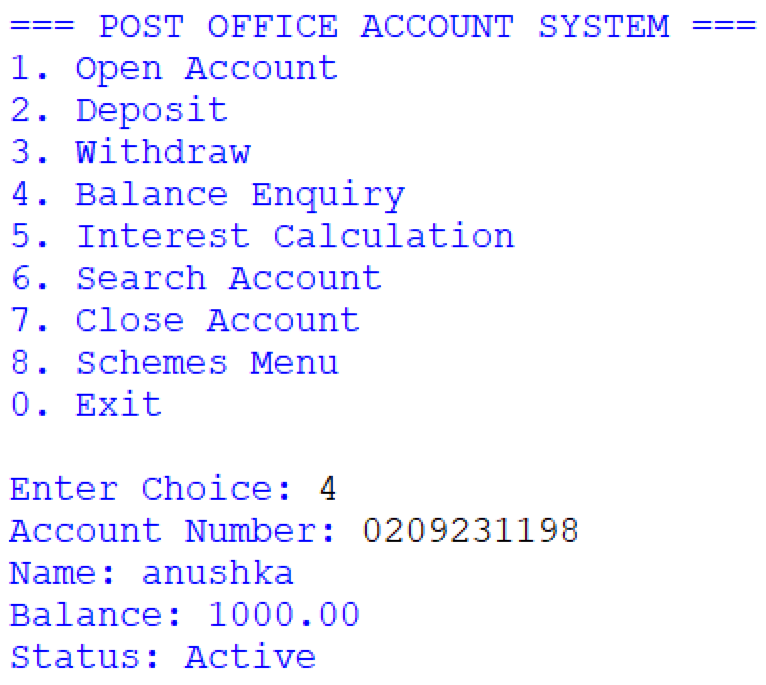
Output Screen 5: Deposit Amount Output

```
=== POST OFFICE ACCOUNT SYSTEM ===
1. Open Account
2. Deposit
3. Withdraw
4. Balance Enquiry
5. Interest Calculation
6. Search Account
7. Close Account
8. Schemes Menu
0. Exit

Enter Choice: 2
Account Number: 0209231198
Amount: 1000
Deposit Successful
```

Output Screen 6: Withdraw Amount Output

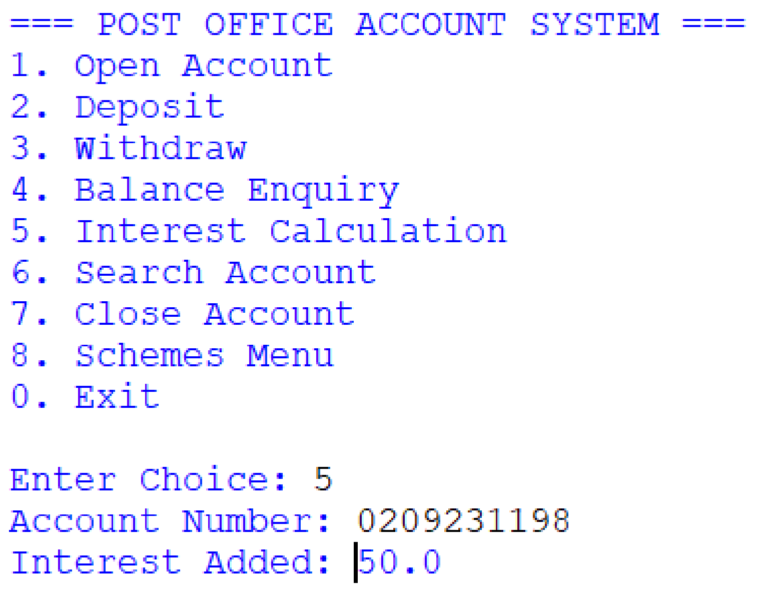
```
=== POST OFFICE ACCOUNT SYSTEM ===
1. Open Account
2. Deposit
3. Withdraw
4. Balance Enquiry
5. Interest Calculation
6. Search Account
7. Close Account
8. Schemes Menu
0. Exit

Enter Choice: 3
Account Number: 0209231198
Amount: 1000
Withdrawal Successful
```

Output Screen 7: Balance Enquiry Output



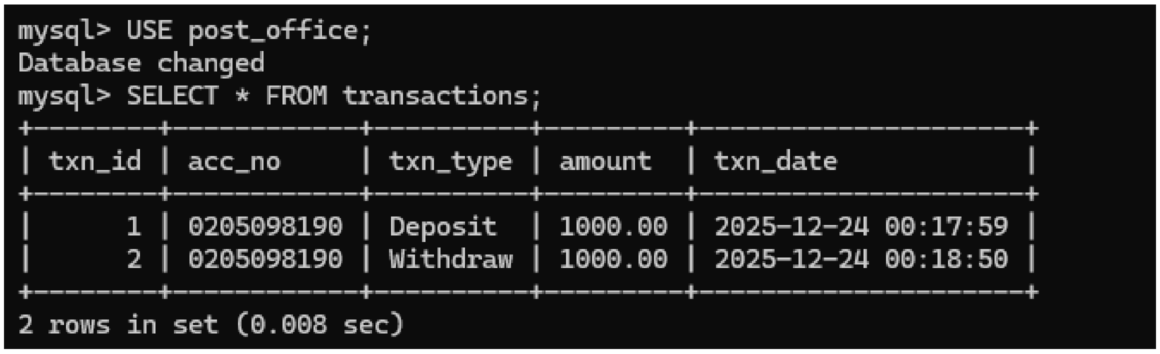
```
=== POST OFFICE ACCOUNT SYSTEM ===
1. Open Account
2. Deposit
3. Withdraw
4. Balance Enquiry
5. Interest Calculation
6. Search Account
7. Close Account
8. Schemes Menu
0. Exit

Enter Choice: 4
Account Number: 0209231198
Name: anushka
Balance: 1000.00
Status: Active
```

Output Screen 8: Interest Calculation Output



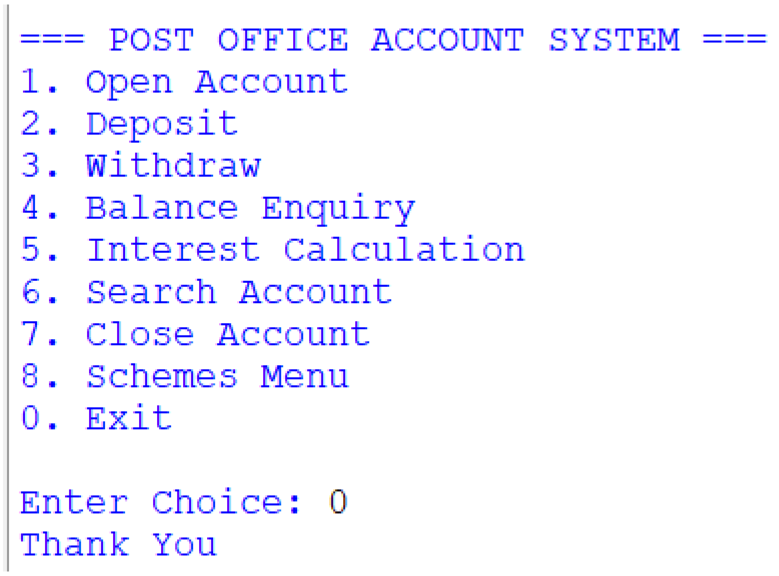
```
=== POST OFFICE ACCOUNT SYSTEM ===
1. Open Account
2. Deposit
3. Withdraw
4. Balance Enquiry
5. Interest Calculation
6. Search Account
7. Close Account
8. Schemes Menu
0. Exit

Enter Choice: 5
Account Number: 0209231198
Interest Added: 50.0
```

Output Screen 9: Transaction Records in MySQL Table



```
mysql> USE post_office;
Database changed
mysql> SELECT * FROM transactions;
+--------+------------+----------+---------+---------------------+
| txn_id | acc_no     | txn_type | amount  | txn_date            |
+--------+------------+----------+---------+---------------------+
|      1 | 0205098190 | Deposit  | 1000.00 | 2025-12-24 00:17:59 |
|      2 | 0205098190 | Withdraw | 1000.00 | 2025-12-24 00:18:50 |
+--------+------------+----------+---------+---------------------+
2 rows in set (0.008 sec)
```

Output Screen 10: Program Exit Screen



```
=== POST OFFICE ACCOUNT SYSTEM ===
1. Open Account
2. Deposit
3. Withdraw
4. Balance Enquiry
5. Interest Calculation
6. Search Account
7. Close Account
8. Schemes Menu
0. Exit

Enter Choice: 0
Thank You
```

▯ ADVANTAGES OF THE PROJECT

1. **Reduces manual work**
   The system automates post office account management, reducing paperwork and manual record keeping.
2. **Fast and accurate operations**
   All calculations such as balance updates and interest calculation are performed automatically, reducing chances of human error.
3. **User-friendly menu-driven interface**
   The menu-driven design makes the system easy to use even for beginners.
4. **Efficient data management**
   The use of MySQL database ensures that account records are stored securely and can be retrieved easily.
5. **Scheme-wise organization**
   Separate handling of SB, RD, and TD schemes improves clarity and better understanding of different account types.
6. **Input validation and data security**
   Validation checks such as mobile number length and minimum balance rules help maintain correct and reliable data.
7. **Realistic banking simulation**
   The project closely simulates real post office banking operations like RD monthly deposits and maturity transfers.
8. **Easy maintenance and scalability**
   The modular design allows future enhancements without major changes in the existing code.
9. **Educational value**
   The project helps in understanding Python programming, SQL queries, and database connectivity practically.
10. **Cost-effective solution**
    Since the project uses open-source tools, it is economical and easy to implement.

.

▯ LIMITATIONS OF THE PROJECT

1. **No graphical user interface (GUI)**
   The system is console-based, which may not be as user-friendly as graphical applications.
2. **Single-user system**
   The project supports only one user at a time and does not include multi-user login functionality.
3. **No real-time SMS or OTP services**
   Advanced security features like real SMS alerts and OTP authentication are not included.
4. **Limited security features**
   The system does not implement encryption or password-based authentication.
5. **Fixed interest rates**
   Interest rates are hard-coded and cannot be changed dynamically.
6. **No transaction history report**
   Detailed transaction statements are not generated in the current version.
7. **Internet-independent but local only**
   The system works only on the local machine and cannot be accessed remotely.
8. **Academic-level scope**
   The project is designed for learning purposes and is not suitable for real-world deployment without enhancements.

▯ FUTURE SCOPE

The current system is designed for academic purposes, but it can be further enhanced in the future by adding advanced features such as:

- Graphical User Interface (GUI) for better user interaction
- Web-based version of the application
- Transaction history and account statement generation
- Enhanced security features like OTP, PIN, and user authentication
- Integration with real SMS and email notification services
- Multi-user access with role-based permissions

These enhancements can make the system more secure, scalable, and suitable for real-world deployment.

▯ CONCLUSION

The Post Office Account System project successfully demonstrates the practical application of Python programming and MySQL database in developing a real-life banking system. The project automates important post office operations such as account creation, deposit, withdrawal, balance enquiry, interest calculation, and scheme-wise management of SB, RD, and TD accounts.

The menu-driven and modular design of the system makes it user-friendly and easy to understand. Proper input validation, account status checks, and interest calculations ensure data accuracy and reliability. The project fulfills all academic requirements of the Class 12 Computer Science curriculum and provides a strong foundation for understanding database-based applications.

Overall, this project helped in gaining hands-on experience in database connectivity, SQL operations, and logical problem solving using Python.

▯ REFERENCES

The following resources were referred to during the development of this project:

1. Python Official Documentation
   https://docs.python.org
2. MySQL Official Documentation
   https://dev.mysql.com/doc
3. mysql-connector-python Documentation
   https://dev.mysql.com/doc/connector-python/en
4. CBSE Computer Science Curriculum (Class XII)
5. Online tutorials and learning resources for Python and MySQL