UNIVERSITY INSTITUTE of COMPUTING
Asia's Fastest Growing University

NAAC GRADE A+
Accredited University

PL/SQL PROJECT

STUDENT MANAGEMENT SYSTEM

MASTERS OF COMPUTER APPLICATION

[AIML]

SUBMITTED BY :-

RITINDER KAUR

UID : 24MCI10092

BRANCH: MCA(AIML)

SECTION/GROUP : 24MAM-2A

SUBJECT CODE : 24CAP-602

SUBMITTED TO:-

MR. MANDEEP

E8730

ASSISITANT PROFESSOR

1. Aim of the Code:

The aim of this code is to create a Student Management System application using Python's tkinter for the GUI and pymysql for database interaction. The application enables users to manage student records stored in a MySQL database, performing basic CRUD (Create, Read, Update, Delete) operations.

2. Task to be Done:

1) Display and Manage Student Data

- **Data Entry Form:** Collects student information, including Roll No, Name, Class, Address, Gender, Section, and DOB.
- **Gender Selection:** Uses a dropdown for gender selection with options for "Male," "Female," and "Others."

2) Database Operations (CRUD)

- **Create:** The add_func function inserts a new student record into the MySQL database.
- **Read:** The display_data function retrieves and displays all records from the database, updating the display table (Treeview) in the GUI.
- **Update:** The update_fun function updates a selected student's details based on Roll No.
- **Delete:** The delete_fun function removes a student's record from the database by Roll No.

- **Search:** The search_fun function searches the database for records that match criteria chosen in the dropdown, using values from fields like Roll No, Name, Class, etc.

3) Table Display (Treeview)

- Displays student records in a structured table, allowing the user to view and select data easily.
- Scrollbars: Vertical and horizontal scrollbars are configured for the table to handle large data sets.

4) Interactive Features

- **Selecting Records:** Clicking a row in the table loads that record's details into the data entry form, enabling easy updates.
- **Clear Form:** The clear function clears the data entry form, resetting input fields.
- **Exit:** The iExit function prompts the user for confirmation before closing the application.

5) Search and Filter

- **Search Functionality:** Allows users to search for a specific record by selecting a criterion (like Roll No or Name) and inputting the search value.
- **Show All:** Refreshes the display to show all records after a search.

6) Error Handling

- Displays messages for various conditions, such as missing input fields during record addition or updating and confirming record deletion.

3. Implementation:

[1]:
```
import tkinter as tk
from tkinter import ttk, messagebox
import sys
sys.path.insert(0, "C:/Users/sneha/AppData/Roaming/Python/Python311/site-packages")
import pymysql


# Initialize main window
win = tk.Tk()
win.geometry("1350x700+0+0")
win.title("Student Management System")

# Temporary in-memory database
students = []
'''
# Functions
def add_student():
    if not rollno.get() or not name.get():
        messagebox.showerror("Error", "Roll No and Name are required!")
        return
    student = {
        "rollno": rollno.get(),
        "name": name.get(),
        "class": class_var.get(),
        "address": address.get(),
        "section": section.get(),
        "dob": DOB.get(),
        "gender": gender.get()
    }
    students.append(student)
    update_table()
    clear_fields()
```

```
def update_student():
    selected = student_table.focus()
    if not selected:
        messagebox.showerror("Error", "No student selected!")
        return
    index = student_table.index(selected)
    students[index] = {
        "rollno": rollno.get(),
        "name": name.get(),
        "class": class_var.get(),
        "address": address.get(),
        "section": section.get(),
        "dob": DOB.get(),
        "gender": gender.get()
    }
    update_table()
    clear_fields()

def delete_student():
    selected = student_table.focus()
    if not selected:
        messagebox.showerror("Error", "No student selected!")
        return
    index = student_table.index(selected)
    del students[index]
    update_table()
    clear_fields()
```

```
def clear_fields():
    rollno.set("")
    name.set("")
    class_var.set("")
    address.set("")
    section.set("")
    DOB.set("")
    gender.set("")

def show_all():
    update_table()

def search_student():
    """Search students based on the selected field and entered keyword."""
    criterion = search_by.get().lower()
    keyword = search_keyword.get().strip().lower()

    if not keyword:
        messagebox.showwarning("Warning", "Please enter a search keyword!")
        return

    filtered_students = [
        student for student in students
        if keyword in str(student.get(criterion, "")).lower()
    ]
    update_table(filtered_students)

def update_table():
    student_table.delete(*student_table.get_children())
    for student in students:
        student_table.insert("", tk.END, values=(
            student["rollno"], student["name"], student["class"],
            student["address"], student["section"], student["dob"], student["gender"]
```

```
    #----variables----#
rollno = tk.StringVar()
name = tk.StringVar()
class_var = tk.StringVar()
address = tk.StringVar()
gender = tk.StringVar()
section = tk.StringVar()
dob = tk.StringVar()
search_by = tk.StringVar()

# GUI Elements
title_label = tk.Label(win, text="Student Management System", font=("Arial", 30, "bold"), border=12,
                       relief=tk.GROOVE, bg="blue", foreground="yellow")
title_label.pack(side=tk.TOP, fill=tk.X)

detail_frame = tk.LabelFrame(win, text="Enter Details", font=("Arial", 20), bd=12, relief=tk.GROOVE, bg="lightgrey")
detail_frame.place(x=20, y=90, width=420, height=575)

data_frame = tk.Frame(win, bd=12, bg="lightgrey", relief=tk.GROOVE)
data_frame.place(x=440, y=90, width=810, height=575)

# Entry Widgets
fields = [("Roll No", rollno), ("Name", name), ("Class", class_var),
          ("Address", address), ("Section", section), ("DOB", dob)]

for i, (label, var) in enumerate(fields):
    tk.Label(detail_frame, text=label, font=('Arial', 15), bg="lightgrey").grid(row=i, column=0, padx=2, pady=2)
    tk.Entry(detail_frame, bd=7, font=('Arial', 15), textvariable=var).grid(row=i, column=1, padx=2, pady=2)

tk.Label(detail_frame, text="Gender", font=('Arial', 15), bg="lightgrey").grid(row=6, column=0, padx=2, pady=2)
gender_ent = ttk.Combobox(detail_frame, font=('Arial', 15), state="readonly", textvariable=gender)
gender_ent['values'] = ("Male", "Female", "Others")
gender_ent.grid(row=6, column=1, padx=2, pady=2)
```

```
#---------------- Function -------------------

def display_data():
    conn = pymysql.connect(host="localhost",user="root",password="",database="sms1")
    curr = conn.cursor()
    curr.execute("SELECT * FROM data")
    rows = curr.fetchall()
    if len(rows)!=0:
        student_table.delete(*student_table.get_children())
        for row in rows:
            student_table.insert('',tk.END,values=row)
        conn.commit()
    conn.close()

def add_func():
    if rollno.get() == "" or class_var.get() == "":
        messagebox.showerror("Error!","Please fill all the fields!")
    else:
        conn = pymysql.connect(host="localhost",user="root",password="",database="sms1")
        curr = conn.cursor()
        curr.execute("INSERT INTO data VALUES(%s,%s,%s,%s,%s,%s,%s)",(rollno.get(),name.get(),class_var.get(),address.get(),gender.get(),section.get(),do
        conn.commit()
        conn.close()
        display_data()
        messagebox.showerror("Data Entry Form","Add Record Sucessfully.")
```

```
def get_cursor(event):
    cursor_row = student_table.focus()
    content = student_table.item(cursor_row)
    row = content['values']
    rollno.set(row[0])
    name.set(row[1])
    class_var.set(row[2])
    address.set(row[3])
    gender.set(row[4])
    section.set(row[5])
    dob.set(row[6])

def clear():
    rollno.set("")
    name.set("")
    class_var.set("")
    address.set("")
    gender.set("")
    section.set("")
    dob.set("")

def update_fun():
    conn = pymysql.connect(host="localhost",user="root",password="",database="sms1")
    curr = conn.cursor()
    curr.execute("UPDATE data SET name=%s, class_var=%s, address=%s, gender=%s, section=%s, dob=%s where rollno=%s",(name.get(), class_var.get(), address
    conn.commit()
    conn.close()
    display_data()
    messagebox.showerror("Data Entry Form","Data Updated Sucessfully.")
    clear()
```

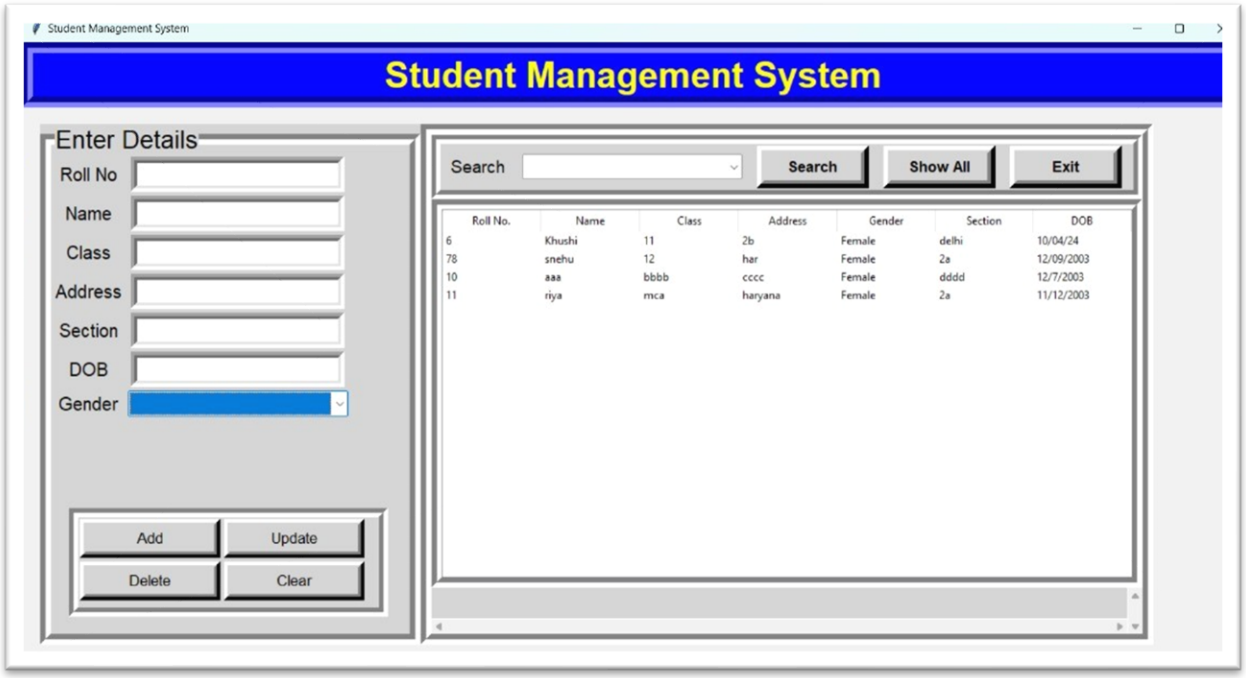
```
def delete_fun():
    conn = pymysql.connect(host="localhost",user="root",password="",database="sms1")
    curr = conn.cursor()
    curr.execute("DELETE FROM data WHERE rollno=%s",rollno.get())
    conn.commit()
    conn.close()
    display_data()
    messagebox.showerror("Data Entry Form","Record Sucessfully Deleted.")
    clear()

def search_fun():
    try:
        conn = pymysql.connect(host="localhost",user="root",password="",database="sms1")
        curr = conn.cursor()
        curr.execute("SELECT * FROM data WHERE =%s",rollno.get())
        row = curr.fetchone()
        rollno.set(row[0])
        name.set(row[1])
        class_var.set(row[2])
        address.set(row[3])
        gender.set(row[4])
        section.set(row[5])
        dob.set(row[6])

        conn.commit()

    except:
        messagebox.showerror("Error", "No student selected!")

    conn.close()
```

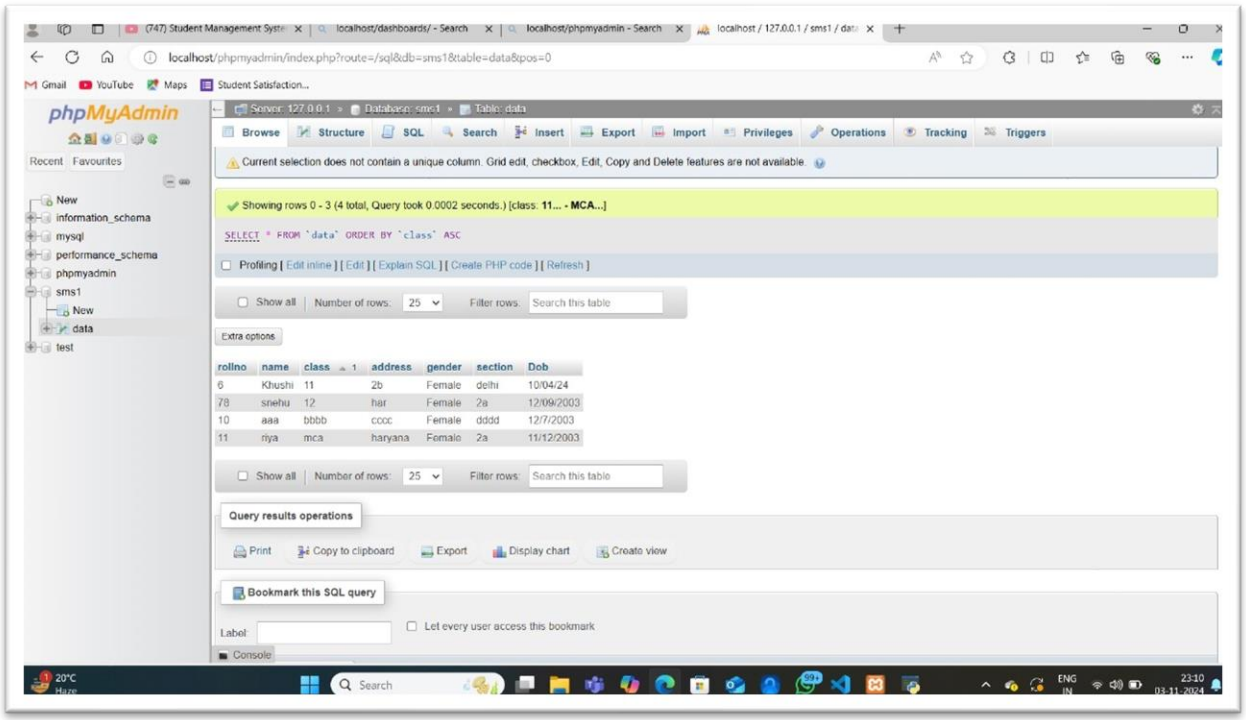
```
    iExit = messagebox.showerror("MySQL Conntion", "Confirm if you want to exit!")
    if iExit > '0':
       win.destroy()
       return

'''
def delete_fun():
    selected = student_table.focus()
    if not selected:
        messagebox.showerror("Error", "No student selected!")
        return
    else:
        index = student_table.index(selected)
        del students[index]
        update_fun()
        clear()'''


# ---------------------------------------------
# Buttons
btn_frame = tk.Frame(detail_frame, bg="lightgrey", bd=10, relief=tk.GROOVE)
btn_frame.place(x=20, y=390, width=352, height=120)

tk.Button(btn_frame, bg="lightgrey", text="Add", bd=7, font=("Arial", 13), width=15, command=add_func).grid(row=0, column=0, padx=2, pady=2)
tk.Button(btn_frame, bg="lightgrey", text="Update", bd=7, font=("Arial", 13), width=15, command=update_fun).grid(row=0, column=1, padx=2, pady=2)
tk.Button(btn_frame, bg="lightgrey", text="Delete", bd=7, font=("Arial", 13), width=15, command=delete_fun).grid(row=1, column=0, padx=2, pady=2)
tk.Button(btn_frame, bg="lightgrey", text="Clear", bd=7, font=("Arial", 13), width=15, command=clear).grid(row=1, column=1, padx=2, pady=2)

# Search Frame
search_Frame= tk.Frame(data_frame,bg="lightgrey",bd=10,relief=tk.GROOVE)
search_Frame.pack(side=tk.TOP,fill=tk.X)
```

```
search_Lbl= tk.Label(search_Frame,text="Search" ,bg="lightgrey",font=("Arial",14))
search_Lbl.grid(row=0,column=0,padx=8,pady=2)

search_in =ttk.Combobox(search_Frame,font=("Arial",14,'bold'),state="readonly",textvariable=search_by)
search_in['values']=("Name","Roll No","class","Address","Section","DOB")
search_in.grid(row=0,column=1,padx=8,pady=2)

search_btn= tk.Button(search_Frame,text="Search",font=("Arial",13,'bold'),bd=9,width=10,command=search_fun,bg="lightgrey")
search_btn.grid(row=0,column=2,padx=8,pady=2)

showall_btn=tk.Button(search_Frame,text="Show All",font=("Arial",13,'bold'),bd=9,width=10,command=display_data,bg="lightgrey")
showall_btn.grid(row=0,column=3,padx=8,pady=2)

showall_btn=tk.Button(search_Frame,text="Exit",font=("Arial",13,'bold'),bd=9,width=10,command=iExit,bg="lightgrey")
showall_btn.grid(row=0,column=4,padx=8,pady=2)


# Data Display
main_Frame = tk.Frame(data_frame, bg="lightgrey", bd=11, relief=tk.GROOVE)
main_Frame.pack(fill=tk.BOTH, expand=True)

y_scroll = tk.Scrollbar(data_frame, orient=tk.VERTICAL)
x_scroll = tk.Scrollbar(data_frame, orient=tk.HORIZONTAL)

student_table = ttk.Treeview(main_Frame, columns=("Roll No.", "Name", "Class", "Address", "Section", "DOB", "Gender"),
                             yscrollcommand=y_scroll.set, xscrollcommand=x_scroll.set)
y_scroll.config(command=student_table.yview)
x_scroll.config(command=student_table.xview)
y_scroll.pack(side=tk.RIGHT, fill=tk.Y)
x_scroll.pack(side=tk.BOTTOM, fill=tk.X)

for col in ("Roll No.", "Name", "Class", "Gender", "Address", "Section", "DOB"):
    student_table.heading(col, text=col)
    student_table.column(col, width=100)
```

```
for col in ("Roll No.", "Name", "Class", "Gender", "Address", "Section", "DOB"):
    student_table.heading(col, text=col)
    student_table.column(col, width=100)

student_table['show'] = 'headings'
student_table.pack(fill=tk.BOTH, expand=True)


# display_data()
student_table.bind("<ButtonRelease-1>",get_cursor)
win.mainloop()
```

4. Output:





5. Conclusions:

1. Student Management Operations:

- Add Student: Adds a student to the in-memory list (students).
- Update Student: Modifies the details of a selected student.
- Delete Student: Removes a student from the list.
- Search Student: Filters the students based on user input.
- Show All: Displays the complete list of students. • Clear Fields: Resets all input fields after operations.

2. Graphical User Interface (GUI):

- Entry Forms: Input fields to enter student details (e.g., Roll No, Name, Class, Gender).
- Treeview Widget: Displays student records in a tabular format with headings.
- Search and Filtering Options: Allows the user to search by various criteria like Roll No, Name, or Class.
- Buttons for Actions: Buttons for Add, Update, Delete, Clear, Search, and Show All.

3. Validation and Feedback:

- Error Handling: Provides warning/error pop-ups using messagebox (e.g., if a field is left empty or no student is selected).
- Dropdown Menu for Gender: Uses a ttk.Combobox to select predefined gender options.

• Core Concepts Demonstrated:

1. tkinter Widgets:

- Labels, Entry Widgets, and Buttons to capture input and perform actions.
- ttk.Treeview to display tabular data.
- Scrollbars linked to the Treeview for easy navigation.
- ttk.Combobox for dropdown options.

2. CRUD Operations:

- Create, Read, Update, and Delete operations are performed on a student list, representing basic data management.

3. In-Memory Data Storage:

- All student data is temporarily stored in the students list, which gets reset every time the application restarts.

4. Modular Functions:

- The code separates logic into individual functions like add_student(), update_student(), and search_student() for clarity and maintainability.

• Potential Improvements:

1. Persistent Storage:

- Add functionality to save the student data in a database or file (e.g., SQLite or CSV) to preserve records across sessions.

2. Input Validation:

- Add stricter validations for fields (e.g., date format for DOB, numeric-only Roll No).

3. UX Improvements:

- Provide confirmation pop-ups before deleting a student.
- Highlight the selected row in the Treeview for better user experience.

4. Sorting:

- Add sorting functionality by clicking on Treeview column headers.

6.Future Frameworks:

1.Backend Database Integration

To make the application more reliable and persistent:

- SQLite / MySQL / PostgreSQL: Store student data in a database for permanent storage.
    - Reason: Currently, the system stores data only in memory, which resets after each session.
    - Action: Integrate SQL queries with sqlite3 or any other RDBMS to save and retrieve records.

2. User Authentication and Roles

Introduce user login functionality with different roles:

- Admin: Can perform all CRUD operations.
- Teacher/User: Limited permissions (e.g., view and update only).

Authentication Methods:

- Use tkinter login screens with password fields. • Implement hashed passwords using the bcrypt library.

3. Advanced Search and Filters

Enhance the search capabilities:

- Multiple Filters: Allow searching by multiple criteria (e.g., Name and Class).
- Date Filters: Enable range-based searches for DOB. • Sorting: Add sorting functionality to the Treeview by column headers.

4. Data Export and Import Options

Provide functionality to **import/export** data:

- **Export to CSV/Excel:** Save student records to files. • **Import from Excel/CSV:** Load bulk data into the system.

5. GUI Improvements for User Experience (UX)

- Pagination: If the number of students grows large, add pagination to the data table.
- Highlight Selected Rows: Provide better feedback when a student is selected for update.
- Confirmation Dialogs: Add confirmation pop-ups before deleting records. • Themes and Styles: Use ttk.Style for a more polished look.

6. Data Validation and Error Handling

- Stricter Input Validation: Ensure Roll No is numeric, and DOB is in the correct format (e.g., YYYY-MM-DD). • Real-time Form Validation: Display error messages while typing invalid input.

7. Cloud Integration and Web Interface

- Cloud-based Storage: Sync data with a cloud database (e.g., Firebase or AWS RDS) to make it accessible from anywhere.
- Web-based Interface: Use Django/Flask to build a web front-end version of the system, complementing the desktop app.

8. Reporting and Analytics

- Generate **custom reports** on student performance or attendance.
- **Graphical Analysis:** Display charts (e.g., using matplotlib) for gender distribution, class-wise strength, etc.

9. Backup and Security

- Automated Backups: Save data regularly to avoid data loss.

- Encryption: Encrypt sensitive data (e.g., DOB, address). • Access Logs: Maintain logs of all operations (e.g., student deletions).

10. Mobile App Integration

Create a mobile version using a framework like:

- Kivy: For mobile GUI. • Flutter or React Native: If moving to a hybrid web/mobile framework.

6. Learning outcomes:

1. Building GUIs with Tkinter:

- You gain experience in using Tkinter to build user interfaces. The use of widgets like Frame, Label, Entry, and Button shows how to design a complete GUI application.

2. Layout Management:

- The code demonstrates the use of pack(), place(), and grid() geometry managers, teaching how to organize and align widgets within the window effectively.

3. Data Binding with Variables:

- The use of tk.StringVar() helps you learn about data binding between variables and widgets, ensuring dynamic updates in the application.

4. Using Treeview for Data Representation:

- Implementing ttk.Treeview teaches how to present data in a tabular format with scrollbars, an essential skill for applications requiring data display.

5. Modularity through Frames:

- The code is organized into different frames, promoting a modular design approach, which makes the interface more manageable and easier to expand.

6.Working with ComboBox Widgets:

• You learn how to use ttk.Combobox to provide selectable options, which adds flexibility to user input (e.g., selecting gender or search criteria).

7. Basic GUI Event Handling:

• Although the button functionalities (e.g., Add, Update, Delete) are not yet implemented, the code introduces the concept of event handling—a core concept in interactive applications.

8. Search Functionality Implementation:

• Implementing a search feature with multiple criteria demonstrates how to filter and retrieve specific data efficiently using widgets like Combobox and Treeview.

9.Foundations for CRUD Operations:

• The interface design prepares you to implement Create, Read, Update, and Delete (CRUD) operations. Learning how to map these operations to widgets is critical for data management applications.

10. Handling Scrollbars in Tkinter:

• The implementation of both vertical and horizontal scrollbars with the Treeview provides insight into improving the usability of the GUI for large datasets.

11. Planning for Database Integration:

• The layout hints at future database integration (e.g., through SQLite). This reinforces how GUIs interact with databases for managing information in real-world applications.

12. Color and Style Customization:

• Customizing widget styles with parameters like background color (bg), foreground color (foreground), and fonts provides experience with UI design concepts.