Infosys Campus Connect Project - Modules 4 & 5

Name of the Student: Nelapudi Ajay Raj
Name of the Project: Bank Application

Problem Statement:

We are aware of banking services and would have done some transactions like deposit, withdrawal, transfer of funds and so on. A bank typically has large number of customers and maintains the details of each customer i.e. what is the name of my customer, his address, account number, how much money is there in his account etc. We also hear about multiple types of accounts like Saving account, Current account. The project is to develop a banking application for a bank that has multiple customers spread across the country.

Development Platform:

Programming Language: Python 3.6.2
Development Tool: PyCharm Community Edition 2017.2.3
Database: MySQL 10.1.37
Operating System: Mac OS X Sierra 10.12.5

Project Development:

The first step taken is to decide on the database schema. All of given operations and specifications are considered and the database is designed. The database is written and then normalised. The tables are written as .sql script and then the database is imported or executed to create the database in the tables.

The next task is to write a class. The approach is to separate the database functions from regular code. This class connects to the database using pymysql python library and then the functions written in the class perform various queries. Additionally, when reports were to be added in module 5, the only change of code is to write new functions and make function calls in the appropriate database class. This clearly states that the code is very well managed.

Now, the menu is developed by following the guidelines in the material provided under Foundation Program 5.0. The menu is printed and the task is performed by separate functions. When functionality needs to be added, making the appropriate function calls would make the code meet the specification.

The ids' of all database entries are generated by another class which takes care of generating customer id, account no and loan no. These are uniquely generated ids.

The code is written in separate files, imported to the menu.py file and necessary calls to objects and functions are made to develop the application. Coding standards are followed such as naming the variables and functions to its near exact meaning and following the underscore notation since the code is in python.

Screenshots of Project Development:

Database Queries

createdb.sql — CarRentals

UNREGISTERED

OPEN FILES

createdb.sql

FOLDERS

CarRentals

```sql
CREATE DATABASE Bank;

USE Bank;

CREATE TABLE Customer (
    c_id INTEGER PRIMARY KEY,
    c_password VARCHAR(20) NOT NULL,
    c_name VARCHAR(50) NOT NULL,
    c_addr VARCHAR(100) NOT NULL
);

CREATE TABLE Account (
    acc_no INTEGER PRIMARY KEY,
    acc_balance BIGINT NOT NULL,
    acc_type VARCHAR(10) NOT NULL
);

CREATE TABLE CustomerAccount (
    c_id INTEGER NOT NULL,
    acc_no INTEGER NOT NULL
);

ALTER TABLE CustomerAccount
ADD CONSTRAINT c_id_CustomerAccount FOREIGN KEY(c_id) REFERENCES Customer(c_id),
ADD CONSTRAINT acc_no_CustomerAccount FOREIGN KEY(acc_no) REFERENCES Account(acc_no);

CREATE TABLE AccountTransaction (
    acc_no INTEGER NOT NULL,
    t_type VARCHAR(10) NOT NULL,
    t_amount BIGINT NOT NULL,
    t_timestamp TIMESTAMP NOT NULL
);
```

Line 59, Column 12 master 6 Tab Size: 4 SQL

createdb.sql — CarRentals

UNREGISTERED

OPEN FILES

createdb.sql

FOLDERS

CarRentals

```sql
    t_type VARCHAR(10) NOT NULL,
    t_amount BIGINT NOT NULL,
    t_timestamp TIMESTAMP NOT NULL
);

ALTER TABLE AccountTransaction
ADD CONSTRAINT acc_no_AccountTransaction FOREIGN KEY(acc_no) REFERENCES Account(acc_no);

CREATE TABLE Loan (
    loan_id INTEGER PRIMARY KEY,
    loan_acc_no INTEGER NOT NULL,
    loan_amount INTEGER NOT NULL,
    loan_term INTEGER NOT NULL
);

ALTER TABLE Loan
ADD CONSTRAINT loan_acc_no_Loan FOREIGN KEY(loan_acc_no) REFERENCES Account(acc_no);

CREATE TABLE Administrator (
    admin_id INTEGER PRIMARY KEY,
    admin_password VARCHAR(20) NOT NULL
);

CREATE TABLE ClosedAccount (
    acc_no INTEGER PRIMARY KEY,
    closing_date TIMESTAMP NOT NULL
);

INSERT INTO Administrator
VALUES
(1, 'key');
```

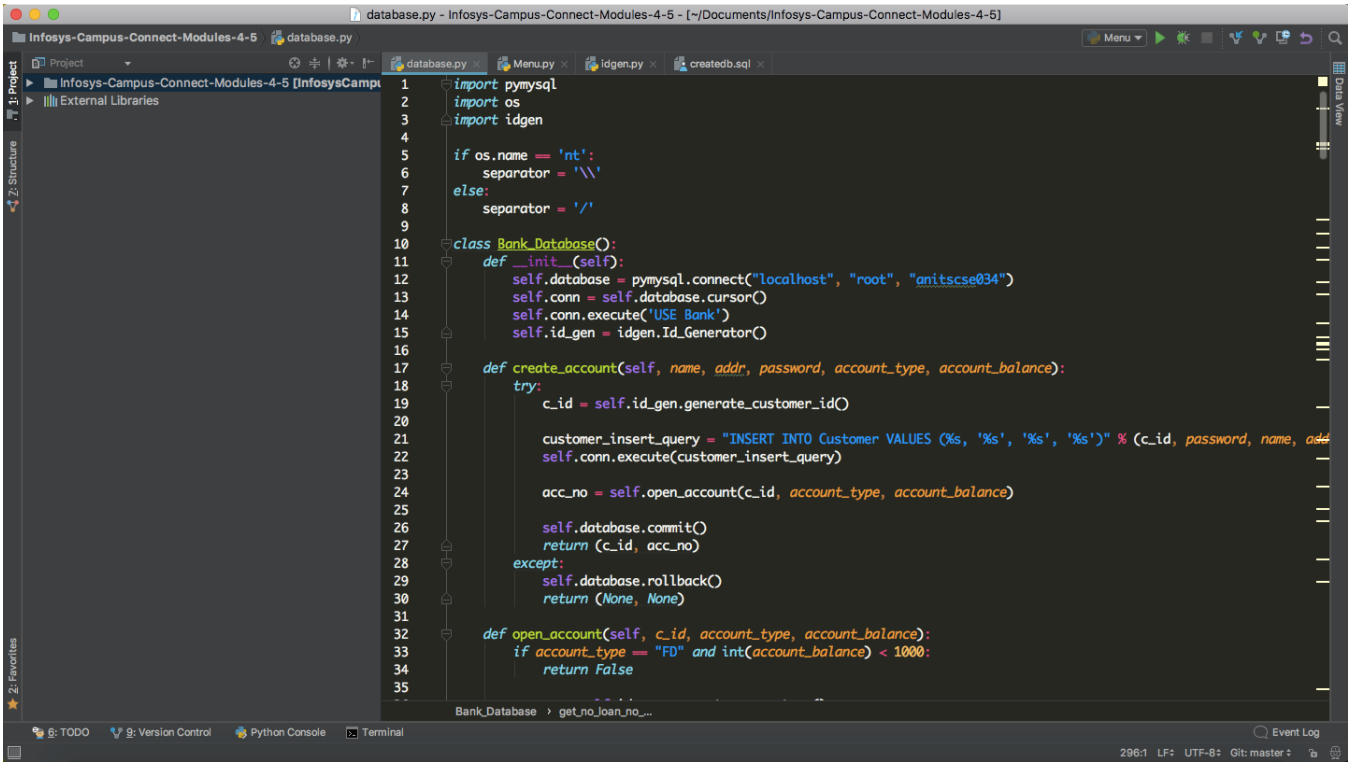
Line 59, Column 12 master 7 Tab Size: 4 SQL

```
ajayraj — mysql -uroot -p — 80×24
[ajayrajs-MacBook-Air:~ Ajay$ mysql -uroot -p
[Enter password:
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 9
Server version: 10.1.37-MariaDB Source distribution

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

[MariaDB [(none)]> SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| Bookaholic         |
| CarRentals         |
| CodeDuel           |
| Student            |
| information_schema |
| mysql              |
| performance_schema |
| phpmyadmin         |
| test               |
+--------------------+
```

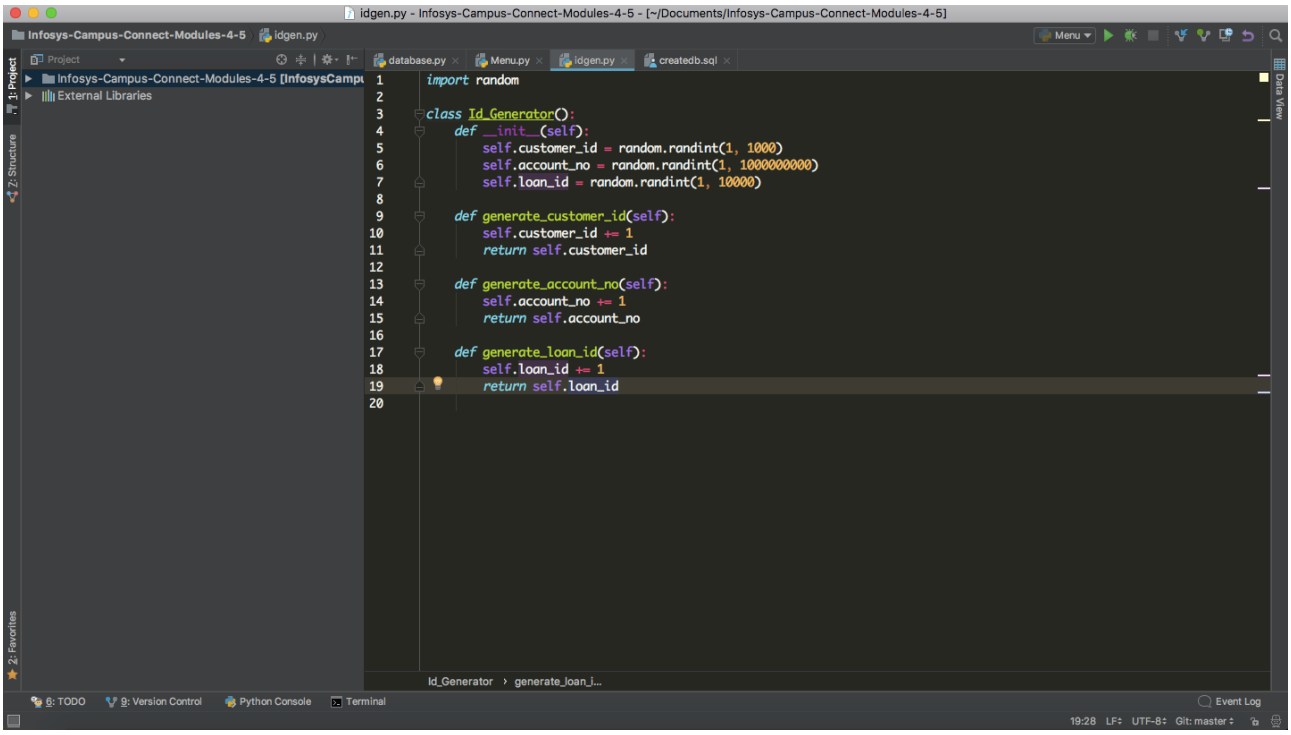
```
ajayraj — mysql -uroot -p — 80×24
[MariaDB [(none)]> SOURCE /Users/ajayraj/Documents/Infosys-Campus-Connect-Modules
-4-5/createdb.sql
Query OK, 1 row affected (0.00 sec)

Database changed
Query OK, 0 rows affected (0.03 sec)

Query OK, 0 rows affected (0.02 sec)

Query OK, 0 rows affected (0.02 sec)

Query OK, 0 rows affected (0.04 sec)
Records: 0  Duplicates: 0  Warnings: 0

Query OK, 0 rows affected (0.03 sec)

Query OK, 0 rows affected (0.04 sec)
Records: 0  Duplicates: 0  Warnings: 0

Query OK, 0 rows affected (0.02 sec)

Query OK, 0 rows affected (0.02 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

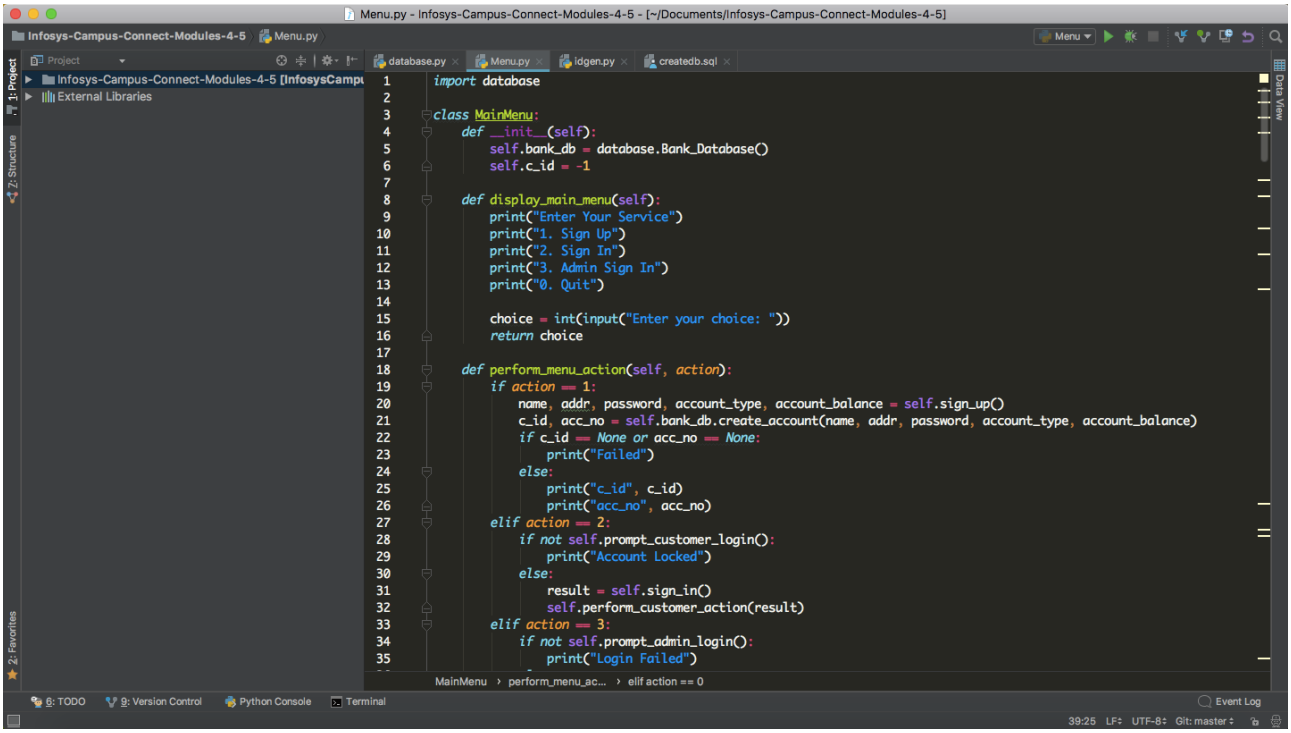
```
ajayraj — mysql -uroot -p — 80×24
Query OK, 0 rows affected (0.02 sec)
Records: 0  Duplicates: 0  Warnings: 0

Query OK, 0 rows affected (0.02 sec)

Query OK, 0 rows affected (0.02 sec)

Query OK, 1 row affected (0.00 sec)

MariaDB [Bank]> SHOW TABLES;
+----------------------+
| Tables_in_Bank       |
+----------------------+
| Account              |
| AccountTransaction   |
| Administrator        |
| ClosedAccount        |
| Customer             |
| CustomerAccount      |
| Loan                 |
+----------------------+
7 rows in set (0.01 sec)

MariaDB [Bank]>
```

Code Snapshots



Code Snapshots





Source Code:

```
import pymysql
import os
import idgen

if os.name == 'nt':
    separator = '\\'
else:
    separator = '/'

class Bank_Database():
    def __init__(self):
        self.database = pymysql.connect("localhost", "root", "anitscse034")
        self.conn = self.database.cursor()
        self.conn.execute('USE Bank')
        self.id_gen = idgen.Id_Generator()

           def   create_account(self,   name,   addr,   password,   account_type,
account_balance):
        try:
            c_id = self.id_gen.generate_customer_id()

               customer_insert_query = "INSERT INTO Customer VALUES (%s, '%s',
'%s', '%s')" % (c_id, password, name, addr)
            self.conn.execute(customer_insert_query)

            acc_no = self.open_account(c_id, account_type, account_balance)

            self.database.commit()
            return (c_id, acc_no)
        except:
            self.database.rollback()
            return (None, None)

    def open_account(self, c_id, account_type, account_balance):
        if account_type == "FD" and int(account_balance) < 1000:
            return False

        acc_no = self.id_gen.generate_account_no()

          account_insert_query = "INSERT INTO Account VALUES (%s, %s, '%s')" %
(acc_no, account_balance, account_type)
        self.conn.execute(account_insert_query)
```

```
        customer_account_insert_query = "INSERT INTO CustomerAccount VALUES (%s,
%s)" % (c_id, acc_no)
        self.conn.execute(customer_account_insert_query)

        return acc_no

    def validate_login(self, c_id, password):
        query = "SELECT * FROM Customer WHERE c_id = %s AND c_password = '%s'" %
(c_id, password)
        if self.conn.execute(query) == 1:
            return True
        return False

    def validate_admin_login(self, admin_id, admin_password):
             query = "SELECT * FROM Administrator WHERE admin_id = %s AND
admin_password = '%s'" % (admin_id, admin_password)
        if self.conn.execute(query) == 1:
            return True
        return False

    def change_address(self, c_id, new_addr):
        try:
                query = "UPDATE Customer SET c_addr = '%s' WHERE c_id = %s" %
(new_addr, c_id)
            self.conn.execute(query)

            self.database.commit()
            return True
        except:
            self.database.rollback()
            return False

    def deposit_money(self, acc_no, amount, commit_on_success=True):
        try:
             deposit_query = "UPDATE Account SET acc_balance = acc_balance + %s
WHERE acc_no = %s" % (amount, acc_no)
             transaction_log_query = "INSERT INTO AccountTransaction VALUES (%s,
'deposit', %s, NOW())" % (acc_no, amount)
            self.conn.execute(deposit_query)
            self.conn.execute(transaction_log_query)

            if commit_on_success:
                self.database.commit()
```

```
            return True
        except:
            self.database.rollback()
            return False

    def withdraw_money(self, acc_no, amount, commit_on_success=True):
        try:
                enquire_balance_query = "SELECT acc_balance FROM Account WHERE
acc_no = %s" % (acc_no)
            self.conn.execute(enquire_balance_query)
            acc_balance = self.conn.fetchone()[0]
            if acc_balance - int(amount) > 500:
                acc_balance = acc_balance - int(amount)
            else:
                return False

            withdraw_query = "UPDATE Account SET acc_balance = %s WHERE acc_no =
%s" % (acc_balance, acc_no)
            self.conn.execute(withdraw_query)

             transaction_log_query = "INSERT INTO AccountTransaction VALUES (%s,
'withdrawl', %s, NOW())" % (acc_no, amount)
            self.conn.execute(transaction_log_query)

            if commit_on_success:
                self.database.commit()
            return True
        except:
            self.database.rollback()
            return False

    def transfer_money(self, src_acc_no, dest_acc_no, amount):
                  withdraw_status  =  self.withdraw_money(src_acc_no,  amount,
commit_on_success=False)
                   deposit_status  =  self.deposit_money(dest_acc_no,  amount,
commit_on_success=False)

        if withdraw_status and deposit_status:
            self.database.commit()
            return True
        return False

    def get_statement(self, acc_no):
        query = "SELECT * FROM AccountTransaction WHERE acc_no = %s" % (acc_no)
```

```
        self.conn.execute(query)
        statement = self.conn.fetchall()
        return statement

    def close_account(self, acc_no):
        try:
             delete_customer_account_query = "DELETE FROM CustomerAccount WHERE
acc_no = %s" % (acc_no)
            self.conn.execute(delete_customer_account_query)

             #delete_account_query = "DELETE FROM Account WHERE acc_no = %s" %
(acc_no)
            #self.conn.execute(delete_account_query)

            insert_closed_account_query = "INSERT INTO ClosedAccount VALUES (%s,
NOW())" % (acc_no)
            self.conn.execute(insert_closed_account_query)

            self.database.commit()
            return True
        except:
            self.database.rollback()
            return False

    def is_associated_account(self, c_id, acc_no):
         query = "SELECT * FROM CustomerAccount WHERE c_id = %s AND acc_no = %s"
% (c_id, acc_no)
        status = self.conn.execute(query)
        if status == 1:
            return True
        return False

    def get_closed_accounts(self):
        query = '''
                SELECT A.*
                FROM ClosedAccount CA, Account A
                WHERE A.acc_no = CA.acc_no
                ORDER BY CA.closing_date
        '''
        self.conn.execute(query)
        closed_accounts = self.conn.fetchall()
        return closed_accounts

    def avail_loan(self, acc_no, loan_amount):
```

```
        try:
             loan_amount_query = "SELECT acc_balance FROM Account WHERE acc_no =
%s AND acc_type = 'SA'" % (acc_no)
            self.conn.execute(loan_amount_query)
            max_loan_amount = self.conn.fetchone()[0] * 2

            if int(loan_amount) > max_loan_amount:
                raise ValueError()

            loan_id = self.id_gen.generate_loan_id()
                 loan_insert_query = "INSERT INTO Loan VALUES (%s, %s, %s)" %
(loan_id, acc_no, loan_amount)
            self.conn.execute(loan_insert_query)

            if not self.deposit_money(acc_no, loan_amount):
                raise ValueError()

            self.database.commit()
            return True
        except:
            self.database.rollback()
            return False

    def get_fd_report(self, c_id):
        query = '''SELECT A.*
                   FROM Account A, CustomerAccount CA
                    WHERE A.acc_no = CA.acc_no AND A.acc_type = 'FD' AND CA.c_id
= %s

        ''' % (c_id)
        self.conn.execute(query)
        report = self.conn.fetchall()

        return report

    def get_fd_report_vis_a_vis_customer(self, c_id):
        query = '''
                SELECT C.c_id, A.*
                FROM CustomerAccount CA, Account A
                WHERE CA.acc_no = A.acc_no AND A.acc_type = 'FD'
                AND A.acc_balance > (SELECT SUM(__A__.acc_balance)
                                     FROM Account __A__, CustomerAccount __CA__
                                              WHERE __A__.acc_type = 'FD' AND
__CA__.acc_no = __A__.acc_no AND __CA__.c_id = %s)
```

```
        ''' % (c_id)
        self.conn.execute(query)
        report = self.conn.fetchall()

        return report

    def get_fd_report_wrt_amount(self, amount):
        query = '''
                SELECT C.c_id, C.c_name, A.acc_balance
                FROM Customer C, Account A, CustomerAccount CA
                        WHERE C.c_id = CA.c_id AND CA.acc_no = A.acc_no AND
A.acc_balance > %s
        ''' % (amount)
        self.conn.execute(query)
        report = self.conn.fetchall()

        return report

    def get_loan_report_of_customer(self, c_id):
        query = '''
                SELECT CA.c_id, L.*
                FROM CustomerAccount CA, Loan L
                 WHERE CA.c_id = %s AND CA.acc_no = A.acc_no AND L.loan_acc_no =
A.acc_no
        '''
        self.conn.execute(query)
        report = self.conn.fetchall()

        return report

    def get_loan_report_vis_a_vis_customer(self, c_id):
        query = '''
                SELECT CA.c_id, L.*
                FROM Account A, CustomerAccount CA, Loan L
                WHERE CA.acc_no = L.loan_acc_no
                AND L.loan_amount > (SELECT SUM(__L__.loan_amount)
                                     FROM Loan __L__, CustomerAccount __CA__
                                      WHERE __CA__.c_id = %s __L__.loan_acc_no =
__CA__.acc_no)
        ''' % (c_id)
        self.conn.execute(query)
        report = self.conn.fetchall()

        return report
```

```
    def get_loan_report_wrt_amount(self, amount):
        query = '''
                SELECT C.c_id, C.c_name, L.loan_amount
                FROM Customer C, Loan L, CustomerAccount CA
                      WHERE C.c_id = CA.c_id AND CA.acc_no = L.loan_acc_no AND
L.loan_amount > %s
        ''' % (amount)
        self.conn.execute(query)
        report = self.conn.fetchall()

        return report

    def get_loan_fd_report(self):
        query = '''
                SELECT C.c_id, C.c_name, SUM(L.loan_amount), SUM(A.acc_balance)
                FROM Customer C, Account A, Loan L, CustomerAccount CA
                WHERE C.c_id = CA.c_id AND CA.acc_no = A.acc_no and A.acc_type =
'FD'
                AND SUM(L.loan_amount) > SUM(A.acc_balance)
        '''
        self.conn.execute(query)
        report = self.conn.fetchall()

        return report

    def get_no_loan_customers(self):
        query = '''
                SELECT C.c_id, C.c_name
                FROM Customer C, Account A, CustomerAccount CA, Loan L
                WHERE C.c_id = CA.c_id
                AND CA.acc_no NOT IN (SELECT L.loan_acc_no FROM Loan L)
        '''
        self.conn.execute(query)
        report = self.conn.fetchall()

        return report

    def get_no_fd_acc_customers(self):
        query = '''
                SELECT C.c_id, C.c_name
                FROM Customer C, Account A, CustomerAccount CA
                WHERE C.c_id = CA.C_id
                AND CA.acc_no NOT IN (SELECT __A__.acc_no FROM Account __A__
```

```
                                    WHERE __A__.acc_type = 'FD')
        '''
        self.conn.execute(query)
        report = self.conn.fetchall()

        return report

    def get_no_loan_no_fd_customers(self):
        query = '''
                SELECT __C__.c_id
                FROM Customer __C__, CustomerAccount __CA__, Account __A__, Loan
__L__
                WHERE __C__.c_id = __CA__.c_id AND __CA__.acc_no = __A__.acc_no
                    AND __A__.acc_type != 'FD' AND __A__.acc_no NOT IN (SELECT
L.loan_acc_no FROM Loan L)
        '''
        self.conn.execute(query)
        report = self.conn.fetchall()

        return report
```

```
import random

class Id_Generator():
    def __init__(self):
        self.customer_id = random.randint(1, 1000)
        self.account_no = random.randint(1, 1000000000)
        self.loan_id = random.randint(1, 10000)

    def generate_customer_id(self):
        self.customer_id += 1
        return self.customer_id

    def generate_account_no(self):
        self.account_no += 1
        return self.account_no

    def generate_loan_id(self):
        self.loan_id += 1
        return self.loan_id
```

```
import database

class MainMenu:
    def __init__(self):
        self.bank_db = database.Bank_Database()
        self.c_id = -1

    def display_main_menu(self):
        print("Enter Your Service")
        print("1. Sign Up")
        print("2. Sign In")
        print("3. Admin Sign In")
        print("0. Quit")

        choice = int(input("Enter your choice: "))
        return choice

    def perform_menu_action(self, action):
        if action == 1:
            name, addr, password, account_type, account_balance = self.sign_up()
               c_id, acc_no = self.bank_db.create_account(name, addr, password,
account_type, account_balance)
            if c_id == None or acc_no == None:
                print("Failed")
            else:
                print("c_id", c_id)
                print("acc_no", acc_no)
        elif action == 2:
            if not self.prompt_customer_login():
                print("Account Locked")
            else:
                result = self.sign_in()
                self.perform_customer_action(result)
        elif action == 3:
            if not self.prompt_admin_login():
                print("Login Failed")
            else:
                result = self.admin_sign_in()
                self.perform_admin_action(result)
        elif action == 0:
            exit()
        else:
            print("Enter proper choice")
```

```
    def sign_up(self):
        name = input("Enter your name: ")
        password = input("Enter password: ")
        addr = input("Enter your address: ")
        account_balance = input("Enter amount to deposit: ")
        account_type = input("Enter account type (CA / SA / FD): ")

        return (name, addr, password, account_type, account_balance)

    def prompt_customer_login(self):
        for i in range(3):
            c_id = input("Enter customer id: ")
            password = input("Enter password: ")

            if self.bank_db.validate_login(c_id, password):
                self.c_id = c_id
                return True
            else:
                print(3 - (i + 1), 'attempts left')
        else:
            return False

    def sign_in(self):
        print("Select Customer Service")
        print("1. Address Change")
        print("2. Create Account")
        print("3. Money Deposit")
        print("4. Money Widthdrawal")
        print("5. Print Statement")
        print("6. Transfer Money")
        print("7. Account Closure")
        print("8. Avail Loan")
        print("0. Logout")

        choice = int(input())
        return choice

    def perform_customer_action(self, action):
        while True:
            if action == 1:
                new_addr = input("Enter new address: ")
                if self.bank_db.change_address(self.c_id, new_addr):
                    print("Success")
                else:
```

```
                print("Failed")
            elif action == 2:
                print("1. Open Savings Account")
                print("2. Open Current Account")
                print("3. Open Fixed Deposit Account")

                choice = int(input("Enter your choice: "))

                accounts = {1: "SA", 2: "CA", 3: "FD"}
                account_type = accounts[choice]

                accont_balance = input("Enter amount: ")

                    acc_no = self.bank_db.open_account(self.c_id, account_type,
accont_balance)
                if acc_no:
                    print("Success")
                    print("Account No:", acc_no)
                else:
                    print("Failed")

            elif action == 3:
                acc_no = input("Enter account no: ")

                if not self.bank_db.is_associated_account(self.c_id, acc_no):
                    print("Not associated account")
                    continue

                amount = input("Enter amount: ")
                if self.bank_db.deposit_money(acc_no, amount):
                    print("Success")
                else:
                    print("Failed")
            elif action == 4:
                acc_no = input("Enter account no: ")

                if not self.bank_db.is_associated_account(self.c_id, acc_no):
                    print("Not associated account")
                    continue

                amount = input("Enter amount: ")
                if self.bank_db.withdraw_money(acc_no, amount):
                    print("Success")
                else:
```

```
            print("Failed")
        elif action == 5:
            acc_no = input("Enter account no: ")

            if not self.bank_db.is_associated_account(self.c_id, acc_no):
                print("Not associated account")
                continue

            statement = self.bank_db.get_statement(acc_no)
            print(statement)
        elif action == 6:
            src_acc = input("Enter source account no: ")

            if not self.bank_db.is_associated_account(self.c_id, src_acc):
                print("Not associated account")
                continue

            dest_acc = input("Enter destination account no: ")
            amount = input("Enter amount to transfer: ")
            if self.bank_db.transfer_money(src_acc, dest_acc, amount):
                print("Success")
            else:
                print("Failed")
        elif action == 7:
            acc_no = input("Enter account no: ")

            if not self.bank_db.is_associated_account(self.c_id, acc_no):
                print("Not associated account")
                continue

            if self.bank_db.close_account(acc_no):
                print("Success")
            else:
                print("Failed")
        elif action == 8:
            acc_no = input("Enter account no: ")

            if not self.bank_db.is_associated_account(self.c_id, acc_no):
                print("Not associated account")
                continue

            loan_amount = input("Enter loan amount: ")

            loan_sanctioned = self.bank_db.avail_loan(acc_no, loan_amount)
```

```
                if loan_sanctioned:
                    print("Success")
                else:
                    print("Failed")
            elif action == 0:
                self.logout()
                print("Logged Out")
                break

            action = int(input("Enter your choice: "))

    def prompt_admin_login(self):
        admin_id = input("Enter admin id: ")
        admin_password = input("Enter admin password: ")

        return self.bank_db.validate_admin_login(admin_id, admin_password)

    def admin_sign_in(self):
        print("Select Admin Service")
        print("1. Print Closure Account History")
        print("2. FD Report of a customer")
        print("3. FD Report of Customers vis-à-vis another Customer")
        print("4. FD Report of Customers w.r.t a particular FD Amount")
        print("5. Loan Report of a Customer")
        print("6. Loan Report of Customers vis-à-vis another Customer")
        print("7. Loan Report of Customers w.r.t a particular Loan Amount")
        print("8. Loan – FD Report of Customers")
        print("9. Report of Customers who are yet to avail a loan (customer id,
first name, last name)")
            print("10. Report of Customers who are yet to open a FD account
(customer id, first name, last name)")
        print("11. Report of Customers who neither have a loan nor a FD account
with the bank (customer id, first name, last name)")
        print("0. Admin Logout")

        choice = int(input())
        return choice

    def perform_admin_action(self, action):
        while True:
            if action == 1:
                report = self.bank_db.get_closed_accounts()
                print(report)
            elif action == 2:
```

```
            c_id = input("Enter c_id: ")
            report = self.bank_db.get_fd_report(c_id)
            print(report)
        elif action == 3:
            c_id = input("Enter c_id: ")
            report = self.bank_db.get_fd_report_vis_a_vis_customer(c_id)
            print(report)
        elif action == 4:
            amount = input("Enter amount: ")
            report = self.bank_db.get_fd_report_wrt_amount(amount)
            print(report)
        elif action == 5:
            c_id = input("Enter c_id: ")
            report = self.bank_db.get_loan_report_of_customer(c_id)
            print(report)
        elif action == 6:
            c_id = input("Enter c_id: ")
            report = self.bank_db.get_fd_report_vis_a_vis_customer(c_id)
            print(report)
        elif action == 7:
            amount = input("Enter amount: ")
            report = self.bank_db.get_loan_report_wrt_amount(amount)
            print(report)
        elif action == 8:
            report = self.bank_db.get_loan_fd_report()
            print(report)
        elif action == 9:
            report = self.bank_db.get_no_loan_customers()
            print(report)
        elif action == 10:
            report = self.bank_db.get_no_fd_acc_customers()
            print(report)
        elif action == 11:
            report = self.bank_db.get_no_loan_no_fd_customers()
            print(report)
        elif action == 0:
            break

        action = int(input("Enter your choice: "))

    def logout(self):
        self.c_id = -1
```

```
menu = MainMenu()
while True:
    choice = menu.display_main_menu()
    menu.perform_menu_action(choice)
```

Output Screenshots: