import mysql.connector

from mysql.connector.locales.eng import client\_error

import sys

from tkinter import \*

from tkinter.font import Font

from tkinter import ttk

from tkinter import messagebox

from PIL import ImageTk, Image

import time

class Connection(mysql.connector.connection.MySQLConnection):

def \_\_init\_\_(self, host, username, password, \*\*kwargs):

super().\_\_init\_\_(host=host, user=username, password=password)

self.crs = self.cursor(buffered=True)

def create\_database(self):

# Checking if 'vectorgaming' database exists

try:

self.crs.execute('USE Bank\_Management;')

# if the database does not exist, create database

except mysql.connector.errors.ProgrammingError:

# If vectorgaming database does not exist

self.crs.execute('CREATE DATABASE Bank\_Management;')

self.crs.execute('USE Bank\_Management;')

self.commit()

class SampleApp(Tk):

def \_\_init\_\_(self, \*args, \*\*kwargs):

Tk.\_\_init\_\_(self, \*args, \*\*kwargs)

self.title('Bank Accounts Manager')

self.iconphoto(False,PhotoImage(file='images/bank.png'))

global c

c=ImageTk.PhotoImage(Image.open("images/canvas.png"))

heading\_label = Label(self,

text='NEUTRINOVAULT BANK',

font=('orbitron',40,'bold'),

foreground='#ffffff',

background='#545454')

heading\_label.pack(pady=5)

frame\_1 = Frame(self,bg="#737373")

frame\_1.pack(fill='both',expand=True)

canvas = Canvas(frame\_1, bd=0, highlightthickness=0)

canvas.create\_image(0,0, image=c, anchor="nw")

canvas.pack(fill="both",expand=True)

def resizer(e):

global bg, resized\_bg, new\_bg

bg=Image.open("images/canvas.png")

resized\_bg=bg.resize((e.width,e.height),Image.ANTIALIAS)

new\_bg=ImageTk.PhotoImage(resized\_bg)

canvas.create\_image(0,0, image=new\_bg, anchor="nw")

frame\_1.bind('<Configure>',resizer)

self.username\_label = Label(

master=canvas, text="Enter MySQL username:",

font=Font(family="system", size=13))

self.username\_box = Entry(canvas, width=40)

self.password\_label =Label(

master=canvas, text="Enter MySQL Password:",

font=Font(family="system", size=13))

self.password\_box = Entry(canvas, show="\*", width=40)

def login\_result():

"""

checks if the given username and password is correct

"""

global password, username

password = self.password\_box.get()

username = self.username\_box.get()

try:

# try connecting to the mysql server with the entered username and password

con = mysql.connector.connect(

host="localhost",

user=username,

password=password

)

except:

# if password is wrong, display a warning

wrong\_password = Label(

canvas, text='Incorrect Password !', fg='red')

wrong\_password.grid(row=2, column=0)

else:

# if the password is correct, close the password window and display main window

con = mysql.connector.connect(

host="localhost",

user=username,

password=password

)

self.con = Connection('localhost', username, password)

self.con.create\_database()

canvas.destroy()

# Create a database or connect to one that exists

con = mysql.connector.connect(

host="localhost",

user=username,

password=password

)

# Create a cursor instance

c = con.cursor()

c.execute('USE Bank\_Management;')

# Create Table

c.execute("""CREATE TABLE if not exists Bank\_Accounts (

first\_name text,

last\_name text,

id integer,

address text,

mobile\_no bigint,

current\_balance decimal(21,1) not null default 0)

""")

# Commit changes

con.commit()

# Close our connection

con.close()

def query\_database():

# Clear the Treeview

for record in my\_tree.get\_children():

my\_tree.delete(record)

# Create a database or connect to one that exists

con = mysql.connector.connect(

host="localhost",

user=username,

password=password

)

# Create a cursor instance

c = con.cursor()

c.execute('USE Bank\_Management;')

c.execute("SELECT \* FROM Bank\_Accounts")

records = c.fetchall()

# Add our data to the screen

global count

count = 0

#for record in records:

# print(record)

for record in records:

if count % 2 == 0:

my\_tree.insert(parent='', index='end', iid=count, text='', values=(record[0], record[1], record[2], record[3], record[4], record[5]), tags=('evenrow',))

else:

my\_tree.insert(parent='', index='end', iid=count, text='', values=(record[0], record[1], record[2], record[3], record[4], record[5]), tags=('oddrow',))

# increment counter

count += 1

# Commit changes

con.commit()

# Close our connection

con.close()

def search\_records():

lookup\_record = search\_entry.get()

# close the search box

search.destroy()

# Clear the Treeview

for record in my\_tree.get\_children():

my\_tree.delete(record)

# Create a database or connect to one that exists

con = mysql.connector.connect(

host="localhost",

user=username,

password=password

)

# Create a cursor instance

c = con.cursor()

c.execute('USE Bank\_Management;')

search\_query="SELECT \* FROM Bank\_Accounts WHERE id = %s"

values\_s=tuple(lookup\_record)

c.execute(search\_query,values\_s)

records = c.fetchall()

# Add our data to the screen

global count

count = 0

#for record in records:

# print(record)

for record in records:

if count % 2 == 0:

my\_tree.insert(parent='', index='end', iid=count, text='', values=(record[0], record[1], record[2], record[3], record[4], record[5]), tags=('evenrow',))

else:

my\_tree.insert(parent='', index='end', iid=count, text='', values=(record[0], record[1], record[2], record[3], record[4], record[5]), tags=('oddrow',))

# increment counter

count += 1

# Commit changes

con.commit()

# Close our connection

con.close()

def lookup\_records():

global search\_entry, search

search = Toplevel(self)

search.title("Check Records")

search.geometry("400x200")

search.iconbitmap('images/bank.ico')

# Create label frame

search\_frame = LabelFrame(search, text="Enter User ID")

search\_frame.pack(padx=10, pady=10)

# Add entry box

search\_entry = Entry(search\_frame, font=("Helvetica", 18))

search\_entry.pack(pady=20, padx=20)

# Add button

search\_button = Button(search, text="Search Record", command=search\_records)

search\_button.pack(padx=20, pady=20)

# Add Menu

my\_menu = Menu(self)

self.config(menu=my\_menu)

#Search Menu

search\_menu = Menu(my\_menu, tearoff=0)

my\_menu.add\_cascade(label="Search", menu=search\_menu)

# Drop down menu

search\_menu.add\_command(label="Search", command=lookup\_records)

search\_menu.add\_separator()

search\_menu.add\_command(label="Reset", command=query\_database)

# Add Some Style

style = ttk.Style()

# Pick A Theme

style.theme\_use('default')

# Configure the Treeview Colors

style.configure("Treeview",

background="#D3D3D3",

foreground="black",

rowheight=25,

fieldbackground="#D3D3D3")

# Change Selected Color

style.map('Treeview',

background=[('selected', "#347083")])

# Create a Treeview Frame

tree\_frame = Frame(frame\_1)

tree\_frame.pack(pady=10)

# Create a Treeview Scrollbar

tree\_scroll = Scrollbar(tree\_frame)

tree\_scroll.pack(side=RIGHT, fill=Y)

# Create The Treeview

my\_tree = ttk.Treeview(tree\_frame, yscrollcommand=tree\_scroll.set, selectmode="extended")

my\_tree.pack()

# Configure the Scrollbar

tree\_scroll.config(command=my\_tree.yview)

# Define Our Columns

my\_tree['columns'] = ("First Name", "Last Name", "ID", "Address", "Mobile No.", "Current Balance")

# Format Our Columns

my\_tree.column("#0", width=0, stretch=NO)

my\_tree.column("First Name", anchor=W, width=200)

my\_tree.column("Last Name", anchor=W, width=200)

my\_tree.column("ID", anchor=CENTER, width=100)

my\_tree.column("Address", anchor=CENTER, width=360)

my\_tree.column("Mobile No.", anchor=CENTER, width=250)

my\_tree.column("Current Balance", anchor=CENTER, width=200)

# Create Headings

my\_tree.heading("#0", text="", anchor=W)

my\_tree.heading("First Name", text="First Name", anchor=W)

my\_tree.heading("Last Name", text="Last Name", anchor=W)

my\_tree.heading("ID", text="ID", anchor=CENTER)

my\_tree.heading("Address", text="Address", anchor=CENTER)

my\_tree.heading("Mobile No.", text="Mobile No.", anchor=CENTER)

my\_tree.heading("Current Balance", text="Current Balance", anchor=CENTER)

# Create Striped Row Tags

my\_tree.tag\_configure('oddrow', background="white")

my\_tree.tag\_configure('evenrow', background='#a6a6a6')

# Add Record Entry Boxes

data\_frame = LabelFrame(frame\_1, text='Record')

data\_frame.pack(fill="x", expand="yes", padx=20)

h1\_label = Label(data\_frame, text="User Details")

h1\_label.grid(row=0, column=0, padx=10, pady=5)

fn\_label = Label(data\_frame, text="First Name")

fn\_label.grid(row=1, column=0, padx=10, pady=5)

fn\_entry = Entry(data\_frame)

fn\_entry.grid(row=1, column=1, padx=10, pady=5)

ln\_label = Label(data\_frame, text="Last Name")

ln\_label.grid(row=1, column=2, padx=10, pady=5)

ln\_entry = Entry(data\_frame)

ln\_entry.grid(row=1, column=3, padx=10, pady=5)

id\_label = Label(data\_frame, text="ID")

id\_label.grid(row=1, column=4, padx=10, pady=5)

id\_entry = Entry(data\_frame)

id\_entry.grid(row=1, column=5, padx=10, pady=5)

id\_desc\_1 = Label(data\_frame, text="\*ID of a user should be unique\nand ∈ N and can not be updated")

id\_desc\_1.grid(row=2, column=5, padx=10, pady=5)

address\_label = Label(data\_frame, text="Address")

address\_label.grid(row=1, column=6, padx=10, pady=5)

address\_entry = Entry(data\_frame)

address\_entry.grid(row=1, column=7, padx=10, pady=5)

mobile\_label = Label(data\_frame, text="Mobile No.")

mobile\_label.grid(row=1, column=8, padx=10, pady=5)

mobile\_entry = Entry(data\_frame)

mobile\_entry.grid(row=1, column=9, padx=10, pady=5)

h2\_label = Label(data\_frame, text="Transaction Details")

h2\_label.grid(row=2, column=0, padx=10, pady=10)

withdraw\_label = Label(data\_frame, text="Withdrawn Amount")

withdraw\_label.grid(row=3, column=0, padx=10, pady=10)

withdraw\_entry = Entry(data\_frame)

withdraw\_entry.grid(row=3, column=1, padx=10, pady=10)

deposit\_label = Label(data\_frame, text="Deposited Amount")

deposit\_label.grid(row=3, column=2, padx=10, pady=10)

deposit\_entry = Entry(data\_frame)

deposit\_entry.grid(row=3, column=3, padx=10, pady=10)

# Move Row Up

def up():

rows = my\_tree.selection()

for row in rows:

my\_tree.move(row, my\_tree.parent(row), my\_tree.index(row)-1)

# Move Rown Down

def down():

rows = my\_tree.selection()

for row in reversed(rows):

my\_tree.move(row, my\_tree.parent(row), my\_tree.index(row)+1)

# Select Record

def select\_record(e):

# Clear entry boxes

fn\_entry.delete(0, END)

ln\_entry.delete(0, END)

id\_entry.delete(0, END)

address\_entry.delete(0, END)

mobile\_entry.delete(0, END)

withdraw\_entry.delete(0, END)

deposit\_entry.delete(0, END)

# Grab record Number

selected = my\_tree.focus()

# Grab record values

values = my\_tree.item(selected, 'values')

# outpus to entry boxes

fn\_entry.insert(0, values[0])

ln\_entry.insert(0, values[1])

id\_entry.insert(0, values[2])

address\_entry.insert(0, values[3])

mobile\_entry.insert(0, values[4])

# Remove one record

def remove\_one():

x = my\_tree.selection()[0]

my\_tree.delete(x)

oid = id\_entry.get()

# Create a database or connect to one that exists

con = mysql.connector.connect(

host="localhost",

user=username,

password=password

)

# Create a cursor instance

c = con.cursor()

c.execute('USE Bank\_Management;')

# Delete From Database

delete\_query="DELETE FROM Bank\_Accounts WHERE id = %s"

values\_d=tuple(oid)

c.execute(delete\_query,values\_d)

# Commit changes

con.commit()

# Close our connection

con.close()

# Clear The Entry Boxes

clear\_entries()

# Add a little message box for fun

messagebox.showinfo("Deleted!", "Your Record Has Been Deleted!")

# Remove all records

def remove\_all():

# Add a little message box for fun

response = messagebox.askyesno("WARNING!!!!", "This Will Delete EVERYTHING From The Table\nAre You Sure?!")

#Add logic for message box

if response == 1:

# Clear the Treeview

for record in my\_tree.get\_children():

my\_tree.delete(record)

# Create a database or connect to one that exists

con = mysql.connector.connect(

host="localhost",

user=username,

password=password

)

# Create a cursor instance

c = con.cursor()

c.execute('USE Bank\_Management;')

# Delete Everything From The Table

c.execute("DROP TABLE Bank\_Accounts")

# Commit changes

con.commit()

# Close our connection

con.close()

# Clear entry boxes if filled

clear\_entries()

# Recreate The Table

create\_table\_again()

# Clear entry boxes

def clear\_entries():

# Clear entry boxes

fn\_entry.delete(0, END)

ln\_entry.delete(0, END)

id\_entry.delete(0, END)

address\_entry.delete(0, END)

mobile\_entry.delete(0, END)

withdraw\_entry.delete(0, END)

deposit\_entry.delete(0, END)

# Update record

def update\_record():

# Grab the record number

selected = my\_tree.focus()

# Update record

first = fn\_entry.get()

last = ln\_entry.get()

oid = id\_entry.get()

address = address\_entry.get()

mobile = mobile\_entry.get()

withdraw = withdraw\_entry.get()

deposit = deposit\_entry.get()

# Update the database

# Create a database or connect to one that exists

con = mysql.connector.connect(

host="localhost",

user=username,

password=password

)

# Create a cursor instance

c = con.cursor()

c.execute('USE Bank\_Management;')

c.execute("SELECT \* FROM Bank\_Accounts")

records = c.fetchall()

for i in records:

if int(i[2])==int(oid):

cb=i[5]

my\_tree.item(selected, text="", values=(fn\_entry.get(), ln\_entry.get(), id\_entry.get(), address\_entry.get(), mobile\_entry.get(), cb))

update\_query="""UPDATE Bank\_Accounts SET

first\_name = %s,

last\_name = %s,

address = %s,

mobile\_no = %s

WHERE id = %s"""

vals=(first,last,address,mobile,oid)

c.execute(update\_query,vals)

con.commit()

he=Label(frame\_1, background="#737373", width=80, height=4)

he.place(relx = 0.5, rely = 1, anchor = 'center')

if withdraw != '' or deposit != '':

l=[withdraw,deposit]

count=-1

for i in l:

count+=1

if i == '':

l[count]=0

ub=float(cb)+float(l[1])-float(l[0])

if ub>=0:

my\_tree.item(selected, text="", values=(fn\_entry.get(), ln\_entry.get(), id\_entry.get(), address\_entry.get(), mobile\_entry.get(), ub))

update\_query\_1="""UPDATE Bank\_Accounts SET

current\_balance = %s

WHERE id = %s"""

values\_u=(ub,oid)

c.execute(update\_query\_1,values\_u)

he.place(relx = 0.5, rely = 1, anchor = 'center')

else:

se=Label(frame\_1, text="Error! Current balance is not sufficient to execute this transaction\n \n ", font=("Helvetica", 10, "bold"), foreground="#FFFFFF", background="#737373", width=70, height=3)

se.place(relx = 0.5, rely = 1, anchor = 'center')

# Commit changes

con.commit()

else:

pass

# Close our connection

con.close()

# Clear entry boxes

fn\_entry.delete(0, END)

ln\_entry.delete(0, END)

id\_entry.delete(0, END)

address\_entry.delete(0, END)

mobile\_entry.delete(0, END)

withdraw\_entry.delete(0, END)

deposit\_entry.delete(0, END)

# add new record to database

def add\_record():

# Update the database

# Create a database or connect to one that exists

con = mysql.connector.connect(

host="localhost",

user=username,

password=password

)

first = fn\_entry.get()

last = ln\_entry.get()

oid = id\_entry.get()

address = address\_entry.get()

mobile = mobile\_entry.get()

withdraw = withdraw\_entry.get()

deposit = deposit\_entry.get()

l1=[withdraw,deposit]

if withdraw == '' or deposit == '':

count=-1

for i in l1:

count+=1

if i == '':

l1[count]=0

else:

pass

nb=float(l1[1])-float(l1[0])

# Create a cursor instance

c = con.cursor()

c.execute('USE Bank\_Management;')

# Add New Record

if nb>=0:

insert\_query = "INSERT INTO Bank\_Accounts VALUES (%s,%s,%s,%s,%s,%s)"

values\_a=(first,last,oid,address,mobile,nb)

c.execute(insert\_query,values\_a)

he1=Label(frame\_1, background="#737373", width=80, height=4)

he1.place(relx = 0.5, rely = 1, anchor = 'center')

else:

se1=Label(frame\_1, text="Error! Net balance of an account can not be less than 0\n \n ", font=("Helvetica", 10, "bold"), foreground="#FFFFFF", background="#737373", width=70, height=3)

se1.place(relx = 0.5, rely = 1, anchor = 'center')

# Commit changes

con.commit()

# Close our connection

con.close()

# Clear entry boxes

fn\_entry.delete(0, END)

ln\_entry.delete(0, END)

id\_entry.delete(0, END)

address\_entry.delete(0, END)

mobile\_entry.delete(0, END)

withdraw\_entry.delete(0, END)

deposit\_entry.delete(0, END)

# Clear The Treeview Table

my\_tree.delete(\*my\_tree.get\_children())

# Run to pull data from database on start

query\_database()

def create\_table\_again():

# Create a database or connect to one that exists

con = mysql.connector.connect(

host="localhost",

user=username,

password=password

)

# Create a cursor instance

c = con.cursor()

c.execute('USE Bank\_Management;')

# Create Table

c.execute("""CREATE TABLE if not exists Bank\_Accounts (

first\_name text,

last\_name text,

id integer,

address text,

mobile\_no bigint,

current\_balance decimal(21,1) default 0)

""")

# Commit changes

con.commit()

# Close our connection

con.close()

def exit():

sys.exit()

# Add Buttons

button\_frame = LabelFrame(frame\_1, text="Commands")

button\_frame.pack(fill="x", expand="yes", padx=20)

update\_button = Button(button\_frame, text="Update Record", command=update\_record)

update\_button.grid(row=0, column=0, padx=10, pady=5)

add\_button = Button(button\_frame, text="Add Record", command=add\_record)

add\_button.grid(row=0, column=1, padx=10, pady=5)

add\_desc = Label(button\_frame, text="\*Every user details is required to add a new user")

add\_desc.grid(row=1, column=1, padx=10, pady=5)

remove\_all\_button = Button(button\_frame, text="Remove All Records", command=remove\_all)

remove\_all\_button.grid(row=0, column=2, padx=10, pady=5)

remove\_one\_button = Button(button\_frame, text="Remove One Selected", command=remove\_one)

remove\_one\_button.grid(row=0, column=3, padx=10, pady=5)

move\_up\_button = Button(button\_frame, text="Move Up", command=up)

move\_up\_button.grid(row=0, column=4, padx=10, pady=5)

move\_down\_button = Button(button\_frame, text="Move Down", command=down)

move\_down\_button.grid(row=0, column=5, padx=10, pady=5)

select\_record\_button = Button(button\_frame, text="Clear Entry Boxes", command=clear\_entries)

select\_record\_button.grid(row=0, column=6, padx=10, pady=5)

exit = Button(button\_frame, text="Exit", command=exit)

exit.grid(row=0, column=7, padx=10, pady=5)

error\_frame = LabelFrame(frame\_1, bg="#737373")

error\_frame.pack(pady=1)

e\_label = Label(error\_frame, text="Current balance is not sufficient to execute this transaction")

e\_label.grid(row=0, column=0, padx=10, pady=1)

he=Label(frame\_1, background="#737373", width=80, height=4)

he.place(relx = 0.5, rely = 1, anchor = 'center')

# Bind the treeview

my\_tree.bind("<ButtonRelease-1>", select\_record)

# Run to pull data from database on start

query\_database()

def cancel():

sys.exit()

self.ok\_btn = Button(canvas, text="Log in", command=login\_result)

self.cancel\_btn = Button(canvas, text="Cancel", command=cancel)

# positioning all the labels, input boxes and buttons

self.username\_label.grid(row=0, column=0, padx=20, pady=30)

self.username\_box.grid(row=0, column=1)

self.password\_label.grid(row=1, column=0, padx=20, pady=30)

self.password\_box.grid(row=1, column=1)

self.ok\_btn.grid(row=2, column=2)

self.cancel\_btn.grid(row=2, column=1)

bottom\_frame = Frame(self,borderwidth=3)

bottom\_frame.pack(fill='x',side='bottom')

visa\_photo = PhotoImage(file='images/visa.png')

visa\_label = Label(bottom\_frame,image=visa\_photo)

visa\_label.pack(side='left')

visa\_label.image = visa\_photo

mastercard\_photo = PhotoImage(file='images/mastercard.png')

mastercard\_label = Label(bottom\_frame,image=mastercard\_photo)

mastercard\_label.pack(side='left')

mastercard\_label.image = mastercard\_photo

american\_express\_photo = PhotoImage(file='images/american-express.png')

american\_express\_label = Label(bottom\_frame,image=american\_express\_photo)

american\_express\_label.pack(side='left')

american\_express\_label.image = american\_express\_photo

def tick():

current\_time = time.strftime('%I:%M %p').lstrip('0').replace(' 0',' ')

time\_label.config(text=current\_time)

time\_label.after(200,tick)

time\_label = Label(bottom\_frame,font=('orbitron',12))

time\_label.pack(side='right')

tick()

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

app = SampleApp()

app.geometry("1370x700")

app.resizable(False,False)

app.configure(bg='#545454')

app.mainloop()