

19CS51C - JAVA PROGRAMMING

TITLE: STUDENT GRADING SYSTEM

NAME: NARMADA M – 2212028

Design (5)	Code (5)	Screenshots (5)	Time Management (5)	Total (20)

PROJECT DESCRIPTION:

The Student Grading System is a desktop-based application designed to manage and track student records efficiently. Built using Java Swing for the graphical user interface (GUI) and MySQL for the backend database, this system allows the user to perform CRUD (Create, Read, Update, Delete) operations on student data, including their name, roll number, marks, and corresponding grades. The system also includes features like searching student records, exporting data to CSV files, and calculating grades based on marks.

KEY FEATURES:

Add Student:

- Users can add new student details including name, roll number, and marks.
- The system automatically calculates the grade based on the student's marks.

Update Student:

Existing student records can be updated by selecting the student from the table and editing their details.

Delete Student:

Users can remove student records from the database by selecting the student from the table.

Search Functionality:

Users can search for students by name or roll number, with instant results shown in the table.

Export to CSV:

The system allows users to export the entire list of students into a CSV file for external usage and record keeping.

Grade Calculation:

The system automatically assigns grades to students based on their marks:

Marks ≥ 90 : Grade A

Marks ≥ 75 : Grade B

Marks ≥ 60 : Grade C

Marks ≥ 50 : Grade D

Marks < 50 : Grade F

Data Highlighting:

The student records table is color-coded, with failing students (marks below 50) highlighted in red for easy identification.

Technologies Used:

- Java Swing: For building the user interface, including forms and tables.
- MySQL: As the relational database to store student records.
- JDBC: To facilitate communication between the Java application and the MySQL database.
- File Handling (CSV): To export student data into CSV format for external usage.

- **MVC Architecture:** The application follows the Model-View-Controller pattern to separate the logic and UI components.

COMPONENTS:

Student Model (Student.java):

Represents the structure of the student entity with attributes like id, name, roll number, marks, and grade.

Student DAO (StudentDAO.java):

Provides database operations for adding, updating, deleting, and fetching students from the MySQL database.

Includes logic for grade calculation and searching students by name or roll number.

Database Connection (DatabaseConnection.java):

Manages the connection between the Java application and the MySQL database using JDBC.

Includes methods to open and close the database connection and check if the connection is active.

User Interface (StudentGradingSystemUI.java):

- The core user interface of the application, designed with Java Swing.
- Allows the user to add, edit, delete, and search student records.
- Features like "Export to CSV" and visual highlighting of failing students are implemented in the UI.

Database Configuration (db.properties):

Stores the database connection information, such as the URL, username, and password, making the application easy to configure.

How It Works:

- **Adding Students:** The user enters a student's name, roll number, and marks, and clicks the "Add Student" button. The system calculates the grade based on marks and stores the data in the MySQL database.
- **Updating and Deleting Students:** The user can select a student from the table, update the details, or delete the record as needed.
- **Searching:** The user can type a name or roll number in the search box, and matching records are instantly displayed.

Export to CSV:

The user can export all student records into a students.csv file, which can be opened in spreadsheet applications like Excel.

SYSTEM REQUIREMENTS:

JDK Version: JDK 17
Database: MySQL
IDE: Visual Studio Code

DATABASE DESIGN:

Database Schema:

Table: students
id (int, primary key, auto-increment)
name (varchar)
roll_number (varchar)
marks (int)
grade (varchar)

The database is designed to manage student information, including personal details, academic marks, and grades. The database consists of a single table called students, which stores all necessary data related to students and their performance. Below is an explanation of the database and its structure.

1. Database Name: student_db

The student_db database is the core of the system. It holds the student information in a table that records individual student data, including their name, roll number, marks, and grade.

2. Table: students

The students table is the primary table in this database. It contains fields that store key student information like their unique identifier (ID), name, roll number, marks, and grade.

Column Name	Data Type	Constraints	Description
id	INT	PRIMARY KEY, AUTO_INCREMENT	Unique identifier for each student. Auto-generated by the database.
name	VARCHAR(255)	NOT NULL	Full name of the student.
roll_number	VARCHAR(100)	NOT NULL, UNIQUE	Unique roll number assigned to each student.
marks	INT	NOT NULL	The total marks obtained by the student in the examination.
grade	VARCHAR(2)		The grade assigned to the student based on their marks.

Column Details:

id:

This is a unique identifier for each student and is automatically generated by the database (auto-incremented). It ensures each record is uniquely identified.

name:

The full name of the student. It's stored as a string (up to 255 characters), and it's required for every student (cannot be null).

roll_number:

The student's roll number, a unique identifier assigned to each student (cannot be duplicated). It is used to identify students more easily and quickly.

marks:

The integer field that stores the marks obtained by the student. This field is mandatory, and the value is used to calculate the student's grade.

grade:

This column stores the student's grade based on their marks. The grade is calculated using the following system:

- A: Marks ≥ 90
- B: $75 \leq \text{Marks} < 90$
- C: $60 \leq \text{Marks} < 75$
- D: $50 \leq \text{Marks} < 60$
- F: Marks < 50

3. Relationships:

Since this project only has one table (students), there are no complex relationships like foreign keys, joins, or other entities. This database is designed to be simple and efficient for storing student information and retrieving data related to their marks and grades.

4. Data Flow:

Adding a Student:

When a new student is added, their name, roll_number, marks, and grade are stored in the students table.

Retrieving Students:

The system fetches student records from the database, either showing all students or allowing searches based on name or roll number.

Updating Students:

The marks and grade of a student can be updated if necessary, and the new information is reflected in the students table.

Deleting Students:

When a student record is no longer needed, it can be removed from the students table

by deleting the corresponding entry.

5. SQL Queries:

Create Table Query:

```
CREATE TABLE students (  
    id INT AUTO_INCREMENT PRIMARY KEY,  
    name VARCHAR(255) NOT NULL,  
    roll_number VARCHAR(100) NOT NULL UNIQUE,  
    marks INT NOT NULL,  
    grade VARCHAR(2)  
);
```

Insert Data Query:

```
INSERT INTO students (name, roll_number, marks, grade)  
VALUES ('John Doe', 'A123', 85, 'B');
```

Select All Students Query:

```
SELECT * FROM students;
```

Update Student Marks Query:

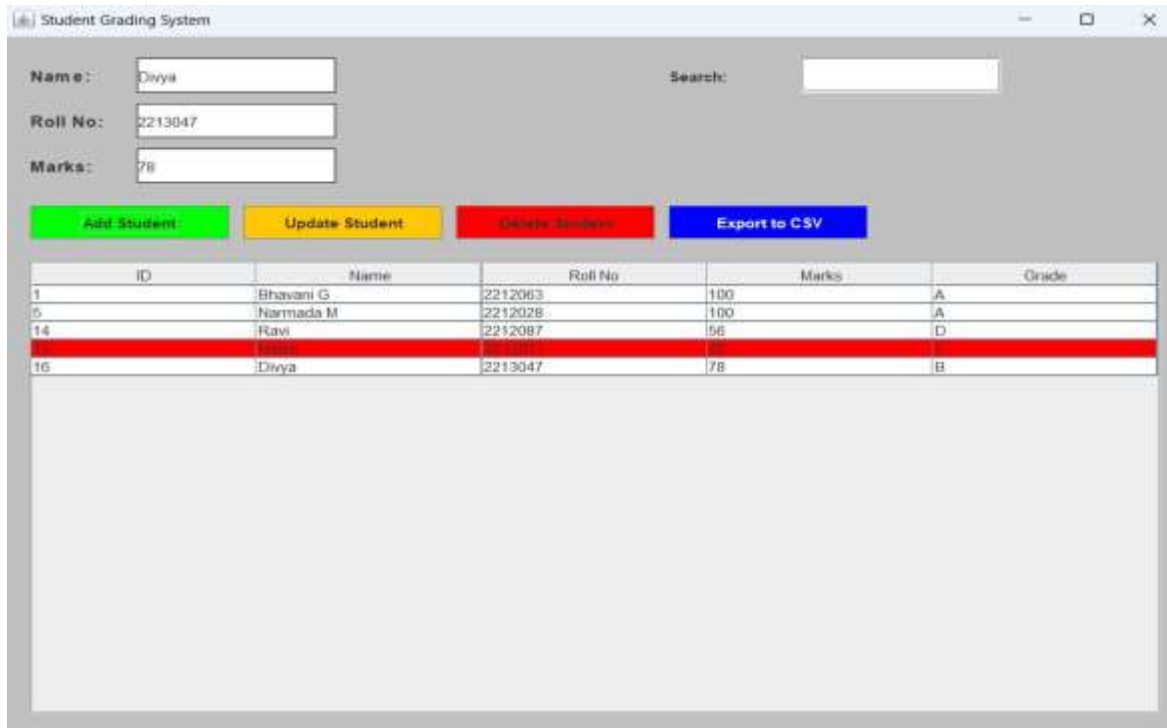
```
UPDATE students  
SET marks = 90, grade = 'A'  
WHERE id = 1;
```

Delete Student Query:

```
DELETE FROM students  
WHERE id = 1;
```

UI DESIGN:

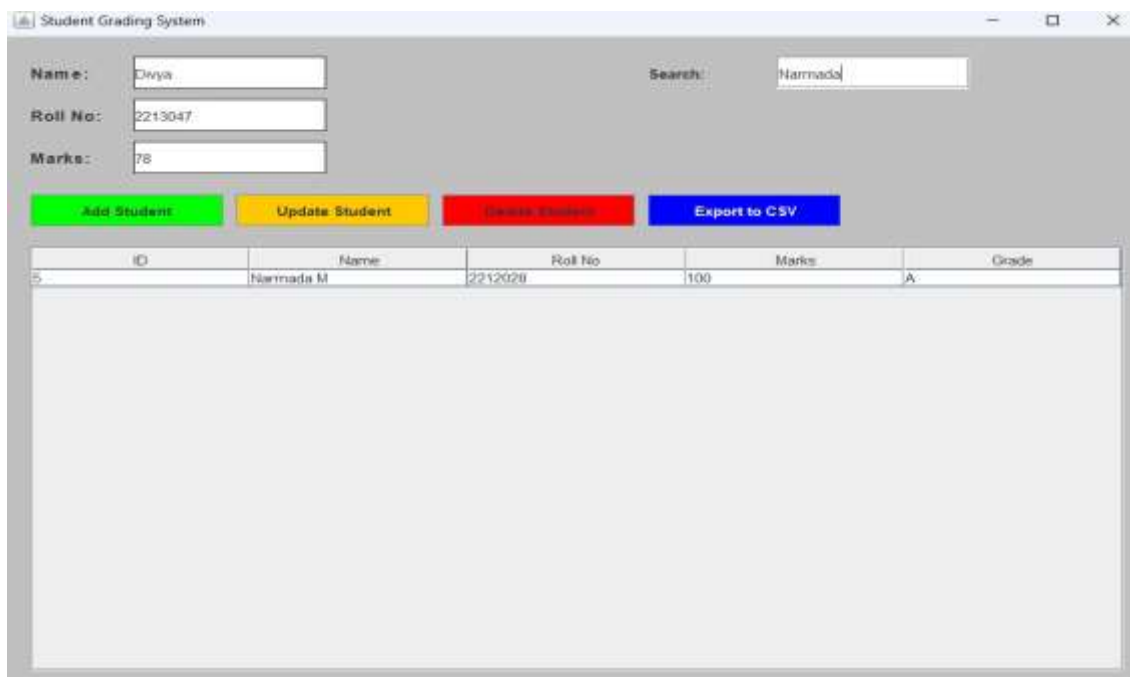
When the student's grade is added it highlights the mark below the average



The screenshot shows the 'Student Grading System' window. It features input fields for Name, Roll No, and Marks, along with a Search field. Below these are four buttons: 'Add Student' (green), 'Update Student' (yellow), 'Delete Student' (red), and 'Export to CSV' (blue). A table displays student data with columns for ID, Name, Roll No, Marks, and Grade. The row for student ID 14 (Ravi) is highlighted in red, indicating a mark below the average.

ID	Name	Roll No	Marks	Grade
1	Bhavani G	2212063	100	A
5	Narmada M	2212028	100	A
14	Ravi	2212087	56	D
15	Divya	2213047	78	B
16	Divya	2213047	78	B

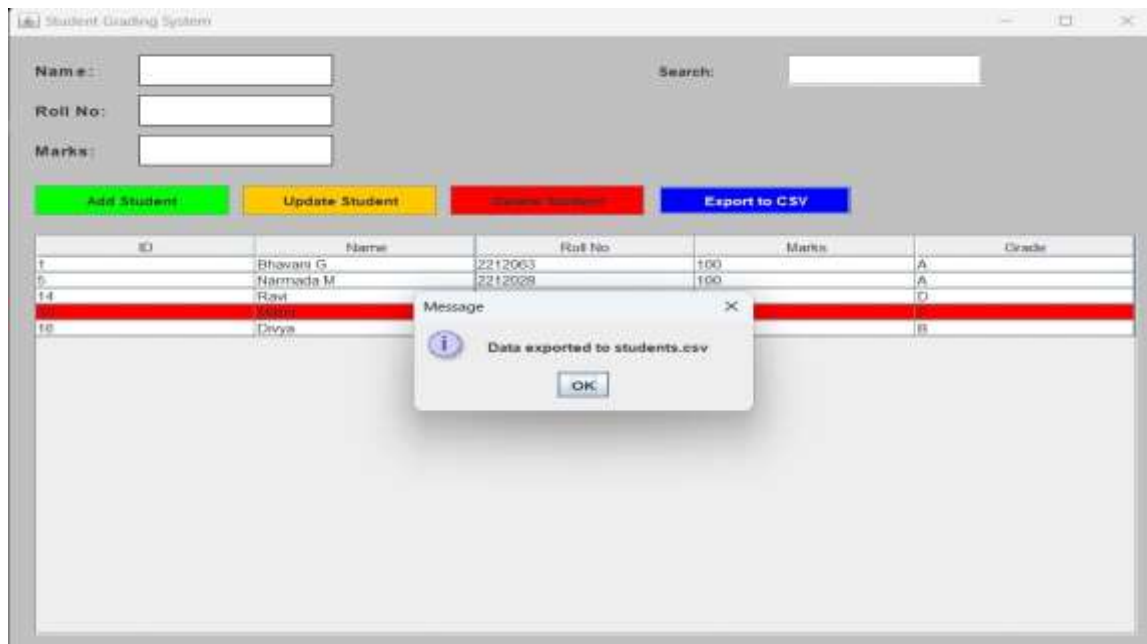
Searches the name of the student provided in the search box



The screenshot shows the 'Student Grading System' window with the search field populated with 'Narmada'. The table now displays only the student with the name 'Narmada M'.

ID	Name	Roll No	Marks	Grade
5	Narmada M	2212028	100	A

It displays the dialog box that the data is exported to student.csv



Data was added to the student.csv successfully

	A	B	C	D	E
1	1	Bhavani G	2212063	100	A
2	5	Narmada M	2212028	100	A
3	14	Ravi	2212088	56	D
4	15	Malini	2213011	45	F
5	16	Divya	2213047	78	B
6					

CODE:

StudentGradingSystem.java

```
import ui.StudentGradingSystemUI;
public class StudentGradingSystem {
    public static void main(String[] args) {
        StudentGradingSystemUI ui = new StudentGradingSystemUI();
        ui.setVisible(true);
    }
}
```



```
}
```

DatabaseConnection.java

```
package db;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.util.Properties;
import java.io.FileInputStream;
import java.io.IOException;

public class DatabaseConnection {
    private static Connection conn;

    // Method to check if connection is closed
    private static boolean isConnectionClosed() throws SQLException {
        return conn == null || conn.isClosed();
    }

    public static Connection getConnection() {
        try {
            // Check if connection is null or closed
            if (isConnectionClosed()) {
                Properties props = new Properties();
                FileInputStream fis = new FileInputStream("src/db.properties");
                props.load(fis);

                String url = props.getProperty("db.url");
                String user = props.getProperty("db.user");
                String password = props.getProperty("db.password");

                conn = DriverManager.getConnection(url, user, password);
            }
        } catch (SQLException | IOException e) {
            e.printStackTrace();
        }
        return conn;
    }

    public static void closeConnection() {
        if (conn != null) {
            try {
                conn.close();
            }
        }
    }
}
```

```
        conn = null; // Set to null to avoid using a closed connection
    } catch (SQLException e) {
        e.printStackTrace();
    }
}
}
```

Student.java:

package model;

```
public class Student {
    private int id;
    private String name;
    private String rollNumber;
    private int marks;
    private String grade;
    public Student(int id, String name, String rollNumber, int marks, String grade) {
        this.id = id;
        this.name = name;
        this.rollNumber = rollNumber;
        this.marks = marks;
        this.grade = grade;
    }
    public int getId() {
        return id;
    }
    public String getName() {
        return name;
    }

    public String getRollNumber() {
        return rollNumber;
    }

    public int getMarks() {
        return marks;
    }
    public String getGrade() {
        return grade;
    }

    public void setGrade(String grade) {
```

```
        this.grade = grade;
    }
}
```

StudentDAO.java

```
package dao;
```

```
import db.DatabaseConnection;
import model.Student;
```

```
import java.sql.*;
import java.util.ArrayList;
import java.util.List;
```

```
public class StudentDAO {
```

```
    public void addStudent(Student student) {
        String sql = "INSERT INTO students (name, roll_number, marks, grade) VALUES (?, ?, ?, ?)";
```

```
        try (Connection conn = DatabaseConnection.getConnection();
            PreparedStatement stmt = conn.prepareStatement(sql)) {
```

```
            stmt.setString(1, student.getName());
            stmt.setString(2, student.getRollNumber());
            stmt.setInt(3, student.getMarks());
            stmt.setString(4, calculateGrade(student.getMarks()));
```

```
            stmt.executeUpdate();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
```

```
    public List<Student> getAllStudents() {
        List<Student> students = new ArrayList<>();
        String sql = "SELECT * FROM students";
```

```
        try (Connection conn = DatabaseConnection.getConnection();
            Statement stmt = conn.createStatement();
            ResultSet rs = stmt.executeQuery(sql)) {
```

```

        while (rs.next()) {
            int id = rs.getInt("id");
            String name = rs.getString("name");
            String rollNumber = rs.getString("roll_number");
            int marks = rs.getInt("marks");
            String grade = rs.getString("grade");

            students.add(new Student(id, name, rollNumber, marks, grade));
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
    return students;
}

public List<Student> searchStudents(String searchText) {
    List<Student> students = new ArrayList<>();
    String sql = "SELECT * FROM students WHERE name LIKE ? OR roll_number LIKE ?";

    try (Connection conn = DatabaseConnection.getConnection();
        PreparedStatement stmt = conn.prepareStatement(sql)) {

        stmt.setString(1, "%" + searchText + "%");
        stmt.setString(2, "%" + searchText + "%");

        try (ResultSet rs = stmt.executeQuery()) {
            while (rs.next()) {
                int id = rs.getInt("id");
                String name = rs.getString("name");
                String rollNumber = rs.getString("roll_number");
                int marks = rs.getInt("marks");
                String grade = rs.getString("grade");

                students.add(new Student(id, name, rollNumber, marks, grade));
            }
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
    return students;
}

public void updateStudent(Student student) {

```

```
String sql = "UPDATE students SET name = ?, roll_number = ?, marks = ?, grade = ?  
WHERE id = ?";
```

```
try (Connection conn = DatabaseConnection.getConnection();  
     PreparedStatement stmt = conn.prepareStatement(sql)) {  
  
    stmt.setString(1, student.getName());  
    stmt.setString(2, student.getRollNumber());  
    stmt.setInt(3, student.getMarks());  
    stmt.setString(4, calculateGrade(student.getMarks()));  
    stmt.setInt(5, student.getId());  
  
    stmt.executeUpdate();  
} catch (SQLException e) {  
    e.printStackTrace();  
}  
}
```

```
public void deleteStudent(int id) {  
    String sql = "DELETE FROM students WHERE id = ?";  
  
    try (Connection conn = DatabaseConnection.getConnection();  
         PreparedStatement stmt = conn.prepareStatement(sql)) {  
  
        stmt.setInt(1, id);  
        stmt.executeUpdate();  
    } catch (SQLException e) {  
        e.printStackTrace();  
    }  
}
```

```
private String calculateGrade(int marks) {  
    if (marks >= 90)  
        return "A";  
    else if (marks >= 75)  
        return "B";  
    else if (marks >= 60)  
        return "C";  
    else if (marks >= 50)  
        return "D";  
    else  
        return "F";  
}  
}
```

StudentGradingSystemUI.java

```
package ui;

import dao.StudentDAO;
import model.Student;

import javax.swing.*.*;
import javax.swing.event.DocumentEvent;
import javax.swing.event.DocumentListener;
import javax.swing.table.DefaultTableCellRenderer;
import javax.swing.table.DefaultTableModel;
import java.awt.*.*;
import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.IOException;
import java.util.List;

public class StudentGradingSystemUI extends JFrame {

    private StudentDAO studentDAO;
    private DefaultTableModel tableModel;

    public StudentGradingSystemUI() {
        studentDAO = new StudentDAO();
        initializeUI();
    }

    private void initializeUI() {
        setTitle("Student Grading System");
        setSize(900, 700);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLayout(null);
        getContentPane().setBackground(Color.LIGHT_GRAY); // Set background color

        JLabel nameLabel = new JLabel("Name:");
        nameLabel.setBounds(20, 20, 100, 30);
        nameLabel.setFont(new Font("Arial", Font.BOLD, 14));
        add(nameLabel);

        JTextField nameField = new JTextField();
        nameField.setBounds(100, 20, 150, 30);
```

```
nameField.setBorder(BorderFactory.createLineBorder(Color.BLACK));  
add(nameField);
```

```
JLabel rollLabel = new JLabel("Roll No:");  
rollLabel.setBounds(20, 60, 100, 30);  
rollLabel.setFont(new Font("Arial", Font.BOLD, 14));  
add(rollLabel);
```

```
JTextField rollField = new JTextField();  
rollField.setBounds(100, 60, 150, 30);  
rollField.setBorder(BorderFactory.createLineBorder(Color.BLACK));  
add(rollField);
```

```
JLabel marksLabel = new JLabel("Marks:");  
marksLabel.setBounds(20, 100, 100, 30);  
marksLabel.setFont(new Font("Arial", Font.BOLD, 14));  
add(marksLabel);
```

```
JTextField marksField = new JTextField();  
marksField.setBounds(100, 100, 150, 30);  
marksField.setBorder(BorderFactory.createLineBorder(Color.BLACK));  
add(marksField);
```

```
JButton addButton = new JButton("Add Student");  
addButton.setBounds(20, 150, 150, 30);  
addButton.setBackground(Color.GREEN);  
add(addButton);
```

```
JButton updateButton = new JButton("Update Student");  
updateButton.setBounds(180, 150, 150, 30);  
updateButton.setBackground(Color.ORANGE);  
add(updateButton);
```

```
JButton deleteButton = new JButton("Delete Student");  
deleteButton.setBounds(340, 150, 150, 30);  
deleteButton.setBackground(Color.RED);  
add(deleteButton);
```

```
JButton exportButton = new JButton("Export to CSV");  
exportButton.setBounds(500, 150, 150, 30);  
exportButton.setBackground(Color.BLUE);  
exportButton.setForeground(Color.WHITE);  
add(exportButton);
```

```
JLabel searchLabel = new JLabel("Search:");
searchLabel.setBounds(500, 20, 100, 30);
add(searchLabel);
```

```
JTextField searchField = new JTextField();
searchField.setBounds(600, 20, 150, 30);
add(searchField);
```

```
String[] columnNames = { "ID", "Name", "Roll No", "Marks", "Grade" };
tableModel = new DefaultTableModel(columnNames, 0);
JTable studentTable = new JTable(tableModel);
```

```
// Highlight rows based on marks
studentTable.setDefaultRenderer(Object.class, new DefaultTableCellRenderer() {
    @Override
    public Component getTableCellRendererComponent(JTable table, Object value,
        boolean isSelected, boolean hasFocus,
        int row, int column) {
        Component c = super.getTableCellRendererComponent(table, value, isSelected,
hasFocus, row, column);
        int marks = (int) tableModel.getValueAt(row, 3);
        if (marks < 50) {
            c.setBackground(Color.RED); // Failing marks
        } else {
            c.setBackground(Color.WHITE);
        }
        return c;
    }
});
```

```
JScrollPane scrollPane = new JScrollPane(studentTable);
scrollPane.setBounds(20, 200, 850, 400);
add(scrollPane);
```

```
// Add functionality for adding students
addButton.addActionListener(e -> {
    String name = nameField.getText();
    String rollNumber = rollField.getText();
    int marks = Integer.parseInt(marksField.getText());

    Student student = new Student(0, name, rollNumber, marks, null);
    studentDAO.addStudent(student);
    fetchStudents();
});
```



```

// Update functionality
updateButton.addActionListener(e -> {
    int selectedRow = studentTable.getSelectedRow();
    if (selectedRow != -1) {
        int id = (int) tableModel.getValueAt(selectedRow, 0);
        String name = (String) tableModel.getValueAt(selectedRow, 1);
        String rollNumber = (String) tableModel.getValueAt(selectedRow, 2);
        int marks = (int) tableModel.getValueAt(selectedRow, 3);

        Student student = new Student(id, name, rollNumber, marks, null);
        studentDAO.updateStudent(student);
        fetchStudents();
    }
});

// Delete functionality
deleteButton.addActionListener(e -> {
    int selectedRow = studentTable.getSelectedRow();
    if (selectedRow != -1) {
        int id = (int) tableModel.getValueAt(selectedRow, 0);
        studentDAO.deleteStudent(id);
        fetchStudents();
    }
});

// Export to CSV functionality
exportButton.addActionListener(e -> {
    try (BufferedWriter writer = new BufferedWriter(new FileWriter("students.csv"))) {
        for (int i = 0; i < tableModel.getRowCount(); i++) {
            writer.write(tableModel.getValueAt(i, 0) + "," +
                tableModel.getValueAt(i, 1) + "," +
                tableModel.getValueAt(i, 2) + "," +
                tableModel.getValueAt(i, 3) + "," +
                tableModel.getValueAt(i, 4));
            writer.newLine();
        }
        JOptionPane.showMessageDialog(this, "Data exported to students.csv");
    } catch (IOException ex) {
        ex.printStackTrace();
    }
});

// Search functionality

```

```

searchField.getDocument().addDocumentListener(new DocumentListener() {
    public void insertUpdate(DocumentEvent e) {
        searchStudents(searchField.getText());
    }

    public void removeUpdate(DocumentEvent e) {
        searchStudents(searchField.getText());
    }

    public void changedUpdate(DocumentEvent e) {
        searchStudents(searchField.getText());
    }
});

fetchStudents();
}

// Fetch and display students
private void fetchStudents() {
    tableModel.setRowCount(0);
    List<Student> students = studentDAO.getAllStudents();
    for (Student student : students) {
        tableModel.addRow(new Object[] {
            student.getId(), student.getName(),
            student.getRollNumber(), student.getMarks(),
            student.getGrade()
        });
    }
}

// Search students based on name or roll number
private void searchStudents(String searchText) {
    tableModel.setRowCount(0);
    List<Student> students = studentDAO.searchStudents(searchText);
    for (Student student : students) {
        tableModel.addRow(new Object[] {
            student.getId(), student.getName(),
            student.getRollNumber(), student.getMarks(),
            student.getGrade()
        });
    }
}
}

```

db.properties

db.url=jdbc:mysql://localhost:3306/student_db

db.user=root

[db.password=BG30@MySQL](#)

CONCLUSION:

The **Student Grading System** database is designed for simplicity, focusing on managing student data efficiently through a single table, students. Each student record includes personal details like name, roll number, and academic performance (marks and grade). The system supports basic CRUD operations (Create, Read, Update, Delete) and features such as grade calculation based on marks. Its lightweight design ensures easy scalability and adaptability, making it ideal for small to medium-sized educational systems. The database is robust enough to handle common student management tasks while maintaining data integrity.

19CS51C - JAVA PROGRAMMING

TITLE: STUDENT GRADING SYSTEM

NAME: BHAVANI G - 2212063

Design (5)	Code (5)	Screenshots (5)	Time Management (5)	Total (20)

19CS51C - JAVA PROGRAMMING

TITLE: STUDENT GRADING SYSTEM

NAME: PRIYANKA S - 2212043

Design (5)	Code (5)	Screenshots (5)	Time Management (5)	Total (20)

