**19CS44C - DATABASE MANAGEMENT SYSTEM**

**MINI PROJECT**

**THE BANKING SYSTEM**

**BY**

DHANA KARTHICK S

ATHITHYA KV

SIVARAM V

MEHRAN MUSHTAQ WANI M

**Introduction**

The Banking System is a mini project developed to demonstrate the implementation of a simple database management system using SQLite and Python. The application allows users to manage bank accounts effectively.

**Objective**

The primary objective of this mini project is to understand the fundamental concepts of database connectivity and manipulation using SQLite in Python. By creating a banking application, we aim to demonstrate CRUD operations (Create, Read, Update, Delete) on a SQLite database.

**Technologies Used**

* **SQLite**: A lightweight relational database management system.
* **Python**: A high-level programming language.
* **SQLite3 module in Python**: A built-in module for working with SQLite databases in Python.

**Features**

* **Create Account**: Users can create new bank accounts.
* **View Accounts**: Users can view all bank accounts.
* **Update Account**: Users can update existing account information.
* **Delete Account**: Users can delete accounts.
* **Deposit Money**: Users can deposit money into their accounts.
* **Withdraw Money**: Users can withdraw money from their accounts.

**Implementation**

**Database Schema**

The SQLite database schema consists of a single table named **accounts** with the following columns:

* **id**: Primary key, auto-incremented integer.
* **name**: Name of the account holder (text).
* **balance**: Balance of the account (real).

**Python Script**

Below is the Python script developed to interact with the SQLite database. The script includes functions to perform CRUD operations on the **accounts** table as well as deposit and withdraw functionalities.

**CODE**

import sqlite3

from sqlite3 import Error

def create\_connection(db\_file):

conn = None

try:

conn = sqlite3.connect(db\_file)

print(f"Connected to database {db\_file}")

except Error as e:

print(e)

return conn

def create\_account(conn, account):

sql = '''INSERT INTO accounts(name, balance) VALUES(?,?)'''

cur = conn.cursor()

cur.execute(sql, account)

conn.commit()

return cur.lastrowid

def get\_all\_accounts(conn):

cur = conn.cursor()

cur.execute("SELECT \* FROM accounts")

return cur.fetchall()

def update\_account(conn, account):

sql = '''UPDATE accounts SET name = ?, balance = ? WHERE id = ?'''

cur = conn.cursor()

cur.execute(sql, account)

conn.commit()

def delete\_account(conn, id):

sql = 'DELETE FROM accounts WHERE id=?'

cur = conn.cursor()

cur.execute(sql, (id,))

conn.commit()

def deposit\_money(conn, id, amount):

cur = conn.cursor()

cur.execute("UPDATE accounts SET balance = balance + ? WHERE id = ?", (amount, id))

conn.commit()

def withdraw\_money(conn, id, amount):

cur = conn.cursor()

cur.execute("SELECT balance FROM accounts WHERE id = ?", (id,))

if cur.fetchone()[0] >= amount:

cur.execute("UPDATE accounts SET balance = balance - ? WHERE id = ?", (amount, id))

conn.commit()

return True

return False

def main():

database = r"banking\_system.db"

conn = create\_connection(database)

with conn:

create\_table\_sql = """ CREATE TABLE IF NOT EXISTS accounts (

id integer PRIMARY KEY,

name text NOT NULL,

balance real); """

try:

c = conn.cursor()

c.execute(create\_table\_sql)

except Error as e:

print(e)

account = ('Alice', 1000.0)

account\_id = create\_account(conn, account)

print(f"Account created with id: {account\_id}")

deposit\_money(conn, account\_id, 500.0)

print(f"Deposited 500.0 into account id: {account\_id}")

success = withdraw\_money(conn, account\_id, 200.0)

print(f"Withdrew 200.0 from account id: {account\_id}" if success else f"Insufficient balance to withdraw from account id: {account\_id}")

print("All accounts:")

for row in get\_all\_accounts(conn):

print(row)

update\_account(conn, ('Alice Wonderland', 1500.0, account\_id))

print(f"Account with id {account\_id} updated")

delete\_account(conn, account\_id)

print(f"Account with id {account\_id} deleted")

print("All accounts after deletion:")

for row in get\_all\_accounts(conn):

print(row)

if \_\_name\_\_ == '\_\_main\_\_':

main()

**OUTPUT**

Connected to database banking\_system.db

Account created with id: 1

Deposited 500.0 into account id: 1

Withdrew 200.0 from account id: 1

All accounts:

(1, 'Alice', 1300.0)

Account with id 1 updated

Account with id 1 deleted

All accounts after deletion:

Empty.

## Demonstration

The Banking System application starts by connecting to the SQLite database and creating the **accounts** table. We create a new account for Alice with an initial balance of $1000. We then deposit $500 and withdraw $200, updating the balance accordingly. The application retrieves and displays all accounts, showing the updated information. Alice's account is updated to "Alice Wonderland" with a balance of $1500. Finally, the account is deleted, and the accounts list is displayed to confirm the deletion. Each step provides real-time feedback, demonstrating effective CRUD and transaction operations.

**Conclusion**

The Banking System successfully demonstrates the implementation of a simple database management system using SQLite and Python. By completing this mini project, we gained hands-on experience in database connectivity, CRUD operations, and Python programming.

**Future Enhancements**

* Implement additional features such as transaction history.
* Develop a graphical user interface (GUI) for the banking application.
* Enhance error handling and validation mechanisms.