Student Database Management System

CIS 552-01: Database Design (2024 Spring)

Summary Report

Group - 3

Team Members-

Piyush Kolte (32)

Mythri Krpet Jayaram (35)

Pawan Karthik (27)

Part 1: Planned Database Driven Application Requirements /Use-cases

Application Requirements:

1. Student Database Management System (DBMS):
    - Integrate MySQL, MongoDB, and Neo4j to create a comprehensive DBMS.
    - MySQL for structured data storage (e.g., student profiles, course details).
    - MongoDB for managing unstructured data (e.g., documents, multimedia files).
    - Neo4j for representing complex relationships (e.g., students, courses, instructors).
2. Python-based Frontend:
    - Develop a frontend using Jupyter Notebook.
    - Ensure a seamless and intuitive user experience for administrators, faculty, and students.
3. Efficient Data Handling:
    - Enable efficient storage, retrieval, and manipulation of student-related data.
    - Cater to diverse needs of educational institutions.
4. Advanced Functionality:
    - Enable advanced querying and analysis.
    - Provide insights into student performance, enrollment trends, and academic networks.

Use Cases:

1. Administrators:
    - Manage student profiles, course details, and academic records efficiently.
    - Utilize advanced querying and analysis tools for strategic decision-making.
2. Faculty:
    - Access student information easily for academic planning and evaluation.
    - Analyze student performance and engagement patterns for personalized teaching approaches.
3. Students:
    - Access their academic records and course details conveniently.
    - Benefit from a system that enhances their learning experience through efficient data management and analysis.
4. Institutional Researchers:
    - Utilize the system for in-depth analysis of enrollment trends, academic networks, and performance metrics.
    - Extract valuable insights to inform institutional policies and strategies.
5. System Administrators:
    - Ensure the smooth operation and maintenance of the DBMS.

Part 2: Database Structure

MongoDB is employed in the student database management system to efficiently handle unstructured data like documents and multimedia files associated with students. Its flexibility enables seamless management of diverse data types, enhancing the system's capacity to store and retrieve student-related information effectively within educational institutions.

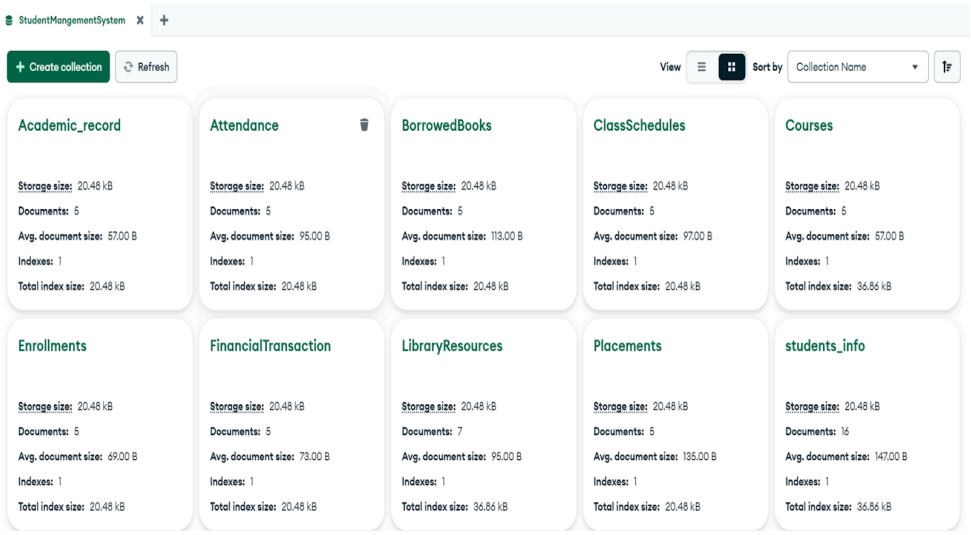

Fig 1 - Collections Created in MongoDB

The development and structure of a student database management system implemented using MongoDB Compass. MongoDB, a NoSQL database, is utilized for its flexible schema and efficient handling of large volumes of structured and unstructured data. MongoDB Compass is the GUI for MongoDB that simplifies the management of database tasks such as schema inspection, query performance analysis, and CRUD (Create, Read, Update, Delete) operations.

Our student database comprises ten collections, each tailored to store specific types of data relevant to student management:

Student_info: Stores comprehensive details about each student.
This collection is structured with individual documents for each student, where every document includes essential information about the student. The field `_id`. `FirstName` and `LastName`, `DateOfBirth`,`ContactNumber` and`Email`. Collectively, these fields assemble a comprehensive profile for each student within the database system.

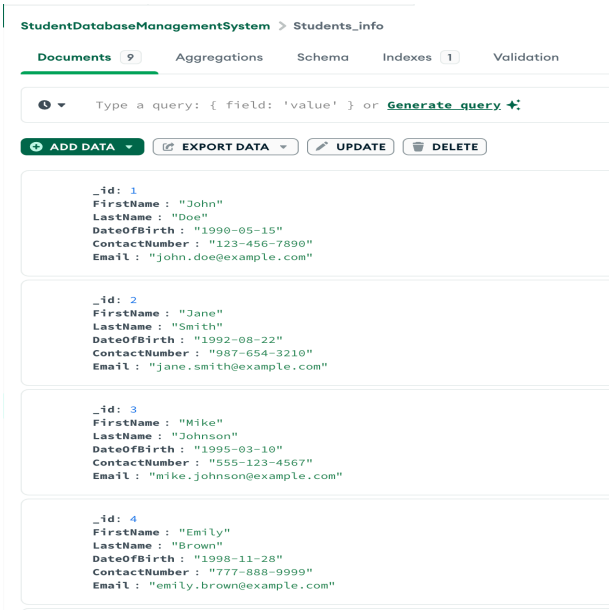

Fig 2 - Student Information Collection

Placements: Records information related to job placements of students.

The "Placements" is detailing various student employment records. Each entry is uniquely identified by an `_id`, paired with a `StudentID` that denotes the specific student involved. The `CompanyName` field indicates the employer, while the `Position` outlines the role or job title granted to the student. The employment period is defined by `StartDate` and `EndDate`, which mark the beginning and end of the placement, respectively.

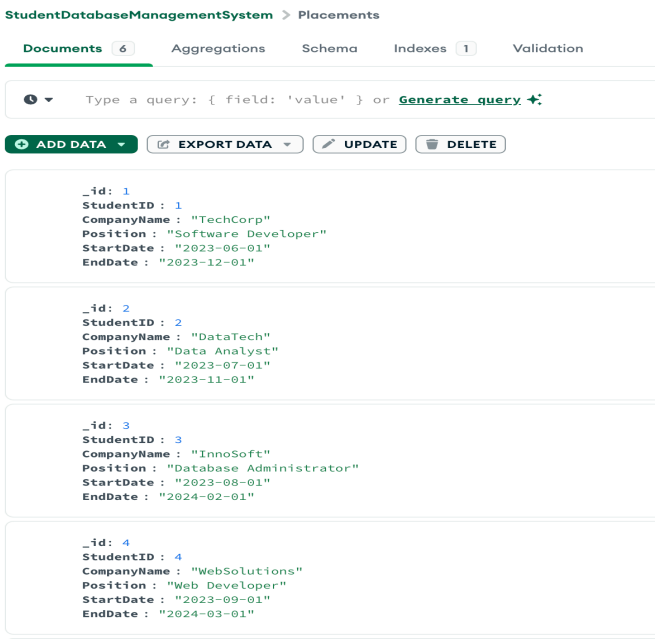

Fig 3 - Placement Collection

LibraryResources: Manages data on library assets and their usage.

The below image shows entries within a database where each record contains several fields. The `_id` field serves as a unique identifier for each document. The `Title` field specifies the name of the book. The `Author` field reveals the name of the individual who authored the book, and the `AvailableCopies` field indicates the quantity of copies of the book that are currently available.

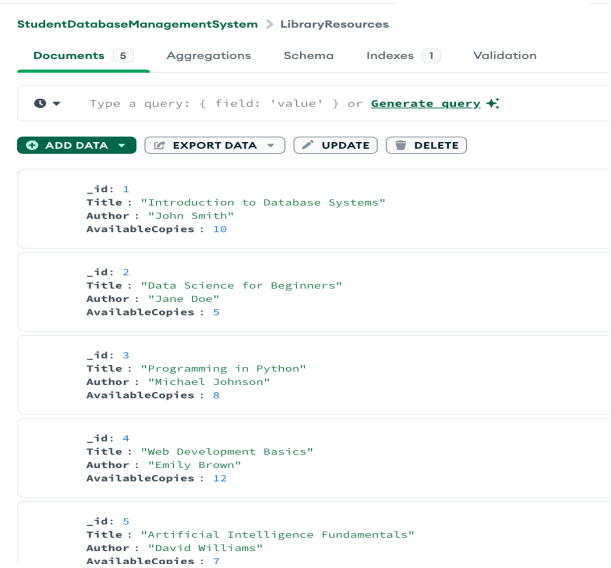

Fig 4 - Library Resources Collection

FinancialTransaction: This collection consists of several fields:_id, which serves as the unique identifier for each transaction; StudentID, which is the identifier linking the transaction to an individual student; TransactionDate, indicating the date the transaction took place; and Amount, denoting the monetary value of the transaction.

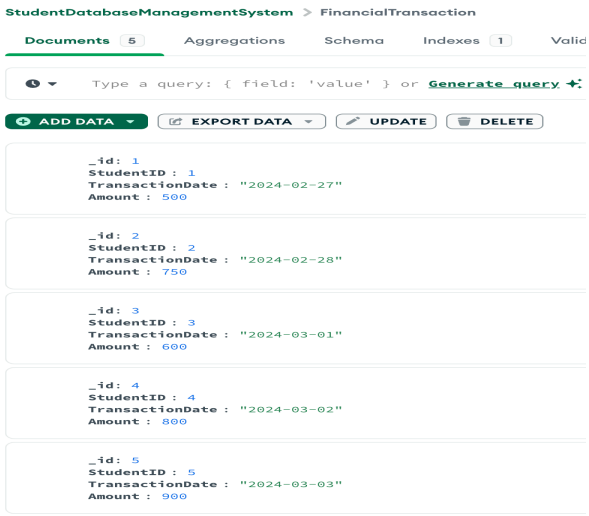

Fig 5 - Financial Transaction Collection

Enrollments: Contains records of students enrolled in various courses. The "Enrollments" collection captures the records of student enrollments. In this collection, each document comprises several fields: `_id` acts as a unique identifier for the record; `StudentID` specifies the numerical identifier of the student; `CourseID` indicates the numerical code of the course in which the student is enrolled; and `Semester` states the academic term for the enrollment.

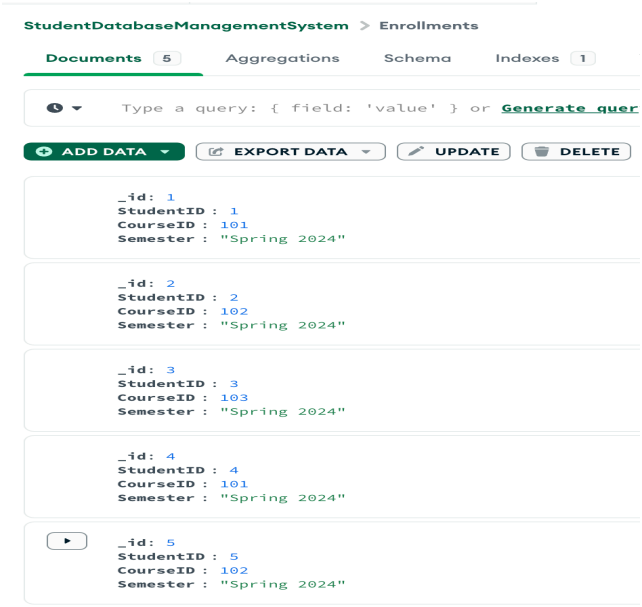

Fig 6 - Enrollment Collection

Courses: Details about courses offered, including syllabi and instructor information. The Courses collection contains the two fields in the database, i.e, _id where it appears to be a numerical identifier assigned to each course and a courseName as it contains the title of the course.

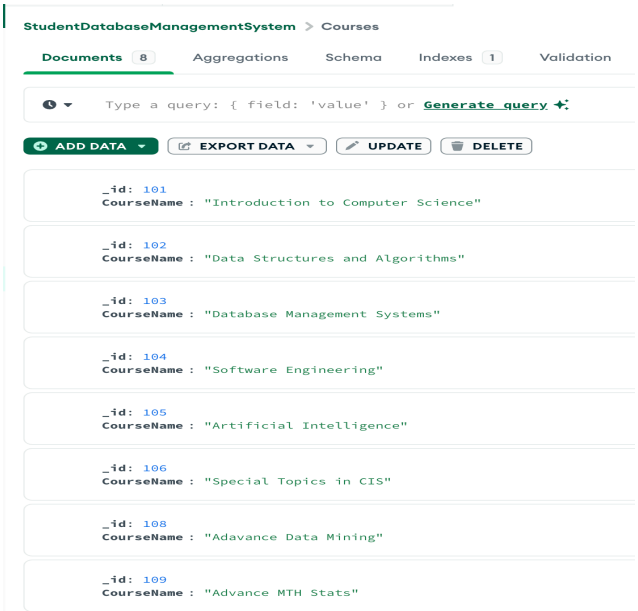

Fig 7 - Courses Collection

ClassSchedules: Scheduling information for all classes. The "ClassSchedule" collection contains several fields such as, `_id` field assigns a unique number to each class entry, ensuring no two classes have the same identifier. The `CourseID` correlates to the specific course's unique number. The `DayOfWeek` designates the day on which the class meets. `StartTime` and `EndTime` fields mark the beginning and conclusion of the class, respectively.

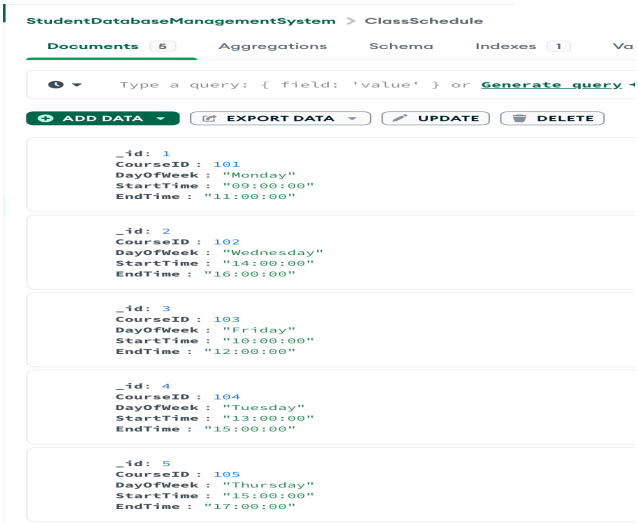

Fig 8 - Class Schedule Collection

BorrowedBooks: Tracks books borrowed by students from the library. The "BorrowedBooks" collection is distinguished by a unique `_id` number, and `StudentID`, which denotes the student who has taken out the book, and the `ResourceID`, which specifies the particular book borrowed. The `BorrowDate` field records the date the book was checked out, while the `DueDate` indicates when it should be returned. The `ReturnedDate` is intended to note the date of return, but a 'null' value signifies the book has not been returned.

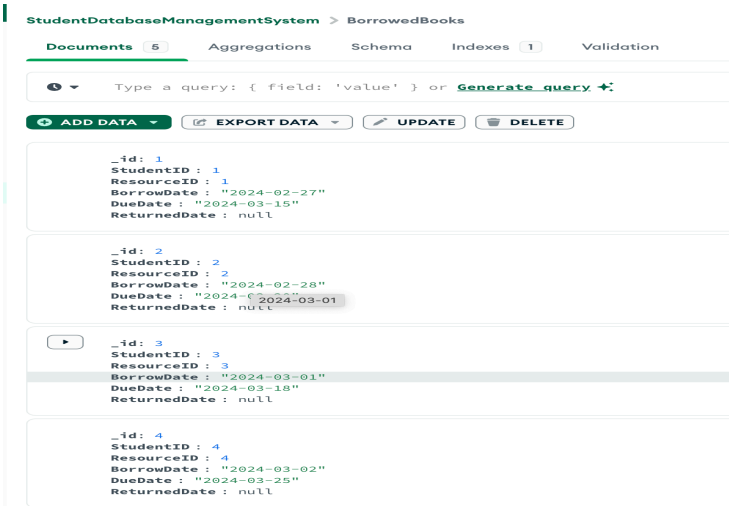

Fig 9 - Borrowed Books Collection

Attendance: Keeps attendance records for students in different classes. The "Attendance" section cataloged with specific fields. The `_id` serves as a distinct identifier for each record, while the `StudentID` notes the ID number of the student. Each record also includes a `ScheduleID`, which corresponds to the class schedule, and an `AttendanceDate`, marking when the attendance was taken. Lastly, the `Status` field indicates the student's attendance condition, either "Present" or "Absent," on the recorded date.

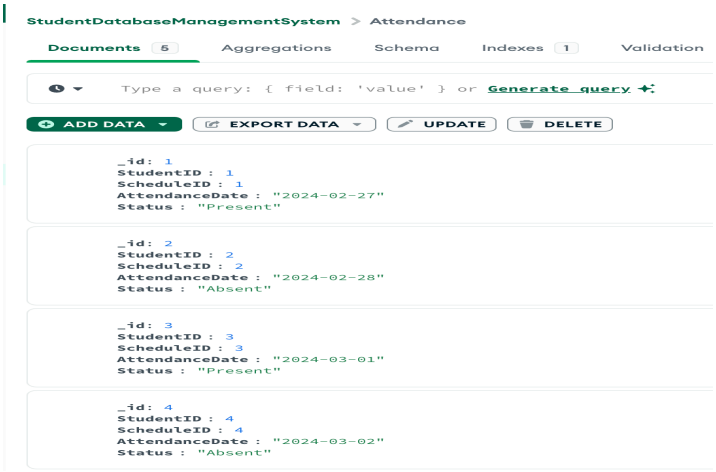

Fig 10 - Attendance Collection

Academic_Record: Maintains academic performance and grade records of students. The "Academic_record" collection consists of several key fields. The `_id` field serves as the unique identifier for each record, ensuring distinct entries for every academic performance logged. The `StudentID` field is assigned to identify the student, linking them to their performance data, while the `CourseID` indicates the specific course to which the grades pertain. Lastly, the `Grade` field records the numerical score achieved by the student in that course.

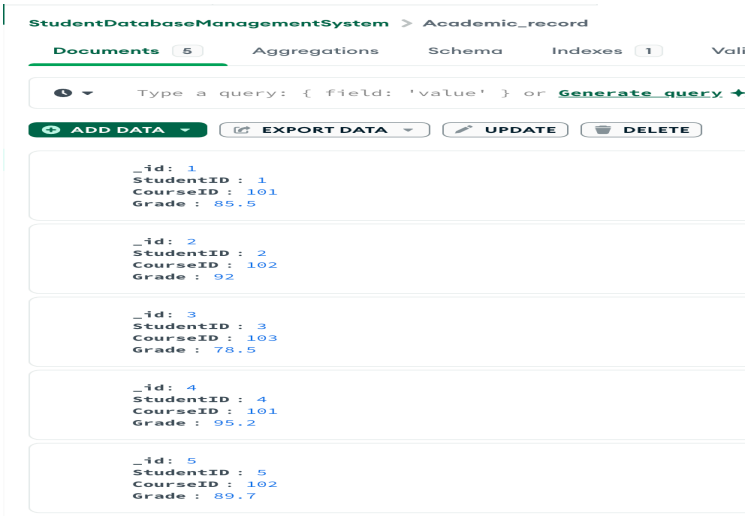

Fig 11 - Academic Record Collection

Part 3: Front-End Interface

The Student Database Management System, developed using Python's Tkinter library, provides an efficient and user-friendly interface for managing student records. The application facilitates Create, Read, Update, and Delete (CRUD) operations, which are fundamental to database management.

The intuitive layout begins with fields for inputting student details such as first name, last name, date of birth, contact number, and email address. These fields are paired with a 'Register Student' button to add new records to the system.

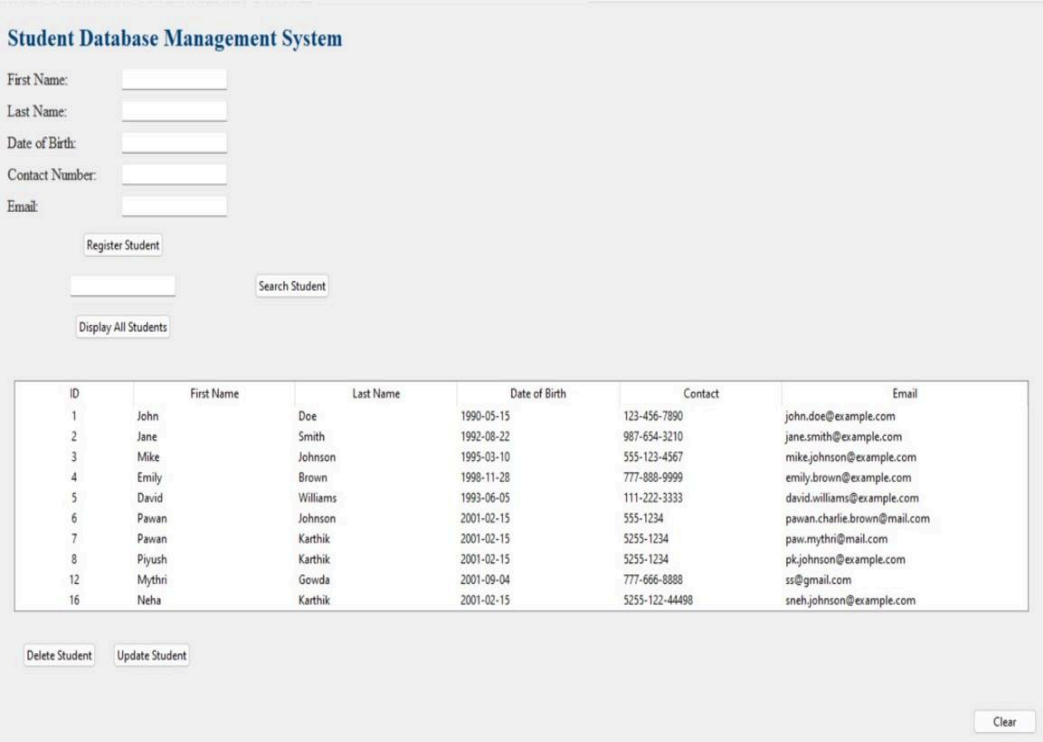

Fig 12 - Frontend GUI, Display all student data that is stored in the MongoDB

The 'Search Student' function allows for quick retrieval of records, while the 'Display All Students' button populates a comprehensive table that lists all student entries. This table includes columns for ID, names, birth dates, contact information, and emails, providing a clear view of student data.

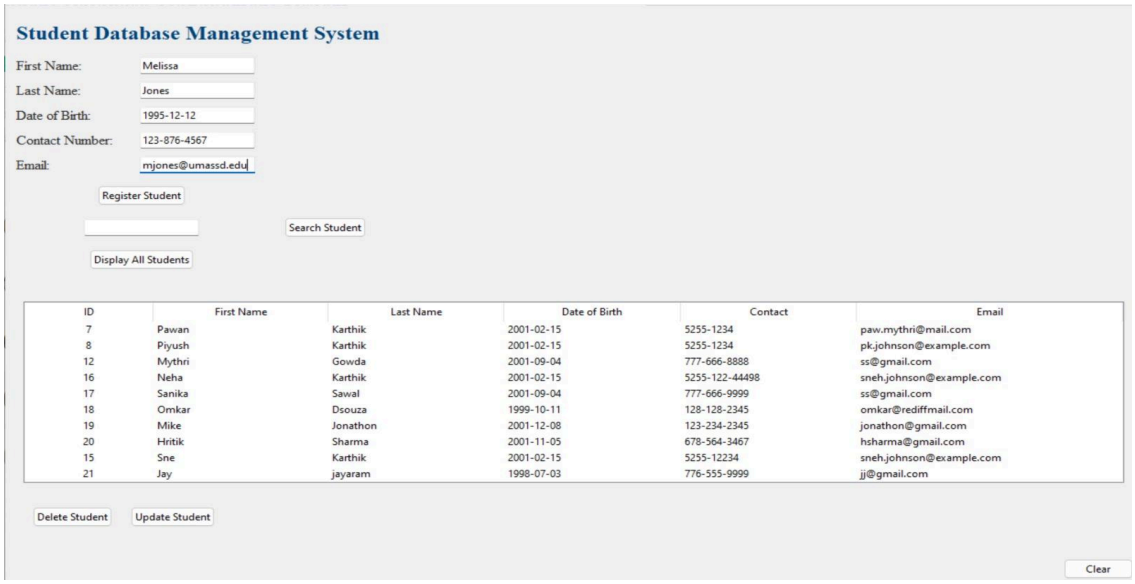

Fig 13 - Student Registration Interface

The interface depicted is a functional module of a Student Database Management System, showcasing an active form populated with a sample student's data, ready for registration. The system's layout is designed for ease of use, with clear entry fields and action buttons to manage student records effectively, reinforcing its utility in educational administration.

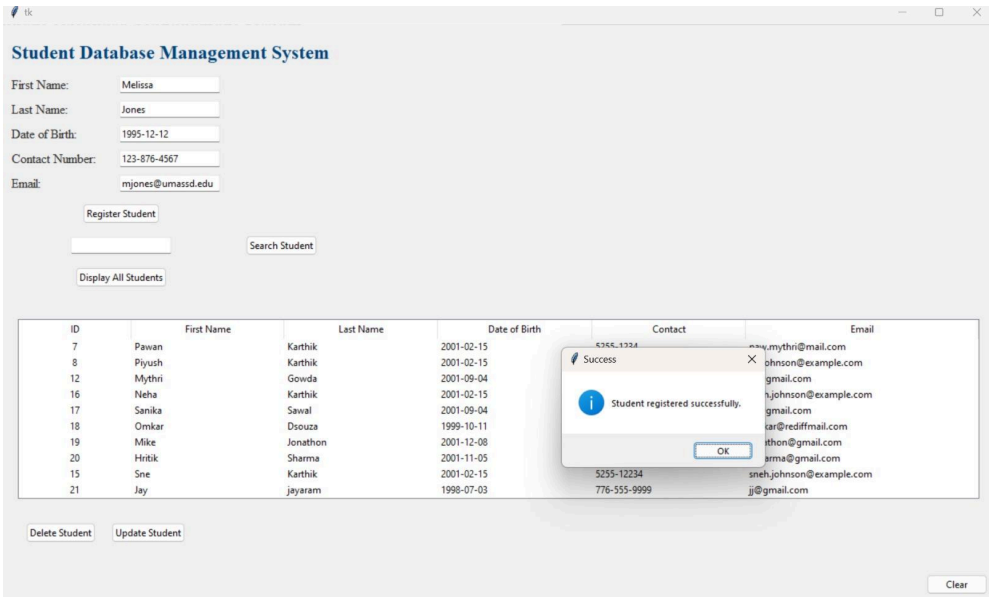

Fig 14 - Confirmation of Successful Student Registration

The interface captures a successful student registration notification within the Student Database Management System, indicating a seamless addition to the database. This confirmation dialog enhances user experience by providing clear feedback on the action performed, affirming the system's reliability and effectiveness in handling student records.

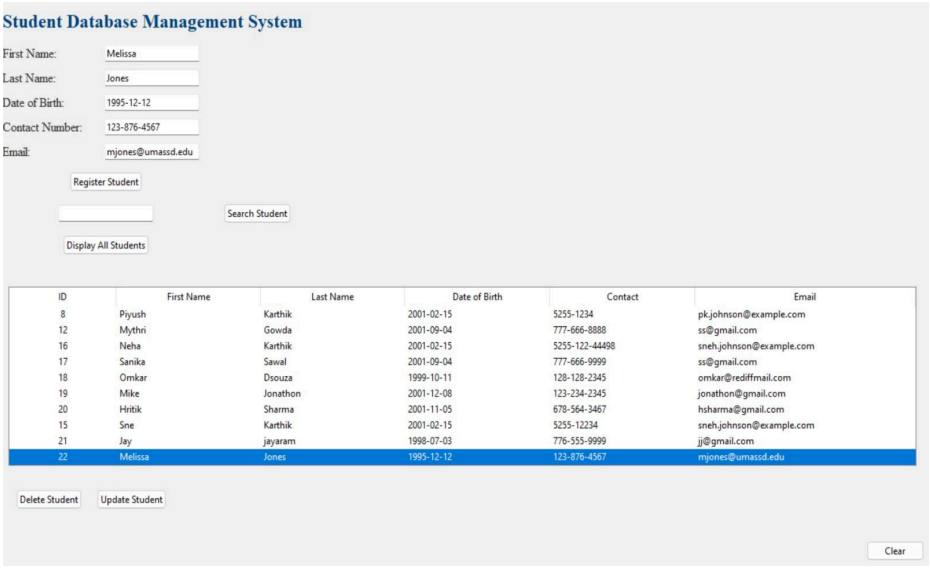

Fig 15 - Updated Student List Post-Registration

The image shows a database interface after adding a new student record. The form is pre-filled with Melissa Jones's information, indicating her recent registration. Below, the student list table has been updated to include her details, reflecting the system's immediate data processing and display capabilities.

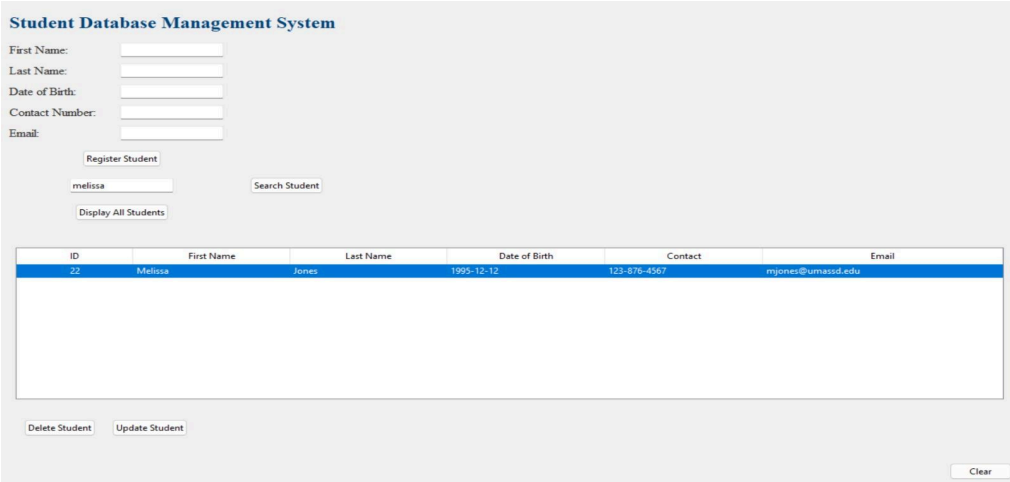

Fig 16 - Student Search Functionality Display

The image showcases the Student Database Management System with a single entry displayed. It highlights the search functionality, where the name 'melissa' has been entered, resulting in the corresponding student record for Melissa Jones being filtered and displayed, with her details visible in the table.

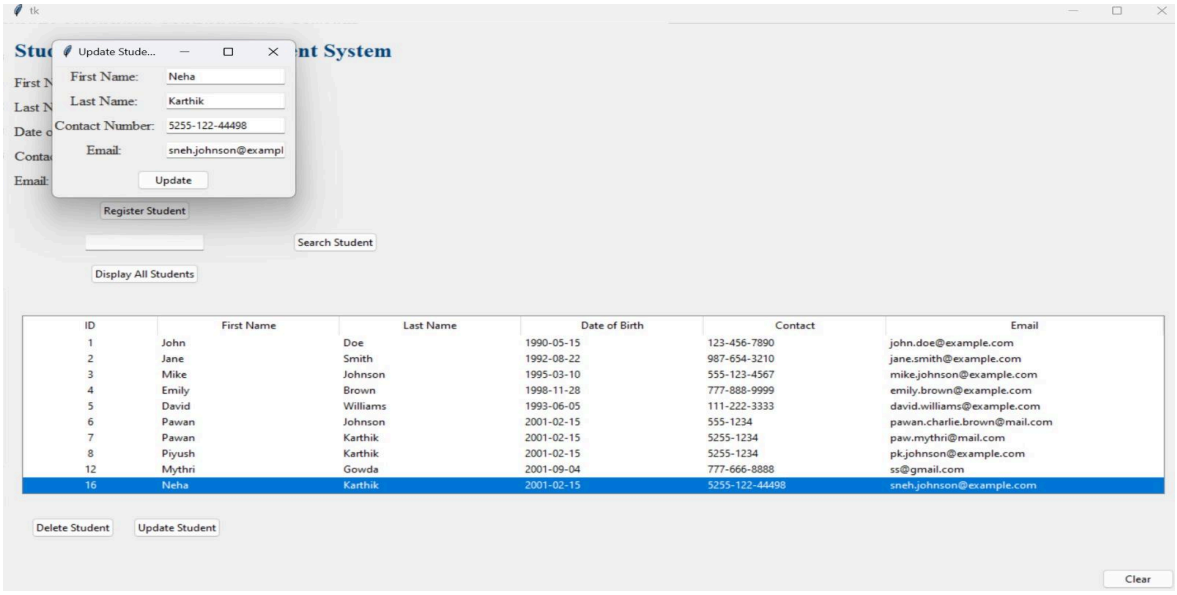

Fig 17 - Student Information Update Dialog

The image depicts a dialog box for updating student information within the Student Database Management System. Prior to submitting changes, the form is filled with Neha Karthik's details, reflecting the preparatory step of editing her contact number and email. This functionality underscores the system's capacity for maintaining up-to-date student records.

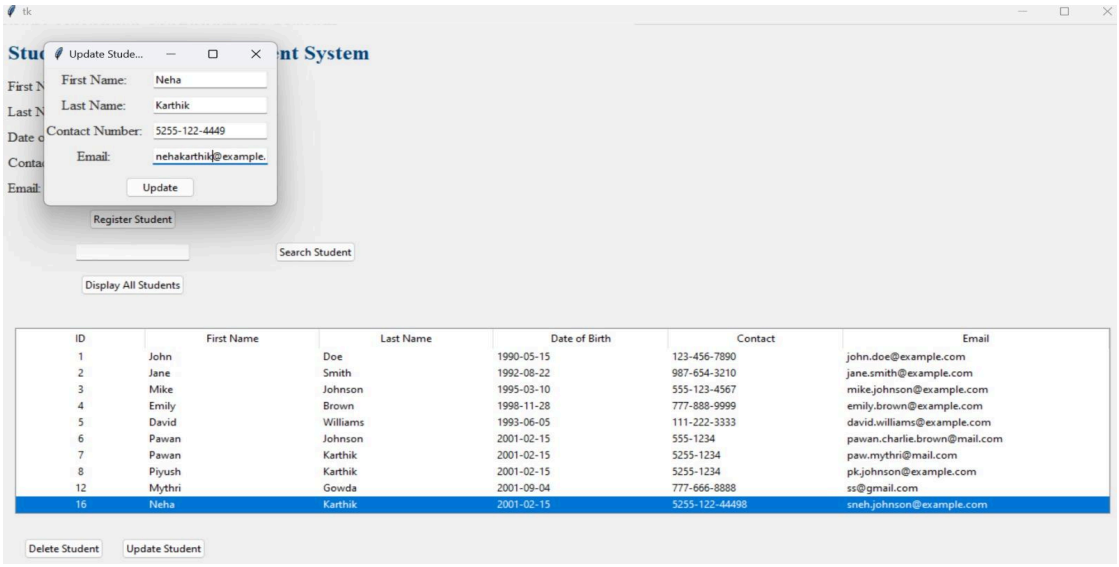

Fig 18 - Modifying Student Contact Details

The below image illustrates an 'Update Student' interface where modifications to Neha Karthik's record are being made. The form displays updated contact and email information, ready to be saved. This update function is vital for ensuring that the student database reflects the most current and accurate information available.

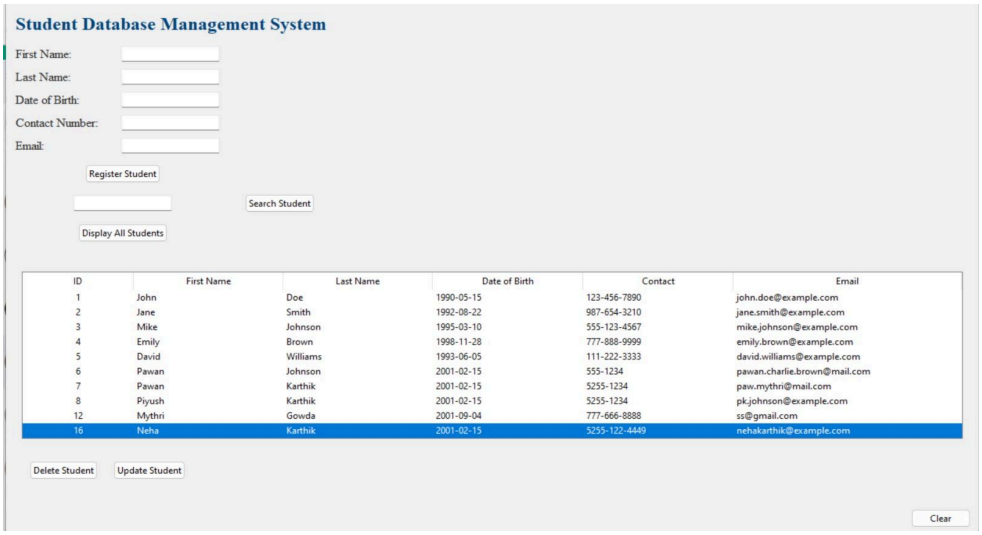

Fig 19 - Student email ID is successfully updated

This image illustrates the student record interface before executing a deletion operation. The selected student is queued for removal from the Student Database Management System, demonstrating the system's capability to manage and update the student directory as needed

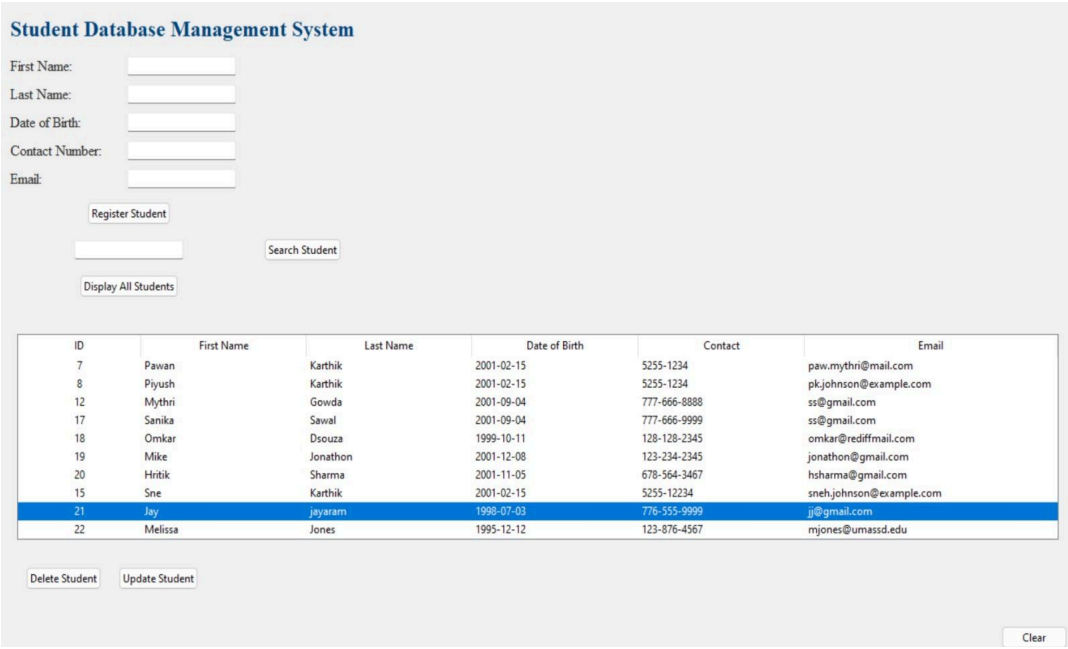

Fig 20 - Pre-Deletion Student Record Interface

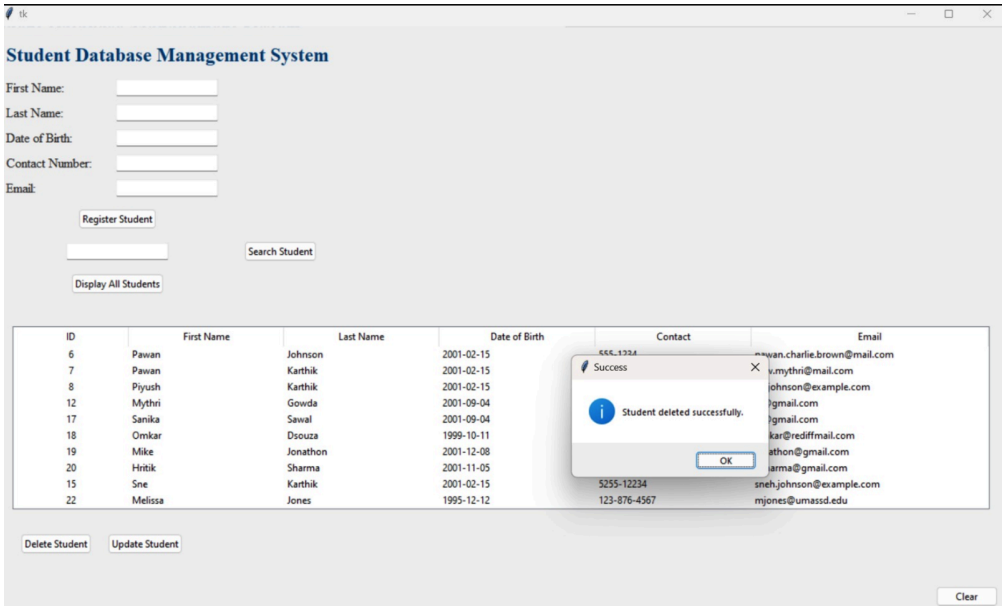

Fig 21 - Deleted the student record from the database

After deleting the student, the system also incorporates 'Delete Student' and 'Update Student' buttons for easy record modification or removal, ensuring the database is always current. A 'Clear' button is also available to reset input fields or clear selections, enhancing the system's usability. This robust system serves as a vital tool for educational administrators, streamlining the management of student information with precision and ease.

Part 4: Source Code

We are utilizing the pymongo library to establish a connection to MongoDB, a popular NoSQL database. The script is designed to access and manage a student information database, which is part of a larger Student Database Management System. The primary goal of this code is to query and display details about students stored within the database, which can be useful for reports, analytics, or administrative purposes. This demonstrates a practical application of Python in handling and retrieving data from a database.

```
import pymongo
from pymongo import MongoClient
pymongo.__version__
```

```
'4.6.3'
```

Connection

```
client = MongoClient('mongodb://localhost:27017/StudentDatabaseManagementSystem')
```

```
#calling the database
db = client.StudentMangementSystem
client.list_database_names()
```

```
['StudentDatabaseManagementSystem',
 'Studentdatabase',
 'admin',
 'config',
 'local']
```

```
db = client['StudentDatabaseManagementSystem']
print(db.list_collection_names())
```

```
['Enrollments', 'Courses', 'BorrowedBooks', 'ClassSchedule', 'LibraryResources', 'Students_info',
```

Fig 22 - Python Script for Connecting to MongoDB Server and Retrieving Student Information

CRUD Operation Code:

```
new_Courses = {
    "_id": 109,
    "CourseName": "Advance MTH Stats"
}
Courses_collection.insert_one(new_Courses)
```

```
InsertOneResult(109, acknowledged=True)
```

Fig 23 - Adding New Course to MongoDB Collection

This code snippet demonstrates the insertion of a new course into a MongoDB collection. It creates a dictionary with an ID and course name for "Advance MTH Stats" and adds it to the Courses_collection. The operation is confirmed successful by the InsertOneResult with acknowledgment.

```python
# Read and iterate through student documents
cursor = db.Courses.find()
for Courses in cursor:
    print(Courses)
```

```
{'_id': 101, 'CourseName': 'Introduction to Computer Science'}
{'_id': 102, 'CourseName': 'Data Structures and Algorithms'}
{'_id': 103, 'CourseName': 'Database Management Systems'}
{'_id': 104, 'CourseName': 'Software Engineering'}
{'_id': 105, 'CourseName': 'Artificial Intelligence'}
{'_id': 106, 'CourseName': 'Special Topics in CIS'}
{'_id': 107, 'CourseName': 'Special Topics in MTH'}
{'_id': 108, 'CourseName': 'Adavance Data Mining'}
{'_id': 109, 'CourseName': 'Advance MTH Stats'}
```

Fig 24 - Verification of Successful Course Insertion into MongoDB Collection

```python
Course_id_to_find = 101  # Assuming you want to find students enrolled in course 101
Enrollments = db.Enrollments.find({"CourseID": Course_id_to_find})

# Assuming you have student IDs in the enrollments and you want to fetch those students
for Enrollment in Enrollments:
    student = db.Students_info.find_one({"_id": Enrollment["StudentID"]})
    print(student)
```

```
{'_id': 1, 'FirstName': 'John', 'LastName': 'Doe', 'DateOfBirth': '1990-05-15', 'ContactNumber': '123-456-7890', 'Emai
{'_id': 4, 'FirstName': 'Emily', 'LastName': 'Brown', 'DateOfBirth': '1998-11-28', 'ContactNumber': '777-888-9999', 'E
```

Fig 25 - Retrieving/ Reading Student Records for a Specific Course in MongoDB

This code snippet is part of a read operation in CRUD, used to retrieve records from a MongoDB database. It searches for students enrolled in a specific course (ID 101) from the Enrollments collection and then fetches each corresponding student's detailed information from the Students_info collection, printing out the results.

```python
Course_id_to_aggregate = 101  # Assuming you're interested in the average grade for course 101

pipeline = [
    {"$match": {"CourseID": Course_id_to_aggregate}},  # Step 1: Filter documents by CourseID
    {"$group": {                                       # Step 2: Group the filtered documents
        "_id": "$CourseID",                            # Grouping key - in this case, CourseID
        "AverageGrade": {"$avg": "$Grade"}             # Calculate the average grade
    }}
]

average_grade = db.Academic_record.aggregate(pipeline)

for result in average_grade:
    print("CourseID:", result["_id"], "Average Grade:", result["AverageGrade"])
```

```
CourseID: 101 Average Grade: 90.35
```

Fig 26 - MongoDB Aggregation for Average Course Grade Calculation

This code performs an aggregation in MongoDB to calculate the average grade for a specific course. Using a pipeline that filters and groups records by CourseID, it computes the average grade and prints the result, showing the average for course ID 101 as 90.35.

```
# Update a student's email
result = db.students_info.update_one(
    {"_id": 7},
    {"$set": {"Email": "pawan.mythri.brown@mail.com"}}
)
print("Number of documents updated:", result.modified_count)
```

Fig 27 - Updating Student Email in MongoDB

This snippet of code updates the email address for a student record with the ID of 7 in the MongoDB students_info collection. It uses the update_one method with the $set operator to change the email to a new value and outputs the count of documents updated.

```
course_id_to_delete = 110  # Specify the CourseID you wish to delete

result = db.Courses.delete_one({"_id": course_id_to_delete})

print("Number of documents deleted:", result.deleted_count)
```

Fig 28 - Deleting a Course Record in MongoDB

This code performs a delete operation, specifically targeting a course in a MongoDB collection. It identifies a course by its ID (110) and removes the corresponding document from the Courses collection using the delete_one method. The operation's success is confirmed by printing out the total number of documents deleted, showcasing an effective way to maintain data integrity by removing obsolete or unwanted entries.

Frontend Source code:

Importing Libraries: Necessary libraries such as pymongo for MongoDB interaction, tkinter for GUI, PIL for image handling, and other standard libraries are imported.

MongoDB Connection Setup: The code establishes a connection to the MongoDB database named StudentDatabaseManagementSystem and sets up a collection named students_info.

```python
#importing necessary libraries
from pymongo import MongoClient
import tkinter as tk
from tkinter import ttk, messagebox
from PIL import Image, ImageTk
from tkinter import font
from tkinter import Label

# MongoDB connection setup
client = MongoClient('mongodb://localhost:27017/')
db = client['StudentDatabaseManagementSystem']
students_collection = db['students_info']
```

register_student(): Registers a new student by inserting their information into the database. Also we have added exception handling, If any error occurs during the process, it catches the exception, displays an error message with the details of the exception, and prevents the registration process from failing silently.

```python
# Function to register a student
def register_student():
    if students_collection.find_one({"FirstName": firstname_entry.get(), "LastName": lastname_entry.get()}):
        messagebox.showerror("Error", "Student with this name already exists!")
    else:
        try:
            max_id_doc = students_collection.find_one(sort=[("_id", -1)])
            next_id = 1 if not max_id_doc else max_id_doc["_id"] + 1

            student_info = {
                "_id": next_id,
                "FirstName": firstname_entry.get(),
                "LastName": lastname_entry.get(),
                "DateOfBirth": dob_entry.get(),
                "ContactNumber": contactnumber_entry.get(),
                "Email": email_entry.get(),
            }
            students_collection.insert_one(student_info)
            messagebox.showinfo("Success", "Student registered successfully.")
            display_all_students()
            clear_entries()
        except Exception as e:
            messagebox.showerror("Error", f"An error occurred: {e}")
```

display_all_students(): This function retrieves all student records from the database and displays them in the GUI's Treeview widget. It configures the columns with appropriate column names such as ID, first name, last name, date of birth, contact, and email. Once we perform this task it displays all students currently stored in the database.

```python
# Function to display all students
def display_all_students():
    result = students_collection.find()
    treeview.delete(*treeview.get_children())  # Clear existing entries

    # Configure the Treeview columns
    treeview['columns'] = ("ID", "First Name", "Last Name", "Date of Birth", "Contact", "Email")
    treeview.heading("ID", text="ID")
    treeview.heading("First Name", text="First Name")
    treeview.heading("Last Name", text="Last Name")
    treeview.heading("Date of Birth", text="Date of Birth")
    treeview.heading("Contact", text="Contact")
    treeview.heading("Email", text="Email")

    # Set column widths and alignment
    treeview.column("ID", anchor='center', width=50)
    treeview.column("First Name", anchor='w', width=100)
    treeview.column("Last Name", anchor='w', width=100)
    treeview.column("Date of Birth", anchor='w', width=100)
    treeview.column("Contact", anchor='w', width=100)
    treeview.column("Email", anchor='w', width=200)

    # Insert results into the Treeview
    for student in result:
        treeview.insert("", 'end', values=(
            student.get("_id", ''),
            student.get("FirstName", ''),
            student.get("LastName", ''),
            student.get("DateOfBirth", ''),
            student.get("ContactNumber", ''),
            student.get("Email", '')
        ))
```

delete_student(): This function is responsible for removing a selected student record from the database. It validates whether a student record is selected in the GUI's Treeview widget; if not, it prompts the user to make a selection. Once a student record is selected, it retrieves the ID of the selected student and deletes the corresponding record from the database. After successful deletion, it updates the GUI by removing the selected student's entry from the Treeview widget. Additionally, it handles the scenario where no student record is selected, providing an error message to guide user interaction effectively. Overall, `delete_student()` deletes a selected student record from the database.

```
# Function to delete a student
def delete_student():
    selected_item = treeview.selection()
    if selected_item:
        selected_student = treeview.item(selected_item[0])['values']
        student_id = int(selected_student[0])
        students_collection.delete_one({"_id": student_id})
        treeview.delete(selected_item[0])
        messagebox.showinfo("Success", "Student deleted successfully.")
    else:
        messagebox.showerror("Error", "Please select a student to delete.")
```

update_student(): Initiates the process of updating a student's information. This function enables users to modify student information stored in the database. It validates the selection of a student record based on the data given. As we can see in the below code the update function only allows the user to modify the student basic data like, First Name, Last Name, contact number & email ID. After a successful update of the record it will show a message like student information is successfully updated.

```
## Function to update a student record
def update_student():
    selected_item = treeview.selection()
    if selected_item:
        update_window = tk.Toplevel(root)
        update_window.title("Update Student Information")

        selected_student_id = treeview.item(selected_item[0])['values'][0]
        student_id = int(selected_student_id)
        student = students_collection.find_one({"_id": student_id})

        # Fields for editable first and last names
        ttk.Label(update_window, text="First Name:", font=label_font, foreground=label_color).grid(row=0, column=0)
        firstname_update_entry = ttk.Entry(update_window)
        firstname_update_entry.grid(row=0, column=1, padx=10, pady=5)
        firstname_update_entry.insert(0, student["FirstName"])

        ttk.Label(update_window, text="Last Name:", font=label_font, foreground=label_color).grid(row=1, column=0)
        lastname_update_entry = ttk.Entry(update_window)
        lastname_update_entry.grid(row=1, column=1, padx=10, pady=5)
        lastname_update_entry.insert(0, student["LastName"])

        # Fields for editable contact number and email
        ttk.Label(update_window, text="Contact Number:", font=label_font, foreground=label_color).grid(row=2, column
        contactnumber_update_entry = ttk.Entry(update_window)
        contactnumber_update_entry.grid(row=2, column=1, padx=10, pady=5)
        contactnumber_update_entry.insert(0, student["ContactNumber"])

        ttk.Label(update_window, text="Email:", font=label_font, foreground=label_color).grid(row=3, column=0)
        email_update_entry = ttk.Entry(update_window)
        email_update_entry.grid(row=3, column=1, padx=10, pady=5)
        email_update_entry.insert(0, student["Email"])

        # Button to submit the updates
        update_button = ttk.Button(update_window, text="Update", command=lambda: submit_update(update_window, studen
        update_button.grid(row=4, column=0, columnspan=2, pady=10)
```

Main GUI: The below provided code snippet initializes the graphical user interface (GUI) for our Student Database Management System project using the Tkinter library in Python. It defines custom fonts and styles, creates frames to organize the layout, and includes input fields for registering new students.

Additionally, it implements buttons for actions such as registering new students, searching for specific students, displaying all students, deleting student records, updating student information, and clearing input fields. A Treeview widget is utilized to display student information in a tabular format, with columns for ID, first name, last name, date of birth, contact, and email. Overall, the code sets up a comprehensive GUI interface for managing student data effectively.

```
# Create the main GUI window
root = tk.Tk()
#root.title("Student Database Management System")

# Defining custom fonts and styles
heading_font = font.Font(family="Times New Roman", size=18, weight="bold", root=root)
label_font = font.Font(family="Times New Roman", size=12, root=root)
label_color = "#2B2B2B"


# Define frames for layout
top_frame = ttk.Frame(root)
top_frame.pack(fill=tk.X, padx=10, pady=10)
bottom_frame = ttk.Frame(root)
bottom_frame.pack(fill=tk.X, padx=10, pady=10)

# Heading label with custom font and color
heading_label = ttk.Label(top_frame, text="Student Database Management System", font=heading_font, foreground="#0045
heading_label.grid(row=0, columnspan=3, pady=10)

# Labels and input fields for student registration
ttk.Label(top_frame, text="First Name:", font=label_font, foreground=label_color).grid(row=1, column=0, sticky='w')
firstname_entry = ttk.Entry(top_frame)
firstname_entry.grid(row=1, column=1, padx=10, pady=5)

ttk.Label(top_frame, text="Last Name:", font=label_font, foreground=label_color).grid(row=2, column=0, sticky='w')
lastname_entry = ttk.Entry(top_frame)
lastname_entry.grid(row=2, column=1, padx=10, pady=5)

ttk.Label(top_frame, text="Date of Birth:", font=label_font, foreground=label_color).grid(row=3, column=0, sticky='w
dob_entry = ttk.Entry(top_frame)
dob_entry.grid(row=3, column=1, padx=10, pady=5)

ttk.Label(top_frame, text="Contact Number:", font=label_font, foreground=label_color).grid(row=4, column=0, sticky='
contactnumber_entry = ttk.Entry(top_frame)
contactnumber_entry.grid(row=4, column=1, padx=10, pady=5)

ttk.Label(top_frame, text="Email:", font=label_font, foreground=label_color).grid(row=5, column=0, sticky='w')
email_entry = ttk.Entry(top_frame)
email_entry.grid(row=5, column=1, padx=10, pady=5)

# Buttons for student registration, display, and search
register_button = ttk.Button(top_frame, text="Register Student", command=register_student)
register_button.grid(row=6, column=0, columnspan=2, pady=10)

search_entry = ttk.Entry(top_frame)
search_entry.grid(row=7, column=0, padx=10, pady=5, columnspan=2)

search_button = ttk.Button(top_frame, text="Search Student", command=search_student)
search_button.grid(row=7, column=2, padx=10, pady=5)

display_all_button = ttk.Button(top_frame, text="Display All Students", command=display_all_students)
display_all_button.grid(row=8, column=0, columnspan=2, pady=10)
```