Python Code

```
# -*- coding: utf-8 -*-
"""
Created on Mon Apr 19 00:51:10 2024

@author: 22000404 & 22000409
"""

import tkinter as tk
from tkinter import messagebox, simpledialog
import sys
sys.path.append(r'C:\Users\ADMIN\AppData\Local\Packages\PythonSoftwareFo
undation.Python.3.11_qbz5n2kfra8p0\LocalCache\local-
packages\Python311\site-packages')
import mysql.connector

root = None  # Define root as a global variable

def connect_to_database():
    try:
        connection = mysql.connector.connect(
            host="localhost",
            user="root",
            password="**********",
            database="dbms_project"
        )
        print("Connected to MySQL database")
        return connection
    except mysql.connector.Error as err:
        print("Error:", err)

# Function to verify login credentials
def verify_login(username, password):
    connection = connect_to_database()
    cursor = connection.cursor()
```

```
    query = "SELECT * FROM login WHERE username = %s AND password =
%s"
    cursor.execute(query, (username, password))
    result = cursor.fetchone()
    cursor.close()
    connection.close()
    return result is not None

# Function to get table names from the database
def get_table_names(connection):
    cursor = connection.cursor()
    query = "SHOW TABLES"
    cursor.execute(query)
    tables = cursor.fetchall()
    # Exclude the 'login' table from the list of tables
    table_names = [table[0] for table in tables if table[0] != 'login']
    cursor.close()
    return table_names


# Function to update Blood_Bank_Available table after donor or patient insertion
def update_blood_bank_available(connection, values, intake=False):
    cursor = connection.cursor()
    try:
        if intake:
            query = "SELECT h_id FROM Registration_Team WHERE r_id = %s"
        else:
            query = "SELECT h_id FROM Registration_Team WHERE r_id = %s"
        cursor.execute(query, (values[6],))
        h_id_result = cursor.fetchone()
        if h_id_result:
            h_id = h_id_result[0]
        else:
            print("Error: No matching registration team found.")
            return

        query = "SELECT b_id FROM Blood_Bank WHERE h_id = %s"
```

```
        cursor.execute(query, (h_id,))
        b_id_result = cursor.fetchone()
        if b_id_result:
            b_id = b_id_result[0]
        else:
            print("Error: No matching blood bank found.")
            return

        query = "SELECT * FROM Blood_Bank_Available WHERE b_id = %s AND
blood_group = %s"
        cursor.execute(query, (b_id, values[4]))
        existing_record = cursor.fetchone()

        if existing_record:
            if intake:
                query = "UPDATE Blood_Bank_Available SET quantity = quantity - %s
WHERE b_id = %s AND blood_group = %s"
            else:
                query = "UPDATE Blood_Bank_Available SET quantity = quantity + %s
WHERE b_id = %s AND blood_group = %s"
            cursor.execute(query, (values[8], b_id, values[4]))
            print("Blood Bank Available updated successfully")
        else:
            if intake:
                print("Error: No availability of blood in the bank for the requested blood
group.")
            else:
                query = "INSERT INTO Blood_Bank_Available (b_id, blood_group,
quantity) VALUES (%s, %s, %s)"
                cursor.execute(query, (b_id, values[4], values[8]))
                print("New record added to Blood Bank Available table")

        connection.commit()
    except mysql.connector.Error as err:
        print("Error:", err)
    cursor.close()
```

```
# Function to insert a record into a specified table
def insert_record(root, parent_window, table_name, values):
    connection = connect_to_database()
    cursor = connection.cursor()
    query = f"INSERT INTO {table_name} VALUES ({','.join(['%s']*len(values))})"
    try:
        cursor.execute(query, values)
        connection.commit()
        print("Record inserted successfully")
        if table_name == "donor":
            update_blood_bank_available(connection, values)
        elif table_name == "patient":
            update_blood_bank_available(connection, values, intake=True)
    except mysql.connector.Error as err:
        print("Error:", err)
    cursor.close()


# Function to display tables and operations menu
def handle_display_tables(root, table_names):
    if hasattr(root, "main_frame"):
        # Clear the existing frame content
        for widget in root.main_frame.winfo_children():
            widget.destroy()
    else:
        root.main_frame = tk.Frame(root)
        root.main_frame.pack(fill=tk.BOTH, expand=True)

        menubar = tk.Menu(root)
        root.config(menu=menubar)

        file_menu = tk.Menu(menubar, tearoff=False)
        file_menu.add_command(label="Log Out", command=log_out)
        file_menu.add_command(label="Exit", command=root.destroy)
        menubar.add_cascade(label="Menu", menu=file_menu)
```

```
        tk.Label(root.main_frame, text="Blood Bank Management System",
font=("Helvetica", 24)).pack(pady=20)

    tk.Label(root.main_frame, text="Tables", font=("Helvetica", 25)).pack(pady=10)

    for table_name in table_names:
        tk.Button(root.main_frame, text=table_name, font=("Helvetica", 16),
command=lambda t=table_name: handle_table_click(root, t)).pack(pady=5)

# Function to handle table click events
def handle_table_click(root, table_name):
    operations_window = tk.Toplevel(root)
    operations_window.title(f"Operations for {table_name}")
    operations_window.geometry("300x200")
    operations_window.state('zoomed')  # Maximize window

    tk.Label(operations_window, text=f"Operations for {table_name}",
font=("Helvetica", 25)).pack(pady=10)

    tk.Button(operations_window, text="Insert Record", font=("Helvetica", 16),
command=lambda: handle_insert_record(root, operations_window,
table_name)).pack(pady=5)
    tk.Button(operations_window, text="Delete Record", font=("Helvetica", 16),
command=lambda: handle_delete_record(root, operations_window,
table_name)).pack(pady=5)
    tk.Button(operations_window, text="Update Record", font=("Helvetica", 16),
command=lambda: handle_update_record(root, operations_window,
table_name)).pack(pady=5)
    tk.Button(operations_window, text="Display Table", font=("Helvetica", 16),
command=lambda: handle_display_table(root, table_name)).pack(pady=5)
    tk.Button(operations_window, text="Back to Main Page", font=("Helvetica",
16), command=operations_window.destroy).pack(pady=5)

# Function to handle record insertion
def handle_insert_record(root, parent_window, table_name):
    insert_window = tk.Toplevel(parent_window)
    insert_window.title("Insert Record")
```

```
    insert_window.geometry("400x300")
    insert_window.state('zoomed')  # Maximize window

    tk.Label(insert_window, text=f"Insert Record into {table_name}",
font=("Helvetica", 16)).pack(pady=10)

    connection = connect_to_database()
    cursor = connection.cursor()
    cursor.execute(f"DESCRIBE {table_name}")
    columns = [column[0] for column in cursor.fetchall()]
    cursor.close()

    input_entries = []

    for column in columns:
        tk.Label(insert_window, text=f"{column}:").pack(pady=5)
        entry = tk.Entry(insert_window)
        entry.pack()
        input_entries.append(entry)

    def insert_record_wrapper():
        values = [entry.get() for entry in input_entries]
        if None in values:
            messagebox.showerror("Error", "Please fill in all fields.")
            return
        insert_record(root, parent_window, table_name, values)
        insert_window.destroy()

    tk.Button(insert_window, text="Insert",
command=insert_record_wrapper).pack(pady=10)
    tk.Button(insert_window, text="Back to Operations",
command=insert_window.destroy).pack(pady=5)

# Function to handle record deletion
def handle_delete_record(root, parent_window, table_name):
    delete_window = tk.Toplevel(parent_window)
    delete_window.title("Delete Record")
```

```
    delete_window.geometry("300x200")
    delete_window.state('zoomed')  # Maximize window

    tk.Label(delete_window, text=f"Delete Record from {table_name}",
font=("Helvetica", 16)).pack(pady=10)

    tk.Label(delete_window, text=f"Enter {table_name[:-1]} ID to
delete:").pack(pady=5)
    entry = tk.Entry(delete_window)
    entry.pack()

    def delete_record():
        primary_key = entry.get()
        if primary_key == "":
            messagebox.showerror("Error", "Please enter the ID.")
            return
        connection = connect_to_database()
        cursor = connection.cursor()
        cursor.execute(f"DELETE FROM {table_name} WHERE
{table_name[0:1]}_id = %s", (primary_key,))
        connection.commit()
        cursor.close()
        connection.close()
        delete_window.destroy()

    tk.Button(delete_window, text="Delete",
command=delete_record).pack(pady=10)
    tk.Button(delete_window, text="Back to Operations",
command=delete_window.destroy).pack(pady=5)

# Function to handle record update
def handle_update_record(root, parent_window, table_name):
    update_window = tk.Toplevel(parent_window)
    update_window.title("Update Record")
    update_window.geometry("400x300")
    update_window.state('zoomed')  # Maximize window
```

```
    tk.Label(update_window, text=f"Update Record in {table_name}",
font=("Helvetica", 16)).pack(pady=10)

    connection = connect_to_database()
    cursor = connection.cursor()
    cursor.execute(f"DESCRIBE {table_name}")
    columns = [column[0] for column in cursor.fetchall()]
    cursor.close()

    input_entries = []

    for column in columns:
        tk.Label(update_window, text=f"{column}:").pack(pady=5)
        entry = tk.Entry(update_window)
        entry.pack()
        input_entries.append(entry)

    def update_record():
        primary_key_value = input_entries[0].get()
        if primary_key_value == "":
            messagebox.showerror("Error", "Please enter the primary key value.")
            return
        values = [entry.get() for entry in input_entries[1:]]  # Exclude the primary key
value
        if None in values:
            messagebox.showerror("Error", "Please fill in all fields.")
            return

        # Construct the UPDATE query dynamically
        update_query = f"UPDATE {table_name} SET "
        update_query += ", ".join([f"{col} = %s" for col in columns[1:]])  # Exclude the
primary key column
        update_query += f" WHERE {table_name[0:1]}_id = %s"

        cursor = connection.cursor()
        cursor.execute(update_query, tuple(values) + (primary_key_value,))
        connection.commit()
```

```
        cursor.close()
        connection.close()
        update_window.destroy()

    tk.Button(update_window, text="Update",
command=update_record).pack(pady=10)
    tk.Button(update_window, text="Back to Operations",
command=update_window.destroy).pack(pady=5)

# Function to display table data
def handle_display_table(root, table_name):
    display_window = tk.Toplevel(root)
    display_window.title(f"{table_name} Data")
    display_window.geometry("400x300")
    display_window.state('zoomed')  # Maximize window

    tk.Label(display_window, text=f"{table_name} Data", font=("Helvetica",
16)).pack(pady=10)

    connection = connect_to_database()
    cursor = connection.cursor()
    cursor.execute(f"SELECT * FROM {table_name}")
    rows = cursor.fetchall()
    for row in rows:
        tk.Label(display_window, text=row).pack()
    cursor.close()
    connection.close()

    tk.Button(display_window, text="Close",
command=display_window.destroy).pack(pady=10)

# Function to navigate back to the main page
def navigate_to_main():
    handle_display_tables(root, get_table_names(connect_to_database()))

# Function to log out
def log_out():
```

```
    global root
    root.destroy()
    main()

# Main function
def main():
    global root
    root = tk.Tk()
    root.title("Blood Bank Management System")
    root.withdraw()  # Hide the root window initially

    screen_width = root.winfo_screenwidth()
    screen_height = root.winfo_screenheight()
    login_window = tk.Toplevel(root)
    login_window.title("Login")
    login_window.geometry("600x400+{}+{}".format(int(screen_width/2 - 300),
int(screen_height/2 - 200)))
    login_window.focus_set()  # Set focus to the login window

    tk.Label(login_window, text="Login", font=("Helvetica", 20)).pack(pady=10)

    username_label = tk.Label(login_window, text="Username:", font=("Helvetica",
16))
    username_label.pack(pady=5)
    username_entry = tk.Entry(login_window)
    username_entry.pack()

    password_label = tk.Label(login_window, text="Password:", font=("Helvetica",
16))
    password_label.pack(pady=5)
    password_entry = tk.Entry(login_window, show='*')
    password_entry.pack()

    def login():
        username = username_entry.get()
        password = password_entry.get()
        if verify_login(username, password):
```

```
            login_window.destroy()
            root.state('zoomed')  # Maximize the root window
            root.deiconify()  # Bring the root window to front
            navigate_to_main()
        else:
            messagebox.showerror("Login Failed", "Invalid username or password.")

    tk.Button(login_window, text="Login", font=("Helvetica", 16),
command=login).pack(pady=10)

    root.mainloop()

if __name__ == "__main__":
    main()
```