REPORT HOMEWORK LAB 4

Student Name: Nguyễn Tiến Anh

Student ID: ITITDK22128

Course: Web Application Development

Lab 4: JSP + MYSQL - CRUD OPERATIONS

Github: https://github.com/TienAnh0108/Lab4_Exercise.git

Ex 5:

```
<form action="list_students.jsp" method="GET" class="search-form">
    <input type="text" name="keyword" placeholder="Search by name or code..."
           value="<%= Objects.toString(request.getParameter("keyword"), "")%>">
    <button type="submit">Search</button>
    <a href="list_students.jsp">Clear</a>
</form>
```

<form action="list_students.jsp" method="GET">: Defines the search form. The action is set to the same page (list_students.jsp), meaning the page reloads itself with the search results. The method="GET" is used so the search term is visible in the URL.

value="<%= Objects.toString(...) %>": This JSP expression dynamically sets the input field's value to the search term the user just entered. This is essential for persisting the search term, so the user knows what they searched for after the page reloads.

<button type="submit">Search</button>: Triggers the form submission, reloading list_students.jsp with the keyword parameter in the URL.

```
String keyword = request.getParameter("keyword");
String sql;
boolean isSearching = false;

if (keyword != null && !keyword.trim().isEmpty()) {
    isSearching = true;
    sql = "SELECT * FROM students WHERE UPPER(full_name) LIKE ? OR UPPER(student_code) LIKE ? OR UPPER
} else {
    // Normal query
    sql = "SELECT * FROM students ORDER BY id DESC";
}

try {
    Class.forName("com.mysql.cj.jdbc.Driver");

    conn = DriverManager.getConnection(
            "jdbc:mysql://localhost:3306/student_management",
            "root",
            "Tienanh0108!"
    );

    pstmt = conn.prepareStatement(sql);

    if (isSearching) {
        String searchParam = "%" + keyword.toUpperCase() + "%";

        pstmt.setString(1, searchParam);
        pstmt.setString(2, searchParam);
        pstmt.setString(3, searchParam);
    }

    rs = pstmt.executeQuery(); // Execute the query
```

String keyword = request.getParameter("keyword"); - Retrieves the search term entered by the user from the URL (if method="GET" was used) using the name keyword.

if (keyword != null && !keyword.trim().isEmpty()) { ... } - Server-Side Validation: Checks if a keyword was submitted and is not just empty spaces. If a valid keyword exists, it sets isSearching = true.

sql = "SELECT * FROM students WHERE UPPER(full_name) LIKE ? OR UPPER(student_code) LIKE ? OR UPPER(major) LIKE ?"; - Search Query: If isSearching is true, this secure SQL query is constructed. It uses the LIKE operator and the UPPER() function (for case-insensitivity) to search across the Full Name, Student Code, and Major fields using three placeholders (?).

else { sql = "SELECT * FROM students ORDER BY id DESC"; } - Default Query: If no keyword is present, it uses the standard query to fetch all students, sorted by id descending.

conn = DriverManager.getConnection(...) - Establishes the connection to the MySQL database (student_management) using the defined credentials (root, "Tienanh0108!").

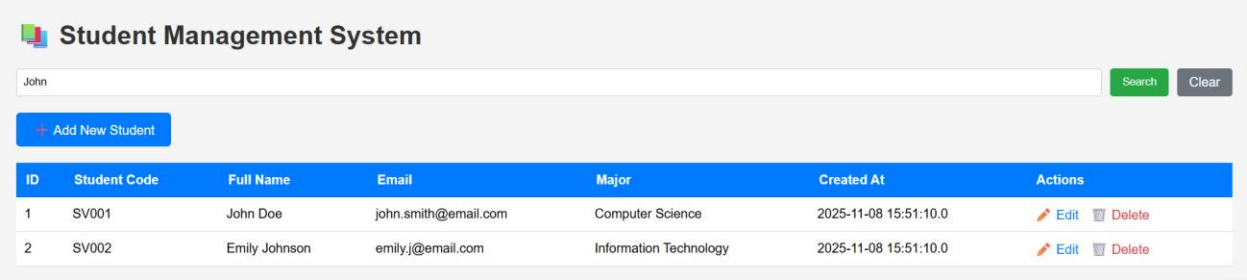
pstmt = conn.prepareStatement(sql); - Creates a PreparedStatement object. This is a crucial security measure against SQL Injection, as it compiles the query structure separately from the data.

if (isSearching) { ... pstmt.setString(N, searchParam); } - Parameter Binding: If a search is active, it creates the search parameter ("%" + keyword.toUpperCase() + "%") and binds it to the three placeholders (?) in the SQL query using setString(1), setString(2), and setString(3).

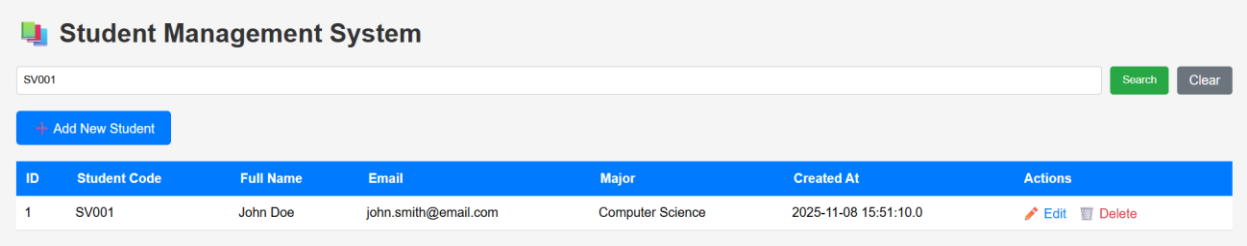
rs = pstmt.executeQuery(); - Executes the final, safe query against the database, storing the resulting student records in the ResultSet (rs) for display in the HTML table.

Result:

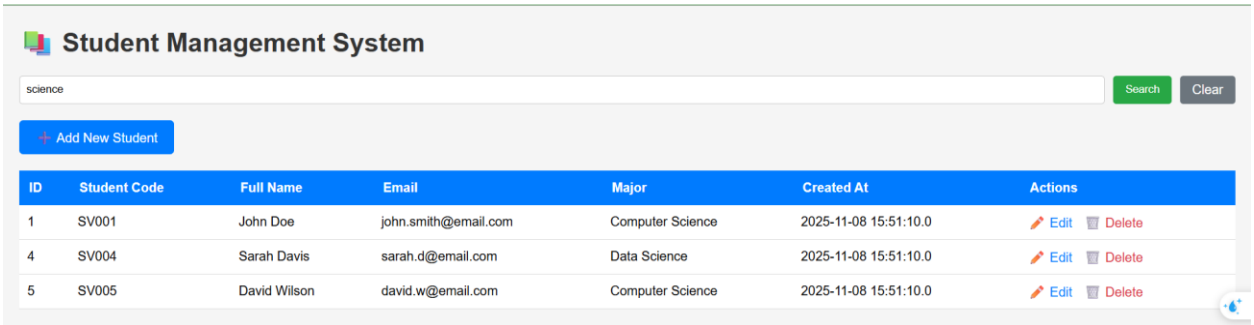
Testcase 1:



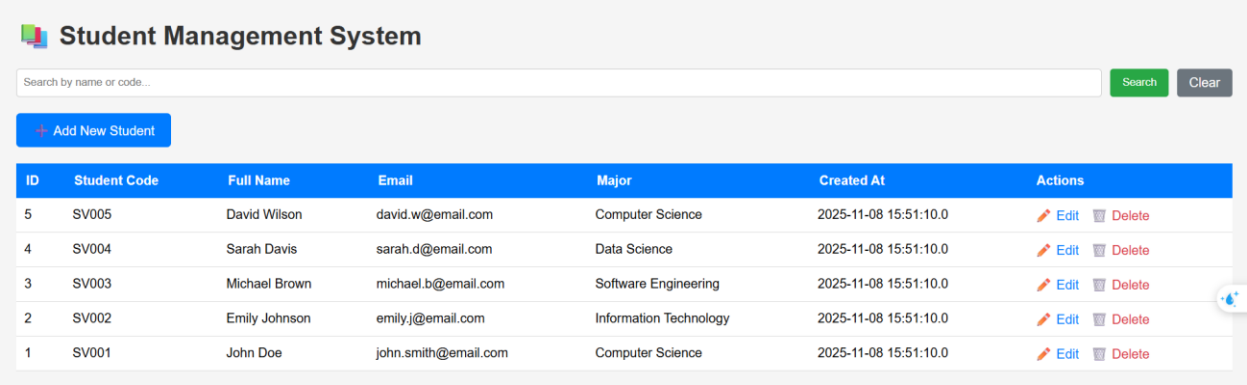
Testcase 2:



Testcase 3:



Testcase 4:



Ex 6.1:

```
if (email != null && !email.trim().isEmpty()) {
    String emailRegex = "^[a-zA-Z0-9+._-]+@[a-zA-Z0-9.-]+\\.[a-zA-Z]{2,}$";
    if (!email.matches(emailRegex)) {
        response.sendRedirect("add_student.jsp?error=Invalid email format");
        return;
    }
}
```

if (email != null && !email.trim().isEmpty()) { - **email != null**: Ensures the variable holds a value and is not null (i.e., the parameter exists); **&&**: Logical AND; **!email.trim().isEmpty()**: Ensures that after removing any leading or trailing whitespace (trim()), the string is **not empty**.

String emailRegex = "^[a-zA-Z0-9+._-]+@[a-zA-Z0-9.-]+\\.[a-zA-Z]{2,}$"; -

- ^: Matches the beginning of the string.
- [a-zA-Z0-9+._-]+: Matches one or more standard characters, numbers, plus signs, dots, underscores, or hyphens (the local part before the @).
- @: Matches the literal @ symbol.
- [a-zA-Z0-9.-]+: Matches one or more characters, numbers, dots, or hyphens (the domain name).
- \\.: Matches a literal dot (the separator before the domain extension).
- [a-zA-Z]{2,}$: Matches the domain extension (TLD, e.g., "com", "net", "uk"), ensuring it has at least two letters and matches the end of the string ($).

if (!email.matches(emailRegex)) { - This is a Java method that returns true if the entire email string matches the defined emailRegex pattern; !: The negation operator. The inner block executes **only if the match fails** (i.e., the email format is invalid).

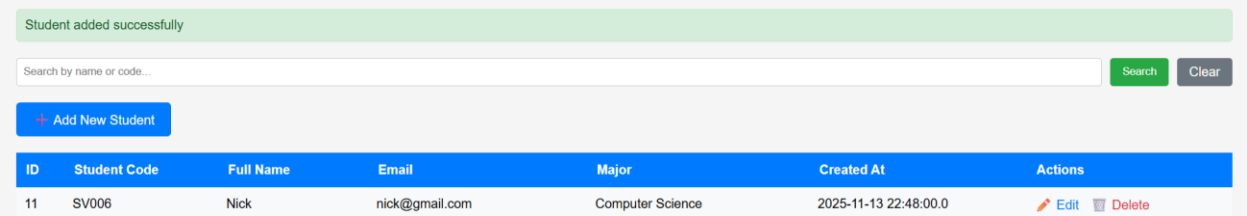
response.sendRedirect(...): The user is immediately redirected back to the form page (add_student.jsp).

?error=Invalid email format: An error message is appended to the URL as a query parameter so the JSP can display the validation failure to the user.

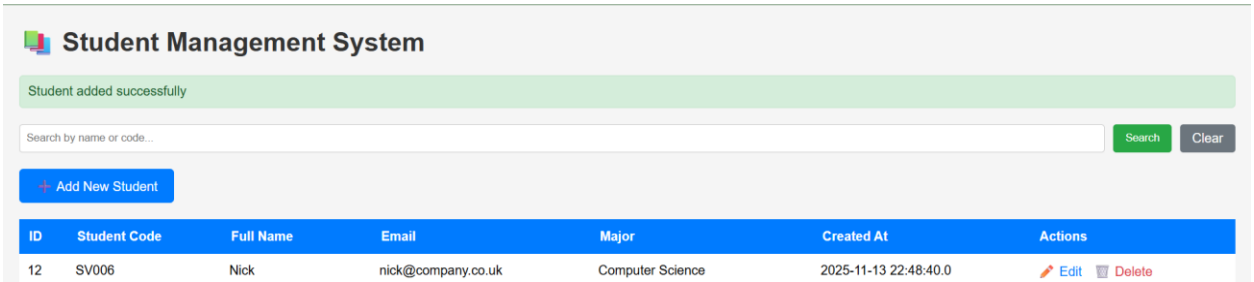
return;: Stops the execution of the JSP or Servlet method, preventing the invalid data from reaching the database.

Result:

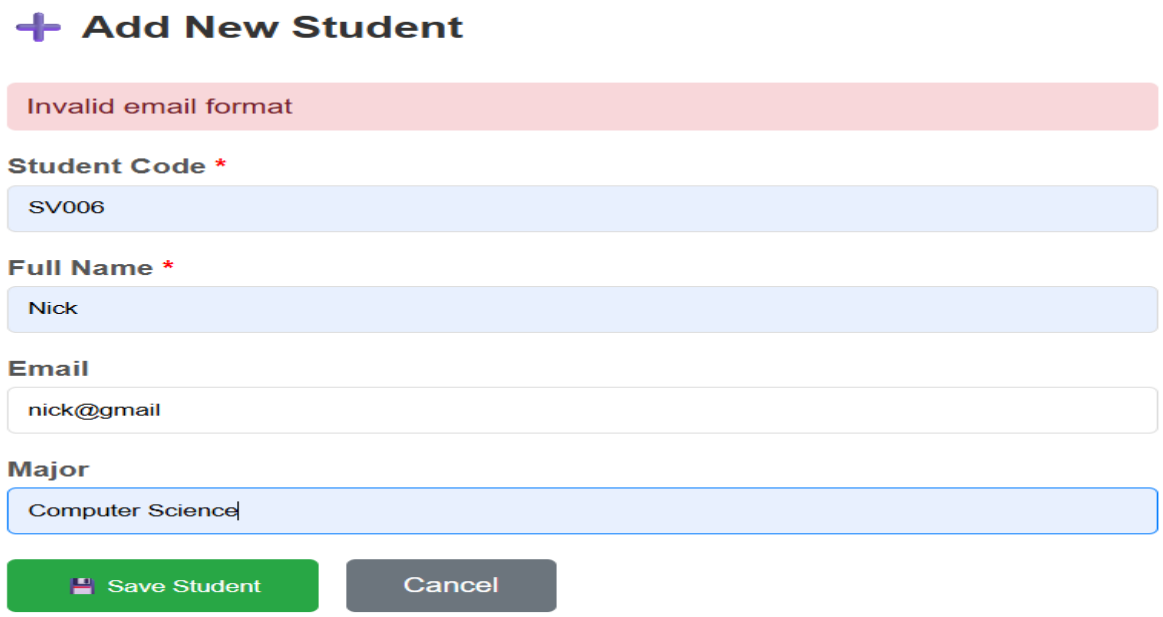
Testcase 1:



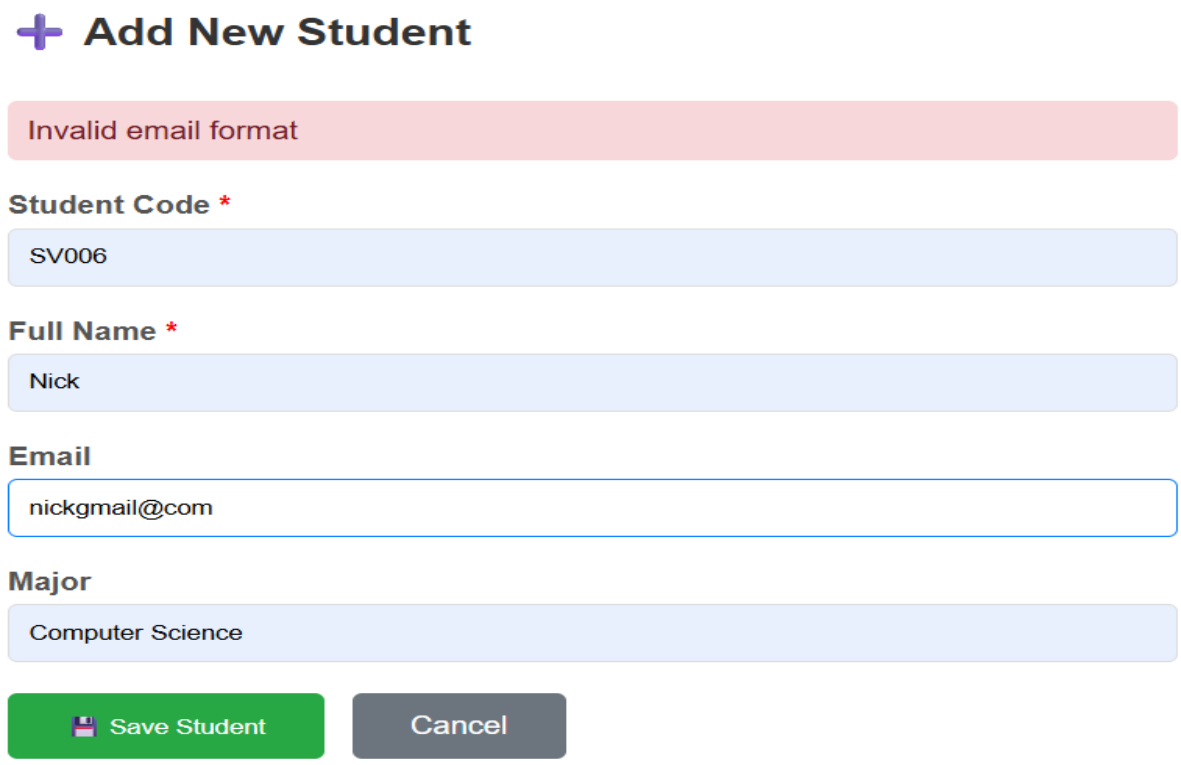
Testcase 2:



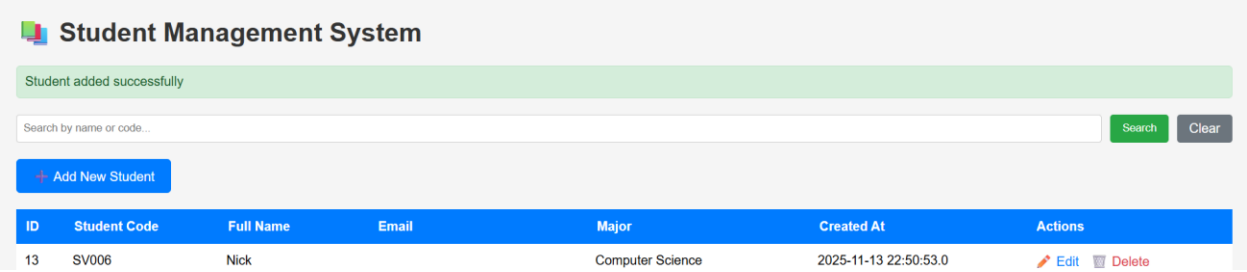
Testcase 3:



Testcase 4:



Testcase 5:



Ex 6.2:

```
if (studentCode != null) {
    String codeRegex = "[A-Z]{2}[0-9]{3,}";
    if (!studentCode.matches(codeRegex)) {
        response.sendRedirect("edit_student.jsp?id=" + idParam + "&error=Invalid Student Code format. Must be 2 upperca
        return;
    }
}
```

if (studentCode != null) { - The logic block begins by checking if the **studentCode** variable is **not null**. In the context of a form submission, this means the parameter for the student code was received from the client.

String codeRegex = "[A-Z]{2}[0-9]{3,}";

This line defines the **Regular Expression** that the student code must match. The pattern enforces the following structure:

- **[A-Z]{2}**: Exactly **two** uppercase English letters (A through Z).
- **[0-9]{3,}**: Followed by a minimum of **three** digits (0 through 9). The comma means it matches three or more digits.

if (!studentCode.matches(codeRegex)) {

This line performs the validation:

- **studentCode.matches(codeRegex)**: Checks if the entire studentCode string conforms to the defined pattern.
- **!**: The negation operator. The inner block executes **only if the code does NOT match** the required format.

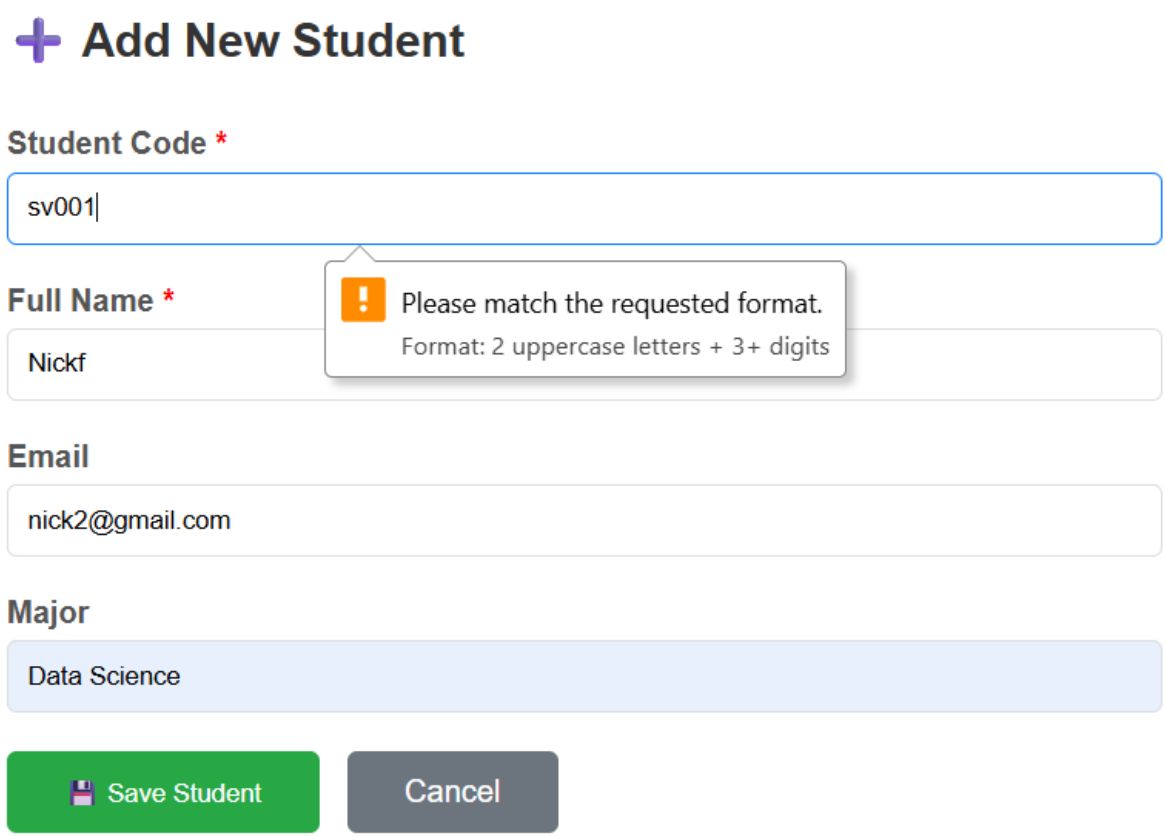
response.sendRedirect("edit_student.jsp?id=" + idParam + "&error=Invalid Student Code format. Must be 2 upperca...");

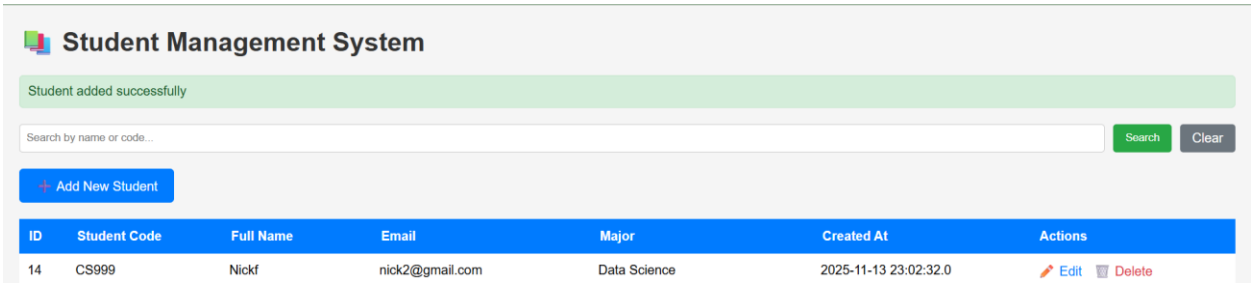
return;

If the validation fails:

- **response.sendRedirect(...)**: The user is immediately redirected back to the edit form (edit_student.jsp).
- **id=" + idParam**: The original student ID (idParam) is crucial and is passed back as a query parameter so the edit form knows which student data to reload and display.
- **&error=Invalid Student Code format...**: A clear error message is passed to the form to inform the user exactly what rule was violated.
- **return;**: This halts the execution of the current script, preventing the invalid code from being used in the subsequent database UPDATE query.

Testcase:




Student Management System
Student added successfully
Search by name or code...
Search
Clear
Add New Student
ID
Student Code
Full Name
Email
Major
Created At
Actions
14
CS999
Nickf
nick2@gmail.com
Data Science
2025-11-13 23:02:32.0
Edit
Delete