‘’’ Henry Chan ‘’’

import sqlite3

from prettytable import PrettyTable

def connect\_db():

conn = sqlite3.connect(r'c:\sqlite\db\bank.db')

return conn

def list\_all\_branches(conn):

cursor = conn.cursor()

cursor.execute("SELECT \* FROM branch")

headers = [description[0] for description in cursor.description]

table = PrettyTable(headers)

rows = cursor.fetchall()

for row in rows:

table.add\_row(row)

print(table)

def count\_accounts\_greater\_than(conn, amount):

cursor = conn.cursor()

cursor.execute("SELECT COUNT(\*) FROM account WHERE balance > ?", (amount,))

count = cursor.fetchone()[0]

print(f'Number of accounts with balance greater than {amount}: {count}')

def find\_customers\_with\_deposit(conn):

cursor = conn.cursor()

cursor.execute("""

SELECT customer.customer\_name, customer.customer\_street, customer.customer\_city,

depositor.account\_number, account.balance

FROM customer

JOIN depositor ON customer.customer\_name = depositor.customer\_name

JOIN account ON depositor.account\_number = account.account\_number

""")

headers = [description[0] for description in cursor.description]

table = PrettyTable(headers)

rows = cursor.fetchall()

for row in rows:

table.add\_row(row)

print(table)

def add\_new\_customer(conn, customer\_name, customer\_street, customer\_city):

cursor = conn.cursor()

cursor.execute("""

INSERT INTO customer (customer\_name, customer\_street, customer\_city)

VALUES (?, ?, ?)

""", (customer\_name, customer\_street, customer\_city))

conn.commit()

print(f"New customer {customer\_name} added successfully.")

def add\_new\_account(conn, account\_number, balance):

cursor = conn.cursor()

cursor.execute("""

INSERT INTO account (account\_number, balance)

VALUES (?, ?)

""", (account\_number, balance))

conn.commit()

print(f"New account {account\_number} with balance {balance} added successfully.")

def link\_customer\_account(conn, customer\_name, account\_number):

cursor = conn.cursor()

cursor.execute("""

INSERT INTO depositor (customer\_name, account\_number)

VALUES (?, ?)

""", (customer\_name, account\_number))

conn.commit()

print(f"Linked customer {customer\_name} to account {account\_number}.")

def update\_customer\_address(conn, customer\_name, new\_street, new\_city):

cursor = conn.cursor()

cursor.execute("""

UPDATE customer

SET customer\_street = ?, customer\_city = ?

WHERE customer\_name = ?

""", (new\_street, new\_city, customer\_name))

conn.commit()

print(f"Address for customer {customer\_name} updated successfully.")

def make\_deposit(conn, customer\_name, account\_number, deposit\_amount):

cursor = conn.cursor()

cursor.execute("""

UPDATE account

SET balance = balance + ?

WHERE account\_number = ? AND account\_number IN (

SELECT account\_number FROM depositor WHERE customer\_name = ?

)

""", (deposit\_amount, account\_number, customer\_name))

print(f"{cursor.rowcount} rows updated.")

conn.commit()

print(f"Deposit of {deposit\_amount} made to account {account\_number} of customer {customer\_name}.")

def list\_all\_customers(conn):

cursor = conn.cursor()

cursor.execute("SELECT \* FROM customer")

headers = [description[0] for description in cursor.description]

table = PrettyTable(headers)

rows = cursor.fetchall()

for row in rows:

table.add\_row(row)

print(table)

def main():

conn = connect\_db()

while True:

print("""

1. List all branches

2. Count accounts with balance greater than specific amount

3. Find customers with deposit

4. Add new customer

5. Update customer address

6. Make a deposit

7. List all customers

0. Exit

""")

choice = input("Enter your choice: ")

if choice == '1':

list\_all\_branches(conn)

elif choice == '2':

amount = int(input("Enter the amount: "))

count\_accounts\_greater\_than(conn, amount)

elif choice == '3':

find\_customers\_with\_deposit(conn)

elif choice == '4':

name = input("Enter customer name: ")

street = input("Enter customer street: ")

city = input("Enter customer city: ")

add\_new\_customer(conn, name, street, city)

account\_number = input("Enter new account number: ")

balance = float(input("Enter initial balance: "))

add\_new\_account(conn, account\_number, balance)

link\_customer\_account(conn, name, account\_number)

elif choice == '5':

name = input("Enter customer name: ")

new\_street = input("Enter new street: ")

new\_city = input("Enter new city: ")

update\_customer\_address(conn, name, new\_street, new\_city)

elif choice == '6':

name = input("Enter customer name: ")

account\_number = input("Enter account number: ")

deposit\_amount = float(input("Enter deposit amount: "))

make\_deposit(conn, name, account\_number, deposit\_amount)

elif choice == '7':

list\_all\_customers(conn)

elif choice == '0':

break

else:

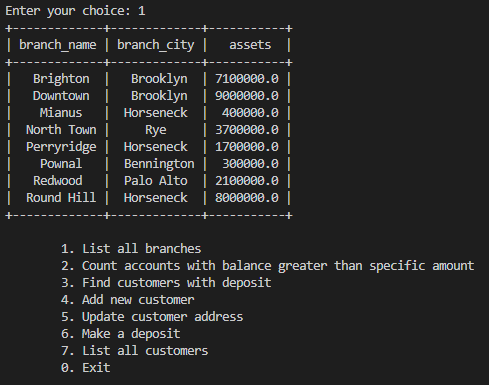
print("Invalid choice. Please choose a valid option.")

if \_\_name\_\_ == "\_\_main\_\_":

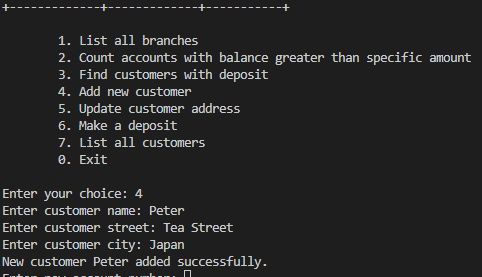
main()

The above Python code is a banking system that allows users to perform various operations such as listing all branches of the bank, counting accounts with balance greater than a specific amount, finding customers with deposits, adding new customers to the bank's database, updating customer addresses in the database and making deposits. The code uses a database connection to store and retrieve data from the bank's database.

The main function of the code contains a while loop that displays a menu of options for the user to choose from. Depending on the user's selection, different functions are called to perform specific operations. For example, if the user selects option 1 from the menu, then the function list\_all\_branches is called which retrieves all branch information from the database and displays it in a table format using PrettyTable library.



Similarly, if the user selects option 4 from the menu to add a new customer to the bank's database then they are prompted to enter their name and address which is then added to the database using add\_new\_customer function.



Overall this Python code provides an efficient way for banks to manage their customers' accounts and transactions by automating various banking operations.

