UNIVERSITY INSTITUTE of COMPUTING
Asia's Fastest Growing University

NAAC GRADE A+
ACCREDITED UNIVERSITY

Case Study
On
Student Result Management System (SRMS)

CHANDIGARH UNIVERSITY
Discover. Learn. Empower.

Subject Code : 24CAH-606

MASTERS IN COMPUTER APPLICATION

Submitted By:

Name-Priyam Yadav
UID- 24MCA20377
Branch- MCA
Section- 6-A

Submitted To:

Mrs. Palwinder Kaur Mangat

(Assistant Professor)

Case Study: Student Result Management System

1. Introduction

Managing student results is a critical task in educational institutions. With advancements in technology, manual record-keeping has become obsolete, replaced by digital systems that streamline and automate the process. This case study discusses the development of a **Student Result Management System** using Python with the **Tkinter** library for the GUI and **SQLite** for database management.

2. Objective

The primary objective is to design a user-friendly, efficient, and secure system that allows administrators to manage student details, courses, and results. The system aims to provide functionalities such as:

- Adding, updating, and deleting course information.
- Managing student records.
- Viewing student results.
- Providing an intuitive and visually appealing graphical user interface (GUI).

3. System Requirements

3.1 Software Requirements

- **Python**: Programming language for logic implementation.
- **Tkinter**: Built-in Python library for creating the graphical user interface.
- **SQLite**: Lightweight, embedded database for efficient data storage and management.

- **PIL (Python Imaging Library)**: For image handling and display.

3.2 Hardware Requirements

- A standard PC with a minimum of 4GB RAM.
- Disk space of at least 50MB for the SQLite database and application files.

4. System Architecture

The system is developed as a desktop application with a layered architecture:

1. **Presentation Layer**: Built using Tkinter for an interactive GUI.
2. **Database Layer**: Uses SQLite to store data related to courses, students, and results.
3. **Business Logic Layer**: Python functions and classes handle the core operations, including database management and user input processing.

5. Design and Implementation

5.1 Database Design

The database consists of tables like course, student, and result. The course table is created using SQL commands to store course details:

sql

Copy code

```
CREATE TABLE IF NOT EXISTS course (
    cid INTEGER PRIMARY KEY AUTOINCREMENT,
    name TEXT,
    duration TEXT,
    charges TEXT,
    description TEXT
);
```

Other tables include:

- **student**: Stores student information (name, roll number, class, etc.).
- **result**: Stores examination results, including marks, grades, and overall status.

5.2 Graphical User Interface (GUI)

The GUI is developed using Tkinter and provides the following features:

- **Title Bar**: Displays the name of the application.
- **Menu Bar**: Contains buttons like Course, Student, Result, View Student Result, Logout, and Exit.
- **Dashboard**: Displays background images and statistical information about total courses, students, and results.
- **Footer**: Shows the application's contact information for technical support.

5.3 Modules

1. **Course Management Module**
    - Allows administrators to add, view, update, and delete course details.
    - Includes fields like course name, duration, charges, and description.
2. **Student Management Module**
    - Handles student records, including adding and updating student details.

- Provides a user-friendly form for data entry.

3. **Result Management Module**
    - Enables administrators to add and manage examination results.
    - Facilitates the retrieval and display of results for students.

6. Key Features and Functionalities

1. **User Authentication and Security**
    - Login functionality can be added to ensure that only authorized users have access to the system.
2. **Data Integrity and Validation**
    - Input fields include validation mechanisms to prevent incorrect or incomplete data entry.
3. **Image Handling**
    - The PIL library is used to load and display images, enhancing the user experience with a visually appealing interface.
4. **Responsive Layout**
    - The application layout is designed to adapt to different screen sizes and resolutions.

7. Advantages and Disadvantages

7.1 Advantages

- **User-Friendly Interface**: The use of Tkinter provides an intuitive GUI for easy navigation.
- **Lightweight Database**: SQLite is efficient for small to medium-sized applications, requiring minimal configuration.

- **Cross-Platform Compatibility**: The system can run on different operating systems, including Windows, macOS, and Linux.
- **Scalable Design**: The architecture allows for easy extension and addition of new features.

7.2 Disadvantages

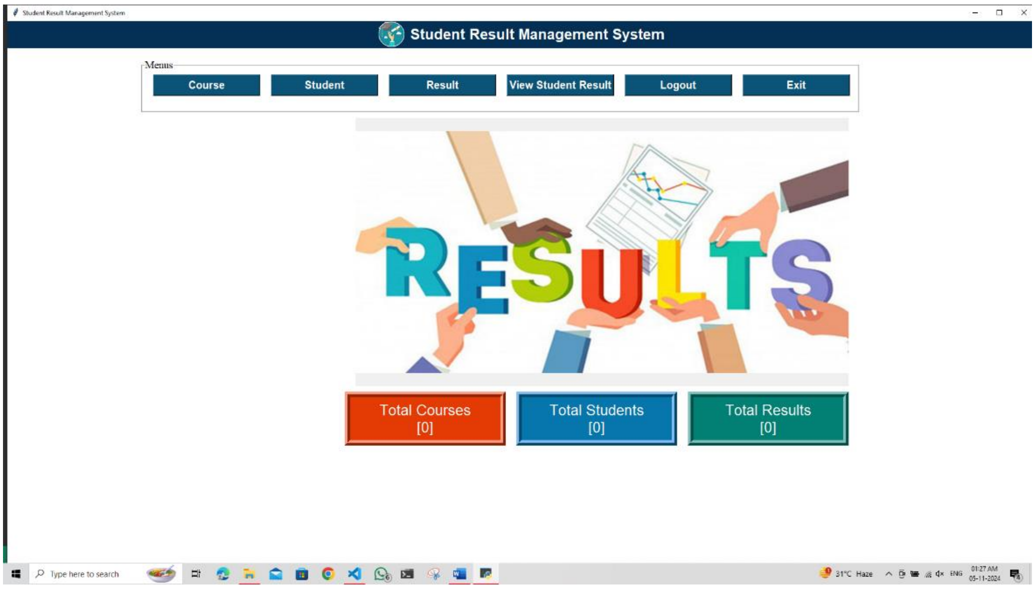
- **Limited Database Size**: SQLite may not be suitable for very large datasets or concurrent multi-user access.
- **Performance Overhead**: The Tkinter GUI can become less responsive when handling a large volume of data.
- **Security Concerns**: Basic implementations may lack advanced security measures, making the system vulnerable to data breaches if not appropriately handled.

8. Testing and Evaluation

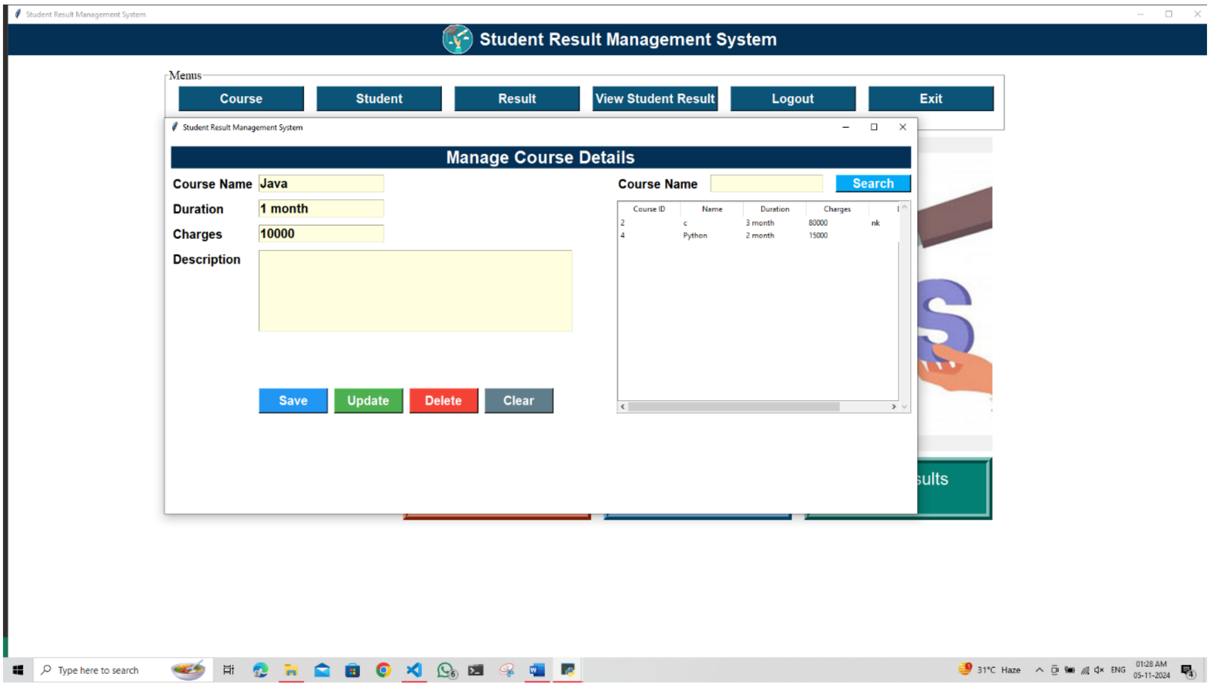
The system underwent extensive testing, including:

1. **Functional Testing**: Ensured that all features work as expected.
2. **Usability Testing**: Assessed the user experience and made improvements based on feedback.
3. **Performance Testing**: Measured the response time of the application under different data loads.

9. Output



- ## Output Of Course Page



Course Name Java

Duration 1 month

Charges 10000

Description

Course Name

Course ID	Name	Duration	Charges	
2	c	3 month	80000	nk
4	Python	2 month	15000	

10. Code –

10.1 Code of Dashboard Page (Home Page)

```
from tkinter import *
from PIL import Image, ImageTk
import os
from Course import courseClass
from student import studentClass
class RMS:
def __init__(self, root):
self.root = root
self.root.title("Student Result Management System")
# self.root.geometry("1350x700+0+0")
scrn_width= self.root.winfo_screenwidth()
scrn_height=self.root.winfo_screenheight()
self.root.geometry(f"{scrn_width}x{scrn_height}+0+0")
self.root.config(bg="white")
```

```
# Load the logo image

logo_path = "images/logo_p.png"

if os.path.exists(logo_path):

self.logo_dash = ImageTk.PhotoImage(file=logo_path)

# Title

title = Label(self.root, text="Student Result Management System", padx=10, compound=LEFT, image=self.logo_dash, font=("goudy old style", 20, "bold"), bg="#033054", fg="white")

title.place(x=0, y=0, relwidth=1, height=50)

# Menu Frame

M_Frame = LabelFrame(self.root, text="Menus", font=("times new roman", 15), bg="white")

M_Frame.place(x=250, y=70, width=1340, height=100)

# Buttons

btn_course = Button(M_Frame, text="Course", font=("goudy old style", 15, "bold"), bg="#0b5377", fg="white", cursor="hand2",command=self.add_course)

btn_course.place(x=20, y=5, width=200, height=40)

btn_student = Button(M_Frame, text="Student", font=("goudy old style", 15, "bold"), bg="#0b5377", fg="white", cursor="hand2")

btn_student.place(x=240, y=5, width=200, height=40)

btn_result = Button(M_Frame, text="Result", font=("goudy old style", 15, "bold"), bg="#0b5377", fg="white", cursor="hand2")

btn_result.place(x=460, y=5, width=200, height=40)

btn_view = Button(M_Frame, text="View Student Result", font=("goudy old style", 15, "bold"), bg="#0b5377", fg="white", cursor="hand2")

btn_view.place(x=680, y=5, width=200, height=40)

btn_logout = Button(M_Frame, text="Logout", font=("goudy old style", 15, "bold"), bg="#0b5377", fg="white", cursor="hand2")

btn_logout.place(x=900, y=5, width=200, height=40)

btn_exit = Button(M_Frame, text="Exit", font=("goudy old style", 15, "bold"), bg="#0b5377", fg="white", cursor="hand2")

btn_exit.place(x=1120, y=5, width=200, height=40)

# Background Image

bg_img_path = "images/bg.jpg"
```

```
if os.path.exists(bg_img_path):
self.bg_img = Image.open(bg_img_path)
# Increased height to 450
self.bg_img = self.bg_img.resize((920, 450), Image.LANCZOS)
self.bg_img = ImageTk.PhotoImage(self.bg_img)

# Display Background Image
self.lbl_bg = Label(self.root, image=self.bg_img)
self.lbl_bg.place(x=650, y=180, width=920, height=500)
# Statistics Labels
self.lbl_course = Label(self.root, text="Total Courses\n[0]", font=("goudy old style", 20), bd=10, relief=RIDGE, bg="#e43b05", fg="white")
self.lbl_course.place(x=630, y=690, width=300, height=100)
self.lbl_student = Label(self.root, text="Total Students\n[0]", font=("goudy old style", 20), bd=10, relief=RIDGE, bg="#0676ad", fg="white")
self.lbl_student.place(x=950, y=690, width=300, height=100)
self.lbl_result = Label(self.root, text="Total Results\n[0]", font=("goudy old style", 20), bd=10, relief=RIDGE, bg="#038074", fg="white")
self.lbl_result.place(x=1270, y=690, width=300, height=100)
# Footer
footer = Label(self.root, text="Student Result Management System\nContact Us for any Technical Issue: 8318388719", font=("goudy old style", 12), bg="#262626", fg="white")
footer.pack(side=BOTTOM, fill=X)
def add_course(self):
self.new_win=Toplevel(self.root)
self.new_obj=courseClass(self.new_win)
def add_student(self):
self.new_win=Toplevel(self.root)
self.new_obj=studentClass(self.new_win)
if __name__ == "__main__":
root = Tk()
obj = RMS(root)
```

```
root.mainloop()
```

10.2 Code of Course Page

```
from tkinter import *
from PIL import Image, ImageTk
import sqlite3
from tkinter import ttk, messagebox
class courseClass:
def __init__(self, root):
self.root = root
self.root.title("Student Result Management System")
self.root.geometry("1200x600+250+180")
self.root.config(bg="white")
self.root.focus_force()
# Title
title = Label(self.root, text="Manage Course Details", font=("goudy old style", 20, "bold"), bg="#033054", fg="white")
title.place(x=10, y=15, width=1180, height=35)
# Variables
self.var_course = StringVar()
self.var_duration = StringVar()
self.var_charges = StringVar()
# Widgets
lbl_courseName = Label(self.root, text="Course Name", font=("goudy old style", 15, 'bold'), bg='white')
lbl_courseName.place(x=10, y=60)
lbl_duration = Label(self.root, text="Duration", font=("goudy old style", 15, 'bold'), bg='white')
lbl_duration.place(x=10, y=100)
lbl_charges = Label(self.root, text="Charges", font=("goudy old style", 15, 'bold'), bg='white')
lbl_charges.place(x=10, y=140)
lbl_description = Label(self.root, text="Description", font=("goudy old style", 15, 'bold'), bg='white')
lbl_description.place(x=10, y=180)
# Input Fields
self.txt_courseName = Entry(self.root, textvariable=self.var_course, font=("goudy old style", 15, 'bold'), bg='lightyellow')
self.txt_courseName.place(x=150, y=60, width=200)
self.txt_duration = Entry(self.root, textvariable=self.var_duration, font=("goudy old style", 15, 'bold'), bg='lightyellow')
self.txt_duration.place(x=150, y=100, width=200)
self.txt_charges = Entry(self.root, textvariable=self.var_charges, font=("goudy old style", 15, 'bold'), bg='lightyellow')
self.txt_charges.place(x=150, y=140, width=200)
self.txt_description = Text(self.root, font=("goudy old style", 15, 'bold'), bg='lightyellow')
self.txt_description.place(x=150, y=180, width=500, height=130)

# Buttons with different positions
self.btn_add = Button(self.root, text='Save', font=("goudy old style", 15, "bold"), bg='#2196f3', fg="white", cursor="hand2", command=self.add)
self.btn_add.place(x=150, y=400, width=110, height=40)
self.btn_update = Button(self.root, text='Update', font=("goudy old style", 15, "bold"), bg='#4caf50', fg="white", cursor="hand2", command=self.update)
self.btn_update.place(x=270, y=400, width=110, height=40)
self.btn_delete = Button(self.root, text='Delete', font=("goudy old style", 15, "bold"), bg='#f44336', fg="white", cursor="hand2", command=self.delete)
self.btn_delete.place(x=390, y=400, width=110, height=40)
self.btn_clear = Button(self.root, text='Clear', font=("goudy old style", 15, "bold"), bg='#607d8b', fg="white", cursor="hand2", command=self.clear)
self.btn_clear.place(x=510, y=400, width=110, height=40)
# Search Panel
self.var_search = StringVar()
lbl_search_courseName = Label(self.root, text="Course Name", font=("goudy old style", 15, 'bold'), bg='white')
lbl_search_courseName.place(x=720, y=60)
txt_search_courseName = Entry(self.root, textvariable=self.var_search, font=("goudy old style", 15, 'bold'), bg='lightyellow')
txt_search_courseName.place(x=870, y=60, width=180)
btn_search = Button(self.root, text='Search', font=("goudy old style", 15, "bold"), bg='#03a9f4', fg="white", cursor="hand2", command=self.search)
btn_search.place(x=1070, y=60, width=120, height=28)
# Content Frame
self.C_Frame = Frame(self.root, bd=2, relief=RIDGE)
self.C_Frame.place(x=720, y=100, width=470, height=340)
# Treeview for displaying courses
self.CourseTable = ttk.Treeview(self.C_Frame, columns=("cid", "name", "duration", "charges", "description"))
self.CourseTable.heading("cid", text="Course ID")
self.CourseTable.heading("name", text="Name")
self.CourseTable.heading("duration", text="Duration")
self.CourseTable.heading("charges", text="Charges")
self.CourseTable.heading("description", text="Description")
self.CourseTable['show'] = 'headings'
# Column widths
self.CourseTable.column("cid", width=100)
self.CourseTable.column("name", width=100)
self.CourseTable.column("duration", width=100)
self.CourseTable.column("charges", width=100)
self.CourseTable.column("description", width=150)

# Bind to get data
```

```
self.CourseTable.bind("<ButtonRelease-1>", self.get_data)
self.show()
# Scrollbars
scrolly=Scrollbar(self.C_Frame,orient=VERTICAL,
command=self.CourseTable.yview)
scrollx=Scrollbar(self.C_Frame,orient=HORIZONTAL,
command=self.CourseTable.xview)
self.CourseTable.configure(yscrollcommand=scrolly.set, xscrollcommand=scrollx.set)
scrolly.pack(side=RIGHT, fill=Y)
scrollx.pack(side=BOTTOM, fill=X)
self.CourseTable.pack(fill=BOTH, expand=1)
# Fetch data for update
def get_data(self, ev):
r = self.CourseTable.focus()
content = self.CourseTable.item(r)
row = content["values"]
if row:
self.var_course.set(row[1])
self.var_duration.set(row[2])
self.var_charges.set(row[3])
self.txt_description.delete('1.0', END)
self.txt_description.insert(END, row[4])
self.txt_courseName.config(state='readonly')
# Add course
def add(self):
con = sqlite3.connect(database="rms.db")
cur = con.cursor()
try:
if self.var_course.get() == "":
messagebox.showerror("Error", "Course Name Should Be Required", parent=self.root)
else:
cur.execute("select * from course where name=?", (self.var_course.get(),))
row = cur.fetchone()
if row:
messagebox.showerror("Error", "Course Name Already Present", parent=self.root)
else:
cur.execute("insert into course(name, duration, charges, description) values (?, ?, ?, ?)",
(
self.var_course.get(),
self.var_duration.get(),
self.var_charges.get(),
self.txt_description.get("1.0", END).strip()
))
con.commit()
messagebox.showinfo("Success", "Course Added Successfully", parent=self.root)
self.show()
```

```
except Exception as ex:
messagebox.showerror("Error", f"Error due to {str(ex)}", parent=self.root)
finally:
con.close()
# Show added courses
def show(self):
con = sqlite3.connect(database="rms.db")
cur = con.cursor()
try:
cur.execute("select * from course")
rows = cur.fetchall()
self.CourseTable.delete(*self.CourseTable.get_children())
for row in rows:
self.CourseTable.insert('', END, values=row)
except Exception as ex:
messagebox.showerror("Error", f"Error due to {str(ex)}", parent=self.root)
finally:
con.close()
# Update course
def update(self):
con = sqlite3.connect(database="rms.db")
cur = con.cursor()
try:
if self.var_course.get() == "":
messagebox.showerror("Error", "Course Name Should Be Required", parent=self.root)
else:
cur.execute("select * from course where name=?", (self.var_course.get(),))
row = cur.fetchone()
if not row:
messagebox.showerror("Error", "Select Course from List", parent=self.root)
else:
cur.execute("update course set duration=?, charges=?, description=? where name=?", (
self.var_duration.get(),
self.var_charges.get(),
self.txt_description.get("1.0", END).strip(),
self.var_course.get()
))
con.commit()
messagebox.showinfo("Success", "Course Updated Successfully", parent=self.root)
self.show()
except Exception as ex:
messagebox.showerror("Error", f"Error due to {str(ex)}", parent=self.root)
finally:
con.close()
# Delete course
def delete(self):
```

```
con = sqlite3.connect(database="rms.db")
cur = con.cursor()
try:
if self.var_course.get() == "":
messagebox.showerror("Error", "Course Name Should Be Required", parent=self.root)
else:
cur.execute("select * from course where name=?", (self.var_course.get(),))
row = cur.fetchone()
if not row:
messagebox.showerror("Error", "Select Course from List", parent=self.root)
else:
op = messagebox.askyesno("Confirm", "Do you really want to delete?", parent=self.root)
if op:
cur.execute("delete from course where name=?", (self.var_course.get(),))
con.commit()
messagebox.showinfo("Delete", "Course Deleted Successfully", parent=self.root)
self.clear()
except Exception as ex:
messagebox.showerror("Error", f"Error due to {str(ex)}", parent=self.root)
finally:
con.close()
# Clear input fields
def clear(self):
self.show()
self.var_course.set("")
self.var_duration.set("")
self.var_charges.set("")
self.txt_description.delete('1.0', END)
self.var_search.set("")
self.txt_courseName.config(state=NORMAL)
# Search for a course
def search(self):
con = sqlite3.connect(database="rms.db")
cur = con.cursor()
try:
cur.execute("select * from course where name LIKE ?", ('%' + self.var_search.get() + '%',))
rows = cur.fetchall()
self.CourseTable.delete(*self.CourseTable.get_children())
for row in rows:
self.CourseTable.insert('', END, values=row)
except Exception as ex:
messagebox.showerror("Error", f"Error due to {str(ex)}", parent=self.root)
finally:
con.close()
```

```
if __name__ == "__main__":
root = Tk()
obj = courseClass(root)
root.mainloop()
```

10.3 Code of Creation Of Database(create_db.py)

```
import sqlite3
def create_db():
    con = sqlite3.connect(database="rms.db")
    cur = con.cursor()
    cur.execute("""CREATE TABLE IF NOT EXISTS course (
             cid INTEGER PRIMARY KEY AUTOINCREMENT,
             name TEXT,
             duration TEXT,
             charges TEXT,
             description TEXT )""")
   con.commit()

create_db()
```

11. Conclusion

The **Student Result Management System** is an effective solution for managing student records and results. The implementation using Python, Tkinter, and SQLite provides a reliable and scalable platform for educational institutions. However, future enhancements, such as integrating cloud storage and implementing multi-user access control, could further improve the system.

12. References

1. *Python Documentation* - Official Python documentation for libraries like Tkinter and SQLite: https://docs.python.org/

2. Grinberg, M. (2018). *Flask Web Development: Developing Web Applications with Python*. O'Reilly Media.

3. Zelle, J. (2010). *Python Programming: An Introduction to Computer Science*. Franklin, Beedle & Associates Inc.

4. *SQLite Documentation* - Comprehensive guide to SQLite features and best practices: https://www.sqlite.org/