ATM SYSTEM

Program Documentation

1.Overview

This Python based ATM System uses MYSQL as database and transaction history. It supports account creation , login , deposites, withdrawals, transfers, mini statements, PIN change, admin control etc.

2. Technologies Used

- **Python –for implementing application logic.**
- **MySQL Database- to store account data and transactions**
- **Mysql.connector- for Python-MySQL connectivity**
- **CSV- for exporting trasactions as CSV**
- **Datetime- for handling dates and times**

3.Database Design

Datebase Name: atm

Tables:

1. **ac_details**

Column	Type	Description
ac_no(pk)	**VARCHAR(20)**	**Account number**
c_name	**VACRCHAR(100)**	**Customer name**
Pin	**VARCHAR(10)**	**4-digt PIN for authentication**
opening_balance	**DECIMAL**	**Current account balance**

2. **transition_table**

Column	Type	Description
id(pk)	**INT**	**Auto-increment transaction id**
ac_no(fk)	**VARCHAR(20)**	**Account number**
type	**ENUM**	**Transaction type (deposit,withdraw etc)**
amount	**DECIMAL**	**Amount (transaction)**
date	**DATETIME**	**Date and time of transaction**

4.Program Modules

A. Database Setup

- **create_datebase() – Create the atm databae if not present.**
- **Create_tables()- create the required tables (ac_details, transition_tables). Whenever the program runs for the first time, these will ensure everything is ready.**

B. Account Management functions

1.create New Account (create_account)

- **Prompts for Account Number, Name, Pin , opening Balance.**
- **Inserts into ac_details.**

2.Close Account (close_account)

- **Deletes account form ac_details and all related transactions from transition_tables.**

3.Login (login)

Verifies account number and pin for authentication.

4.Change PIN (change_pin)

Allows user to change ATM PIN after verifying current PIN.

C. Banking Operations

1.Check Balance (check_balance)

- **Fetches and displays opening_balance from ac_details.**

2.Deposit (deposit)

- **Increases balance in ac_details.**
- **Inserts a transaction in transition_table.**

3.Withdraw (withdraw)

- **Checks if withdrawal is within daily withdrawal limit (default 20,000).**
- **Updates balance and logs transaction.**

4.Transfer Funds (transfer_funds)

- **Deducts from sender and adds to receiver.**
- **Logs two transactions: transfer_out for sender, transfer_in for receiver.**

5.Credit Interest (credit_interest)

- **Calculates interest = balance × rate / 100.**
- **Deposits into account as interest transaction.**

D. Transaction Features

1.Mini Statement (mini_statement)

- **Displays last 5 transactions for the account.**

2.Search Transaction by Date (transaction_by_date)

- **Queries transactions between a given start and end date.**

3.Export Transactions to CSV (export_transactions)

- **Saves all transactions into a .csv file for printing/record keeping.**

E. Admin Panel Features

1.Admin Login (admin_login)

- **Username: admin**
- **Password: admin123**

2.View All Accounts (view_all_accounts)

- **Lists all customer accounts.**

3.View All Transactions (view_all_transactions)

- **Displays complete transactions report for all accounts.**

5. Program Flow

1.On Start

- **Calls create_database() → Ensures DB exists.**
- **Calls create_tables() → Ensures tables exist.**
- **Runs atm() function.**

2.Main ATM Menu

1. **Login & Use ATM**
2. **Create New Account**
3. **Admin Panel**
4. **Exit**

Inside ATM Login

Once a user logs in, they get:

1. **Check Balance**
2. **Deposit**
3. **Withdraw**
4. **Mini Statement**
5. **Transfer Funds**
6. **Credit Interest**
7. **Change PIN**

8. Search Transactions by Date
9. Export Transactions to CSV
10. Close Account
11. Logout

Admin Panel Menu

If admin logs in:

1. View All Accounts
2. View All Transactions
3. Back

6. Special Features

- ✓ Automatic Database & Table Creation on first run
- ✓ Daily Withdrawal Limit for security
- ✓ Export Transactions to CSV for permanent records
- ✓ Transaction Search by Date
- ✓ Admin Level Reporting
- ✓ Supports Multiple Transaction Types (deposit, withdraw, transfer_in, transfer_out, interest)

7. How to Run

Step 1 – Install MySQL Connector

```
pip install mysql-connector-python
```

Step 2 – Run Program

```
python atm.py
```

Ensure MySQL server is running locally with:

host = localhost

user = root

password = 12345(if you use password in your database otherwise leave it)

8. Database Backup & Maintenance

Since this system is in MySQL, you can back up using:

```
mysqldump -u root -p atm > atm_backup.sql
```

Restore anytime using:

```
mysql -u root -p atm < atm_backup.sql
```

ATM System –Python code

```
import mysql.connector
import csv
from datetime import datetime, date
# ========= Database Connection =========
def connect_db(database=True):
    return mysql.connector.connect(
    host="localhost",
    user="root",
    password="12345",
    database="atm" if database else None )

# ========= Database Setup =========
def create_database():
    conn = connect_db(False)
    cur = conn.cursor()
    cur.execute("CREATE DATABASE IF NOT EXISTS atm")
    conn.commit()
    conn.close()

def create_tables():
    conn = connect_db()
    cur = conn.cursor()
    cur.execute("""CREATE TABLE IF NOT EXISTS ac_details (
        ac_no VARCHAR(20) PRIMARY KEY,
```

```
            c_name VARCHAR(100) NOT NULL,
            pin VARCHAR(10) NOT NULL,
            opening_balance DECIMAL(15, 2) DEFAULT 0 )""")
        cur.execute(""" CREATE TABLE IF NOT EXISTS transition_table (
            id INT AUTO_INCREMENT PRIMARY KEY,
            ac_no VARCHAR(20),
            type ENUM('deposit', 'withdraw', 'transfer_in', 'transfer_out', 'interest')
            NOT NULL,amount DECIMAL(15, 2) NOT NULL,date DATETIME NOT
            NULL,FOREIGN KEY (ac_no) REFERENCES ac_details(ac_no) ON DELETE
            CASCADE ) """)
            conn.commit()
            conn.close()

# ========= Account Management =========
def create_account():
        conn = connect_db()
        cur = conn.cursor()
        ac_no = input("Enter new account number: ")
        name = input("Enter account holder name: ")
        pin = input("Set a 4-digit PIN: ")
        bal = float(input("Enter opening balance: "))
        cur.execute("INSERT INTO ac_details VALUES (%s, %s, %s, %s)", (ac_no,
        name, pin, bal))
        conn.commit()
        conn.close()
        print("□ New account created successfully!")
```

```
def close_account(ac_no):
    conn = connect_db()
    cur = conn.cursor()
    cur.execute("DELETE FROM ac_details WHERE ac_no=%s", (ac_no,))
    conn.commit()
    conn.close()
    print("□ Account closed successfully!")

def login(ac_no, pin):
    conn = connect_db()
    cur = conn.cursor()
    cur.execute("SELECT * FROM ac_details WHERE ac_no=%s AND pin=%s",
    (ac_no, pin)) account = cur.fetchone()
    conn.close()
    return account

def change_pin(ac_no):
    conn = connect_db()
    cur = conn.cursor()
    current_pin = input("Enter current PIN: ")
    cur.execute("SELECT pin FROM ac_details WHERE ac_no=%s", (ac_no,))
    result = cur.fetchone()
    if result and result[0] == current_pin:
        new_pin = input("Enter new 4-digit PIN: ")
```

```
            confirm_pin = input("Confirm new PIN: ")
            if new_pin == confirm_pin:
                cur.execute("UPDATE ac_details SET pin=%s WHERE ac_no=%s",
                (new_pin, ac_no))
                conn.commit()
                print("✅ PIN changed successfully.")
            else:
                print("❌ PINs do not match.")
            else:
                print("❌ Incorrect current PIN.")
            conn.close()

# ========= Transactions =========
def check_balance(ac_no):
    conn = connect_db()
    cur = conn.cursor()
    cur.execute("SELECT opening_balance FROM ac_details WHERE ac_no=%s",
    (ac_no,)) bal = cur.fetchone()
    conn.close()
    print(f"💰 Current Balance: {bal[0]}" if bal else "Account not found.")

def deposit(ac_no, amount, t_type="deposit"):
    conn = connect_db()
    cur = conn.cursor()
```

```
        cur.execute("UPDATE ac_details SET opening_balance=opening_balance+%s
        WHERE ac_no=%s", (amount, ac_no))

        cur.execute("INSERT INTO transition_table (ac_no, type, amount, date)
        VALUES (%s, %s, %s, %s)",(ac_no, t_type, amount, datetime.now()))

        conn.commit()

        conn.close()

        print(f"□ {amount} deposited successfully.")


            def withdraw(ac_no, amount, daily_limit=20000):

               conn = connect_db()

               cur = conn.cursor()
# Check today's withdrawal total

        today = date.today()

        cur.execute("""SELECT SUM(amount) FROM transition_table WHERE
        ac_no=%s AND type='withdraw' AND DATE(date)=%s """, (ac_no, today))

        total_today = cur.fetchone()[0] or 0

            if total_today + amount > daily_limit:

                print(f"⚠ Daily withdrawal limit of {daily_limit} exceeded.")

                conn.close()

                return


        cur.execute("SELECT opening_balance FROM ac_details WHERE ac_no=%s",
        (ac_no,)) bal = cur.fetchone()

            if not bal or bal[0] < amount:

                print("□ Insufficient funds.")

            else:
```

```
            cur.execute("UPDATE ac_details SET
            opening_balance=opening_balance-%s WHERE ac_no=%s", (amount,
            ac_no))

            cur.execute("INSERT INTO transition_table (ac_no, type, amount, date)
            VALUES (%s, 'withdraw', %s, %s)", (ac_no, amount, datetime.now()))

            conn.commit()

            print(f"□ {amount} withdrawn successfully.")

            conn.close()


def transfer_funds(sender, receiver, amount):
        conn = connect_db()

        cur = conn.cursor()

        cur.execute("SELECT opening_balance FROM ac_details WHERE ac_no=%s",
        (sender,)) bal = cur.fetchone()

        if not bal or bal[0] < amount:

            print("□ Insufficient funds.")

            conn.close()

            return

        cur.execute("UPDATE ac_details SET opening_balance=opening_balance-%s
        WHERE ac_no=%s", (amount, sender))

        cur.execute("UPDATE ac_details SET opening_balance=opening_balance+%s
        WHERE ac_no=%s", (amount, receiver))

        cur.execute("INSERT INTO transition_table (ac_no, type, amount, date)
        VALUES (%s, 'transfer_out', %s, %s)",(sender, amount, datetime.now()))

        cur.execute("INSERT INTO transition_table (ac_no, type, amount, date)
        VALUES (%s, 'transfer_in', %s, %s)",(receiver, amount, datetime.now()))

        conn.commit()
```

```
    conn.close()
    print(f"✅ {amount} transferred successfully.")

def credit_interest(ac_no, rate=2.5):
    conn = connect_db()
    cur = conn.cursor()
    cur.execute("SELECT opening_balance FROM ac_details WHERE ac_no=%s",
    (ac_no,)) bal = cur.fetchone()
    if bal:
        interest = bal[0] * rate / 100
        deposit(ac_no, interest, "interest")
        print(f"✅ Interest of {interest} credited.")
    else:
        print("❌ Account not found.")
    conn.close()

def mini_statement(ac_no):
    conn = connect_db()
    cur = conn.cursor()
    cur.execute("SELECT type, amount, date FROM transition_table WHERE
    ac_no=%s ORDER BY date DESC LIMIT 5", (ac_no,)) rows = cur.fetchall()
    conn.close()
    print("\n📜 Last 5 Transactions:")
    for r in rows:
        print(f"{r[2]} | {r[0]} | {r[1]}")
```

```
def transaction_by_date(ac_no, start, end):
    conn = connect_db()
    cur = conn.cursor()
    cur.execute("""SELECT type, amount, date FROM transition_table
        WHERE ac_no=%s AND DATE(date) BETWEEN %s AND %s ORDER BY
        date DESC """, (ac_no, start, end)) rows = cur.fetchall()
    conn.close()
    print(f"\n Transactions from {start} to {end}:")
    for r in rows:
        print(f"{r[2]} | {r[0]} | {r[1]}")


def export_transactions(ac_no):
    conn = connect_db()
    cur = conn.cursor()
    cur.execute("SELECT * FROM transition_table WHERE ac_no=%s", (ac_no,))
    rows = cur.fetchall()
    conn.close()
    filename = f"{ac_no}_transactions.csv"
    with open(filename, "w", newline="") as file:
        writer = csv.writer(file)
        writer.writerow(["ID", "Account No", "Type", "Amount", "Date"])
        writer.writerows(rows)
        print(f" Transactions exported to {filename}")
```

```
# ========= Admin Functions =========
        def admin_login():
            username = input("Admin username: ")
            password = input("Admin password: ")
            return username == "admin" and password == "admin123"


        def view_all_accounts():
            conn = connect_db()
            cur = conn.cursor()
            cur.execute("SELECT * FROM ac_details")
            for row in cur.fetchall():
                print(row)
            conn.close()


        def view_all_transactions():
            conn = connect_db()
            cur = conn.cursor()
            cur.execute("SELECT * FROM transition_table ORDER BY date DESC")
            for row in cur.fetchall():
                print(row)
            conn.close()
# ========= ATM Menu =========
        def atm():
            while True:
                print("\n===== ATM System =====")
```

```
print("1. Login & Use ATM")
print("2. Create New Account")
print("3. Admin Panel")
print("4. Exit")
choice = input("Enter choice: ")
if choice == "1":
    ac = input("Enter Account Number: ")
    pin = input("Enter PIN: ")
    if not login(ac, pin):
        print("❌ Invalid login.")
        continue
    while True:
        print("\n1. Check Balance")
        print("2. Deposit")
        print("3. Withdraw")
        print("4. Mini Statement")
        print("5. Transfer Funds")
        print("6. Credit Interest")
        print("7. Change PIN")
        print("8. Search Transactions by Date")
        print("9. Export Transactions to CSV")
        print("10. Close Account")
        print("11. Logout")
        op = input("Enter choice: ")
        if op == "1": check_balance(ac)
```

```
                    elif op == "2": deposit(ac, float(input("Amount: ")))
                    elif op == "3": withdraw(ac, float(input("Amount: ")))
                    elif op == "4": mini_statement(ac)
                    elif op == "5": transfer_funds(ac, input("Receiver: "),
float(input("Amount: ")))
                    elif op == "6": credit_interest(ac)
                    elif op == "7": change_pin(ac)
                    elif op == "8": transaction_by_date(ac, input("Start date (YYYY-
MM-DD): "), input("End date (YYYY-MM-DD): "))
                    elif op == "9": export_transactions(ac)
                    elif op == "10": close_account(ac); break
                    elif op == "11": break
                    else: print("⚠ Invalid choice.")
            elif choice == "2":
                create_account()
            elif choice == "3":
                if admin_login():
                    while True:
                        print("\n*** ADMIN PANEL ***")
                        print("1. View All Accounts")
                        print("2. View All Transactions")
                        print("3. Back")
                        admin_choice = input("Enter choice: ")
                        if admin_choice == "1": view_all_accounts()
                        elif admin_choice == "2": view_all_transactions()
```

```
                    elif admin_choice == "3": break
                    else: print("❌ Invalid choice.")
                else:
                    print("❌ Access Denied.")
            elif choice == "4":
                print("👋 Goodbye!")
                break
            else:
                print("❌ Invalid choice.")

# ========= Start =========
        if __name__ == "__main__":
            create_database()
            create_tables()
            atm()
```

Explaination:

Here is a line-by-line explanation …….

Imports and Setup

Python code:

```
import mysql.connector

import csv

from datetime import datetime, date
```

- ❖ Imports `mysql.connector` for connecting to MySQL databases.
- ❖ Imports `csv` for exporting transaction data.
- ❖ Imports `datetime` and `date` for handling dates and timestamps.

Database Connection

Python code:

```
def connect_db(database=True):
    return mysql.connector.connect(
        host="localhost",
        user="root",
        password="12345",
        database="atm" if database else None
    )
```

- ➢ Defines a function to connect to MySQL.
- ➢ Connects to the "atm" database if `database=True`, otherwise connects without selecting a database.
- ➢ Uses localhost, with user "root" and password "12345".

Database Setup

Python code:

```
def create_database():
    conn = connect_db(False)
    cur = conn.cursor()
    cur.execute("CREATE DATABASE IF NOT EXISTS atm")
    conn.commit()
    conn.close()
```

- Connects without a database selected.
- Creates the `atm` database if it doesn't exist.
- Commits the change and closes the connection.

Python code

```
def create_tables():
    conn = connect_db()
    cur = conn.cursor()
```

- Connects to the `atm` database.
- Gets a cursor for executing SQL commands.

Python code:

```
    cur.execute("""
    CREATE TABLE IF NOT EXISTS ac_details (
        ac_no VARCHAR(20) PRIMARY KEY,
        c_name VARCHAR(100) NOT NULL,
```

```
    pin VARCHAR(10) NOT NULL,
    opening_balance DECIMAL(15, 2) DEFAULT 0
)
""")
```

- Creates a table `ac_details` for account details.
- Columns:
- `ac_no`: Account number, primary key, string up to 20 characters.
- `c_name`: Customer name, max 100 chars, required.
- `pin`: PIN code, string up to 10 chars, required.
- `opening_balance`: Account balance, decimal with 15 digits and 2 decimal places, default 0.

Python code:

```
cur.execute("""
CREATE TABLE IF NOT EXISTS transition_table (
    id INT AUTO_INCREMENT PRIMARY KEY,
    ac_no VARCHAR(20),
    type ENUM('deposit', 'withdraw', 'transfer_in', 'transfer_out', 'interest') NOT NULL,
    amount DECIMAL(15, 2) NOT NULL,
    date DATETIME NOT NULL,
    FOREIGN KEY (ac_no) REFERENCES ac_details(ac_no) ON DELETE CASCADE
)
""")
```

- Creates `transition_table` to log all transactions.
- Columns:
- `id`: Auto-incremented unique transaction ID (primary key).
- `ac_no`: Account number linked by foreign key to `ac_details`.
- `type`: Type of transaction (deposit, withdraw, transfer_in, transfer_out, interest).
- `amount`: Transaction amount.
- `date`: Timestamp of transaction.
- Foreign key ensures transactions are linked to accounts and get deleted if the account is deleted.

Python code:

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

- Commits changes and closes the connection.

Account Management

python code

```
def create_account():
    conn = connect_db()
    cur = conn.cursor()
    ac_no = input("Enter new account number: ")
    name = input("Enter account holder name: ")
```

```
    pin = input("Set a 4-digit PIN: ")
    bal = float(input("Enter opening balance: "))
    cur.execute("INSERT INTO ac_details VALUES (%s, %s, %s, %s)",
(ac_no, name, pin, bal))
    conn.commit()
    conn.close()
    print("✅ New account created successfully!")
```

- Prompts user for account details.
- Inserts new account into `ac_details`.
- Commits and closes.

Python code:

```
def close_account(ac_no):
    conn = connect_db()
    cur = conn.cursor()
    cur.execute("DELETE FROM ac_details WHERE ac_no=%s", (ac_no,))
    conn.commit()
    conn.close()
    print("✅ Account closed successfully!")
```

- Deletes the specified account from the database.
- Due to cascade, associated transactions get deleted.

Python code:

```
def login(ac_no, pin):
    conn = connect_db()
    cur = conn.cursor()
    cur.execute("SELECT * FROM ac_details WHERE ac_no=%s AND pin=%s", (ac_no, pin))
    account = cur.fetchone()
    conn.close()
    return account
```

- Checks if given account number and PIN match.
- Returns account details if valid, otherwise None.

Python code:

```
def change_pin(ac_no):
    conn = connect_db()
    cur = conn.cursor()
    current_pin = input("Enter current PIN: ")
    cur.execute("SELECT pin FROM ac_details WHERE ac_no=%s", (ac_no,))
    result = cur.fetchone()
    if result and result[0] == current_pin:
        new_pin = input("Enter new 4-digit PIN: ")
        confirm_pin = input("Confirm new PIN: ")
        if new_pin == confirm_pin:
```

```
                cur.execute("UPDATE ac_details SET pin=%s WHERE ac_no=%s",
(new_pin, ac_no))
                conn.commit()
                print("✅ PIN changed successfully.")
            else:
                print("❌ PINs do not match.")
        else:
            print("❌ Incorrect current PIN.")
        conn.close()
```

- Verifies current PIN.
- If correct, lets user set and confirm a new PIN.
- Updates PIN in the database if confirmed.

Transactions

Python code:

```
def check_balance(ac_no):
    conn = connect_db()
    cur = conn.cursor()
    cur.execute("SELECT opening_balance FROM ac_details WHERE
ac_no=%s", (ac_no,))
    bal = cur.fetchone()
    conn.close()
    print(f"💰 Current Balance: {bal[0]}" if bal else "Account not found.")
```

- Gets and prints current balance of the account.

Python code:

```
def deposit(ac_no, amount, t_type="deposit"):
    conn = connect_db()
    cur = conn.cursor()
    cur.execute("UPDATE ac_details SET opening_balance=opening_balance+%s WHERE ac_no=%s", (amount, ac_no))
    cur.execute("INSERT INTO transition_table (ac_no, type, amount, date) VALUES (%s, %s, %s, %s)",
                (ac_no, t_type, amount, datetime.now()))
    conn.commit()
    conn.close()
    print(f"□ {amount} deposited successfully.")
```

- Adds amount to balance.
- Logs the deposit transaction.
- Supports custom type (default "deposit" but also used for interest).

Python code:

```
def withdraw(ac_no, amount, daily_limit=20000):
    conn = connect_db()
    cur = conn.cursor()
    today = date.today()
    cur.execute("""
      SELECT SUM(amount) FROM transition_table
```

```
            WHERE ac_no=%s AND type='withdraw' AND DATE(date)=%s
        """, (ac_no, today))
        total_today = cur.fetchone()[0] or 0

        if total_today + amount > daily_limit:
            print(f"⚠ Daily withdrawal limit of {daily_limit} exceeded.")
            conn.close()
            return

        cur.execute("SELECT opening_balance FROM ac_details WHERE ac_no=%s", (ac_no,))
        bal = cur.fetchone()
        if not bal or bal[0] < amount:
            print("❌ Insufficient funds.")
        else:
            cur.execute("UPDATE ac_details SET opening_balance=opening_balance-%s WHERE ac_no=%s", (amount, ac_no))
            cur.execute("INSERT INTO transition_table (ac_no, type, amount, date) VALUES (%s, 'withdraw', %s, %s)",
                        (ac_no, amount, datetime.now()))
            conn.commit()
            print(f"✅ {amount} withdrawn successfully.")
        conn.close()
```

- Checks daily withdrawal total to enforce daily limit.
- Checks if balance is sufficient.
- Deducts amount and logs withdrawal if allowed.

Python code:

```
def transfer_funds(sender, receiver, amount):
    conn = connect_db()
    cur = conn.cursor()
    cur.execute("SELECT opening_balance FROM ac_details WHERE ac_no=%s", (sender,))
    bal = cur.fetchone()
    if not bal or bal[0] < amount:
        print("⚠ Insufficient funds.")
        conn.close()
        return
    cur.execute("UPDATE ac_details SET opening_balance=opening_balance-%s WHERE ac_no=%s", (amount, sender))
    cur.execute("UPDATE ac_details SET opening_balance=opening_balance+%s WHERE ac_no=%s", (amount, receiver))
    cur.execute("INSERT INTO transition_table (ac_no, type, amount, date) VALUES (%s, 'transfer_out', %s, %s)",
                (sender, amount, datetime.now()))
    cur.execute("INSERT INTO transition_table (ac_no, type, amount, date) VALUES (%s, 'transfer_in', %s, %s)",
                (receiver, amount, datetime.now()))
    conn.commit()
    conn.close()
    print(f"✅ {amount} transferred successfully.")
```

- Checks sender's balance for sufficient funds.

- Deducts amount from sender, adds to receiver.
- Logs two transactions: transfer out and transfer in.

Python code:

```
def credit_interest(ac_no, rate=2.5):
    conn = connect_db()
    cur = conn.cursor()
    cur.execute("SELECT opening_balance FROM ac_details WHERE ac_no=%s", (ac_no,))
    bal = cur.fetchone()
    if bal:
        interest = bal[0] * rate / 100
        deposit(ac_no, interest, "interest")
        print(f"□ Interest of {interest} credited.")
    else:
        print("□ Account not found.")
    conn.close()
```

- Calculates interest on current balance based on given rate (default 2.5%).
- Uses deposit function with type "interest" to add interest.

Python code:

```
def mini_statement(ac_no):
    conn = connect_db()
    cur = conn.cursor()
```

```
        cur.execute("SELECT type, amount, date FROM transition_table
WHERE ac_no=%s ORDER BY date DESC LIMIT 5", (ac_no,))
        rows = cur.fetchall()
        conn.close()
        print("\n Last 5 Transactions:")
        for r in rows:
            print(f"{r[2]} | {r[0]} | {r[1]}")
```

- Fetches and prints last 5 transactions for the account, most recent first.

Python code:

```
def transaction_by_date(ac_no, start, end):
    conn = connect_db()
    cur = conn.cursor()
    cur.execute("""
        SELECT type, amount, date FROM transition_table WHERE
ac_no=%s AND DATE(date) BETWEEN %s AND %s ORDER BY date DESC
    """, (ac_no, start, end))
    rows = cur.fetchall()
    conn.close()
    print(f"\n Transactions from {start} to {end}:")
    for r in rows:
        print(f"{r[2]} | {r[0]} | {r[1]}")
```

- Lists transactions between specified start and end dates

Python code:

```
def export_transactions(ac_no):
    conn = connect_db()
    cur = conn.cursor()
    cur.execute("SELECT * FROM transition_table WHERE ac_no=%s",
(ac_no,))
    rows = cur.fetchall()
    conn.close()
    filename = f"{ac_no}_transactions.csv"
    with open(filename, "w", newline="") as file:
        writer = csv.writer(file)
        writer.writerow(["ID", "Account No", "Type", "Amount", "Date"])
        writer.writerows(rows)
    print(f"□ Transactions exported to {filename}")
```

- Exports all transactions of the account to a CSV file named <account>_transactions.csv.

Admin Functions

Python code:

```
def admin_login():
    username = input("Admin username: ")
    password = input("Admin password: ")
    return username == "admin" and password == "admin123"
```

- Simple admin login with fixed credentials.

Python code:

```
def view_all_accounts():
    conn = connect_db()
    cur = conn.cursor()
    cur.execute("SELECT * FROM ac_details")
    for row in cur.fetchall():
        print(row)
    conn.close()
```

- Lists all account details (admin use).

Python code:

```
def view_all_transactions():
    conn = connect_db()
    cur = conn.cursor()
    cur.execute("SELECT * FROM transition_table ORDER BY date DESC")
    for row in cur.fetchall():
        print(row)
    conn.close()
```

- Lists all transactions sorted by most recent.

ATM Menu and Main Flow

Python code:

```
def atm():
    while True:
        print("\n===== ATM System =====")
        print("1. Login & Use ATM")
```

```
print("2. Create New Account")
print("3. Admin Panel")
print("4. Exit")
choice = input("Enter choice: ")
```

- Starts the main menu in a loop.

Python code:

```
if choice == "1":
    ac = input("Enter Account Number: ")
    pin = input("Enter PIN: ")
    if not login(ac, pin):
        print("□ Invalid login.")
        continue
```

- Option to login; continues if login fails.

Python code:

```
while True:
    print("\n1. Check Balance")
    print("2. Deposit")
    print("3. Withdraw")
    print("4. Mini Statement")
    print("5. Transfer Funds")
    print("6. Credit Interest")
    print("7. Change PIN")
    print("8. Search Transactions by Date")
    print("9. Export Transactions to CSV")
```

```
print("10. Close Account")
print("11. Logout")
op = input("Enter choice: ")
```

- After login, offers user transaction options.

Python code:

```
if op == "1": check_balance(ac)
elif op == "2": deposit(ac, float(input("Amount: ")))
elif op == "3": withdraw(ac, float(input("Amount: ")))
elif op == "4": mini_statement(ac)
elif op == "5": transfer_funds(ac, input("Receiver: "), float(input("Amount: ")))
elif op == "6": credit_interest(ac)
elif op == "7": change_pin(ac)
elif op == "8": transaction_by_date(ac, input("Start date (YYYY-MM-DD): "), input("End date (YYYY-MM-DD): "))
elif op == "9": export_transactions(ac)
elif op == "10": close_account(ac); break
elif op == "11": break
else: print("⚠ Invalid choice.")
```

- Executes the selected operation.
- Option 10 closes account and breaks inner loop (logs user out).
- Option 11 logs user out.

Python code:

```
elif choice == "2":
    create_account()
```

- Option to create a new account.

Python code:

```
elif choice == "3":
    if admin_login():
        while True:
            print("\n*** ADMIN PANEL ***")
            print("1. View All Accounts")
            print("2. View All Transactions")
            print("3. Back")
            admin_choice = input("Enter choice: ")
            if admin_choice == "1": view_all_accounts()
            elif admin_choice == "2": view_all_transactions()
            elif admin_choice == "3": break
            else: print("❌ Invalid choice.")
    else:
        print("❌ Access Denied.")
```

- Admin panel with options to view all accounts or all transactions.
- Requires correct admin login.

Python code:

```
elif choice == "4":
    print("👋 Goodbye!")
    break
else:
    print("❌ Invalid choice.")
```

- Option 4 exits the program.
- Any other input prompts invalid choice message.
- Option 4 exits the program.
- Any other input prompts invalid choice message.
- Program Entry Point

Python code:

```
if __name__ == "__main__":
    create_database()
    create_tables()
    atm()
```

- Creates database and tables if needed.
- Starts the ATM menu loop.