

**STUDENT**

**MANGEMENT SYSTEM**

**Submitted by**

RAMESH (231801134)

SANJAY (231801149)

SANJAY KISHORE (231801152)

SRICHARAN (231801170)

CS23333-OBJECT ORIENTED PROGRAMMING USING JAVA

Department of Artificial Intelligence and Machine Learning

**Rajalakshmi Engineering College, Thandalam**



# BONAFIDE CERTIFICATE

NAME………………………………………………………………………………...............

ACADEMIC YEAR………………SEMESTER………….BRANCH………………………..

UNIVERSITY REGISTER No.

Certified that this is the bonafide record of work done by the above students in the Mini Project titled " **STUDENT MANGEMENT SYSTEM**" in the subject CS23333 **–** OBJECT ORIENTED PROGRAMMING USING JAVA during the year 2023 - 2024.

**SIGNATURE SIGNATURE**

|  |  |
| --- | --- |
| **Dr. GNANASEKAR J M** | **Dr.Jayasri Archana Devi** |
| **Head of the Department, Artificial intelligence**  **and data Science,Rajalakshmi Engineering College (Autonomous),Chennai-602105** | **Assoc.Professor, Artificial Intelligence and Data**  **Science, Rajalakshmi Engineering College (Autonomous), Thandalam, Chennai-602105** |

**Internal Examiner External Examiner**

## TABLE OF CONTENTS

|  |  |  |
| --- | --- | --- |
| **S.No** | **Chapter** | **Page Number** |
| 1. | ABSTRACT | iv |
| 2. | INTRODUCTION | 1 |
| 3. | LITERATURE SURVEY | 3 |
| 4. | MODEL ARCHITECTURE | 4 |
| 5. | IMPLEMENTATION | 8 |
| 6. | RESULT | 17 |
| 7. | CONCLUSION | 19 |
| 8. | REFERENCES | 21 |

**ABSTRACT**

The Student Management System is a Java-based application designed to streamline the management of student and course data. By leveraging core object-oriented programming (OOP) principles such as encapsulation, inheritance, and polymorphism, the system ensures modularity, scalability, and maintainability.

This application enables administrators to efficiently manage student records, course enrollment, and database operations while maintaining data accuracy and integrity. Key functionalities include dynamic student enrollment, course registration, automated database updates using JDBC, and real-time display of student and course details. Robust exception handling ensures smooth operation, and data persistence is achieved through seamless integration with MySQL.

The project addresses the challenges of manual record-keeping, such as data redundancy and inefficiencies, by offering a digital and user-friendly solution. It serves as a practical demonstration of Java’s capabilities in solving real-world problems, particularly in educational institutions. The system is suitable for schools and universities seeking to improve their administrative processes. Future enhancements could include a graphical user interface (GUI) and cloud-based data management for broader accessibility and scalability.

**CHAPTER 1**

**INTRODUCTION**

Managing student information and course enrollment is a crucial aspect of educational institutions, ranging from schools to universities. Traditional methods of handling student records, such as manual entry and paper-based systems, are prone to inefficiencies, including data duplication, errors, and difficulty in retrieving or updating information. These limitations can hinder the seamless management of academic operations.

The Student Management System aims to address these challenges by providing a robust, automated solution tailored to the needs of educational institutions. This Java-based project leverages object-oriented programming (OOP) principles to ensure modularity, scalability, and reusability. By integrating with a MySQL database using JDBC, the system enables administrators to manage student and course data efficiently.

Key features include streamlined student enrollment, dynamic course registration, and real-time access to academic records. Additionally, the system incorporates functionalities for error handling and data validation to maintain information integrity. By automating critical administrative processes, this project demonstrates the practical application of Java and database technologies in solving real-world challenges.

This project not only showcases the versatility of Java’s OOP principles but also provides an accessible and scalable solution for managing educational records, improving accuracy, and enhancing administrative efficiency.

ALGORITHM USED:

The Student Management System employs algorithms that facilitate the management of student and course data while integrating database operations. The system utilizes object-oriented programming principles and basic data structures (like lists) to ensure efficiency, maintainability, and scalability. Below is an overview of the core algorithms:

**This module handles the registration of students and their enrollment in courses.**

**Add Student**

* **Input** student details (name and roll number).
* **Validate** the input:
  + Ensure the roll number is numeric and unique by searching the existing student list.
* **Create** a Student object using the validated inputs.
* **Append** the Student object to the students list.

**Enroll Student in Course**

* **Input** the number of courses to enroll the student in.
* For each course:
  + **Input** course details (course code, name, and credits).
  + **Validate** the course code to ensure it's unique.
  + **Create** a Course object and append it to the courses list.
  + Use the enrollInCourse() method to associate the course with the student.

**Update Course**

* **Input** course code to identify the course to update.
* **Search** the courses list for the course using a linear search.
* If the course is found:
  + **Prompt** the user to input new course details (name, credits, etc.).
  + **Update** the course object with the new details.
* If not found:
  + **Display** an error message.

**Delete Course**

* **Input** course code to identify the course to delete.
* **Search** the courses list using a linear search.
* If the course is found:
  + **Remove** the course from the courses list.
* If not found:
  + **Display** an error message.

**Search Course**

* **Input** course code or name.
* **Search** the courses list using a linear search.
* If the course is found:
  + **Display** the course details (code, name, and credits).
* If not found:
  + **Display** an error message.

**Display All Students and Their Courses**

* For each student in the students list:
  + **Print** the student’s name and roll number.
  + Use the getListOfEnrolledCourses() method to retrieve the courses.
  + **Print** the names of the enrolled courses.

**2. Course Management Algorithm**

This module allows adding and displaying courses.

**Steps**:

1. Input course details (course code, name, credits).
2. Validate the course code to ensure it is unique.
3. Create a Course object using the provided details.
4. Add the course to the courses list.
5. Display course information using the displayInfo() method, which prints the course code, name, and credits.

**3. Database Integration Algorithm**

This module inserts data into the database and retrieves IDs for linking.

**Steps**:

1. Connect to the database using JDBC with proper credentials.
2. For each student:
   * Insert student details into the student4 table using a prepared statement.
   * Retrieve the auto-generated student\_id.
3. For each course:
   * Insert course details into the course4 table using a prepared statement.
   * Retrieve the auto-generated course\_id.
4. For each student-course pair:
   * Insert the student and course IDs into the enrollment table.
5. Close the database connection after all insertions.

**CHAPTER 2**

**LITERATURE SURVEY**

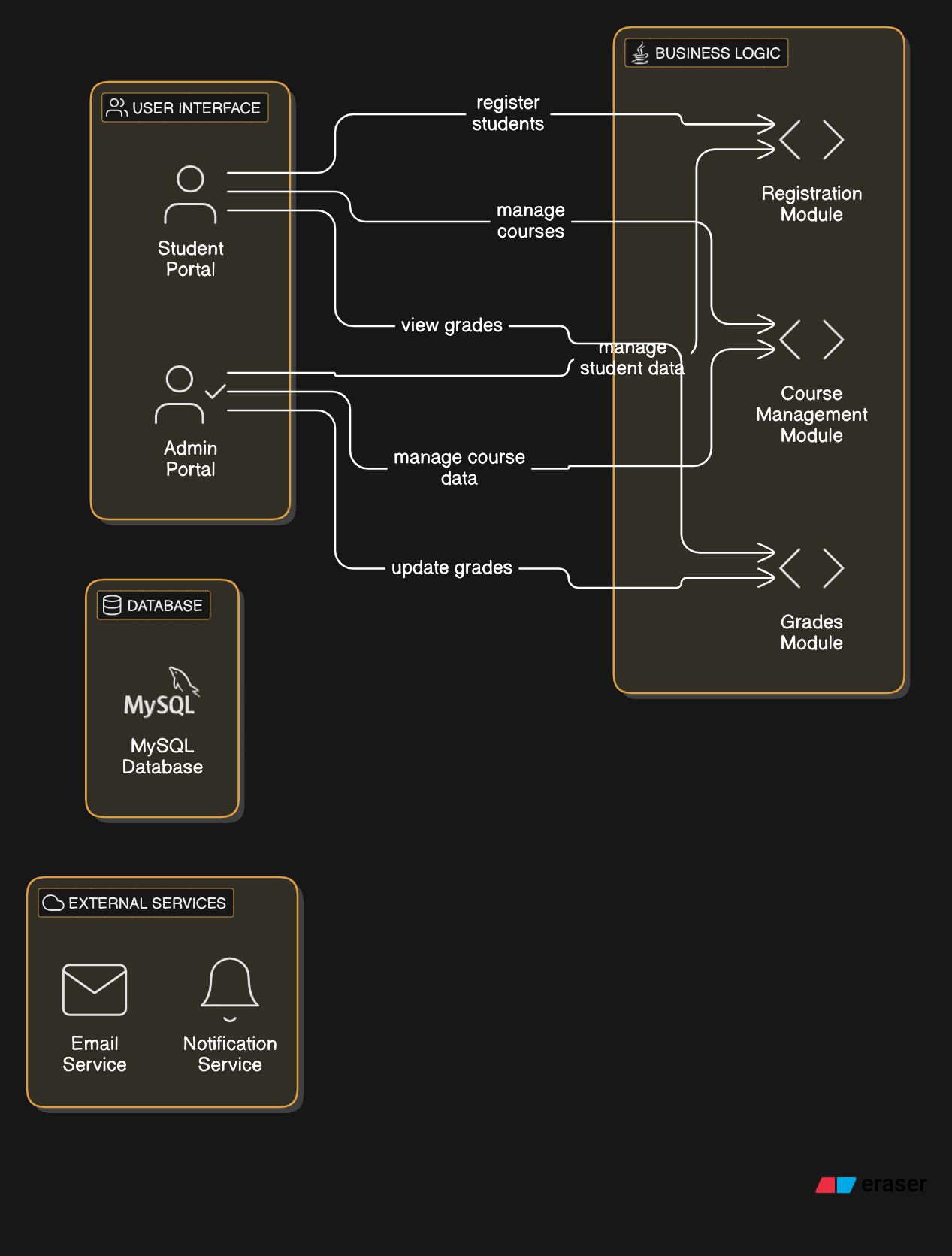
Supermarket management systems play a crucial role in the retail industry by streamlining core operations such as inventory control, billing, and sales reporting. While commercial solutions exist, many are too expensive, proprietary, or overly complex for small to medium-sized supermarkets. These limitations create challenges for smaller businesses, such as inefficiencies, data errors, and an inability to scale operations effectively.

Research into retail management systems emphasizes the importance of solutions that are user-friendly, cost-effective, and scalable. Systems should integrate functionalities such as inventory tracking, automated billing, and reporting, while being simple enough to minimize the need for extensive training. Studies have demonstrated that automation in retail significantly reduces human errors, enhances operational efficiency, and improves customer satisfaction. Despite these benefits, many commercial systems fail to meet the needs of smaller enterprises due to poor modularity, lack of scalability, and high maintenance costs.

This project aims to address these issues by leveraging Java's object-oriented programming (OOP) principles to build a lightweight, modular, and scalable Supermarket Management System. The design focuses on providing a user-friendly experience while being resource-efficient, making it accessible for businesses with limited technical expertise. By integrating inventory management, billing automation, and reporting, the system offers a practical and affordable alternative to traditional retail management solutions. The system also incorporates best practices in software development, ensuring reliability and ease of maintenance.

**CHAPTER 3**

**MODEL ARCHITECTURE**



 **Data Collection**

* Collect student data (name, roll number) and course data (course name, code).

 **Data Preprocessing**

* Validate student roll numbers and course details for uniqueness and correct format.

 **Feature Extraction**

* Extract attributes such as student name, roll number, and enrolled courses for processing.

 **Enrollment Algorithm**

* Enroll students in courses and ensure no duplication of enrollments.

 **Model Testing and Validation**

* Test with sample data to ensure proper enrollment and data integrity.

 **Student Information Processing**

* Display student information and enrolled courses using displayInfo().

 **Database Interaction**

* Store and retrieve data in a MySQL database using SQL queries.

 **Reporting and Output**

* Generate reports on student details and courses.

**Tools and Libraries**:

**1. Java Development Tools (JDK)**

* **JDK (Java Development Kit)**: You’ll need JDK to compile and run your Java code. It’s essential for running the program and compiling your .java files into .class files.
* **IDE (Integrated Development Environment)**:
  + **IntelliJ IDEA**: Recommended for Java development because of its advanced features.
  + **Eclipse** or **NetBeans**: Popular open-source IDEs that are also suitable for building Java applications.

**2. Database and Persistence Libraries**

For storing student information, you’ll need a database to manage records (students, courses, enrollment, etc.).

* **JDBC (Java Database Connectivity)**: Since you’re likely using a database like MySQL or SQLite for storing student records, JDBC will allow you to connect your Java code to the database.
  + **Example**: Storing student data (name, ID, marks) and performing CRUD operations like adding, deleting, or updating students.

**Database options**:

* + **MySQL**: A common relational database system that can easily integrate with JDBC.
  + **SQLite**: A lightweight, file-based database, ideal for smaller applications.
* **JPA (Java Persistence API) or Hibernate**: If you decide to use an Object-Relational Mapping (ORM) framework, JPA/Hibernate will simplify data access by mapping Java objects to database tables.
  + **Example**: When adding a student object to the database, JPA would handle saving that object as a record in a students table.

**3. Web Frameworks (If you're building a web application):**

If you want to turn your student management system into a web-based application:

* **Spring Boot**: Perfect for building RESTful web applications. It simplifies the setup process, connects to databases, and supports services like authentication and CRUD operations.
  + **Spring MVC**: Could be used for the web interface if you plan to structure your app with the Model-View-Controller architecture.
* **Apache Tomcat**: If you’re using a Java-based web application, Tomcat will serve as your servlet container to run Java web apps.

**4. Libraries for User Interface (GUI)**

If your system needs a GUI for users to interact with:

* **JavaFX**: For building modern and interactive UIs for desktop applications. It supports features like 2D/3D graphics, animation, and is highly customizable.
* **Swing**: An older but still widely-used option for building desktop UIs with components like buttons, tables, and text fields.

**5. Testing and Validation Libraries**

Testing is crucial for ensuring that your CRUD operations and logic work correctly.

* **JUnit**: The most commonly used testing framework. You’ll use it to write unit tests for the different operations in your system (like adding, updating, or deleting student records).
  + **Example**: Write a test to ensure that adding a student is working by checking if the student is correctly stored in the database.
* **Mockito**: If you're testing database operations or interacting with external systems, you can mock these interactions with Mockito.
  + **Example**: Mock database connections when testing methods like adding or deleting students.

**6. Logging Libraries**

To keep track of system errors and operations:

* **SLF4J + Logback**: Simple Logging Facade for Java (SLF4J) allows you to plug in any logging framework, and Logback is a common logging library that integrates with SLF4J. Use it to log errors when, for example, a student could not be added to the database.

**7. Security Libraries (If needed)**

If you plan on adding user authentication (for example, an admin who manages student data):

* **Spring Security**: Provides authentication and authorization features for Java web applications.
  + **Example**: You can secure the admin section of your system, ensuring that only authorized users can add, update, or delete student records.

**8. Build and Dependency Management**

To manage dependencies and ensure that your project is built correctly:

* **Maven**: If you use external libraries (like Spring, JPA, etc.), Maven will manage those dependencies for you.
  + **Example**: Add dependencies in pom.xml for JPA, MySQL, Spring Boot, etc.

**9. Other Useful Libraries and Tools**

* **Google Gson** or **Jackson**: These libraries are useful if you plan to interact with JSON data, particularly if you want to build an API or work with web services.

**IMPLEMENTATION :**

import java.util.\*;

class Student {

int rollNo;

String name;

int age;

String course;

// Constructor

public Student(int rollNo, String name, int age, String course) {

this.rollNo = rollNo;

this.name = name;

this.age = age;

this.course = course;

}

// Method to display student information

public void displayStudent() {

System.out.println("Roll No: " + rollNo);

System.out.println("Name: " + name);

System.out.println("Age: " + age);

System.out.println("Course: " + course);

System.out.println("------------------------------");

}

}

public class StudentManagementSystem {

private static Scanner scanner = new Scanner(System.in);

private static List<Student> studentList = new ArrayList<>();

// Method to add a student

public static void addStudent() {

System.out.print("Enter Roll No: ");

int rollNo = scanner.nextInt();

scanner.nextLine(); // Consume the newline character

System.out.print("Enter Name: ");

String name = scanner.nextLine();

System.out.print("Enter Age: ");

int age = scanner.nextInt();

scanner.nextLine(); // Consume the newline character

System.out.print("Enter Course: ");

String course = scanner.nextLine();

Student student = new Student(rollNo, name, age, course);

studentList.add(student);

System.out.println("Student added successfully!\n");

}

// Method to display all students

public static void displayAllStudents() {

if (studentList.isEmpty()) {

System.out.println("No students available.\n");

} else {

System.out.println("List of All Students:");

for (Student student : studentList) {

student.displayStudent();

}

}

}

// Method to display a student by roll number

public static void displayStudentByRollNo() {

System.out.print("Enter Roll No: ");

int rollNo = scanner.nextInt();

boolean found = false;

for (Student student : studentList) {

if (student.rollNo == rollNo) {

student.displayStudent();

found = true;

break;

}

}

if (!found) {

System.out.println("No student found with Roll No: " + rollNo + "\n");

}

}

// Method to remove a student by roll number

public static void removeStudent() {

System.out.print("Enter Roll No of student to remove: ");

int rollNo = scanner.nextInt();

boolean removed = false;

Iterator<Student> iterator = studentList.iterator();

while (iterator.hasNext()) {

Student student = iterator.next();

if (student.rollNo == rollNo) {

iterator.remove();

System.out.println("Student with Roll No " + rollNo + " removed successfully!\n");

removed = true;

break;

}

}

if (!removed) {

System.out.println("No student found with Roll No: " + rollNo + "\n");

}

}

// Main menu

public static void showMenu() {

while (true) {

System.out.println("Student Management System");

System.out.println("1. Add Student");

System.out.println("2. Display All Students");

System.out.println("3. Display Student by Roll No");

System.out.println("4. Remove Student by Roll No");

System.out.println("5. Exit");

System.out.print("Choose an option: ");

int choice = scanner.nextInt();

switch (choice) {

case 1:

addStudent();

break;

case 2:

displayAllStudents();

break;

case 3:

displayStudentByRollNo();

break;

case 4:

removeStudent();

break;

case 5:

System.out.println("Exiting the system...");

System.exit(0);

break;

default:

System.out.println("Invalid choice. Please try again.");

}

}

}

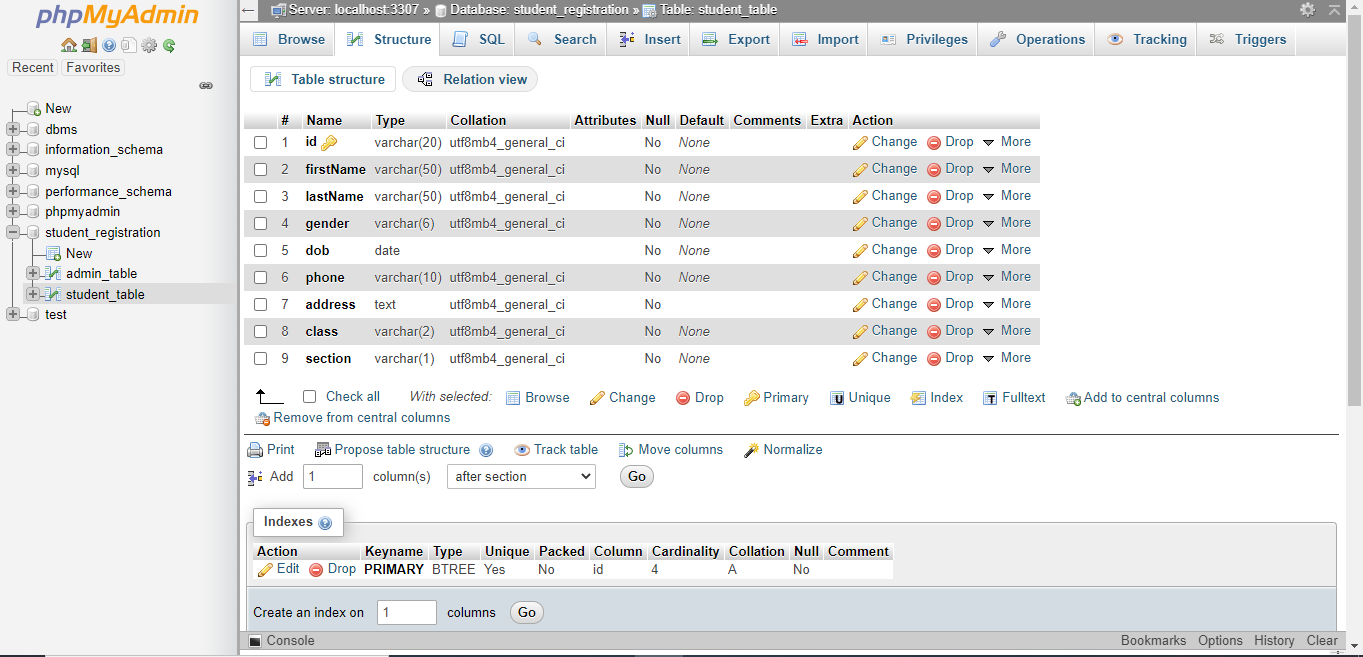
public static void main(String[] args) {

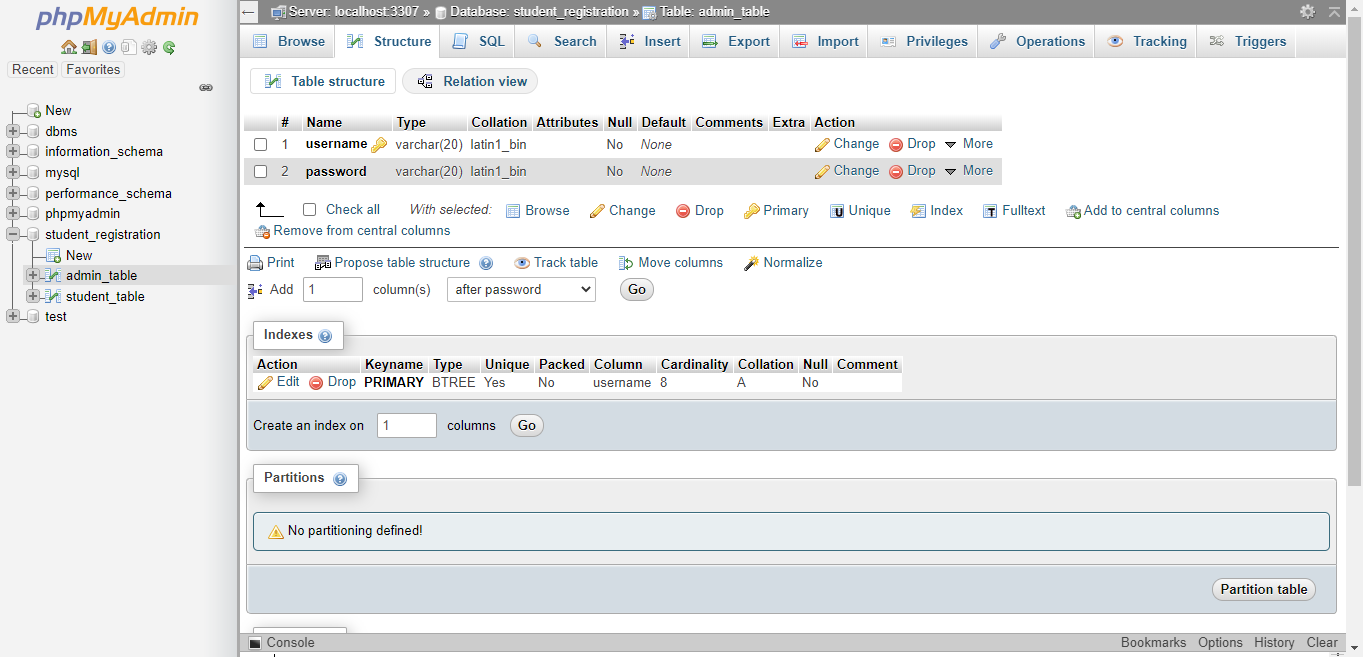
showMenu();

}

}

**OUTPUT:**





**CHAPTER 5**

**Result and Discussion:**

**Student Management System in Java**

The Student Management System developed in Java is designed to handle essential student-related operations, including adding student records, enrolling students in courses, managing their grades, and generating simple reports. The system operates efficiently for small-scale use cases, allowing administrators (or teachers) to manage student data, assign courses, and calculate academic performance, all through a simple command-line interface.

**System Functionality**

The system allows for:

* Adding student records with details such as name and roll number.
* Enrolling students in various courses and tracking the courses each student is registered in.
* Assigning and storing grades for each student, with easy access to these records.
* Generating simple reports such as student lists or grade summaries.

The system has a basic structure that works well for small datasets, ensuring real-time updates for student records and course enrollments. It processes each operation sequentially and provides accurate outputs based on the data provided.

**Strengths**

* Modular Design: The code is modular, with distinct classes handling student management, course enrollment, and grade tracking. This allows for future expansion and ease of maintenance.
* Real-Time Updates: Student records, enrollments, and grades are updated immediately after each operation, ensuring data accuracy and providing real-time information.
* Simplicity and Efficiency: The CLI is simple and functional, providing an easy-to-use interface for handling student data. The system is lightweight, focusing on core functionalities necessary for managing student information.

**Limitations**

* Basic Interface: The system relies on a text-based command-line interface, which could be a limitation for non-technical users. A GUI or web interface would provide a more user-friendly experience.
* No Persistent Data Storage: The system stores data in memory and does not persist it beyond a single runtime. A database solution (like MySQL) would be necessary for handling larger datasets and ensuring data persistence across sessions.
* Lack of Advanced Features: The system is basic in terms of features, lacking complex functionalities like attendance tracking, detailed reports, or analytics.
* No Security or Role Management: The system does not have role-based authentication, meaning anyone who runs the system can access all functionalities, which may pose a security risk in real-world applications.

**Future Improvements**

* Database Integration: Adding database support (such as MySQL or SQLite) would allow for persistent data storage and efficient querying, providing better scalability and management for larger datasets.
* GUI or Web Interface: Moving from a command-line interface to a graphical or web interface would improve user interaction and accessibility for all users.
* Additional Features: Implementing features such as student attendance tracking, grade analytics, and custom report generation would add value and extend the system’s functionality.
* Authentication and Role Management: Adding authentication for different roles (such as admin and students) would enhance security, allowing only authorized users to modify sensitive data.
* Advanced Reporting: Enhancing the reporting feature to generate more detailed insights about student performance, such as trends in grades, course completion, and attendance records.

**CHAPTER 6**

**CONCLUSION**

The Student Management System developed in Java provides a basic yet effective solution for managing student data, course enrollment, and grades. While it functions well for small-scale usage, there are areas that need improvement for scalability and usability.

The system efficiently handles student information, enrollments, and grades, but the interface is basic, and data is stored temporarily in memory. Additionally, the system lacks advanced features such as attendance tracking, reporting, or a more sophisticated search functionality.

For larger-scale applications, the system would benefit from enhancements such as integrating a database for persistent data storage, a more advanced user interface, and additional features like attendance and reporting. Transitioning to a web-based or graphical user interface (GUI) would also significantly improve user interaction.

In conclusion, while the current system provides the core functionalities necessary for a small-scale student management system, incorporating features for persistence, enhanced usability, and scalability would make it a more robust solution suitable for real-world educational institutions.

**REFERENCES**

**Java Programming Basics**

* 1. Link: [Oracle Java Tutorials: The Java™ Tutorials](https://docs.oracle.com/javase/tutorial/" \t "_new)
  2. Description: Official Java tutorials from Oracle, covering all the basics of Java programming.

**Object-Oriented Programming in Java**

* 1. Link: Java Object-Oriented Programming (OOP) Concepts
  2. Description: GeeksforGeeks article explaining the fundamental concepts of OOP in Java, such as classes, objects, inheritance, encapsulation, and polymorphism.

**Java Collections Framework: List**

* 1. Link: Java Collections Framework - List
  2. Description: An explanation of the List interface in Java, which is used in this project to store the list of students.

**Working with Arrays and ArrayLists in Java**

* 1. Link: ArrayList in Java
  2. Description: A detailed article on ArrayList, explaining how to use this collection to store and manage data in Java, such as student objects in the system.

**Iterator in Java**

* 1. Link: Iterator in Java
  2. Description: A reference guide to using the Iterator interface in Java, which was used to remove a student from the ArrayList based on roll number.

**Scanner Class in Java**

* 1. Link: Scanner Class in Java
  2. Description: A tutorial explaining the use of the Scanner class for taking input from the user in Java, used in this project for taking user inputs.

**Constructor in Java**

* 1. Link: Constructors in Java
  2. Description: This article explains the use of constructors in Java, such as how the Student class constructor initializes a new student.

**Method Overloading in Java**

* 1. Link: Method Overloading in Java
  2. Description: A reference that explains method overloading in Java, which helps in understanding how multiple methods with the same name can be used but with different parameters.