

“Online Simple Banking System”

Project Report Submitted by
Name: Vedant R Jaiswal
(2019BEC012)

in partial fulfillment of the requirements for the
award of the degree of

Bachelor of Technology In Electronics & Telecommunication Engineering



**At
Shri Guru Gobind Singhji Institute of Engineering
and Technology, Nanded.**

Department of Electronics and Telecommunication Engineering
2022

Under the Guidance of
Mr. A.V. Nandedkar

CERTIFICATE

This is to certify that the topic entitled "Online Simple Banking System" is an Industrial Training seminar project delivered by Mr.Vedant Jaiswal under our supervision and guidance at the Department of Electronics and Telecommunication Engineering. Shri Guru Gobind Singhji Institute of Engineering and Technology, Nanded for the award of the degree of Bachelor of Technology in Electronics Engineering. The content of this dissertation work, in full or in parts has not been submitted to any other Institute or University for the award of any other degree or diploma...

Date:

Guide:

Mr.M.Bhalerao

Place: Nanded

Mr.A.V.Nandedkar

**Head
Department of Electronics
and Telecommunication
Engineering**

DECLARATION

Shri Guru Gobind Singhji Institute of Engineering and Technology, Nanded.



I hereby declare that the dissertation entitled "Online Simple Banking System", which is being submitted for the award of Bachelor of Technology in Electronics Engineering to the Department of Electronics and Telecommunication Engineering, Shri Guru Gobind Singhji Institute of Engineering & Technology, Nanded is an original and independent work. The contents have not been copied from any source without referring to the same.

**Name: Mr. Vedant Jaiswal
(2019BEC012)**

Signature of Students

ABSTRACT

This is a straightforward **GUI-based program** that is simple to grasp and utilize. The GUI is created with the **Tkinter** package. When it comes to the application, users may easily register an account and log in to manage their bank accounts.

In this project, the **Bank management System Python Project using Tkinter**, the user can create an account by entering information such as his or her username, opening balance, and PIN. The user must then submit those data to gain access to their account. The user can see transaction information, balance inquiries, credit and debit amounts, and more. The user can credit or debit sums by merely specifying a specific amount.

Acknowledgment

We express our sincere gratitude to our concerned teacher and guide **Mr.A.V.Nandedkar**, Professor, Department of Electronics and Telecommunication Engineering, for his valuable and inspiring guidance towards the progress on the topic providing valuable information for the development of my report.

Last but not least I express my sincere and hearty thanks to all those who have directly or indirectly helped me completing this seminar presentation and report successfully.

Date: 20-11-2022

Place: Vishnupuri, Nanded

Contents

Chapter 1

1 Introduction

Chapter 2

2 Pre-requisites for BankingSystem

2.1 Which modules do we require

2.2 SQL pre-requisites

Chapter 3

3 Data Sets

3.1 Types of data sets we need.

3.2 Use of those databases.

Chapter 4

4.1 Main Programming

4.2 OUTPUTs

Chapter 5

5 Application in real life

5.1 Professional sectors that are using this System

Chapter 6

6 Limitations

6.1 limitations of this Banking system.

Conclusion

List of Figures

Image 3.1 to 3.5:- Data Sets

Image 4.1:- Mainframe

Image 4.2:- Deposit, Withdraw, Search Frames

Image 4.3:- List of all accounts.

Chapter 1

Introduction

The Bank Management System project is written in Python. The project file contains a python script (main.py) and a database file. This is a simple console-based system that is very easy to understand and use. Talking about the system contains all the basic functions which include creating a new account, viewing the account holder's record, withdrawing and depositing the amount, balance inquiry, closing an account, and editing account details. In this mini project, there is no such login system. This means he/she can use all those available features easily without any restriction. It is too easy to use, he/she can check the total bank account records easily.

Talking about the features of the Bank Management System, a user can create an account by providing the name of the account holder, and number, selecting the amount type (Saving account or Current account), and providing an initial amount more than or equal to 500. Then the user can also deposit and withdraw money just by providing his/her account and entering the amount. For the certain purpose, he/she can also check for the balance inquiry which displays the account number and amount. He/she can also view all the account holder's lists. Another feature is that he/she can modify their account detail and type if they want to. This simple console-based Bank Management system provides the simplest management of bank accounts and transactions. In short, this project mainly focuses on CRUD. There's an external database connection file used in this mini project to save users' data permanently.

Chapter 2

Pre-requisites for the Banking system

2.1 Which Modules do we require

1 Tkinter import: Tkinter is the standard GUI library for Python. Python when combined with Tkinter provides a fast and easy way to create GUI applications. Tkinter provides a powerful object-oriented interface to the Tk GUI toolkit.

Creating a GUI application using Tkinter is an easy task. All you need to do is perform the following steps –

- Import the *Tkinter* module.
 - Create the GUI application main window.
 - Add one or more of the above-mentioned widgets to the GUI application.
- Enter the main event loop to take action against each event triggered by the user.

2 Tkinter message box: The `tkMessageBox` module is used to display message boxes in your applications. This module provides several functions that you can use to display an appropriate message. Some of these functions are `showinfo`, `showwarning`, `showerror`, `askquestion`, `askokcancel`, `askyesno`, and `askretryignore`.

3 import random

This module implements pseudo-random number generators for various distributions. For integers, there is a uniform selection from a range. For sequences, there is a uniform selection of a random element, a function to generate a random permutation of a list in place, and a function for random sampling without replacement. Almost all module functions depend on the basic function `random()`, which generates a random float uniformly in the semi-open range `[0.0, 1.0)`. Python uses the Mersenne Twister as the core generator. It produces 53-bit precision floats and has a period of $2^{19937}-1$. The underlying implementation in C is both fast and threadsafe. The Mersenne Twister is one of the most extensively tested random number generators in existence. However, being completely deterministic, it is not suitable for all purposes and is completely unsuitable for cryptographic purposes.

4 from MySQL connector

While working with Python we need to work with databases, they may be of different types like MySQL, SQLite, NoSQL, etc. In this article, we will be looking forward to how to connect MySQL databases using MySQL Connector/Python.

MySQL Connector module of Python is used to connect MySQL databases with the Python programs, it does that using the Python Database API Specification v2.0 (PEP 249). It uses the Python standard library and has no dependencies.

2.2 SQL Pre-requisites

MySQL is a relational database management system

Databases are the essential data repository for all software applications. For example, whenever someone conducts a web search, logs in to an account, or completes a transaction, a database system is storing the information so it can be accessed in the future. A **relational database** stores data in separate tables rather than putting all the data in one big storeroom. The database structure is organized into physical files optimized for speed. The logical data model, with objects such as data tables, views, rows, and columns, offers a flexible programming environment. You set up rules governing the relationships between different data fields, such as one-to-one, one-to-many, unique, required, or optional, and “pointers” between different tables. The database enforces these rules so that with a well-designed database your application never sees data that’s inconsistent, duplicated, orphaned, out of date, or missing.

The “SQL” part of “MySQL” stands for “Structured Query Language.” SQL is the most common standardized language used to access databases. Depending on your programming environment, you might enter SQL directly (for example, to generate reports), embed SQL statements into code written in another language, or use a language-specific API that hides the SQL syntax.

Chapter 3

Data Sets and SQL Tables

3.1 Types of data sets we need.

1 Main Database and tables in it

```
mysql> use bank;
Database changed
mysql> show tables;
+-----+
| Tables_in_bank |
+-----+
| account        |
| amount        |
+-----+
2 rows in set (0.01 sec)
```

Image 3.1

2 Table we are going to use:- account

```
mysql> describe account;
+-----+-----+-----+-----+-----+-----+
| Field | Type          | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| Name  | varchar(50)   | YES  |     | NULL    |       |
| ACNO  | varchar(20)   | YES  |     | NULL    |       |
| DOB   | varchar(20)   | YES  |     | NULL    |       |
| PHONE | varchar(20)   | YES  |     | NULL    |       |
| ADDRESS | varchar(100) | YES  |     | NULL    |       |
| OB    | int           | YES  |     | NULL    |       |
+-----+-----+-----+-----+-----+-----+
6 rows in set (0.02 sec)
```

Image 3.2

3 All Existing Values in it.

```
mysql> select * from account;
```

Name	ACNO	DOB	PHONE	ADDRESS	OB
GAURAV	12345	12/09/2008	1234567	AMRAVATI	990
MAYURI	90876	12/08/2003	345236970	NAGPUR	3856
ROHIT	34521	12/11/2009	123678456	WARDHA	10000
Ved	56780	15/06/2005	657832	MUMBAI	10000
Mohit	15075	20/06/2004	950432	AKOLA	9940
KAPIL	45498	27/05/2006	29657	SATARA	10001

6 rows in set (0.00 sec)

Image 3.3

3.2 Use of Those Databases

Database tables are very important in this banking system. Mainly for storing the whole data of bank customers we are using the database hence it reflects on the performance of the system and the speed of the project model too. But specifically speaking for larger data sets we can't use sheets otherwise the performance will decline drastically. Also, the use and modification process through a system-based programming language is very reliable without harming the original set and is very credible in SQL. Due to DBMS now we have Better managed data and Improved data access because of which we can generate better quality information hence on this basis better decisions can be made. Better Data quality improves accuracy, validity, and the time it takes to read data. DBMS does not guarantee data quality, it provides a framework to make it easy to improve data quality. The major purpose of a database system is to provide users with an abstract view of the data. Since many complex algorithms are used by the developers to increase the efficiency of databases that are being hidden by the users through various data abstraction levels to allow users to easily interact with the system.

Chapter 4

Programming

4.1 Main Programs

#Main code

```
from tkinter import *
from Accountdb import *
from tkinter.messagebox import *
import random as r

class openaccountbox(Toplevel):
    def __init__(self):
        super().__init__()
        self.geometry("200x200")
        self.l1=Label(self,text="Name")
        self.nm=StringVar()
        self.b1=IntVar()
        self.dt=StringVar()
        self.phn=IntVar()
        self.adr=StringVar()
        self.l1.grid(row=1,column=0,padx=5,pady=5)
        self.t1=Entry(self,textvariable=self.nm)

        self.t1.grid(row=1,column=1,columnspan=2,padx=5,pady=5)

        self.l2=Label(self,text="Balance")
        self.l2.grid(row=2,column=0,padx=5,pady=5)
        self.t2=Entry(self,textvariable=self.b1)

        self.t2.grid(row=2,column=1,columnspan=2,padx=5,pady=5)

        self.l3=Label(self,text="DOB")
        self.l3.grid(row=3,column=0,padx=5,pady=5)
        self.t3=Entry(self,textvariable=self.dt)
```

```

self.t3.grid(row=3,column=1,columnspan=2,padx=5,pady=5)

        self.l4=Label(self,text="PHONE")
        self.l4.grid(row=4,column=0,padx=5,pady=5)
        self.t4=Entry(self,textvariable=self.phn)

self.t4.grid(row=4,column=1,columnspan=2,padx=5,pady=5)

        self.l5=Label(self,text="ADDRESS")
        self.l5.grid(row=5,column=0,padx=5,pady=5)
        self.t5=Entry(self,textvariable=self.adr)

self.t5.grid(row=5,column=1,columnspan=2,padx=5,pady=5)


self.b1=Button(self,text="Save",command=self.save)
        self.b1.grid(row=6,column=1,padx=5,pady=5)
        self.b2=Button(self,text="cancel")
        self.b2.grid(row=6,column=2,padx=5,pady=5)

    def save(self):
        name=self.nm.get()
        balance=self.b1.get()
        dte=self.dt.get()
        phn=self.phn.get()
        adr=self.adr.get()
        ac=r.randint(10000,99999)

        a=Account()

accno=a.openAccount(name,balance,ac,dte,phn,adr)
        showinfo("Bank","Account Created...\n Accno:
"+str(ac))

```

```

class Depositaccountbox(Toplevel):
    def __init__(self):
        super().__init__()
        self.acn=IntVar()
        self.amt=IntVar()
        self.l1=Label(self,text="Accno.")
        self.l1.grid(row=0,column=0,padx=5,pady=5)
        self.t1=Entry(self,textvariable=self.acn)

self.t1.grid(row=0,column=1,columnspan=2,padx=5,pady=5)
        self.l2=Label(self,text="Amount")
        self.l2.grid(row=1,column=0,padx=5,pady=5)
        self.t2=Entry(self,textvariable=self.amt)

self.t2.grid(row=1,column=1,columnspan=2,padx=5,pady=5)

        self.b1=Button(self,text="Deposit",command=self.save)
        self.b1.grid(row=2,column=1,padx=5,pady=5)
    def save(self):
        an=self.acn.get()
        am=self.amt.get()
        a=Account()
        if a.deposit(an,am):
            showinfo("BANK","Amount is Successfully
Deposted...")
        else:
            showerror("BANK","Failed..")

class Withdrawaccountbox(Toplevel):
    def __init__(self):
        super().__init__()
        self.acn=IntVar()
        self.amt=IntVar()

```

```

        self.l1=Label(self,text="Accno.")
        self.l1.grid(row=0,column=0,padx=5,pady=5)
        self.t1=Entry(self,textvariable=self.acn)

self.t1.grid(row=0,column=1,columnspan=2,padx=5,pady=5)
        self.l2=Label(self,text="Amount")
        self.l2.grid(row=1,column=0,padx=5,pady=5)
        self.t2=Entry(self,textvariable=self.amt)

self.t2.grid(row=1,column=1,columnspan=2,padx=5,pady=5)

self.b1=Button(self,text="Withdraw",command=self.save)
        self.b1.grid(row=2,column=1,padx=5,pady=5)
    def save(self):
        an=self.acn.get()
        am=self.amt.get()
        a=Account()
        if a.withdraw(an,am):
            showinfo("BANK","Amount is Withdraw")
        else:
            showerror("BANK","insuficient Balance")

class searchaccountbox(Toplevel):
    def __init__(self):
        super().__init__()
        self.acn=IntVar()
        self.l1=Label(self,text="Accno.")
        self.l1.grid(row=0,column=0,padx=5,pady=5)
        self.t1=Entry(self,textvariable=self.acn)

self.t1.grid(row=0,column=1,columnspan=2,padx=5,pady=5)

self.b1=Button(self,text="Search",command=self.save)
        self.b1.grid(row=1,column=1,padx=5,pady=5)

```



```

def save(self):
    an=self.acn.get()
    a=Account()
    res=a.search(an)
    if res!=None:
        msg="NAME : "+str(res[0])+"\nACNO : 
"+str(res[1])+"\nDOB : "+str(res[2])+"\nPHONE : 
"+str(res[3])+"\nADDRESS : "+str(res[4])+"\nBalance : 
"+str(res[5])
        showinfo("BANK",msg)
    else:
        showerror("BANK","ACCOUNT NOT FOUND")

class listAccounts(Toplevel):
    def __init__(self):
        super().__init__()
        self.geometry("350x400")
        self.txt=Text(self)
        self.txt.pack()
        a=Account()
        res=a.listAccount()
        for lst in res:
            an=str(lst[0])
            nm=str(lst[1])
            bl=str(lst[5])
            self.txt.insert(1.0,"| "+an+"\t| 
"+nm+"\t\t| "+bl+"\t | \n")
            self.txt.insert(1.0, "| Name\t| Acno\t\t| 
Balance\t| \n")

class mainframe(Tk):
    def __init__(self):

```

```

        super().__init__()
        self.geometry("500x500")

        self.b1=Button(self, text="Open
Account", width=200, bg="skyblue", fg="white", font="arial
20 bold", command=self.funopenaccount)
        self.b1.pack (padx=5, pady=5)

        self.b2=Button(self, text="Deposit", width=200, bg="skyblue", fg="white", font="arial
20 bold", command=self.fundepositaccount)
        self.b2.pack (padx=5, pady=5)

        self.b3=Button(self, text="Withdraw", width=200, bg="skyblue", fg="white", font="arial
20 bold", command=self.funwithdrawaccount)
        self.b3.pack (padx=5, pady=5)

        self.b4=Button(self, text="Search", width=200, bg="skyblue", fg="white", font="arial
20 bold", command=self.funsearchaccount)
        self.b4.pack (padx=5, pady=5)

        self.b5=Button(self, text="List", width=200, bg="skyblue", fg="white", font="arial
20 bold", command=self.funlistaccounts)
        self.b5.pack (padx=5, pady=5)

    def funopenaccount(self):
        box=openaccountbox()
        box.mainloop()

    def funlistaccounts(self):
        box=listAccounts()
        box.mainloop()

```

```
def fundepositaccount(self):  
    box=Depositaccountbox()  
    box.mainloop()  
  
def funwithdrawaccount(self):  
    box=Withdrawaccountbox()  
    box.mainloop()  
  
def funsearchaccount(self):  
    box=searchaccountbox()  
    box.mainloop()  
main=mainframe()  
main.mainloop()
```

#DATABASE CONNECTION

```
import mysql.connector as sql
class Account:
    def __init__(self):

self.conn=sql.connect(user="root",password="Vedant@4352",database="bank",auth_plugin='mysql_native_password')
    print("Connected...")
    def __del__(self):
        self.conn.close()
        print("Connection Closed...")
    def openAccount(self,name,OB,ACNO,dob,pnum,addr):
        stmt="insert into
account (name,OB,ACNO,DOB,PHONE,ADDRESS)
value (%s,%s,%s,%s,%s,%s) "
        self.cur=self.conn.cursor()

self.cur.execute(stmt,[name,OB,ACNO,dob,pnum,addr])
        self.conn.commit()
        stmt2="select * from account order by ACNO DESC
"

        cur2=self.conn.cursor()
        cur2.execute(stmt2)
        res=cur2.fetchone()
        if self.cur.rowcount==1:
            return res[0]
        else:
            print("Failed...")
    def deposit(self,acc,amt):
        stmt="UPDATE account SET OB=OB+%s WHERE
ACNO=%s"

        cur=self.conn.cursor()
        cur.execute(stmt,[amt,acc])
        self.conn.commit()
```

```

        if cur.rowcount==1:
            return True
        else:
            return False
def withdraw(self,acc,amt):
    stmt="select * from account where ACNO=%s"
    cur=self.conn.cursor()
    cur.execute(stmt,[acc])
    res=cur.fetchone()
    bal=int(res[5])
    if bal>=amt:
        stmt1 = "UPDATE account SET OB=OB-%s WHERE
ACNO=%s"

        curl=self.conn.cursor()
        curl.execute(stmt1,[amt,acc])
        self.conn.commit()
        if curl.rowcount==1:
            return True
        else:
            return False
def search(self,acc):
    stmt="select * from account where ACNO=%s"
    cur=self.conn.cursor()
    cur.execute(stmt,[acc])
    res=cur.fetchone()
    return res

def listAccount(self):
    stmt="select * from account"
    cur = self.conn.cursor()
    cur.execute(stmt)
    res = cur.fetchall()
    return res

```

#OUTPUTs

4.2 OUTPUTs

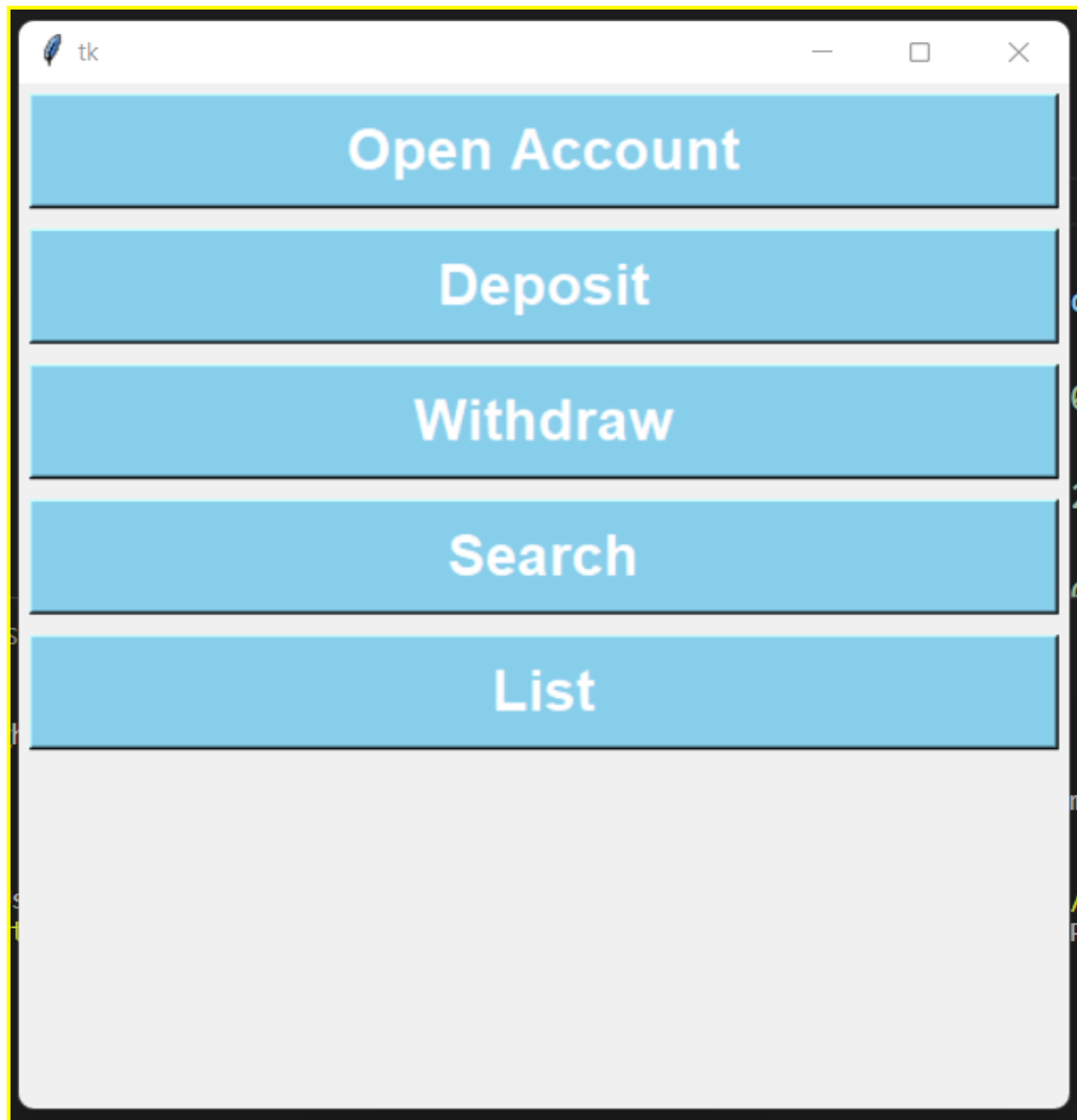


Image 4.1

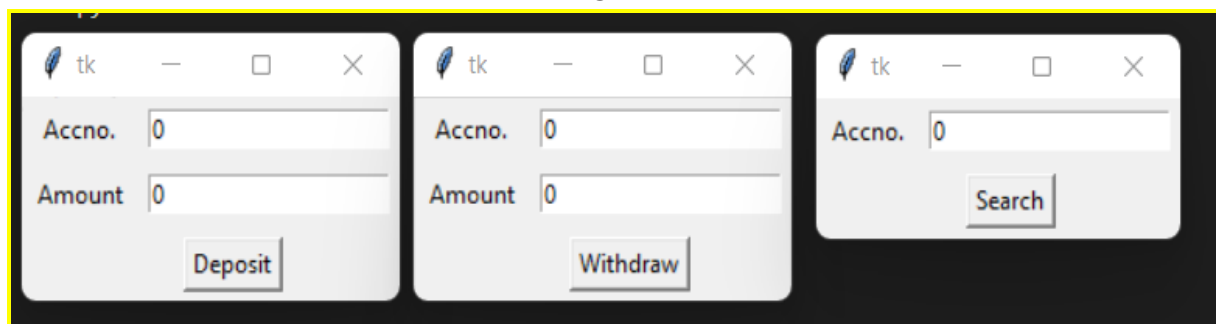
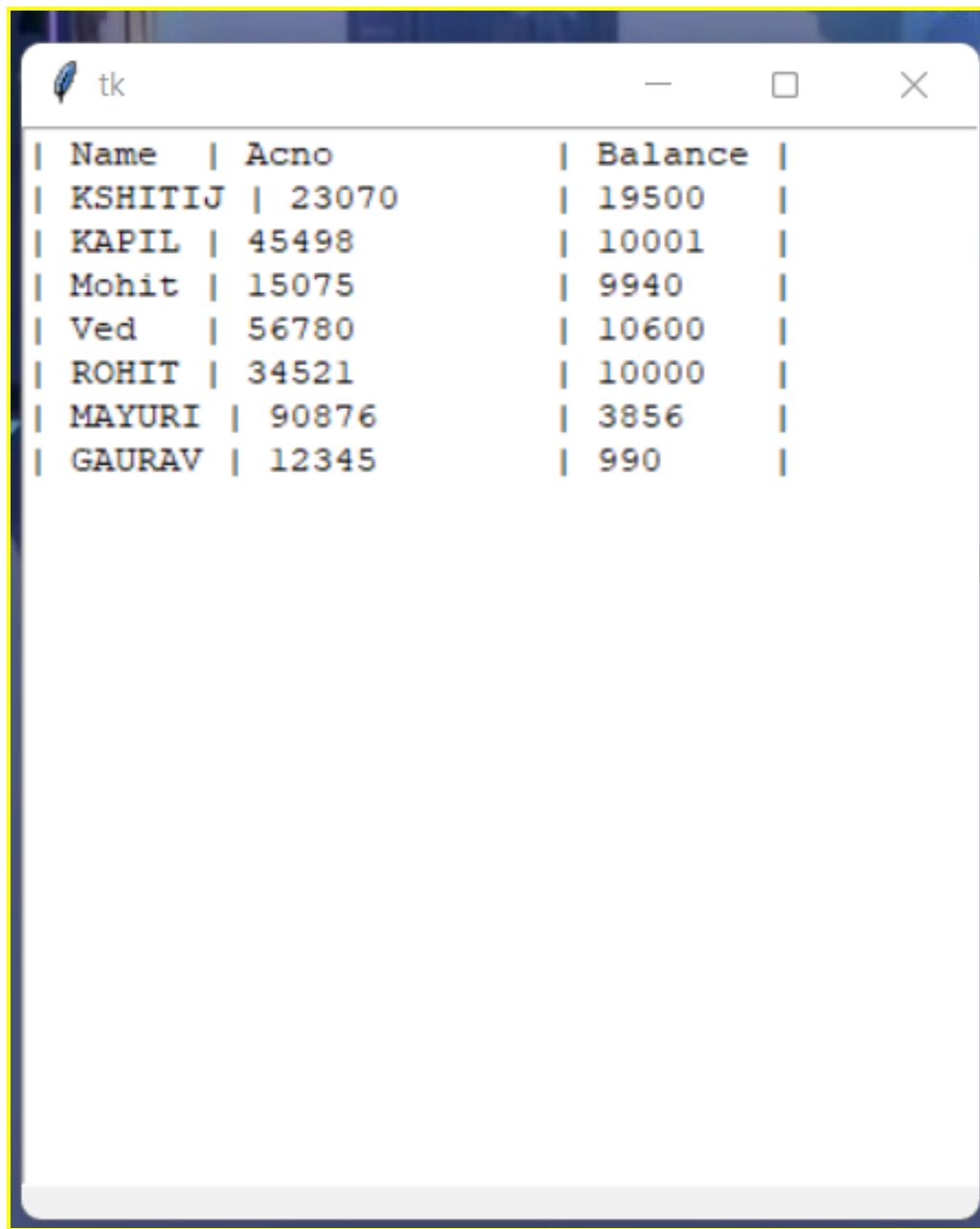


Image 4.2



Name	Acno	Balance
KSHITIJ	23070	19500
KAPIL	45498	10001
Mohit	15075	9940
Ved	56780	10600
ROHIT	34521	10000
MAYURI	90876	3856
GAURAV	12345	990

Image 4.3

Chapter 5

Applications in real life

5.1 Professional sectors that are using this System

e-Commerce

Is an industry where recommendation systems were first widely used. With millions of customers and data on their online behavior, e-commerce companies are best suited to use this system.

Banking

A mass-market product that is consumed digitally by millions. Banking for the masses and SMEs are prime for managing financial activities.

Financial Specialist

Similar to banking aspects financial advisers can also use this simple system. It can loosen some pressure on them and also can help them to manage their data properly in the SQL database without losing them. Also as we are using SQL so it surely has better security and is easy for retrieval.

Chapter 6

Limitations

6.1 Limitations in this Banking Managing System

In an open-source environment, it is hard to measure the success of your system without deploying and receiving feedback from the users. In terms of the banking management system, here the aspect is easy to use but the database here is stored locally so this can be a problem here, but although we solve that problem here the main issue is it's a single user based who has database access or mainly local system invariant so changing the base system requires so much unnecessary manpower.

Conclusion:

Hence we can conclude that this banking managing system can perform the appropriate financial operations which the user's perspective. Hence it can be more efficient but it requires input data from the user.