**PROJECT DETAILED REPORT ON**

**Banking Management System**

**Submitted by**

Prateek Bangwal Shivam Sharma Himanshu Kumar

500044883 500046375 500045966

R-103215048 R-103215070 R-103215026

CSE-BAO V CSE-BAO V CSE-BAO V

***Under the guidance of***

**Mr. Deepak Sharma**

Assistant Professor

School of Computer Science & Engineering,



**Department of Analytic, School of Computer Science & Engineering,**

**UNIVERSITY OF PETROLEUM AND ENERGY STUDIES Dehradun-248007**

**August-2017**

This is to certify that the project work entitled **“Banking Management System”** done by **Prateek Bangwal(R103215048) , Shivam Sharma (R103215070) & Himanshu Kumar(R103215026)** for partial fulfilment of the requirement for the award of the Degree of Bachler of Technology in Computer Science & Engineering with Specialization in Business analytic and optimization to School of Computer Science & Engineering, University of Petroleum & Energy Studies is the bonafide report of the work carried out by them under my guidance and supervision.

To best of my knowledge, the literature embodied in this project work has not been submitted to any other University/Institution for the award of any Degree or Diploma.

**Mr. Deepak Sharma Project Guide SoCSE, UPES**

**CANDIDATE’S DECLARATION**

We hereby declare that the work which is being presented in this dissertation entitled **“Banking Management System”** done by **Prateek bangwal(R103215048) , Shivam Sharma (R103215070) & Himanshu Kumar(R103215026)** for partial fulfilment of the requirement for the award of the degree of Bachler of Technology in Computer Science & Engineering with Specialization in usiness analytic and optimization to School of Computer Science & Engineering, University of Petroleum & Energy Studies, Dehradun, is an authentic record of our work carried out during a period from May, 2017 to August, 2017 under the supervision of **Mr. Deepak Sharma, Assistant Professor, School of Computer Science & Engineering, UPES.**

The matter presented by us in this report has not been submitted by any of us for any other degree to any other institutes.

**Prateek Bangwal**

**(R103215048)**

**Shivam Sharma**

**(R103215070)**

**Himanshu Kumar**

**(R103215026)**

This is to certify that the above statement made by the candidates is correct to the best of my knowledge.

Date – Sep 7, 2017 **Mr. Deepak Sharma**

**Project Guide**

**Assistant Professor**

**School of Computer Science & Engineering**

If words are considered as a symbol of approval and token of appreciation, then let the words play the heralding role of expressing my gratitude.

The satisfaction that accompanies that the successful completion of our task would be incomplete without the mention of people whose ceaseless cooperation made it possible, whose constant guidance and encouragement crown all efforts with success. We are grateful to our project guide **Mr. Deepak Sharma** for the guidance, inspiration and constant suggestions that helped us in the preparation of this project.. We also thank our **colleagues** who have helped in successful completion of this project. We have no words to express our sincere gratitude to our **parents** who have shown faith and for every support they have given.

**Prateek Bangwal (R103215048)**

**Shivam Sharma (R103215070)**

**Himanshu Kumar (R103215026)**

Banking management system which is designed intended to provide glimpse of work of Banks As we all know that currently India to going towards digitalization and Banks are one of the basic entity of a town. Customer data when stored in database is easy to handle, retrieve and helps in generating quick reports and saves time. In order to make this happen, Oracle XE 11G is used for data handling which is integrated with Python 2.7. This mechanism ensures immediate information storage, retrieval, avoid manual calculation error & keeping record of the Customers along with keeping information about various transcation of the Customer

***Keywords:*** *XE, Oracle, Python.*

*Certificate*……………………………………………………………………….…………….i *Candidate’s Declaration*………………………………………………………………...........ii *Acknowledgement*……………………………………………………………….……………iii *Abstract*………………………………………………………………………….……………iv List of Tables, Abbreviation…………………………………………………………………...2

Introduction………………………………………………………………………………...…3

Literature Review…………………………………………………………………….……….4

Problem Statement…………………………………………………………………………....5

Objective………………………………………………………………………….…………..6

Methodology……………………………………………………………………………….....7

Procedure…………………………………………………………………………………….11

Screenshots…………………………………………………………………….…………….12

Results & Declaration………………………………………………………………………..14

Future Scope………………………………………………………………………………….15

References……………………………………………………………………………………16

Tables: Page No.

• Tab.1 - Customer Account Detail……………………………….…………….……..8

• Tab.2 - Saving Account …………………………………………..……………..…….9

• Tab.3 - Current Account Customer…………………..……………...…………………9

• Tab.4 - Customer Transaction …………….………………………..……………....9

• Tab.5 - Admin Account………………………………………….……………….....9

• Tab.6 - Phase Wise Task Completion…………………………………………...…22

• Tab.7 - System Requirement………………………………………………….........22

Abbreviations:

GB **-** Gigabyte

RAM **-** Random Access Memory

HDD **-** Hard Disk Drive

SQL **-** Structured Query Language

FK - Foreign Key

PK - Primary Key

BMS (Banking Management System) is powerful, flexible, and easy to use and is designed and developed to deliver real conceivable benefits to Banks.

Banking Management System is designed for small hospitals, to cover a minimal range of hospital working processes. It is an integrated end-to-end Banking Management System that provides relevant information across the banks to support effective decision making for transaction, deposit and withdrawal of money and basic financial accounting, in a seamless flow. BMS is a software product designed to improve the quality and management of banks management Banking Management System enables you to develop your organization and improve its effectiveness and quality of work. Managing the key processes efficiently is critical to the success of the bank.

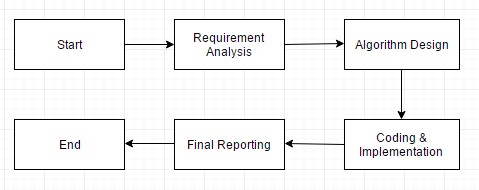
Masoud Nosrati, " Python: An appropriate language for real world programming ", 2011

This paper represents an introduction to Python programming language and prove it as a suitable language for both learning and real-world programming. The use of python for general purposes and its features as a high-level programming language. It has efficient high-level data structures and a simple but effective approach to object-oriented programming. [1]

Existing mechanism in majority of Indian Bank is a primitive way which cause lack of immediate retrievals, lack of immediate information storage, error prone manual calculation.

To develop a Bank management system which provide a quick and efficient retrieval of information and to reduce the problem which occurs when large data is manually entered.

Our methodology involves developing a management system, through which we can achieve a mechanism of quick storage and retrieval of information whenever required.



**Fig. 1 – Advancement of Project**

There are three phases described below: -

**Phase 1: Requirement and Analysis**

• Studying the existing mechanism.

• Finalizing and analyzing the algorithmic design.

• Finalizing the mathematical approach to optimize & implement the algorithms.

**Phase 2: Designing and Development**

Algorithm design is a specific method to create mathematical process to solve a problem and this project aims for the same by attempting to figure out a mathematically optimized algorithm.

Designing and development is further divided into various phases.

This phase starts with the input from the Requirement and Analysis phase which will clarify the problem definition which will lead this project to the model development phase where a

model will be created and further lead to the designing of the algorithm which will focus on desired final product.

After completion of the designing of the algorithm phase, the focus will shift on the analysis of the algorithm based on the overall performance of the algorithm also known as Big O of the algorithm.

**Phase 3: Coding**

On receiving system design documents, the work is divided into modules/unit and distributed among the team members and actual coding is started. Since in this phase the code is produced so it is the main focus of the developers. This is going to be the longest phase in this project. The implementation of this project starts in terms of writing program in the suitable programming language and developing error free executable program efficiently, this phase

primarily focuses on coding, using the decided algorithms in an optimized manner.

**Class Tables:**

**Tab.1 Customer Account Data**

|  |  |  |
| --- | --- | --- |
| **Name** | **Data Type** | **Constraint** |
| Acc\_No | Varchar | PK |
| Fname | Varchar |  |
| Lname | Varchar |  |
| Address Line 1 | Varchar |  |
| Address Line 2 | Varchar |  |
| City | Varchar |  |
| Pin | Int |  |
| Password | Varchar |  |
| STATUS | Varchar |  |
| Validity | Varchar |  |

**Table 2**

**Saving account**

|  |  |  |
| --- | --- | --- |
| **Name** | **Data Type** | **Constraint** |
| Account NO | Varchar | PK |
| Intrest Rate | Int |  |
| Withdrawals | Int |  |
| Last Date | Date |  |
| Balance Number | Int |  |

|  |  |  |
| --- | --- | --- |
| **Name** | **Data Type** | **Constraint** |
| Account NO | Varchar | FK |
| Balance Number | number |  |

**Tab.3 CURRENT ACCOUNT CUSTOMER**

**Tab.4 Customer Transcation**

|  |  |  |
| --- | --- | --- |
| **Name** | **Data Type** | **Constraint** |
| Transaction\_ID | Int | FK |
| Account No | Varchar | PK |
| Type Of Transaction | Varchar |  |
| Transaction Amount Number | Int |  |
| Balance Number | Varchar |  |
| Date of Transaction | Date |  |

**Tab.5 Admin Account**

|  |  |  |
| --- | --- | --- |
| **Name** | **Data type** | **Constraint** |
| Admin Id | Int | PK |
| Dischargedate | Date |  |

**Tab.6 Phase Wise Task Completion**

|  |  |
| --- | --- |
| Phase | Duration |
| Start | 3rd week of July |
| Requirement Analysis | by 4th week of July |
| Algorithm Design | by 1st week of August |
| Coding | by 2nd week of August |
| Reporting | by 3rd week of August |
| End | by 3rd week of August |

|  |  |
| --- | --- |
| Operating System | Windows 8 or above |
| Hardware Configuration | Intel i3 x64, 2 GB RAM, 3 GB HDD |
| Language Compiler | Eclipse IDE with PYdev integration |
| Others | Oracle XE 11G, cx\_Oracle |

**PROCEDURE**

1. New user needs to signup and existing user can log in.

2. After successful login, a menu with various options like money withdrawal , money deposit, print statement ,statement transfer money, account closure ,customer logout will appear.

3. User needs to enter his choice.

4. Relevant sub menu will appear according to user choice and user will be asked to enter the relevant data wherever required.

1. Program will exit only when user needs to exit.

**Solution-**

Following tables are pre created in the pl/sql oracle 11g

All tables are populated via python script, not manually

Only table populated manually is Admin table.

**Tables-**

Create table CustomerAccountDetails(

AccountNo varcahar(40),

Fname varchar(20),

Lname varchar(20),

AddressLine1 varchar(40),

AddressLine2 varchar(40),

City varchar(20),

Pin number,

Password varchar(20),

Status varchar(20) check ( Status IN (‘LOCKED’, ‘UNLOCKED’)),

Validity varchar(20) check ( Validity IN (‘CLOSED’, ‘OPEN’))

);

Create table SavingAccountCustomers(

AccountNo varchar(40),

InterestRate number(4,2),

Withdrawls number,

LastDate Date,

Balance number

);

Create table CurrentAccountCustomers(

AccountNo varchar(40),

Balance number

);

Create table CustomerTransactions(

TransactionId number,

AccountNo varchar(40),

TypeofTransaction varchar( 20) CHECK ( TypeofTransaction IN (‘Deposit’,’Withdrawl’,’Transfer’,’Closure’)),

TransactionAmount number,

Balance number,

DateofTransaction Date

);

Create sequence auto\_increment

Start with 1

Increment by 1

Cache 1000;

Create table AdminAccounts(

adminid varchar(40),

adminpass varchar(40)

);

Insert into AdminAccounts values( ‘user1’,’user123’);

**Code-**

import random

import cx\_Oracle

from collections import Counter

from datetime import datetime

import sys

import calendar

# below is the class for admin sign in which all the related work to admin is performed

class AdminSignIn:

def \_\_init\_\_(self):

print(" do you wish to see the closed Account histories")

answer= raw\_input(" enter Y for YES and N for NO and to be logged out")

if( answer== 'Y'):

con= cx\_Oracle.connect(" user1/user123")

cur= con.cursor()

cur.execute(" select \* from CustomerAccountDetails where Validity= 'CLOSED'")

print( cur.fetchall())

else:

print(" you are being logged out")

return

# the below class is for the Sign Up for a new Customer

class SignUp:

def sending\_to\_database(self):

con= cx\_Oracle.connect("user1/user123")

cur= con.cursor()

cur.execute( " INSERT INTO CustomerAccountDetails values( :param1, :param2, :param3, :param4, :param5 ,:param6, :param7, :param8, :param9, :param10)", { 'param1': self.accountno, 'param2': self.fname, 'param3': self.lname, 'param4': self.addressline1, 'param5': self.addressline2, 'param6':self.city, 'param7':self.pin, 'param8':self.password, 'param9': self.status, 'param10':'OPEN'})

cur= con.cursor()

cur.execute( " select \* from CustomerAccountDetails")

print( cur.fetchall())

if( self.accounttype=='Savings'):

cur.execute(" INSERT INTO SavingAccountCustomers(AccountNo, InterestRate, Withdrawls, Balance) values( :param1, 7.5,10, :param2)", {'param1':self.accountno, 'param2': self.balance})

elif( self.accounttype == 'Current'):

cur.execute( " Insert into CurrentAccountCustomers values( :param1, :param2)", {'param1':self.accountno, 'param2':self.balance})

cur.execute( " select \* from SavingAccountCustomers")

print( cur.fetchall())

cur.execute( " select \* from CurrentAccountCustomers")

print( cur.fetchall())

con.commit()

con.close()

def \_\_str\_\_(self):

return ( self.name + str( self.balance))

def \_\_init\_\_(self):

self.customerid= random.randrange(100000,9999999999, 1)

self.accountno= "BBSCOO"+ str( self.customerid)

self.status= 'UNLOCKED'

self.fname= raw\_input("first name")

self.fname= (self.fname.lower())

self.lname= raw\_input("last name")

self.lname= (self.lname.lower())

self.name= self.fname + self.lname

self.addressline1= raw\_input(" address Line 1")

self.addressline2= raw\_input(" Address Line 2")

self.city= raw\_input("city")

pinflag= False

# the checking off pin is performed

while(pinflag== False):

try:

pint= (raw\_input(" Pin( 6 digits)"))

pcounter=0

for i in pint:

pcounter += 1

pint= int(pint)

if( pcounter == 6):

self.pin= pint

pinflag= True

else:

raise Exception("no pin is greater than 6 digits")

except Exception as err:

print('caught this error', repr(err))

self.address= self.addressline1 + " "+self.addressline2+ " " + self.city + " "+ str( self.pin)

accc= int(raw\_input(" enter the account type you wish for , enter 1 for Savings and 2 for Current"))

if( accc== 1):

self.accounttype= "Savings"

self.balance= 0

elif( accc== 2):

self.accounttype= "Current"

self.balance= 5000

#checking the password

passflag= False

while( passflag== False):

try:

passc= raw\_input("password( 8 alphanumeric to be strong)")

if(len(passc) < 8):

raise Exception('the password must be 8 characters long')

try:

passc= int(passc)

print("password requires atleast 1 alphahabet to be strong and valid")

except Exception as err:

#print(' exception is thrown', repr(err))

print("password accepted !!! remember it")

passflag= True

self.password= passc

#if( isinstance(passc, int)== True ):

# raise Exception("password requires atleast 1 alphahabet to be strong and valid")

except Exception as err:

print('caught this error', repr(err))

passflag= False

# the class is for Sign In it works with the drop down menu for Sign In

class SignIn:

def account\_closure( self, accountnou):

print( "IT WAS A PLEASURE DOING BUSINESS WITH YOU , BAJORIA BANK SERVICES WISHES FOR YOUR RETURN ")

con= cx\_Oracle.connect("user1/user123")

cur= con.cursor()

cur.execute( " update CustomerAccountDetails set Validity= 'CLOSED' where AccountNo= :param1", {'param1':accountnou})

print(" YOUR ACCOUNT HAS BEEN CLOSED")

con.commit()

def logout(self):

print("you are logged out")

return

def transfer\_money( self, accountnofrom):

accountnoto= raw\_input(" enter the accountno in which you want the money to be transfreered")

con=cx\_Oracle.connect("user1/user123")

cur= con.cursor()

try:

cur.execute( " select count(\*) from CustomerAccountDetails where AccountNo= :param1", {'param1':accountnoto})

data= cur.fetchall()

if( int( data[0][0])== 0):

raise Exception(" THE ACCOUNT IN WHICH MONEY HAS TO BE TRANSFERRED IS INVALID")

amount= int( raw\_input( " enter the amount you want to transfer"))

if( amount < 0):

raise Exception(" THE AMOUNT ENTERED IS INVALID")

cur.execute( " select AccountType from CustomerAccountDetails where AccountNo= :param1", { 'param1':accountnofrom})

data1= cur.fetchall()

cur.execute( " select AccountType from CustomerAccountDetails where AccountNo= :param2", { 'param2': accountnoto})

data2= cur.fetchall()

if( str( data1[0][0])== 'Savings'):

cur.execute(" select Balance from SavingAccountCustomers where AccountNo= :param1",{'param1':accountnofrom})

data= cur.fetchall()

previousbalance= int( data[0][0])

if( amount> previousbalance):

raise Exception( "TRANSFER AMOUNT IS GREATER THAN AVAILABLE BALANCE")

finalbalance= previousbalance- amount

cur.execute( " update SavingAccountCustomers set Balance= :param1 where AccountNo= :param2", {'param1':finalbalance,'param2':accountnofrom})

cur.execute("insert into CustomerTransactions values(auto\_increment.nextval, 'Transfer',:param1, :param2, :param3, sysdate)", {'param1':amount, 'param2':finalbalance, 'param3':accountnofrom})

else:

cur.execute(" select Balance from CurrentAccountCustomers where AccountNo= :param1",{'param1':accountnofrom})

data= cur.fetchall()

previousbalance= int( data[0][0])

previousbalance -= 5000

if( amount> previousbalance):

raise Exception( "TRANSFER AMOUNT IS GREATER THAN AVAILABLE BALANCE(we are maintaining min 5k balance)")

finalbalance= previousbalance- amount

finalbalance += 5000

cur.execute( " update CurrentAccountCustomers set Balance= :param1 where AccountNo= :param2", {'param1':finalbalance,'param2':accountnofrom})

cur.execute("insert into CustomerTransactions values(auto\_increment.nextval, 'Transfer',:param1, :param2, :param3, sysdate)", {'param1':amount, 'param2':finalbalance, 'param3':accountnofrom})

if( str( data2[0][0]) == 'Savings'):

cur.execute( " select Balance from SavingAccountCustomers where AccountNo= :param1", {'param1':accountnoto})

data= cur.fetchall()

previousbalance= int( data[0][0])

finalbalance= previousbalance + amount

cur.execute( " update SavingAccountCustomers set Balance= :param1 where AccountNo= :param2", {'param1':finalbalance ,'param2':accountnoto})

cur.execute("insert into CustomerTransactions values(auto\_increment.nextval, 'Transfer',:param1, :param2, :param3, sysdate)", {'param1':amount, 'param2':finalbalance, 'param3':accountnoto})

raise Exception("SUCCESS")

else:

cur.execute( " select Balance from CurrentAccountCustomers where AccountNo= :param1", {'param1':accountnoto})

data= cur.fetchall()

previousbalance= int( data[0][0])

finalbalance= previousbalance + amount

cur.execute( " update CurrentAccountCustomers set Balance= :param1 where AccountNo= :param2", {'param1':finalbalance ,'param2':accountnoto})

cur.execute("insert into CustomerTransactions values(auto\_increment.nextval, 'Transfer',:param1, :param2, :param3, sysdate)", {'param1':amount, 'param2':finalbalance, 'param3':accountnoto})

raise Exception("SUCCESS")

except Exception as err:

con.commit()

print('Caught this error', repr(err))

def print\_statement(self ,accountnou):

print(" enter the dates from where to where you want to print your account statement in the ")

dateflag= False

while( dateflag== False):

date1= raw\_input(" enter the FROM date in the 'YYYY-MM-DD' format")

year1,month1, day1 = (date1.split('-'))

date2= raw\_input(" enter the TO date in the 'YYYY-MM-DD' format")

year2,month2, day2 = (date2.split('-'))

if( year1< year2):

dateflag= True

elif( year1 == year2 and month1< month2):

dateflag= True

elif( year1== year2 and month1== month2 and date1 < date2):

dateflag= True

else:

dateflag= False

con= cx\_Oracle.connect("user1/user123")

cur= con.cursor()

print(" this is your account statement within the time mentioned above")

cur.execute( " select \* from CustomerTransactions where AccountNo= :param1 and DateofTransaction >=:param2 and DateofTransaction <= :param3",{ 'param1':accountnou, 'param2': datetime(int(year1), int(month1),int(day1)) , 'param3':datetime(int(year2), int(month2),int(day2))})

print( cur.fetchall())

def withdraw\_money( self, accountnou):

#accountnou= raw\_input(" enter the account number from which you want to withdraw the money")

con= cx\_Oracle.connect("user1/user123")

cur= con.cursor()

cur.execute( " select count(\*) from CustomerAccountDetails where AccountNo= :param1", {'param1':accountnou})

data= cur.fetchall()

try:

if( data[0][0]==0):

raise Exception(" ACCOUNT DOESN'T EXSISTS , ENTER A VALID ACCOUNT NO")

cur.execute( " select count(\*) from SavingAccountCustomers where AccountNo= :param1", {'param1':accountnou})

data= cur.fetchall()

if( data[0][0]== 1):

cur.execute( " select Balance from SavingAccountCustomers where AccountNo= :param1", { 'param1':accountnou})

data= cur.fetchall()

amount= int( raw\_input(" enter the amount to be withdrwan"))

if( amount< 0):

raise Exception(" the amount is illogical , you cant withdrwa nothing")

if( int(data[0][0]) < amount):

raise Exception( " THE AVAILABLE BALANCE IS LESS THAN THAT WISHED TO BE WITHDRWAN")

cur.execute(" select Lastdate from SavingAccountCustomers where AccountNo= :param1", {'param1':accountnou})

data= cur.fetchall()

if( str(data[0][0])== 'None'):

cur.execute( " select Balance from SavingAccountCustomers where AccountNo= :param1", { 'param1':accountnou})

data= cur.fetchall()

previousbalance= int( data[0][0])

newbalance= previousbalance - amount

print("ch1")

#cur.execute( " insert into CustomerTransactions values( auto\_increment.nextval, 'Withdrawl',:param1, :param2,:param3, sysdate",{ 'param1':amount, 'param2':newbalance,'param3':accountnou})

cur.execute("insert into CustomerTransactions values(auto\_increment.nextval, 'Withdrawl',:param1, :param2, :param3, sysdate)", {'param1':amount, 'param2':newbalance, 'param3':accountnou})

cur.execute( " update SavingAccountCustomers set Balance= :param1, LastDate= sysdate where AccountNo= :param2",{ 'param1':newbalance, 'param2':accountnou})

print("The updated details below")

cur.execute(" select \* from SavingAccountCustomers where AccountNo= :param1", {'param1':accountnou})

print( cur.fetchall())

cur.execute( " select \* from CustomerTransactions where AccountNo= :param1", {'param1': accountnou})

print( cur.fetchall())

con.commit()

raise Exception(" SUCESSS")

cur.execute( " select extract( year from LastDate ), extract( month from LastDate) from SavingAccountCustomers where AccountNo= :param1", {'param1':accountnou})

data= cur.fetchall()

currentyear= datetime.now().year

currentmonth= datetime.now().month

if( int( data[0][0]) < currentyear):

if( int( data[0][1]) < currentmonth):

cur.execute( " select Withdrawls from SavingAccountCustomers where AccountNo= :param1", {'param1': accountnou})

data= cur.fetchall()

if( int( data[0][0]) != 0):

cur.execute( " select Balance from SavingAccountCustomers where AccountNo= :param1", { 'param1':accountnou})

data= cur.fetchall()

previousbalance= int( data[0][0])

newbalance= previousbalance - amount

cur.execute( " insert into CustomerTransactions values( auto\_increment.nextval, 'Withdrawl',:param1, :param2,:param3, sysdate) ",{ 'param1':amount, 'param2':newbalance,'param3':accountnou})

cur.execute( " update SavingAccountCustomers set Balance= :param1, LastDate= sysdate where AccountNo= :param2",{ 'param1':newbalance, 'param2':accountnou})

print("The updated details below")

cur.execute(" select \* from SavingAccountCustomers where AccountNo= :param1", {'param1':accountnou})

print( cur.fetchall())

cur.execute( " select \* from CustomerTransactions where AccountNo= :param1", {'param1': accountnou})

print( cur.fetchall())

con.commit()

raise Exception(" SUCESSS")

else:

raise Exception( " yuo have passed your withdrwal limit")

cur.execute( " select count(\*) from CurrentAccountCustomers where AccountNo= :param1", {'param1':accountnou})

data= cur.fetchall()

if( data[0][0] == 1):

amount= int( raw\_input( " enter the amount you wish to withdraw"))

if( amount< 0):

raise Exception( " enter a valid amount ")

cur.execute( " select Balance from CurrentAccountCustomers where AccountNo= :param1", {'param1': accountnou})

data= cur.fetchall()

allowedbalance= int( data[0][0]) -5000

if( amount> allowedbalance):

raise Exception(" you don't have enough cash fro withdrawl while maintaining min 5K balance")

newbalance= allowedbalance- amount

newbalance += 5000

cur.execute( " update CurrentAccountCustomers set Balance= :param1 where AccountNo= :param2", {'param1': newbalance, 'param2':accountnou})

cur.execute( " insert into CustomerTransactions values( auto\_increment.nextval, 'Withdrawl', :param1, :param2, :param3, sysdate)", {'param1':amount, 'param2':newbalance, 'param3':accountnou})

print("The updated details below")

cur.execute(" select \* from SavingAccountCustomers where AccountNo= :param1", {'param1':accountnou})

print( cur.fetchall())

cur.execute( " select \* from CustomerTransactions where AccountNo= :param1", {'param1': accountnou})

print( cur.fetchall())

con.commit()

raise Exception("Sucess")

except Exception as err:

print('caught this message', repr(err))

return

def password\_check( self, passwordc, passwordcheck):

return ( Counter(passwordc)== Counter(passwordcheck))

def deposit\_money(self):

accountnou= raw\_input(" enter the account number in which you want to deposit the money")

con= cx\_Oracle.connect("user1/user123")

cur= con.cursor()

cur.execute( " select count(\*) from CustomerAccountDetails where AccountNo= :param1", {'param1':accountnou})

data= cur.fetchall()

try:

if( data[0][0]==0):

raise Exception(" ACCOUNT DOESN'T EXSISTS , ENTER A VALID ACCOUNT NO")

amount= int(raw\_input(" enter the amount to be deposited"))

if( amount <=0):

raise Exception( "ENTER VALID AMOUNT FOR DEPOSITION ")

cur.execute( " select count(\*) from SavingAccountCustomers where AccountNo= :param1", {'param1':accountnou})

data= cur.fetchall()

if( data[0][0] == 1):

cur.execute( "select Balance from SavingAccountcustomers where AccountNo= :param1", {'param1':accountnou})

data= cur.fetchall()

print(" previous balance in acount no", accountnou, "is", data[0][0])

previousbalance= int(data[0][0])

newbalance= previousbalance + amount

cur.execute( " update SavingAccountCustomers set Balance= :param1 where AccountNo= :param2", { 'param1':newbalance, 'param2':accountnou})

print("updated details below")

cur.execute( "select \* from SavingAccountcustomers where AccountNo= :param1", {'param1':accountnou})

print(cur.fetchall())

print("recording this transaction in transaction history")

cur.execute("insert into CustomerTransactions values(auto\_increment.nextval, 'Deposit',:param1, :param2, :param3, sysdate)", {'param1':amount, 'param2':newbalance, 'param3':accountnou})

con.commit()

cur.execute( " select count(\*) from CurrentAccountCustomers where AccountNo= :param1", {'param1':accountnou})

data= cur.fetchall()

if( data[0][0] == 1):

cur.execute( "select Balance from CurrentAccountcustomers where AccountNo= :param1", {'param1':accountnou})

data= cur.fetchall()

print(" previous balance in acount no", accountnou, "is", data[0][0])

previousbalance= int(data[0][0])

newbalance= prevoiusbalance + amount

cur.execute( " update CurrentAccountCustomers set Balance= :param1 where AccountNo= :param2", { 'param1':newbalance, 'param2':accountnou})

print("updated details below")

cur.execute( "select \* from CurrentAccountcustomers where AccountNo= :param1", {'param1':accountnou})

print(cur.fetchall())

print("recording this transaction in transaction history")

cur.execute("insert into CustomerTransactions values(auto\_increment.nextval, 'Deposit',:param1, :param2, :param3, sysdate)", {'param1':amount, 'param2':newbalance, 'param3':accountnou})

con.commit()

except Exception as err:

print('caught this error', repr(err))

def address\_change( self, accountno):

newaddressline1= raw\_input(" enter the line 1 of the new address")

newaddressline2= raw\_input(" enter the line2 of the new address")

newcity= raw\_input(" enter new city name")

newpin= raw\_input(" enter the new PIN")

try:

con= cx\_Oracle.connect("user1/user123")

cur= con.cursor()

cur.execute(" update CustomerAccountDetails Set AddressLine1= :param2 , AddressLine2= :param3 , City= :param4 , Pin= :param5 where AccountNo= :param1", { 'param1':accountno, 'param2': newaddressline1, 'param3': newaddressline2, 'param4': newcity, 'param5':newpin})

cur.execute( " select \* from CustomerAccountDetails where AccountNo= :param1", {'param1':accountno})

print(cur.fetchall())

print("SUCESSS")

con.commit()

except Exception as err:

print('caught this error', repr(err))

def \_\_init\_\_(self):

print(" welcome to the BAJORIA BANK SERVICES, inorder to sign in provide your appropiate credentilas as asked for")

accountnoc= raw\_input("enter the account number")

con= cx\_Oracle.connect("user1/user123")

cur= con.cursor()

cur.execute( " select Password,Status from CustomerAccountDetails where AccountNo= :param1", { 'param1':accountnoc})

data= cur.fetchall()

passwordcheck= str(data[0][0])

statuscheck= str( data[0][1])

try:

passflag= False

trials =0

while( passflag == False and trials !=3):

passwordc= raw\_input(" enter the password")

if( statuscheck == 'LOCKED'):

raise Exception(" The Account is locked contact adminstrator for further process")

passflag=self.password\_check( passwordc, passwordcheck)

if(passflag == False):

trials += 1

if( trials==3):

con= cx\_Oracle.connect("user1/user123")

cur= con.cursor()

cur.execute(" update CustomerAccountDetails set Status= 'LOCKED' where AccountNo= :param1", {'param1': accountnoc})

cur.execute(" select \* from CustomerAccountDetails where AccountNo= :param1",{ 'param1' : accountnoc})

print( cur.fetchall())

raise Exception(" 3 consecutive retrials are over contact adminstrator for the account is now locked")

except Exception as err:

print('Caught this error', repr(err))

return

con= cx\_Oracle.connect("user1/user123")

cur= con.cursor()

cur.execute( " select Fname, Lname from CustomerAccountDetails where AccountNo= :param1", { 'param1': accountnoc})

data= cur.fetchall()

print(" welcome ", data[0][1], data[0][0])

print(" select what service you are here to avail based on the serial no")

print("1. Address Change")

print("2. Money Deposit")

print("3. Money withdrwal")

print("4. Print Statement")

print("5. Transfer Money")

print("6. Account Closure")

print("7. Customer LogOut")

choice= int(raw\_input( " select the service you want to avail according to the serial number"))

if( choice== 1):

self.address\_change(accountnoc)

elif( choice== 2):

self.deposit\_money()

elif(choice==3):

self.withdraw\_money(accountnoc)

elif(choice==4):

self.print\_statement(accountnoc)

elif(choice==5):

self.transfer\_money( accountnoc)

elif( choice==6):

self.account\_closure( accountnoc)

elif( choice== 7):

self.logout()

return

print(" BAJORIA BANK SERVICES")

print("MAIN MENU")

print(" enter the choice for what purpose you are here according to the serial no.")

print( "1. Sign Up (New Customer)")

print( "2. Sign In (Exsisting Customer)")

print ( "3. Admin Sign In ")

print( "4. Quit")

choice= int(raw\_input("enter the choice for what purpose you are here according to the serial no."))

if(choice==1):

#creating the child of a SignUp class

print(" welcome to the BAJORIA BANK SERVICES provide the details below to register yourself with us")

SignUpObj= SignUp()

print(SignUpObj)

#now we are sending the values to the database in multiple tables

SignUpObj.sending\_to\_database()

elif( choice== 2):

# creating the child of the SignIn class

SignInObj= SignIn()

elif( choice==3):

# letting signup as the admin

print(" this is the Admin login, there's only 1 adminstrator")

adminid= raw\_input(" enter the admin id")

adminpass= raw\_input(" enter the admin password")

con= cx\_Oracle.connect("user1/user123")

cur= con.cursor()

cur.execute( " select adminpass from AdminAccounts where adminid= :param1", {'param1':adminid})

data= cur.fetchall()

if( str( data[0][0]) != adminpass):

print("ADMINSTRATOR LOGIN DENIED")

else:

AdminSignInObj= AdminSignIn()

elif( choice ==4):

try:

#terminate the whole script"

print("IT WAS A PLEASURE SERVING YOU")

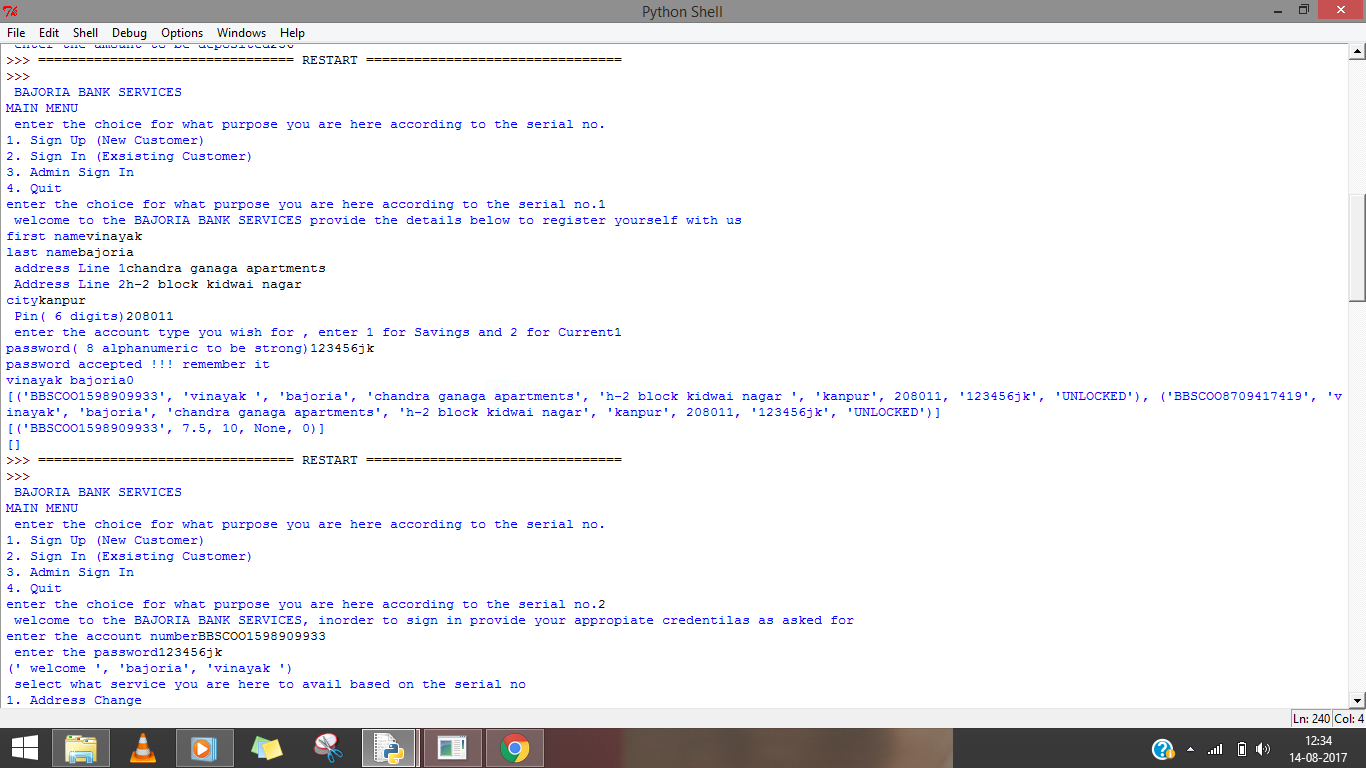
sys.exit()

except:

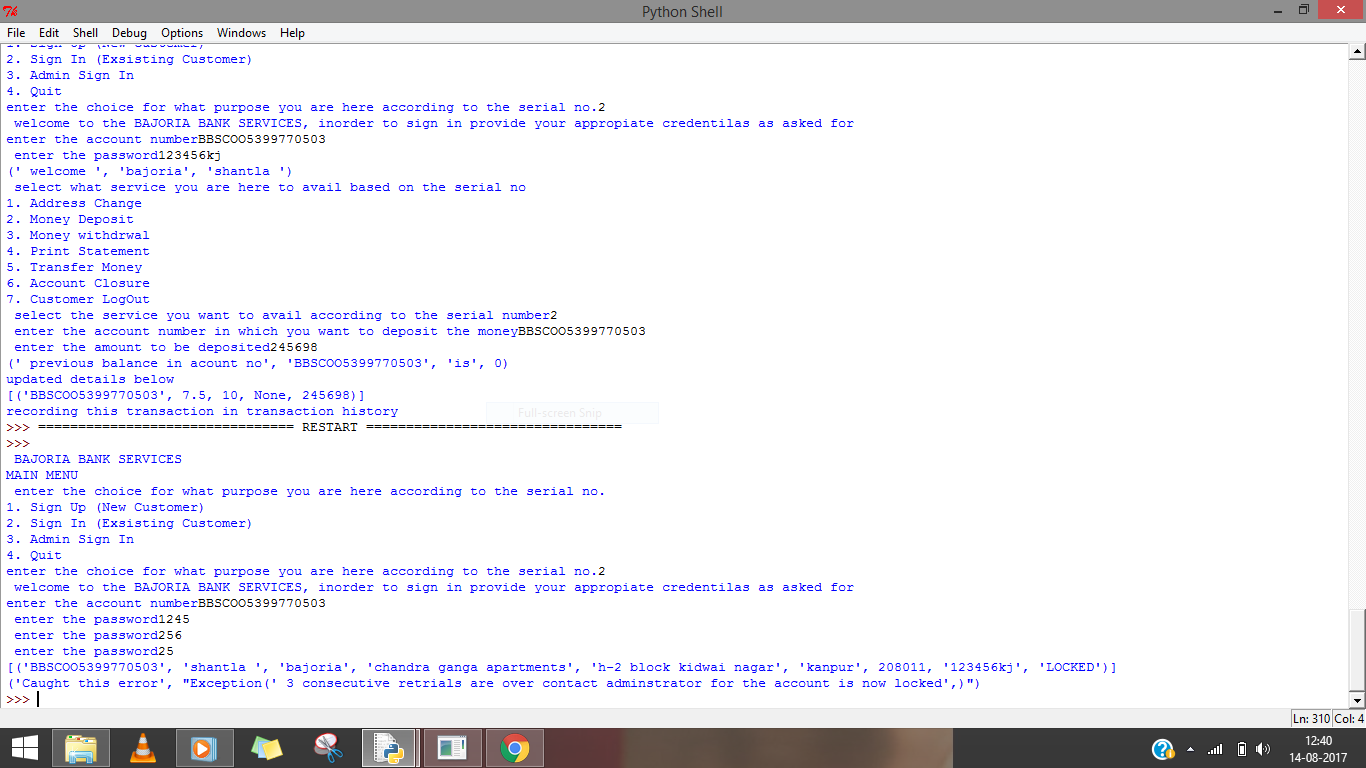
pass

**Output-**

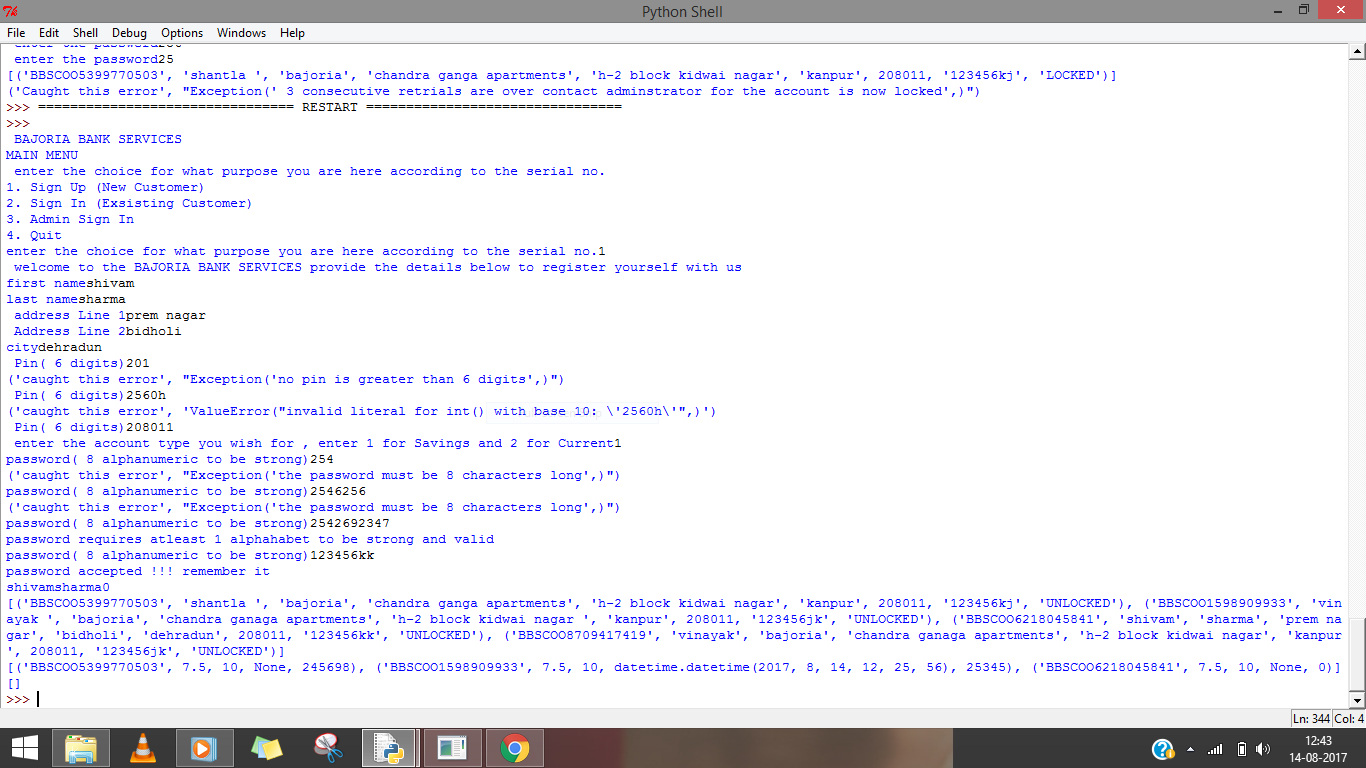
**Signup-**



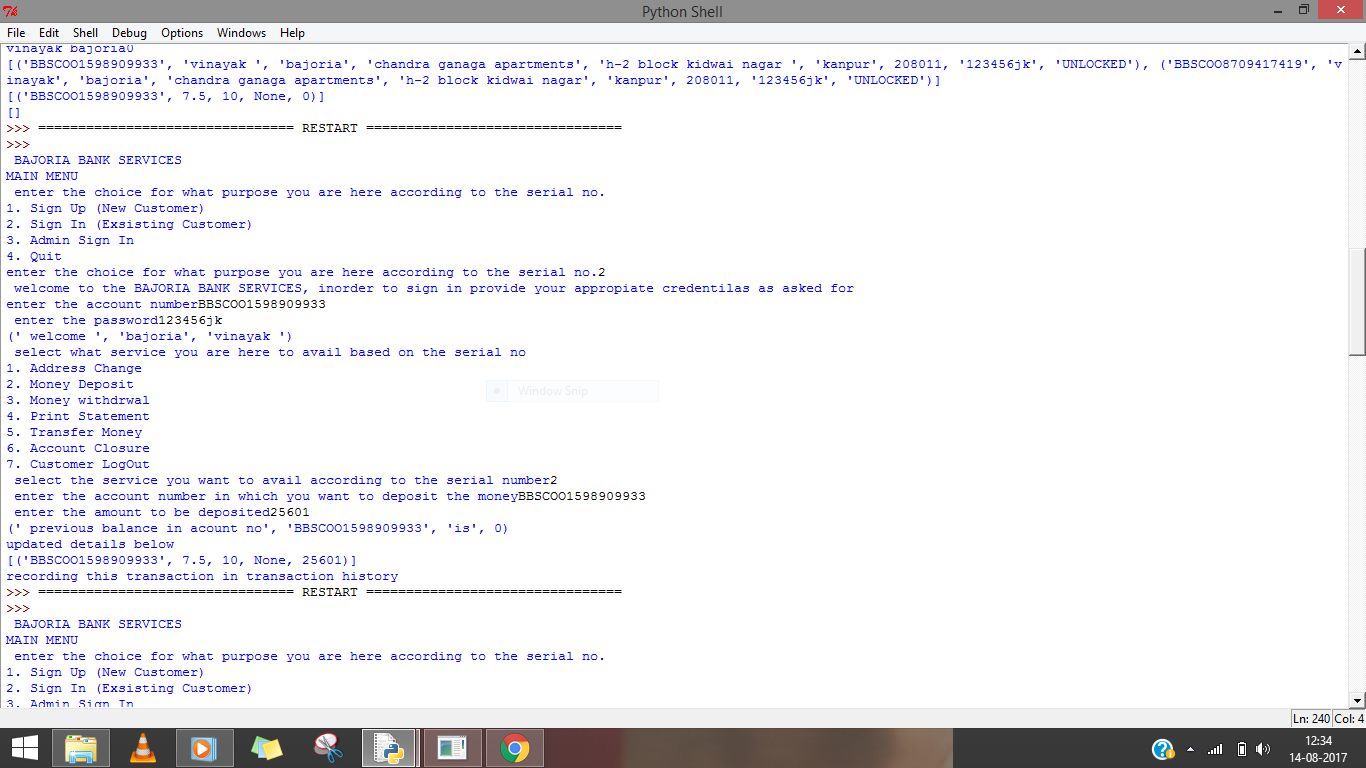
**Passwordtrials-**



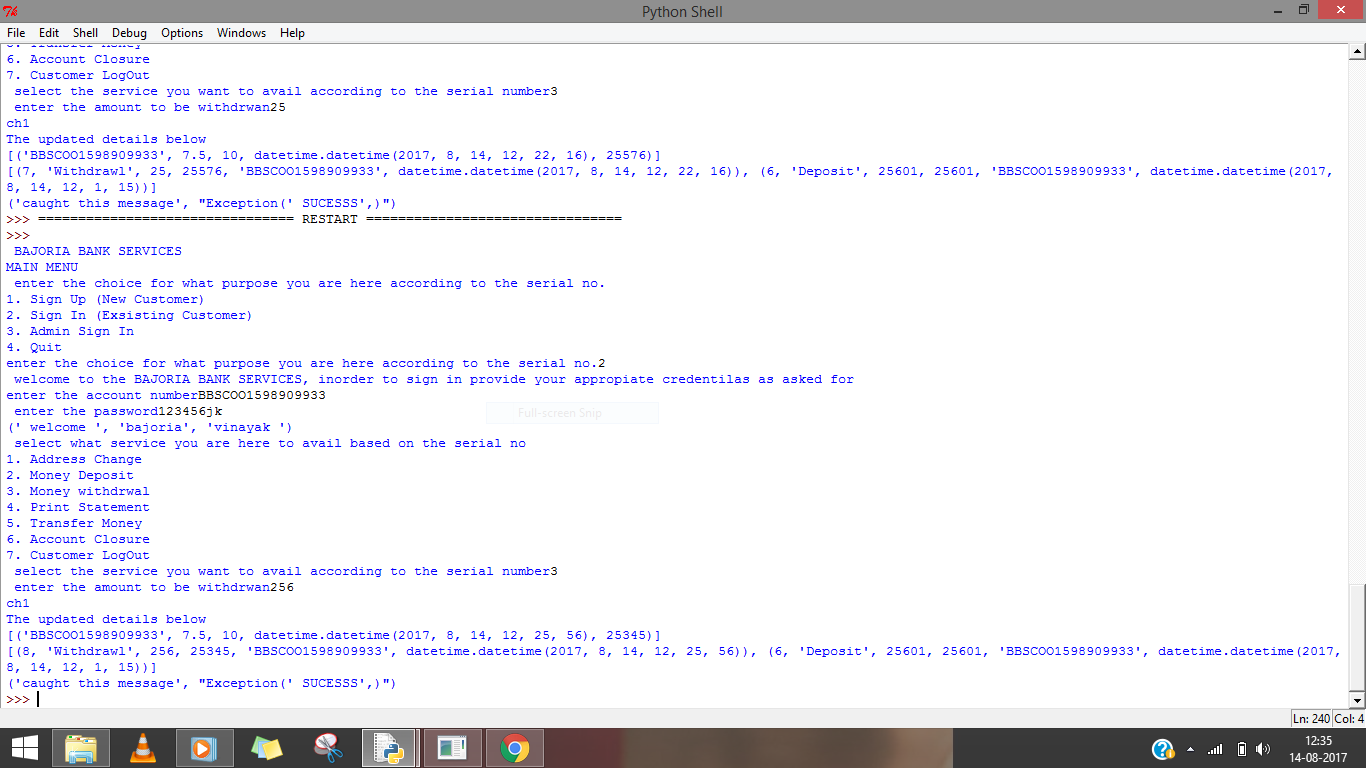
**Pintrials-**

****

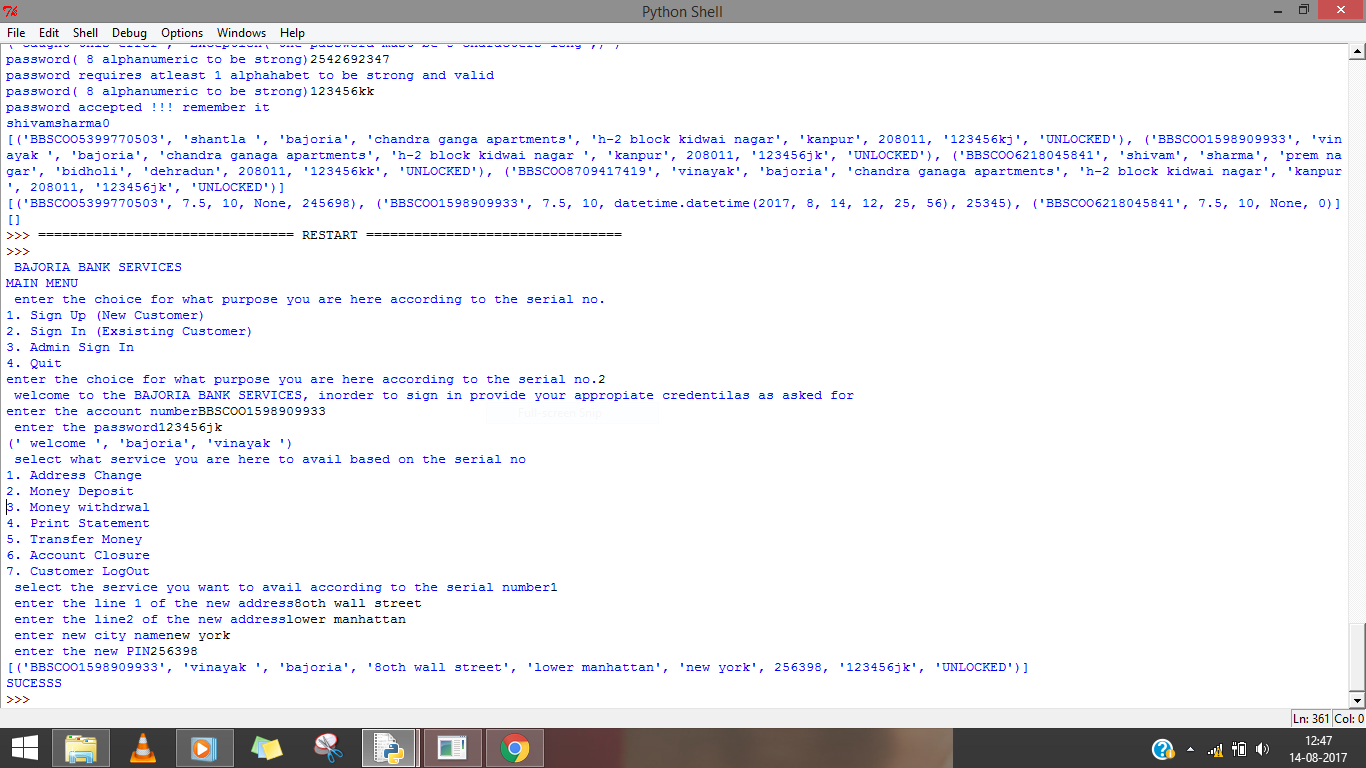
**Deposit after sign in-**

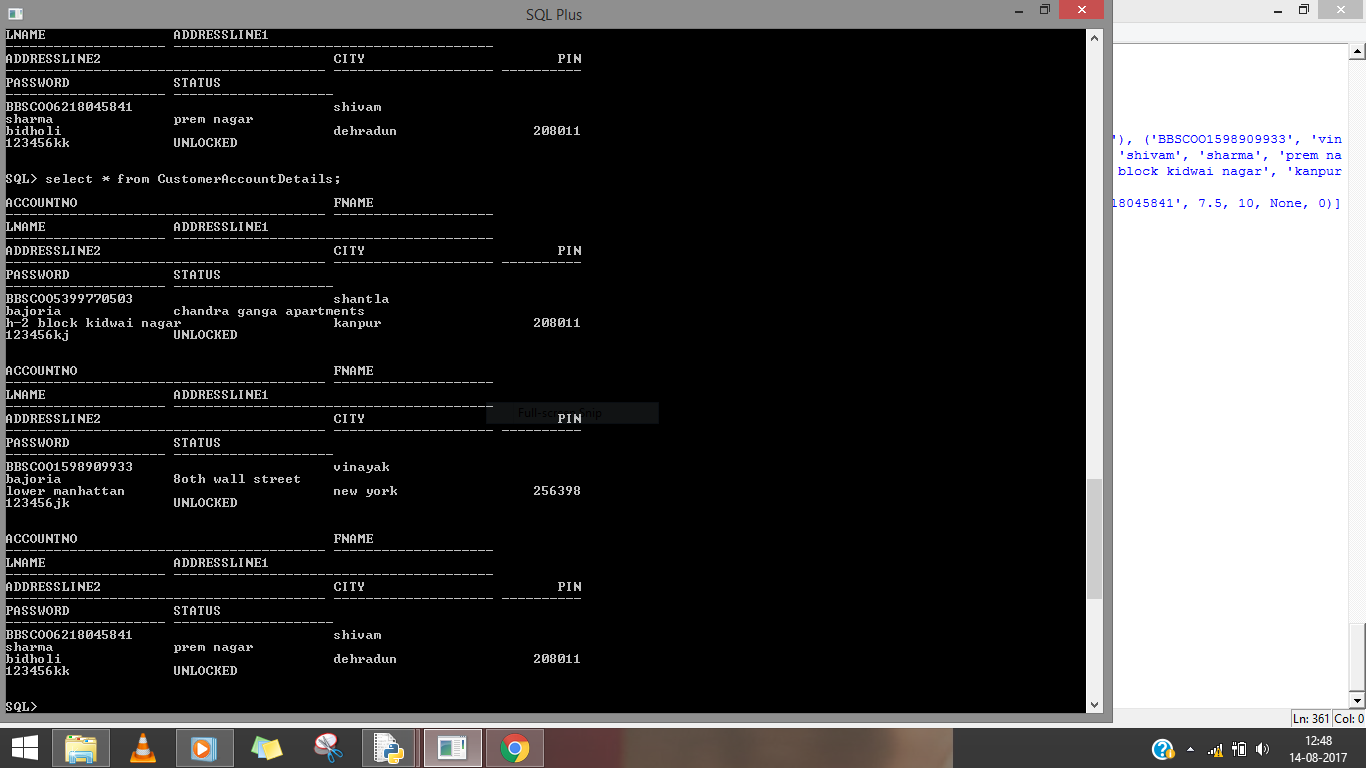
****

**Withdrawl after sign in-**

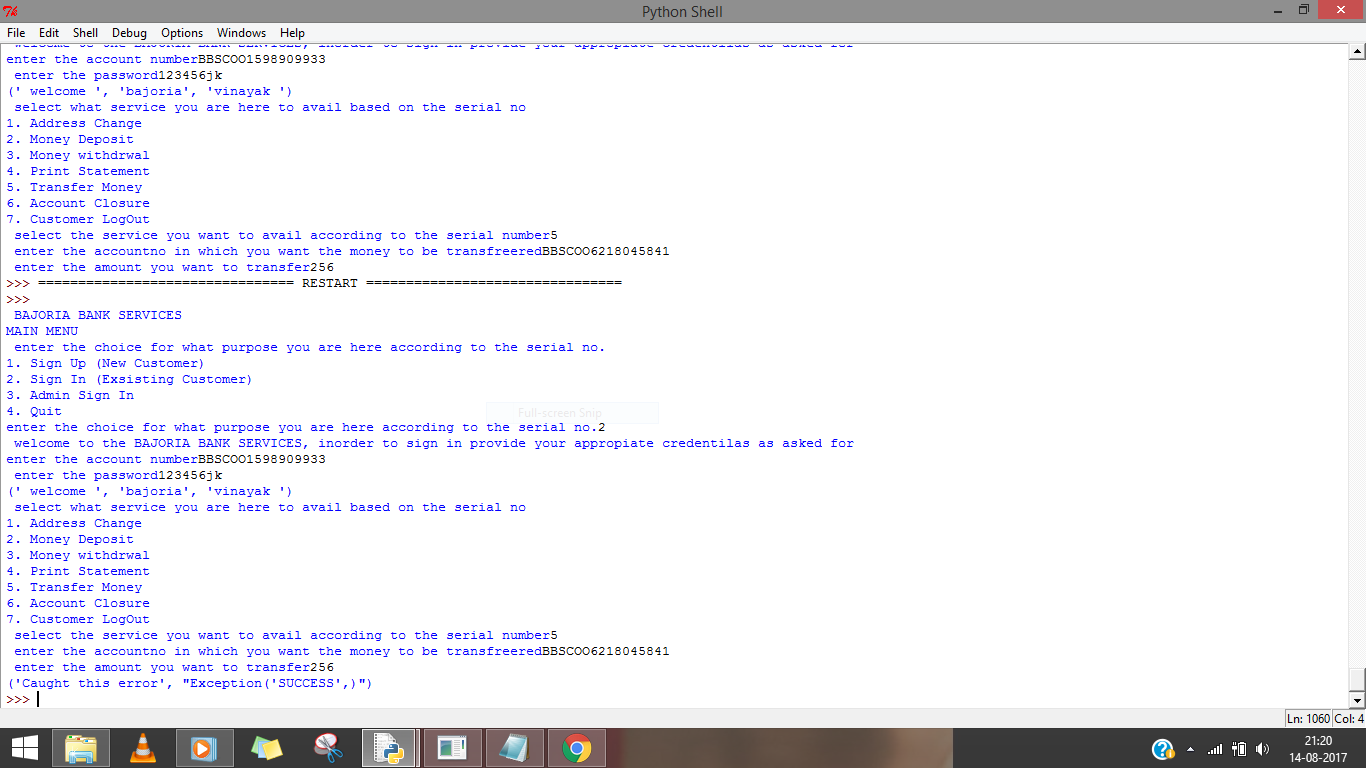
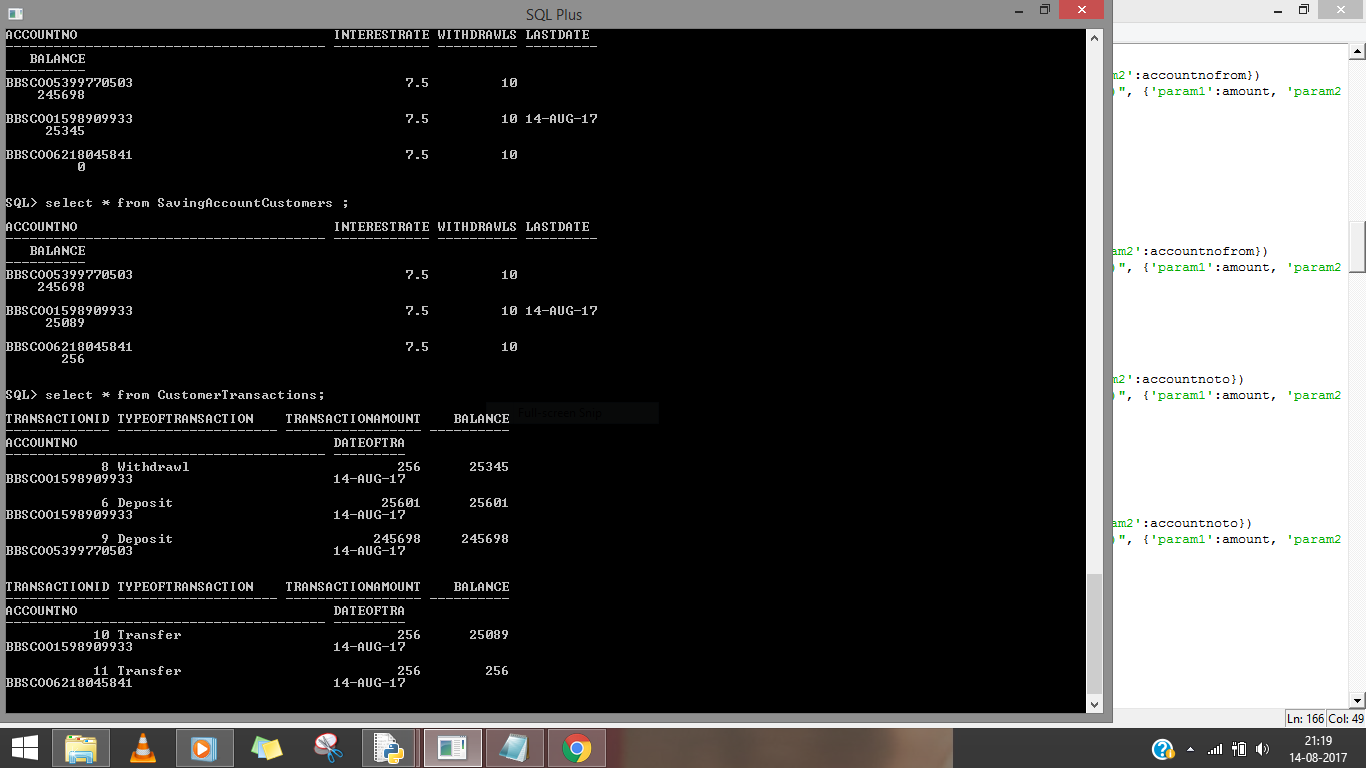


**Address change-**

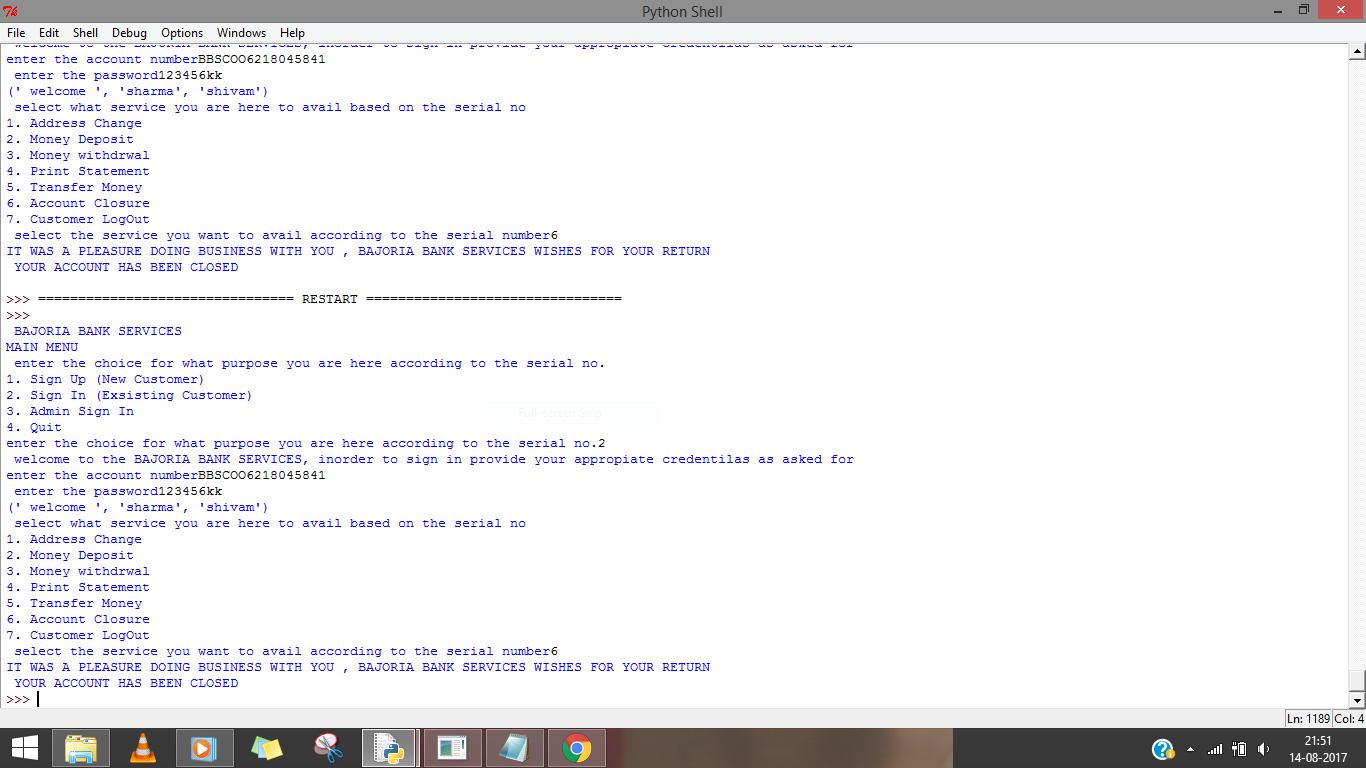
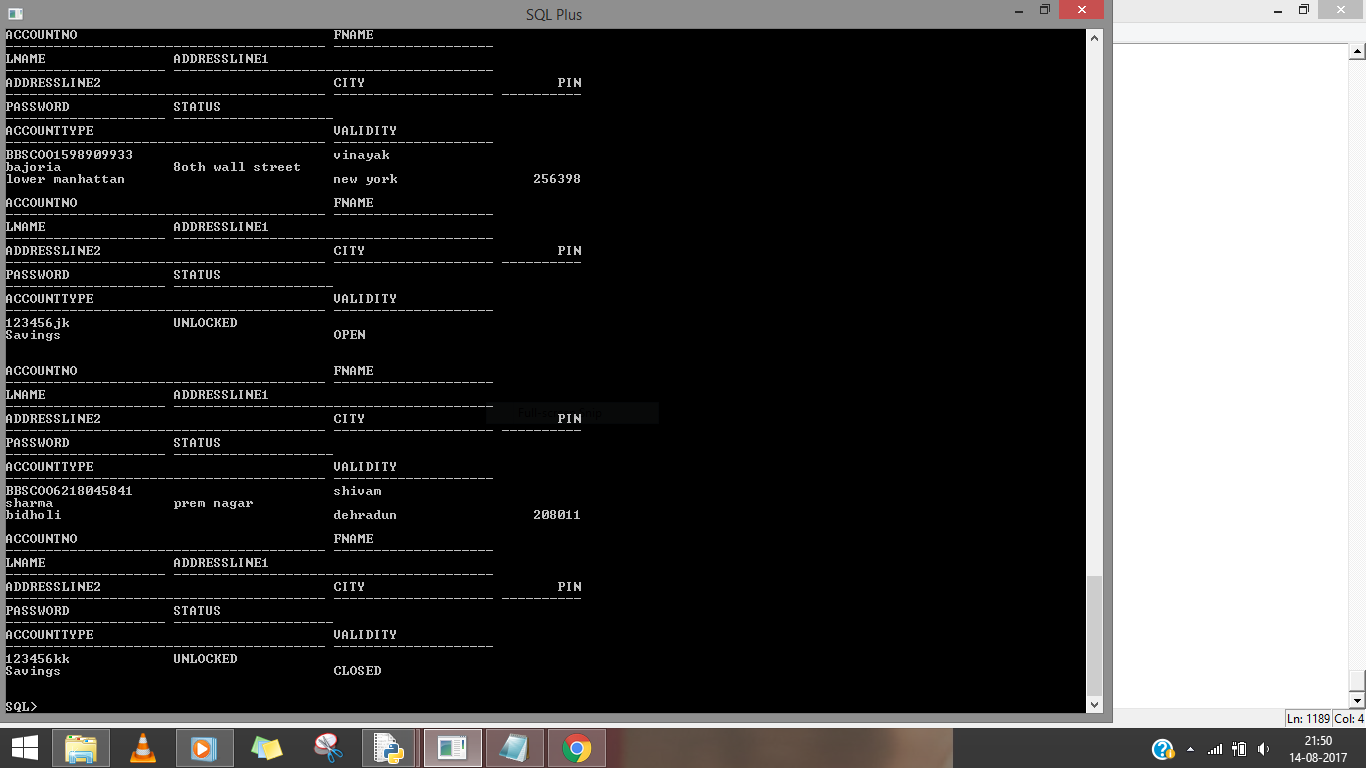
****

****

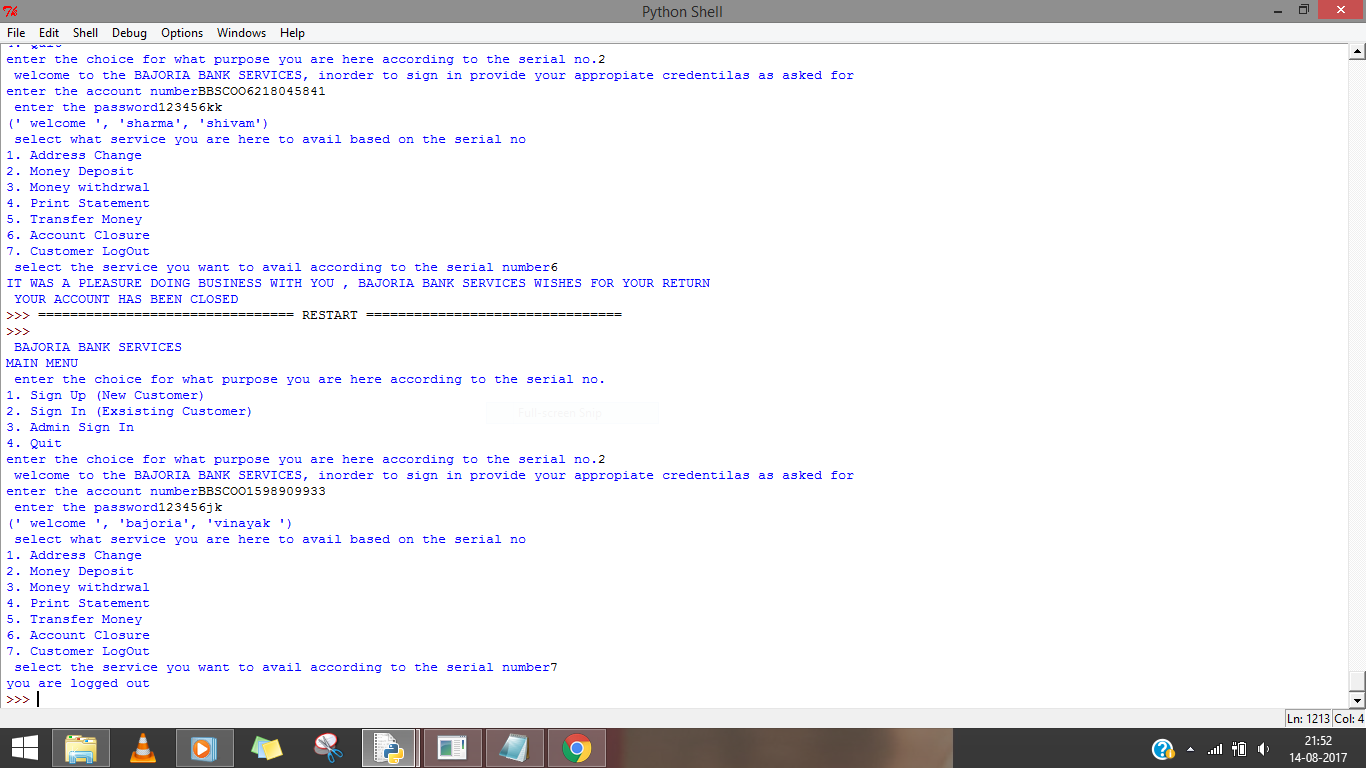
**Transfer-**

****

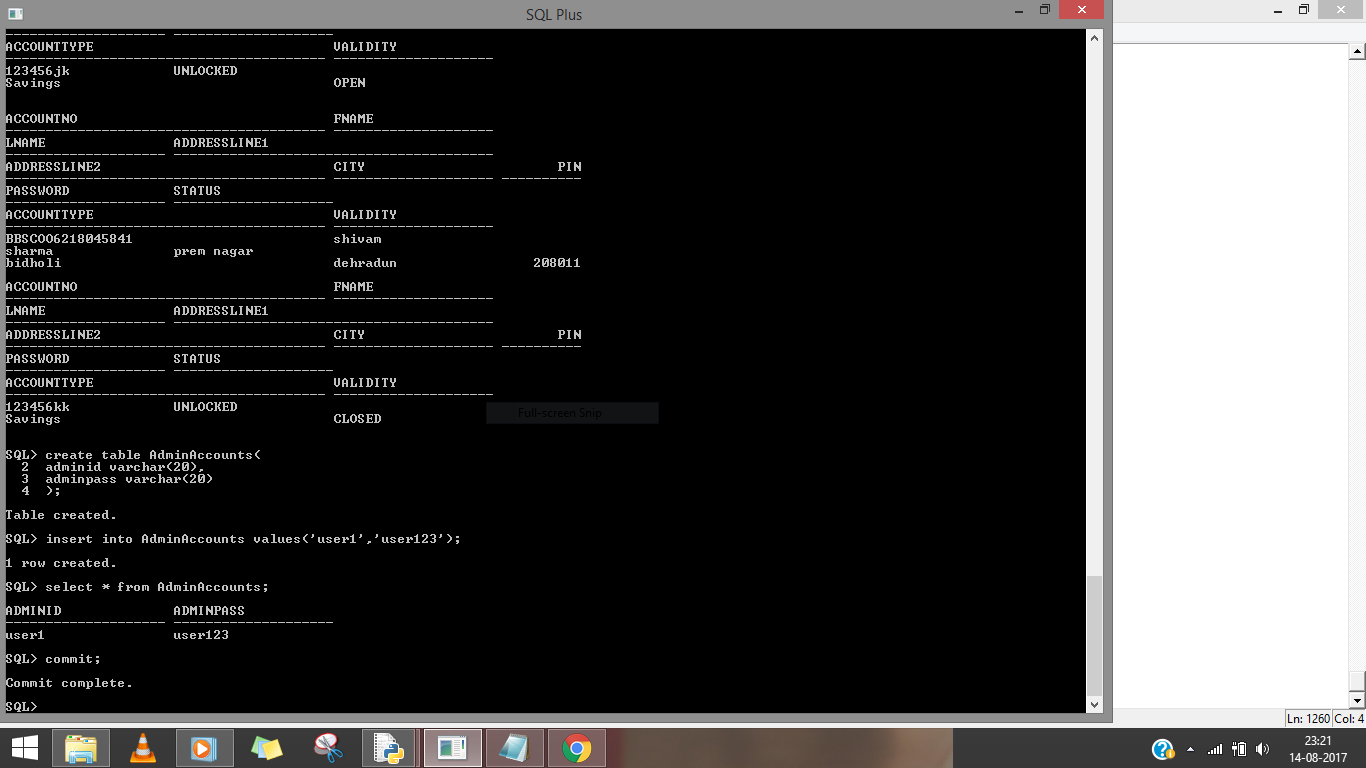
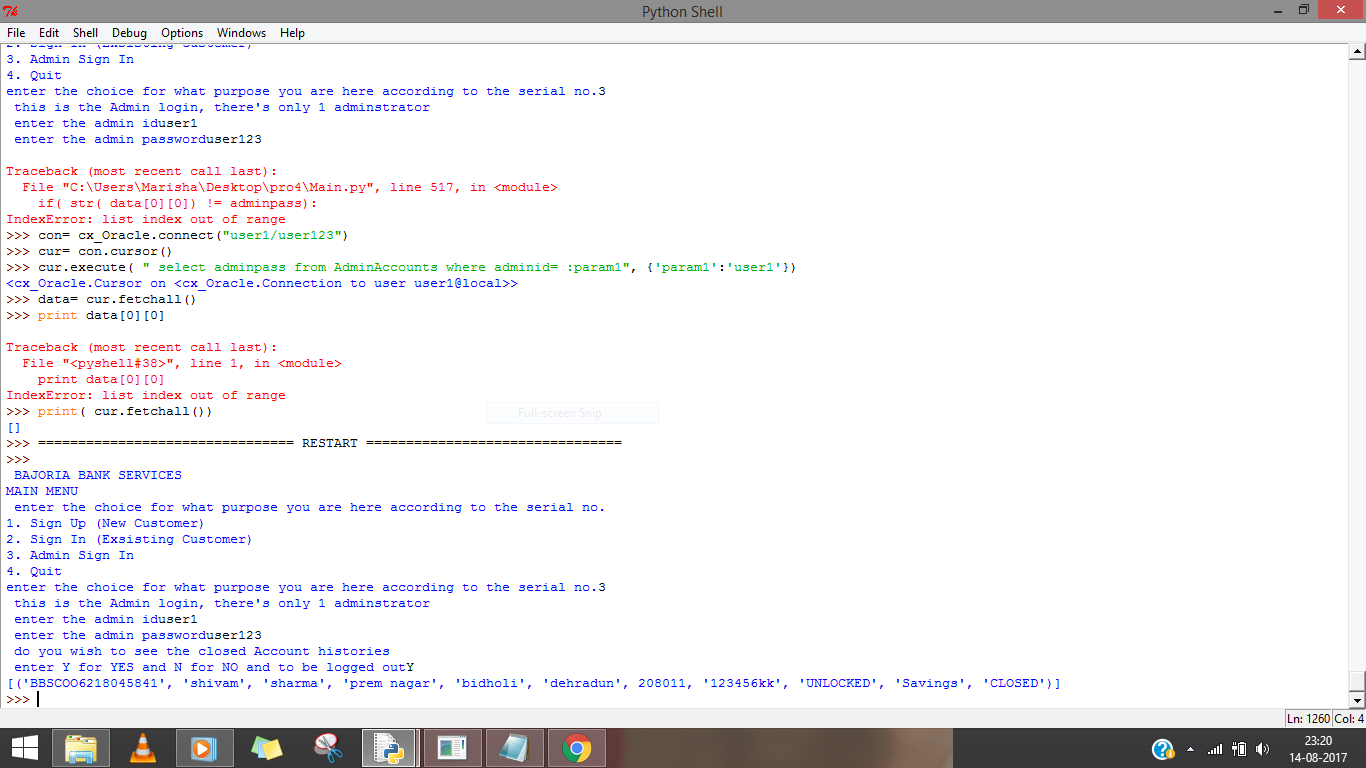
**Closure-**

****

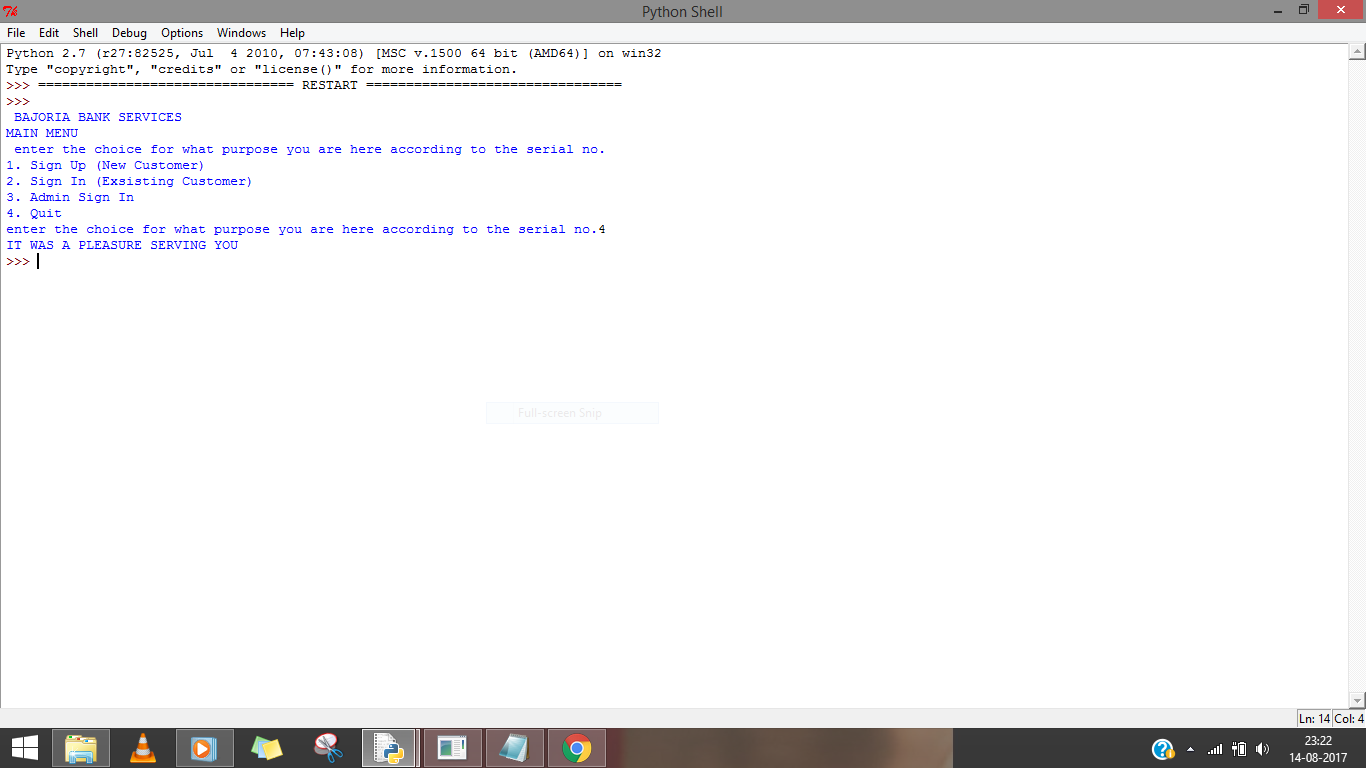
**Logout-**

****

**Admin related work-**

****

**Quit-**

****

**The project ends**

.

[1] M. Nosrati, "Python: An appropriate language for real world programming", World

Applied Programming, vol. 1, no. 2, p. 8, 2011.