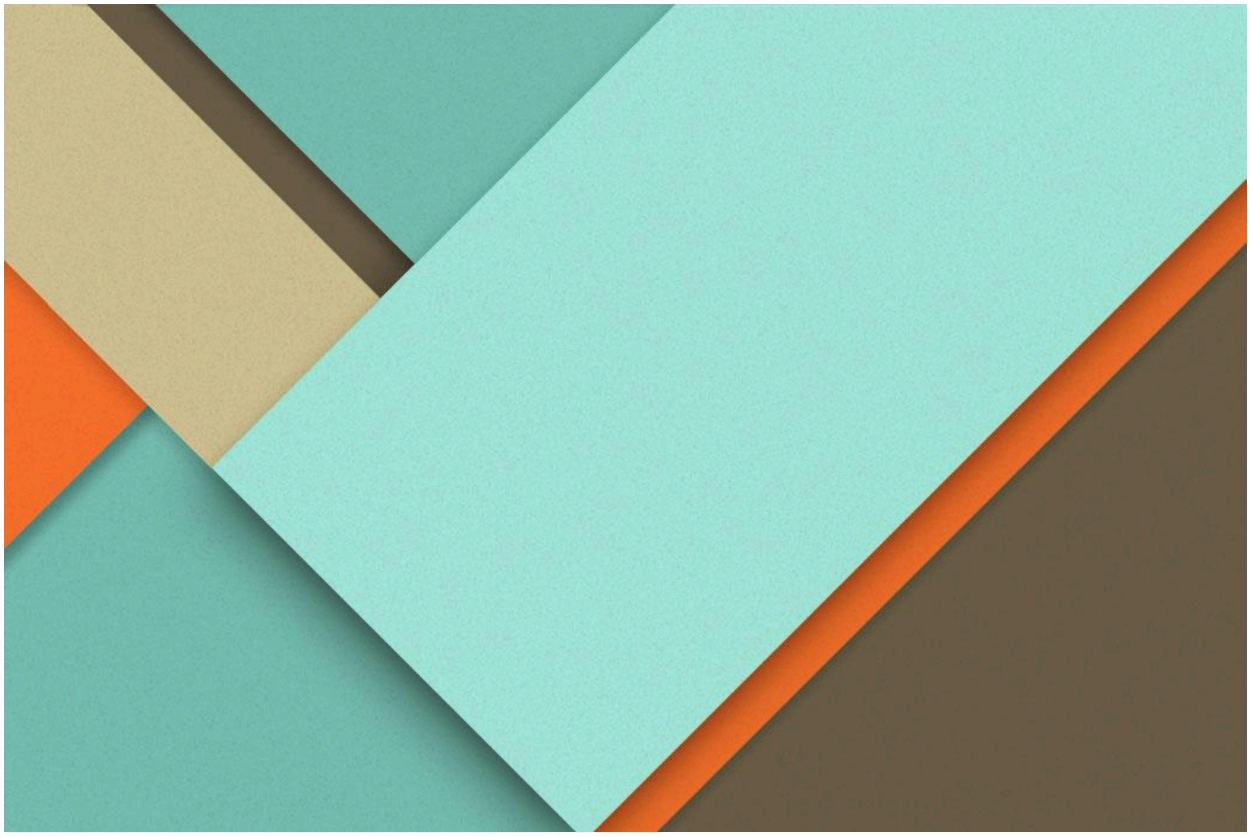


UniSmart

03.02.2024

Malak Sherif
iTi

Overview

UniSmart is a comprehensive and innovative project designed to address the complex data management needs of a university environment. This project combines various technologies, including SQL, PLSQL, Advanced PLSQL, Red Hat, Bash scripting, Java SE, and Object-Oriented Programming (OOP) principles, to create a sophisticated system that seamlessly integrates database management, automation, and application development.

Goals

Efficient Data Handling:

- Streamline data entry, retrieval, and modification processes for students, courses, departments, and grades.

Data Integrity and Normalization:

- Enforce data integrity through the implementation of a normalized relational database schema.
- Minimize redundancy and anomalies in data by adhering to normalization principles.

Reliable Automation:

- Implement robust Bash scripts for routine tasks like database backups and disk space monitoring.
- Ensure timely alerts and notifications for potential issues, contributing to the system's reliability.

User-Friendly Interface:

- Develop an intuitive Java application with a user-friendly interface for seamless interaction.
- Enable users to perform DML operations effortlessly, enhancing the overall user experience.

Accurate GPA Calculation:

- Implement PLSQL procedures for accurate GPA calculation, providing precise academic performance metrics for students.

Comprehensive Reporting:

- Integrate a reporting feature in the Java application to generate comprehensive reports on courses and student performance.
- Provide valuable insights into course enrollment and average GPA.

System Integration:

- Ensure seamless integration between the Java application and the underlying SQL database..

Enhanced Decision Support:

- Provide administrators and stakeholders with valuable data insights through the reporting feature.

UniSmart Specifications

We will need to keep data for

1. Students
2. Courses
3. Departments
4. Students’ Enrollments & Students’ Grades

UniSmart blueprint

Course Enrollment:

- A student can enroll in multiple courses.
- Each course can have multiple enrolled students.
- Student can fail and apply to the same course many times
- Also student can apply to the same course again to improve his grade

Departmental Affiliation:

- Students are affiliated with a specific department.
- Each department can have multiple students.
- A department can be deleted if it has no courses

Course Management:

- Courses are uniquely identified by a course ID.
- Each course is associated with a specific department.
- Course can be deleted if it has no enrolled students .
- Student may not be enrolled at any courses

Department Identification:

- Departments are uniquely identified by a department ID.
- Each department can have many courses.
- Department may have no students or courses in it.

Student Information:

- Student records are uniquely identified by a student ID.
- Student can be deleted if he is not enrolled at any courses.

Grade Recording:

- Grades are recorded for each student in each course.
- Each grade record includes the student's scored grade and enrollment date.
- The student can be recorded in a course without having grade
- The grade for the student can be deleted

Maximum Grade Limit:

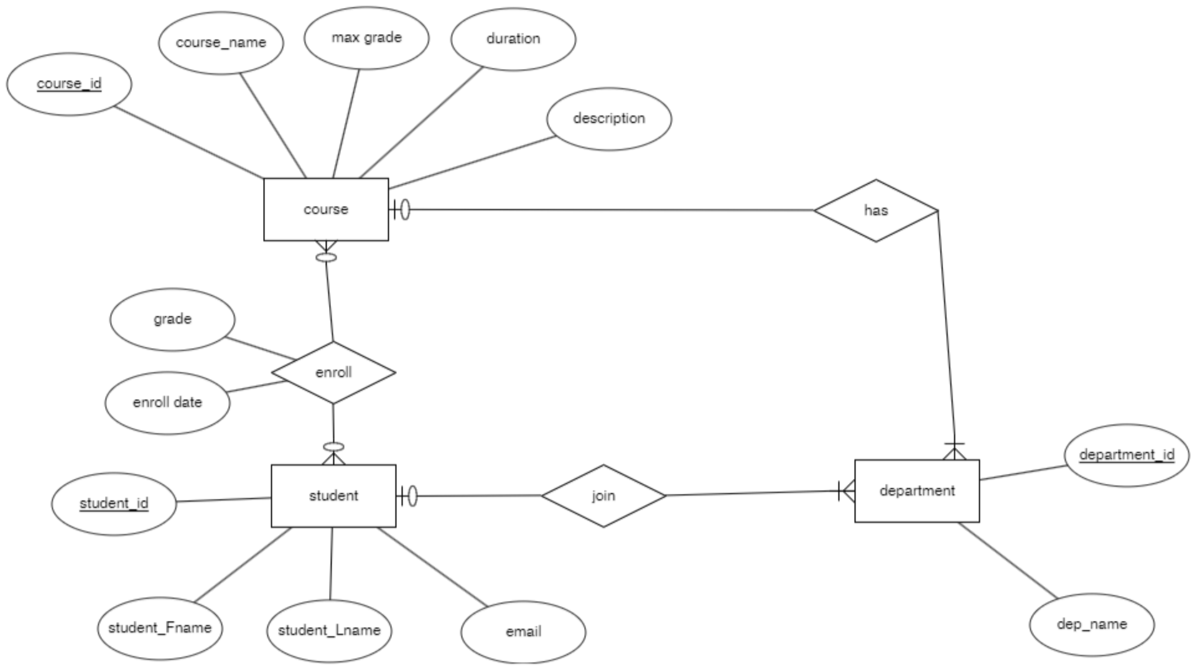
- Each course has a maximum grade limit specified.

GPA Calculation:

- System calculates the Grade Point Average (GPA) for each student based on scored grades and maximum grades.
- System calculates the Grade Point Average (GPA) for each course based on scored grades and maximum grades for each student in enrolled in that course
- Trigger implemented to check and enforce constraints on student scored grades and the max grade before insertion or update.

Milestones

1-Database Design



2-SQL Implementation

Database Schema Creation:

- Develop SQL scripts to create tables for students, courses, departments, and grades.
- Ensure appropriate data types, constraints, and relationships are defined.

Data Population:

- Create SQL scripts to populate the database with sample data.
- Ensure the correctness of the data insertion process.

Database Testing:

- Rigorous testing to validate the integrity and accuracy of the SQL scripts.
- Confirm that the database operates as expected with the provided sample data.

3-PLSQL Implementation

I have the following Database Objects in my CASE schema:

`UPDATE_STUDENT` Procedure:

- Purpose: Updates student information in the `STUDENTS` **table.**
- **Why: Standardizes and efficiently manages the process of updating student details.**
- **How: Takes parameters for student identification and new information.**

`caseGBA` Function:

- Purpose: Calculates GPA for a specific student.
- Why: Provides a streamlined way to assess a student's academic performance.
- How: Takes the student ID, retrieves grades and max grades, and computes the GPA.

`calculate_course_gba` Function:

- Purpose: Computes the average GPA for a specific course.
- Why: Offers insights into the overall performance of students in a course.
- How: Takes the course ID, retrieves student scores, calculates average GPA.

`trg_check_student_scored_grade` Trigger:

- Purpose: Ensures `student_scored_grade` **in** `GRADES` **doesn't exceed course maximum.**
- **Why: Maintains data integrity by preventing entry of grades beyond the course limit.**
- **How: Fires before** `INSERT` **or** `UPDATE` **in** `GRADES`**, checks and raises error if limit is exceeded.**

These components collectively contribute to efficient data management, GPA calculation, and data integrity enforcement in **UniSmart.**

4-Automation Scripts

DISK MONITORING

Bash Script to monitor the Hard Disk usage and send alerts in case a specific threshold is exceeded =>90% in a file named “masg.txt” .

Then used task scheduler to run this script everyday

DATABASE BACKUP

Perform a full backup of the database by exporting the database to file.

You will find references with screenshots in the field called “bashSCREENS”

5-Java Application Development

Java Source Code:

This section encompasses the source code for the Java application, allowing seamless utilization in any code editor for building and running the application.

Client:

This folder contains the Classes that we would use across the applications \

Database:

Within the 'database' directory, you'll find the Singleton class responsible for managing the Database Connection.(data access layer)

GUI:

In the 'gui' directory, discover the code for all front-end (GUI) classes. This directory also holds the source code for the application's backend, providing a comprehensive overview of the system's user interface and underlying functionality.

I used java fx method to design theGUI

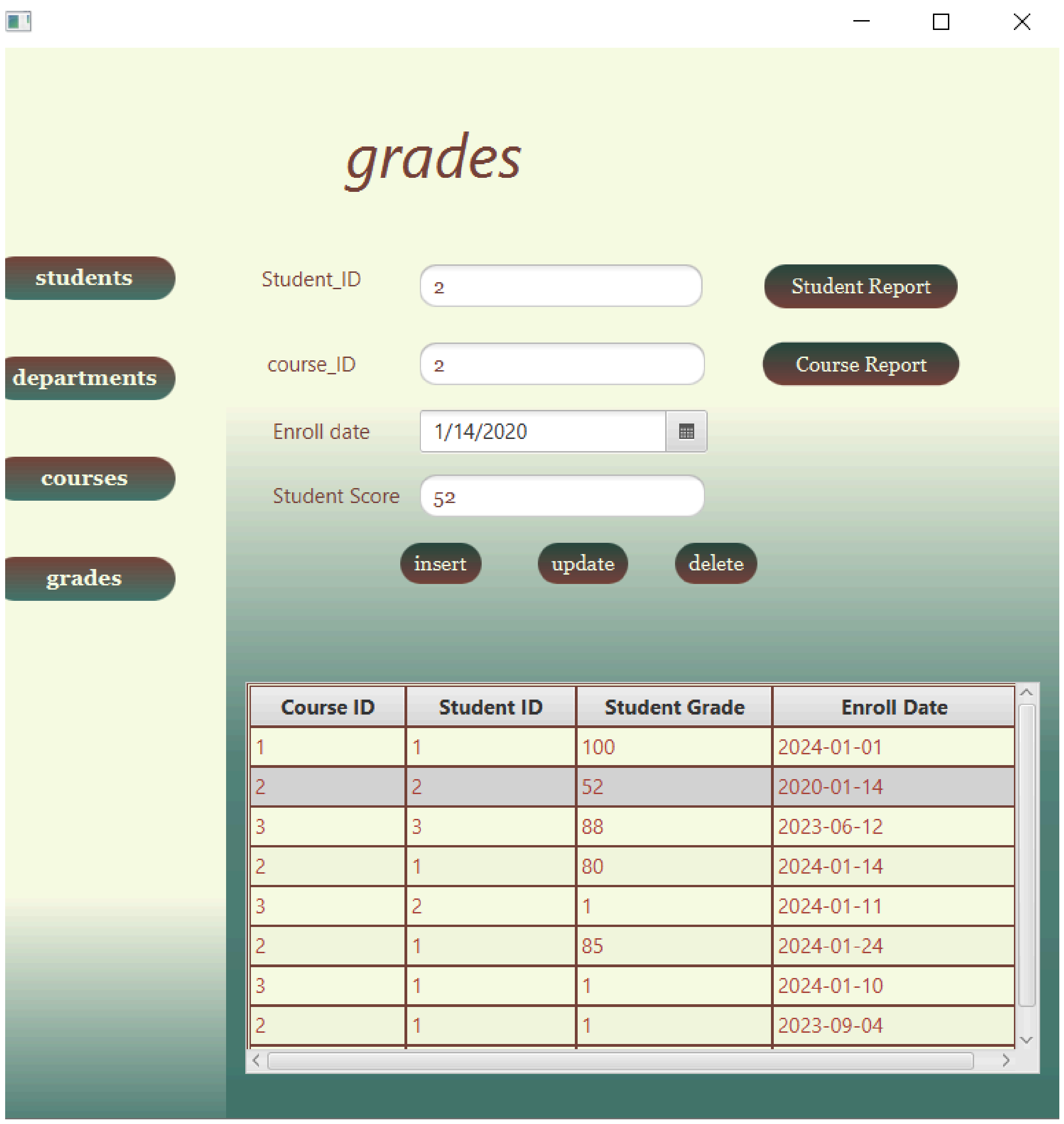
UniSmart has 6 scenes

- Grades
- Student report
- Course report
- Students
- Courses
- Departments

Grades

In this scene, you can see all the students that are enrolled in courses, along with the courses that they are in and their enrollment data and date, if there are any.

You can enroll students in courses or remove students from a course, and you can update the status of the student.



Student_ID: 2

course_ID: 2

Enroll date: 1/14/2020

Student Score: 52

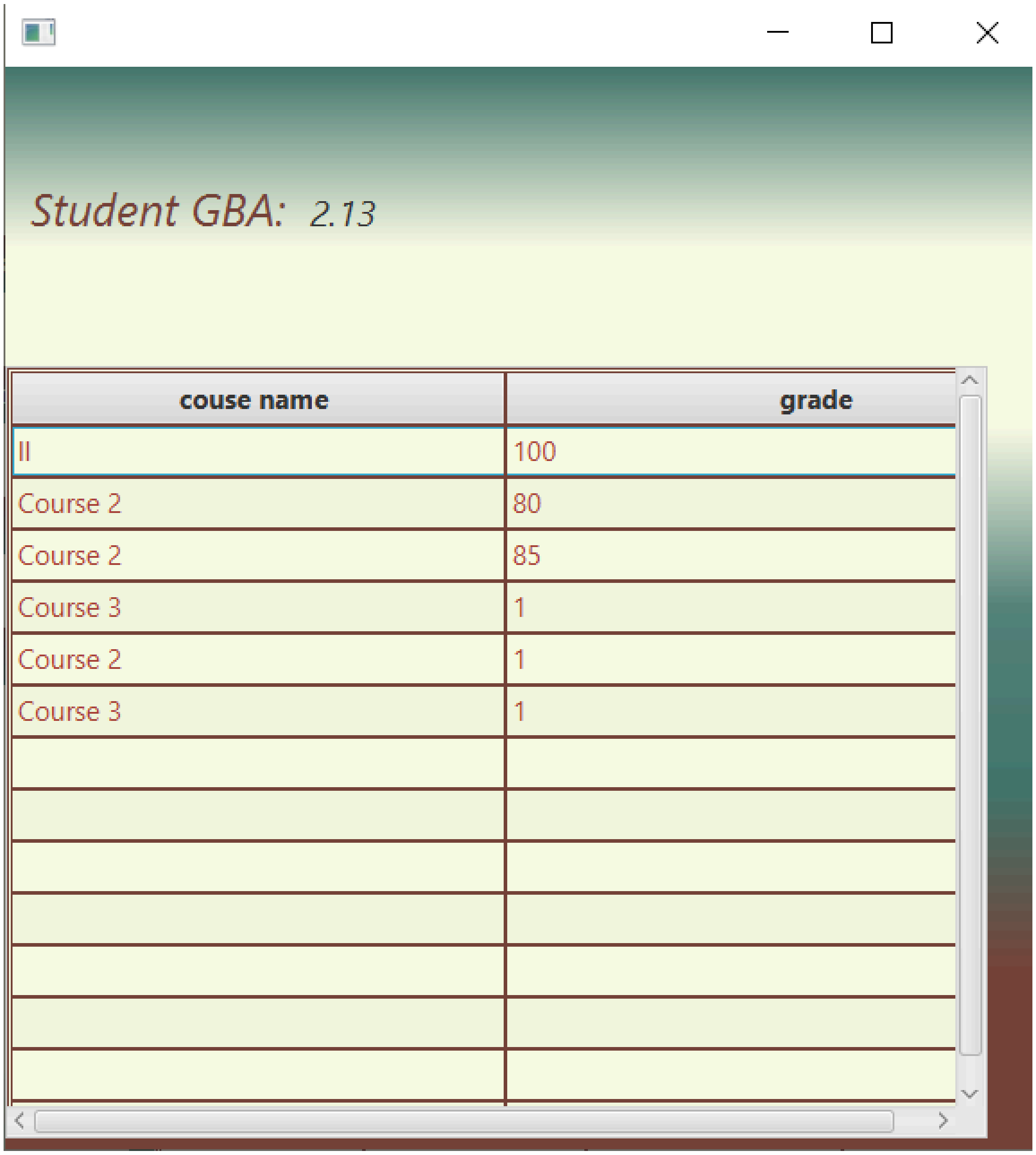
Course ID	Student ID	Student Grade	Enroll Date
1	1	100	2024-01-01
2	2	52	2020-01-14
3	3	88	2023-06-12
2	1	80	2024-01-14
3	2	1	2024-01-11
2	1	85	2024-01-24
3	1	1	2024-01-10
2	1	1	2023-09-04

Also, by clicking the report buttons “student report" and “course report,"

You can view a full report for the selected student or course.

Student report

In this scene, you will see all the course names of the selected student based on his grade in each course, and you will also see the average GPA of this student.



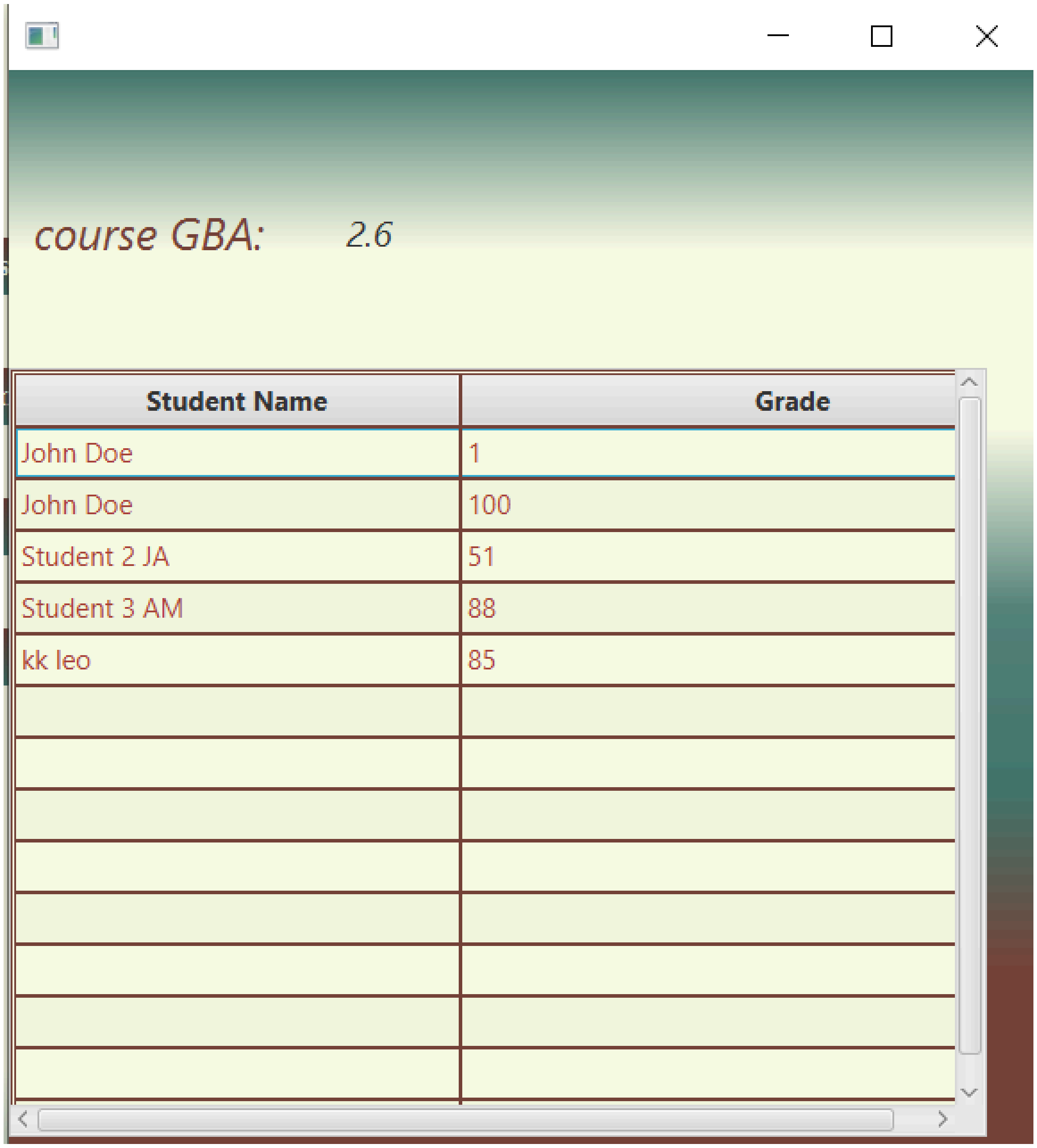
Student GBA: 2.13

couse name	grade
ll	100
Course 2	80
Course 2	85
Course 3	1
Course 2	1
Course 3	1

Course report

In this scene, you will see all the names of the students who are enrolled in that course based on their scores, and you will also see the GBA for that course.

This scene and the student report scene are a result of the data access layer method that uses a callable start to call the PLSQL functions, and another method that uses joins.



course GBA: 2.6

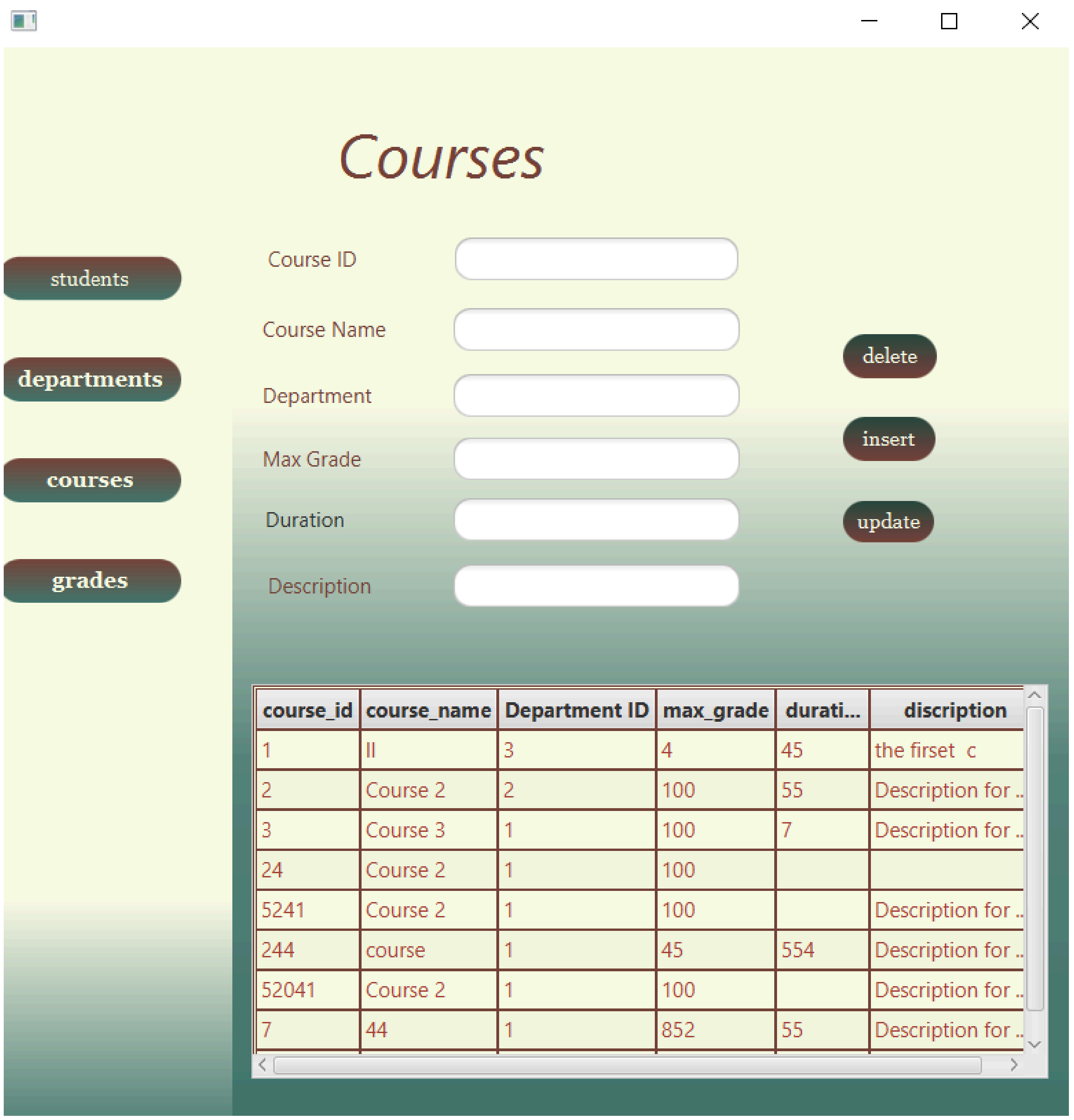
Student Name	Grade
John Doe	1
John Doe	100
Student 2 JA	51
Student 3 AM	88
kk leo	85

COURSES

In this scene, the admin can insert a new course or update existing courses.

The duration and the descriptions are the only columns in the courses table that accept null values, so the user can insert a new course without them. Also, the user can update any number of columns he wants just by using the course_id.

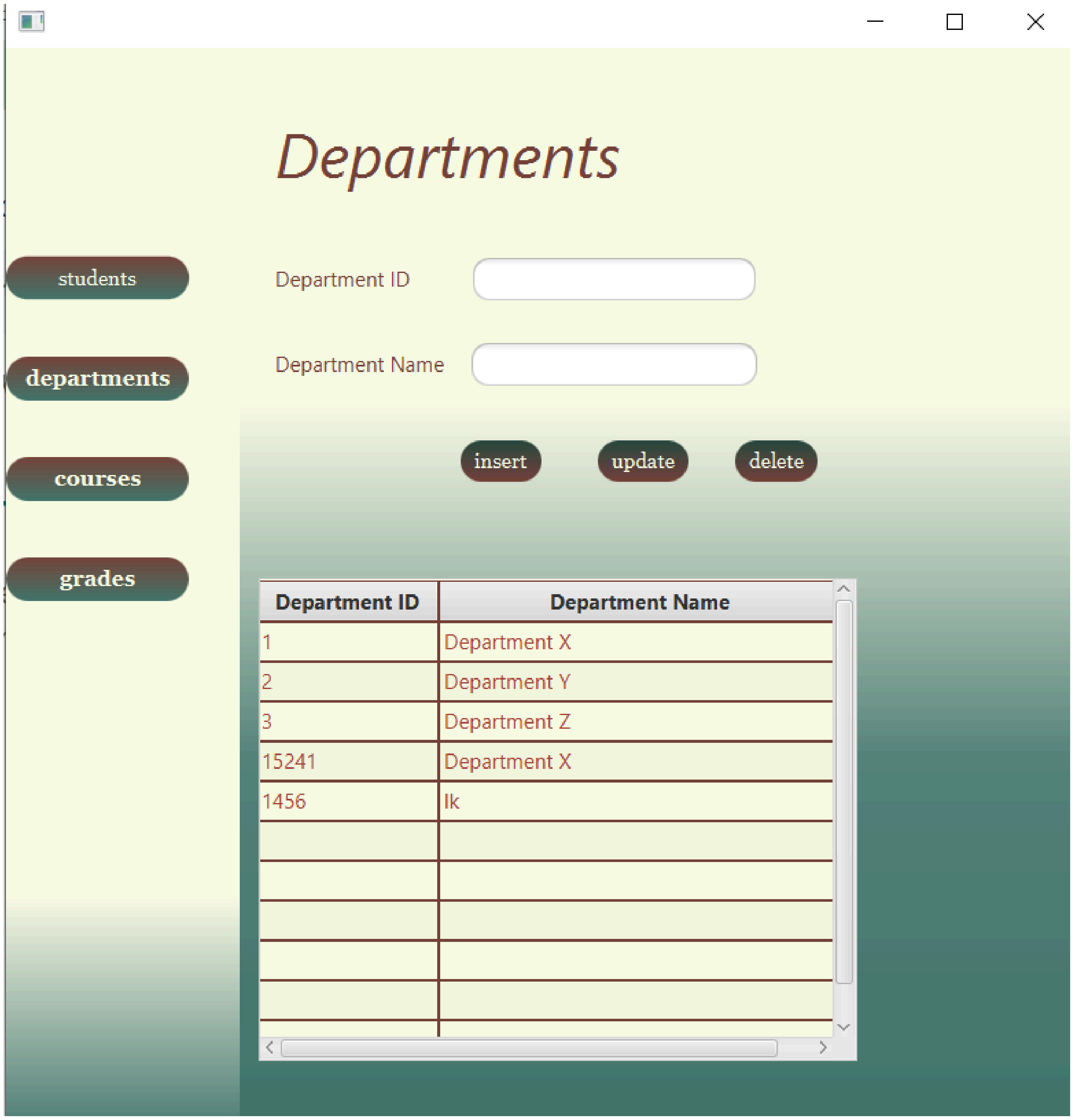
The update function for that scene in the data access layer uses a string builder with complex conditions to allow the user to update any column he wants.



course_id	course_name	Department ID	max_grade	durati...	discription
1	ll	3	4	45	the firset c
2	Course 2	2	100	55	Description for ..
3	Course 3	1	100	7	Description for ..
24	Course 2	1	100		
5241	Course 2	1	100		Description for ..
244	course	1	45	554	Description for ..
52041	Course 2	1	100		Description for ..
7	44	1	852	55	Description for ..

DEPARTMENTS

In this scene, the admin can add a new department. The admin must enter the department ID and name, and the admin can use the update to update the department name.



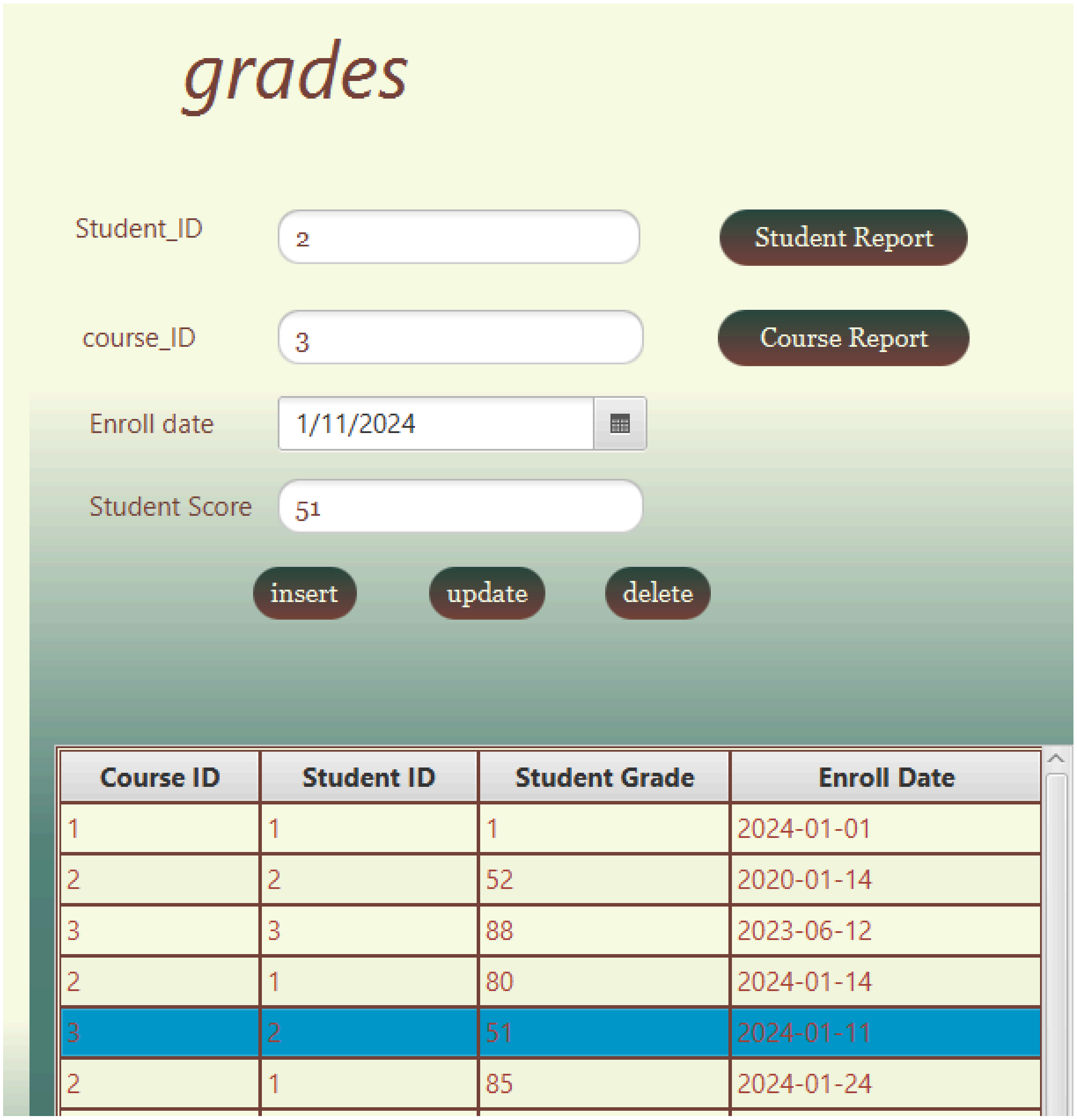
Departments

Department ID

Department Name

Department ID	Department Name
1	Department X
2	Department Y
3	Department Z
15241	Department X
1456	lk

In all the previous scenes, the user table views, and the admin can press on any row in that table view to select data from it, and the data will be automatically written to the text field, so I did not add a select button to them. However, in the student scene, I added a select button instead of the table view to protect students' private information.



Student_ID	2
course_ID	3
Enroll date	1/11/2024
Student Score	51

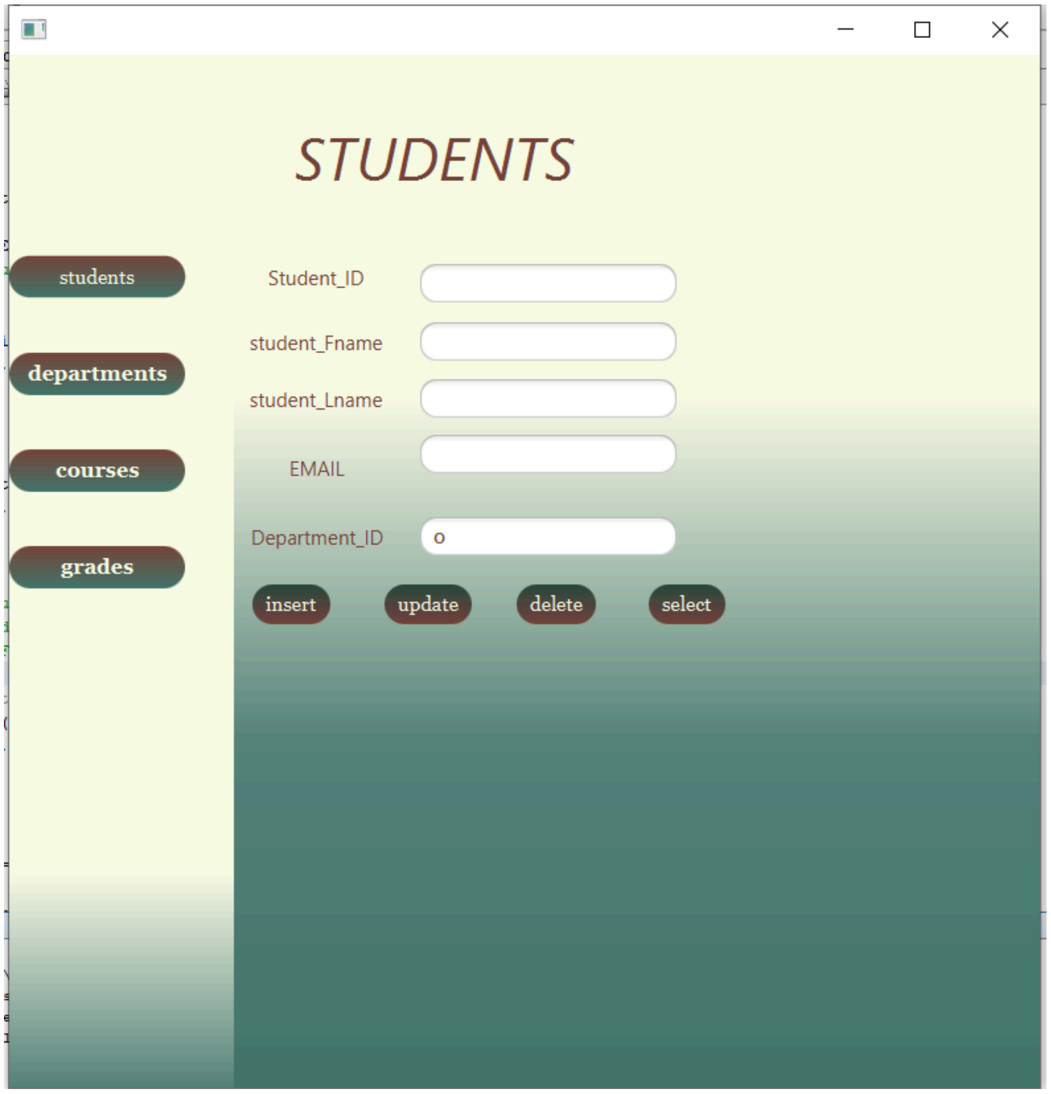
Course ID	Student ID	Student Grade	Enroll Date
1	1	1	2024-01-01
2	2	52	2020-01-14
3	3	88	2023-06-12
2	1	80	2024-01-14
3	2	51	2024-01-11
2	1	85	2024-01-24

When the admin enter the student id and press select , all the other student info will how in the other text fields

STUDENTS

The student scene also has an update function that is similar to that in the course scene and also uses a string builder to enable the user to update any number of columns he wants.

The delete button can delete any student who is not enrolled in any course.



One great feature of the java app is that is provides **User-Friendly App and Error Help**

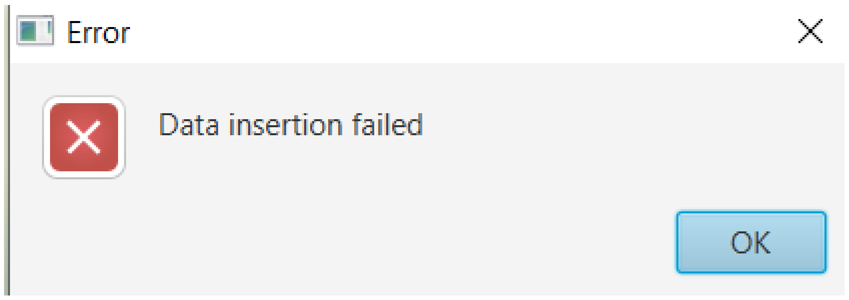
Easy to Use:
The app is designed to be simple and easy for users to understand. You won't get lost!

Clear Forms:
When you fill in information, the app makes sure you know what to put where. Labels tell you what's needed.

Quick Error Feedback:
If you make a mistake, the app tells you right away what went wrong. It's like having a helpful guide.
For numbers, like student IDs, the app checks if you put in the right kind of information. It tells you if you use letters when you shouldn't.

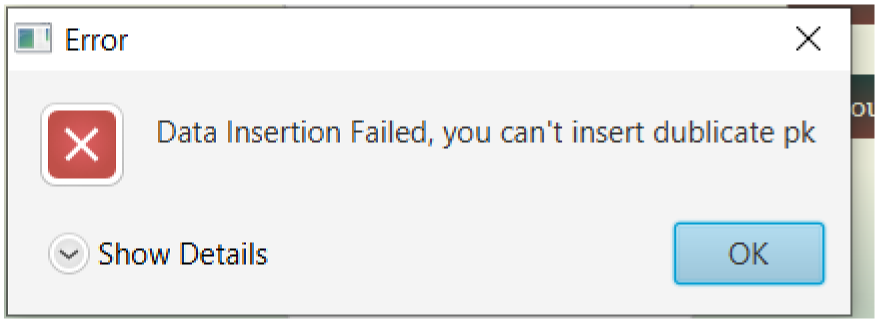
Simple Error Alerts:
If something goes wrong, the app tells you in simple words what happened. No confusing messages.
If you make a mistake, the app always talks to you in the same way. No surprises!

Stop Common Mistakes:
It stops you if you forget to put in important things, like a student ID, so you don't mess up.
Here is some massage error examples



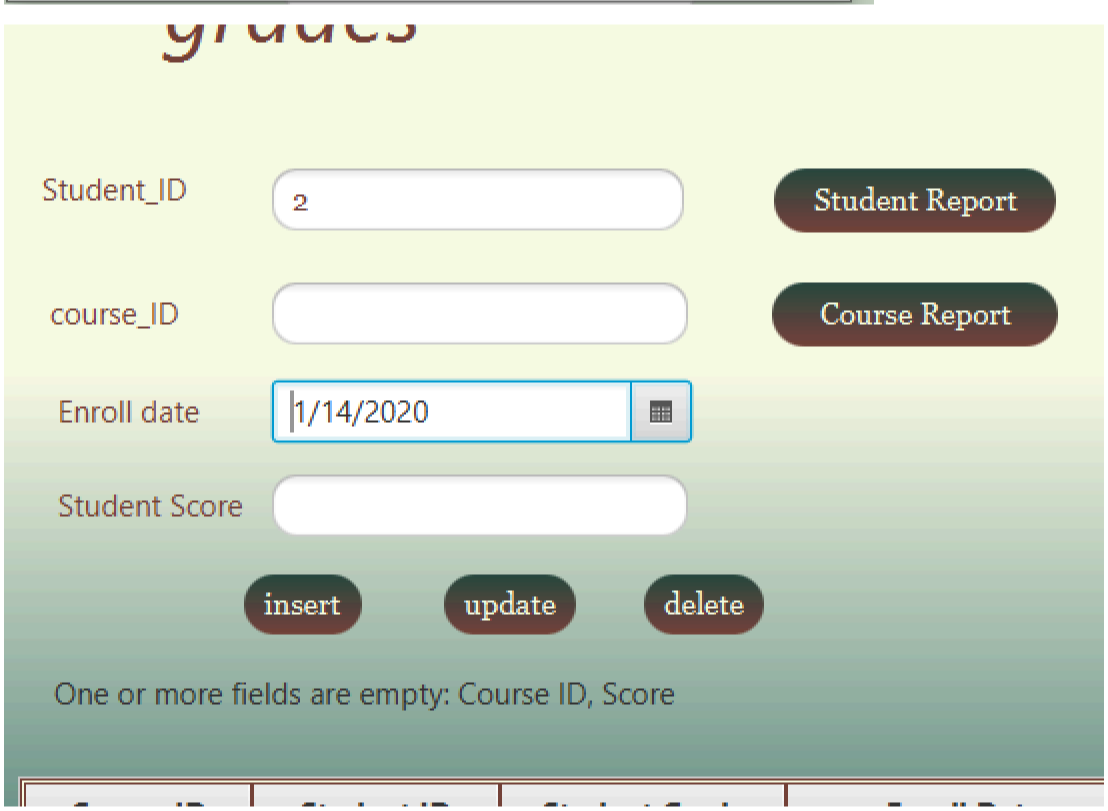
Error

Data insertion failed



Error

Data Insertion Failed, you can't insert dublicate pk



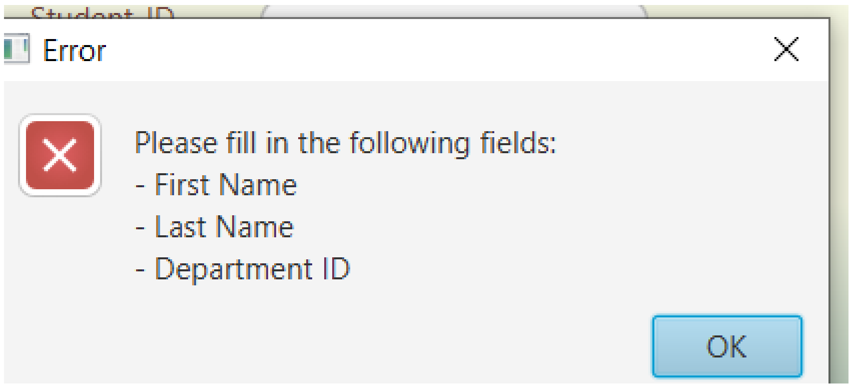
Student_ID: 2

course_ID:

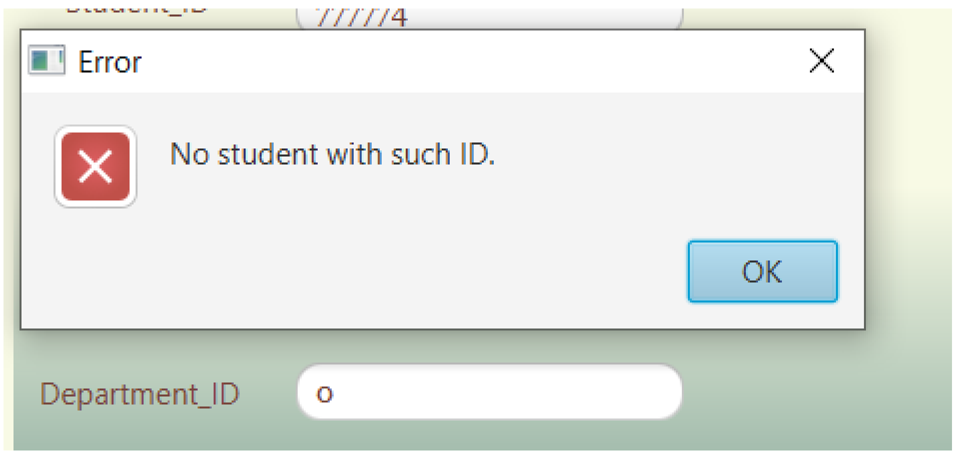
Enroll date: 1/14/2020

Student Score:

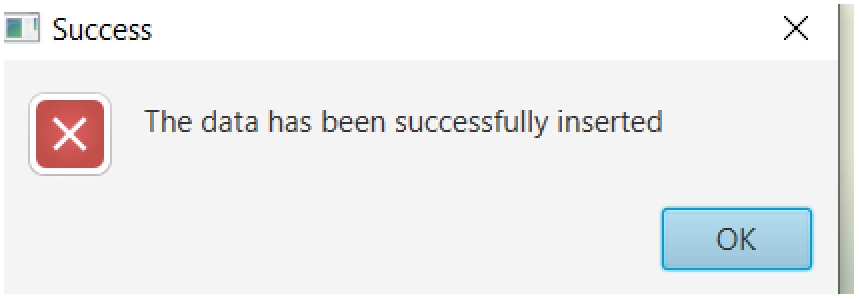
One or more fields are empty: Course ID, Score


Error

Please fill in the following fields:
- First Name
- Last Name
- Department ID

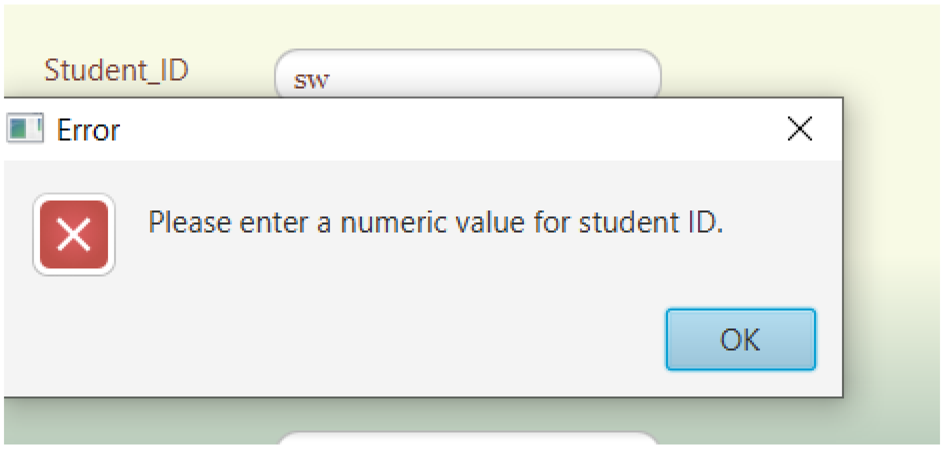

Error

No student with such ID.

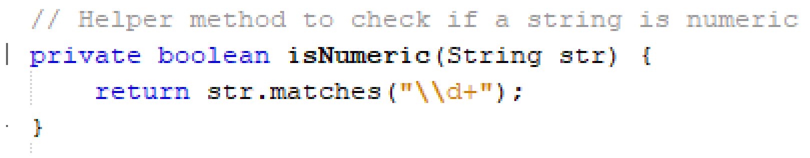

Success

The data has been successfully inserted

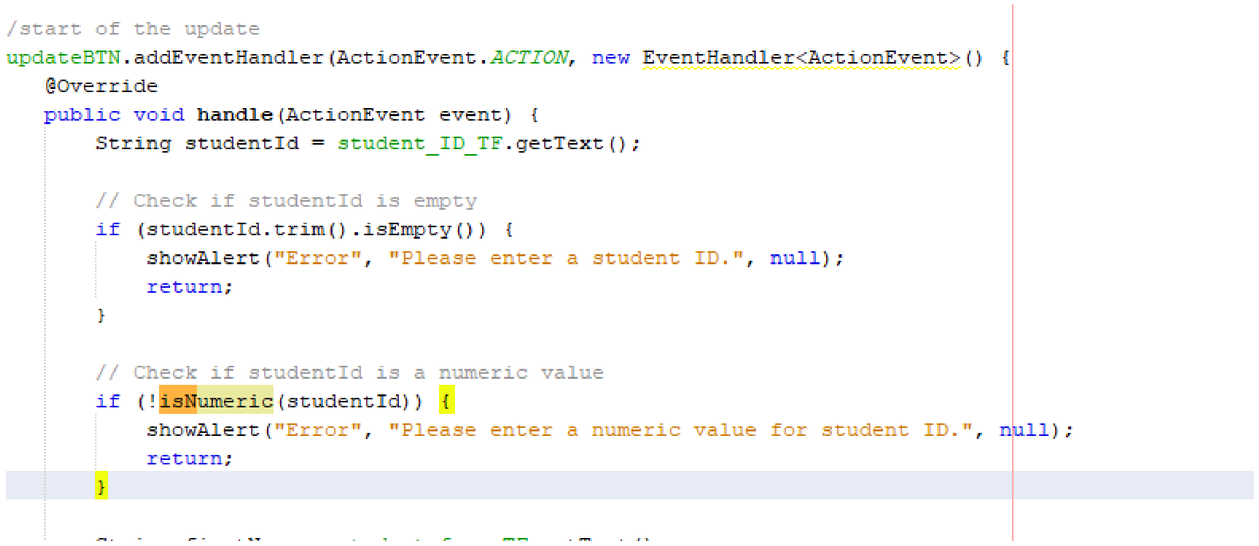
One method that helped a lot was "isNumeric", it prints that error message when the user enter any char in the student_id text filed



Please enter a numeric value for student ID.



```
// Helper method to check if a string is numeric
private boolean isNumeric(String str) {
    return str.matches("\\d+");
}
```



```
/start of the update
updateBTN.addEventHandler(ActionEvent.ACTION, new EventHandler<ActionEvent>() {
   @Override
   public void handle(ActionEvent event) {
       String studentId = student_ID_TF.getText();

       // Check if studentId is empty
       if (studentId.trim().isEmpty()) {
           showAlert("Error", "Please enter a student ID.", null);
           return;
       }

       // Check if studentId is a numeric value
       if (!isNumeric(studentId)) {
           showAlert("Error", "Please enter a numeric value for student ID.", null);
           return;
       }
```