Class XII Computer Science Project Source Code

November 2, 2022

[ ]:
```
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()
                    print(lookup_record)
                    # 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()
```