Name: Nethmi Bandara

Index: 16195

Mini Project – Student Database Application

Introduction

This project is a console-based Java application that manages student information using a MySQL database. The app demonstrates the use of JDBC for database connectivity and implements basic CRUD operations (Create, Read, Update, Delete).

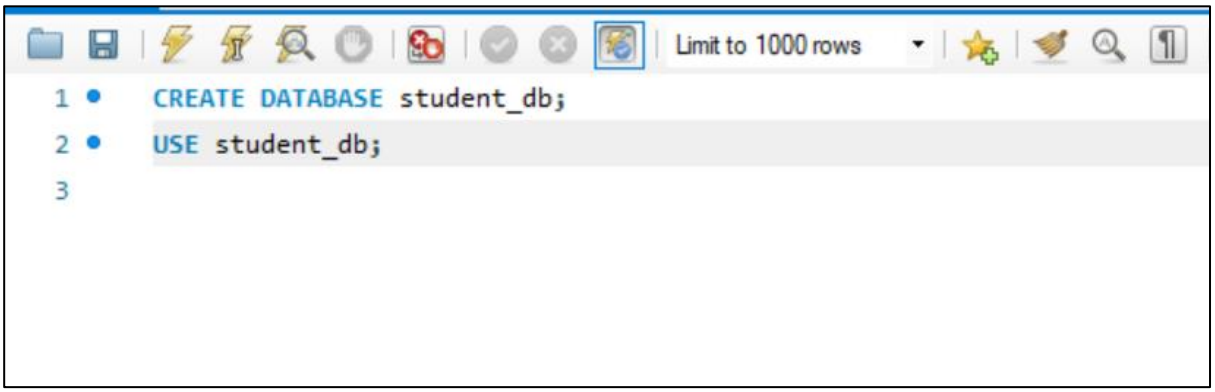
Objectives

1. Store student information (ID, Name, Year, Email, Stream) in a database.
2. Allow the user to add, view, update, or delete students.
3. Handle errors gracefully (invalid input, non-existing IDs).

Tools and Technologies

1. Java 17 – Programming language
2. IntelliJ IDEA – IDE for Java development
3. MySQL 8.0 – Database
4. MySQL Workbench – Database management tool
5. JDBC (Connector/J) – Java Database Connectivity

Database Design



```sql
CREATE TABLE students (
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    year INT,
    email VARCHAR(100),
    stream VARCHAR(50)
) AUTO_INCREMENT=16000;
```

Code Snippets and Explanation

1. DBConnection.java

   Connects Java to the MySQL database using JDBC.

   Returns a Connection object used by DAO classes

```java
public class DBConnection {
    1 usage
    private static final String URL = "jdbc:mysql://localhost:3306/student_db";
    1 usage
    private static final String USER = "root";
    1 usage
    private static final String PASSWORD = "root123";
    5 usages
    public static Connection getConnection() throws SQLException {
        return DriverManager.getConnection(URL, USER, PASSWORD);
    }
    public static void main(String[] args) {
        try (Connection conn = getConnection()) {
            System.out.println("Connected to database!");
        } catch (SQLException e) {
            System.out.println("Connection failed: " + e.getMessage());
        }
    }
```

2. Student.java

   Represents a student object.

   Contains fields, constructors, getters/setters, and a toString() for displaying students in the console.

```
2 usages
public Student(int id, String name, int year, String email, String stream) {
    this.id = id;
    this.name = name;
    this.year = year;
    this.email = email;
    this.stream = stream;
}
```

3. StudentDAO.java

   Interact with the database and perform all operations related to the students table.

```
// Add a student
1 usage
public void addStudent(Student student) {
    String sql = "INSERT INTO students (name, year, email, stream) VALUES (?, ?, ?, ?)";
    try (Connection conn = DBConnection.getConnection();
         PreparedStatement stmt = conn.prepareStatement(sql)) {
        stmt.setString( parameterIndex: 1, student.getName());
        stmt.setInt( parameterIndex: 2, student.getYear());
        stmt.setString( parameterIndex: 3, student.getEmail());
        stmt.setString( parameterIndex: 4, student.getStream());
        stmt.executeUpdate();
        System.out.println("Student added successfully!");
    } catch (SQLException e) {
        System.out.println("Error adding student: " + e.getMessage());
    }
}
```

```java
// View all students
1 usage
public List<Student> getAllStudents() {
    List<Student> students = new ArrayList<>();
    String sql = "SELECT * FROM students";
    try (Connection conn = DBConnection.getConnection();
         Statement stmt = conn.createStatement();
         ResultSet rs = stmt.executeQuery(sql)) {
        while (rs.next()) {
            students.add(new Student(
                    rs.getInt( columnLabel: "id"),
                    rs.getString( columnLabel: "name"),
                    rs.getInt( columnLabel: "year"),
                    rs.getString( columnLabel: "email"),
                    rs.getString( columnLabel: "stream")
            ));
        }
    } catch (SQLException e) {
        System.out.println("Error fetching students: " + e.getMessage());
    }
    return students;
}
```

```java
// Update a student
1 usage
public void updateStudent(Student student) {
    String checkSql = "SELECT COUNT(*) FROM students WHERE id=?";
    String sql = "UPDATE students SET name=?, year=?, email=?, stream=? WHERE id=?";
    try (Connection conn = DBConnection.getConnection();
         PreparedStatement checkStmt = conn.prepareStatement(checkSql)) {
        checkStmt.setInt( parameterIndex: 1, student.getId());
        ResultSet rs = checkStmt.executeQuery();
        rs.next();
        if (rs.getInt( columnIndex: 1) == 0) {
            System.out.println("⚠ No student found with ID " + student.getId());
            return;
        }
        try (PreparedStatement stmt = conn.prepareStatement(sql)) {
            stmt.setString( parameterIndex: 1, student.getName());
            stmt.setInt( parameterIndex: 2, student.getYear());
            stmt.setString( parameterIndex: 3, student.getEmail());
            stmt.setString( parameterIndex: 4, student.getStream());
            stmt.setInt( parameterIndex: 5, student.getId());
            stmt.executeUpdate();
            System.out.println("Student updated successfully!");
        }
    } catch (SQLException e) {
        System.out.println("Error updating student: " + e.getMessage());
    }
```

```
    // Delete a student
    1 usage
    public void deleteStudent(int id) {
        String checkSql = "SELECT COUNT(*) FROM students WHERE id=?";
        String sql = "DELETE FROM students WHERE id=?";
        try (Connection conn = DBConnection.getConnection();
             PreparedStatement checkStmt = conn.prepareStatement(checkSql)) {

            checkStmt.setInt( parameterIndex: 1, id);
            ResultSet rs = checkStmt.executeQuery();
            rs.next();
            if (rs.getInt( columnIndex: 1) == 0) {
                System.out.println("⚠ No student found with ID " + id);
                return;
            }
            try (PreparedStatement stmt = conn.prepareStatement(sql)) {
                stmt.setInt( parameterIndex: 1, id);
                stmt.executeUpdate();
                System.out.println("Student deleted successfully!");
            }
        } catch (SQLException e) {
            System.out.println("Error deleting student: " + e.getMessage());
        }
    }
}
```

4. Main.java (Menu)

   Console-based menu allows the user to choose operations.

   Calls appropriate methods from StudentDAO for CRUD operations.

   Handles invalid input and non-existing IDs with proper messages.

```
System.out.println("\n--- Student Database Menu ---");
System.out.println("1. Add Student");
System.out.println("2. View Students");
System.out.println("3. Update Student");
System.out.println("4. Delete Student");
System.out.println("5. Exit");
System.out.print("Enter your choice: ");
```

Error Handling

- Invalid input: Uses try-catch blocks to prevent the program from crashing when letters are entered instead of numbers.
- Non-existing ID: Update and delete methods first check if the student ID exists before performing operations.
- Empty database: Viewing students shows a message if no records exist.

Conclusion

- The project demonstrates Java + JDBC + MySQL integration.
- Implements a fully functional CRUD system for student management.
- Handles errors and invalid input gracefully.

Sample Outputs

View Students

```
--- Student Database Menu ---
1. Add Student
2. View Students
3. Update Student
4. Delete Student
5. Exit
Enter your choice: 2

ID | Name | Year | Email | Stream
16000 | Nethmi Bandara | 3 | nethmib@stu.cmb.ac.lk | Physical Science
16001 | Lahiru Amarasekara | 3 | lahirua@stu.cmb.ac.lk | Computer Science
16002 | Kavindu Dissanayake | 3 | kavindud@stu.cmb.ac.lk | Physical Science
16003 | Himansa Peiris | 3 | himansap@stu.cmb.ac.lk | Physical Science
16004 | Binura Gamage | 2 | binurag@stu.cmb.ac.lk | Physical Science
16005 | Yasith Perera | 2 | yasithp@stu.cmb.ac.lk | Bio Science
16006 | Kesith balasooriya | 3 | kesithb@stu.cmb.ac.lk | Bio Science
16007 | Shehara Madawela | 2 | sheharam@stu.cmb.ac.lk | Bio Science
16008 | Sayuri Gunasekara | 3 | sayurig@stu.cmb.ac.lk | Computer Science
16009 | Yevin Udawatte | 1 | yevinu@stu.cmb.ac.lk | Physical Science
16010 | Nihara Gamage | 6 | niharag@stu.cmb.ac.lk | Medicine
```

Update Student

```
--- Student Database Menu ---
1. Add Student
2. View Students
3. Update Student
4. Delete Student
5. Exit
Enter your choice: 3
Enter ID to update: 16010
New Name: Nihara Gamage
New Year: 4
New Email: niharag@stu.cmb.ac.lk
New Stream: Medicine
Student updated successfully!
```

Update Student

```
--- Student Database Menu ---
1. Add Student
2. View Students
3. Update Student
4. Delete Student
5. Exit
Enter your choice: 1
Name: Uvindu Amarasinghe
Year: 1
Email: uvindua@stu.cmb.ac.lk
Stream: Physical Science
Student added successfully!
```

Delete Student

```
--- Student Database Menu ---
1. Add Student
2. View Students
3. Update Student
4. Delete Student
5. Exit
Enter your choice: 4
Enter ID to delete: 16007
Student deleted successfully!
```

```
--- Student Database Menu ---
1. Add Student
2. View Students
3. Update Student
4. Delete Student
5. Exit
Enter your choice: 2

ID | Name | Year | Email | Stream
16000 | Nethmi Bandara | 3 | nethmib@stu.cmb.ac.lk | Physical Science
16001 | Lahiru Amarasekara | 3 | lahirua@stu.cmb.ac.lk | Computer Science
16002 | Kavindu Dissanayake | 3 | kavindud@stu.cmb.ac.lk | Physical Science
16003 | Himansa Peiris | 3 | himansap@stu.cmb.ac.lk | Physical Science
16004 | Binura Gamage | 2 | binurag@stu.cmb.ac.lk | Physical Science
16005 | Yasith Perera | 2 | yasithp@stu.cmb.ac.lk | Bio Science
16006 | Kesith balasooriya | 3 | kesithb@stu.cmb.ac.lk | Bio Science
16008 | Sayuri Gunasekara | 3 | sayurig@stu.cmb.ac.lk | Computer Science
16009 | Yevin Udawatte | 1 | yevinu@stu.cmb.ac.lk | Physical Science
16010 | Nihara Gamage | 4 | niharag@stu.cmb.ac.lk | Medicine
16011 | Uvindu Amarasinghe | 1 | uvindua@stu.cmb.ac.lk | Physical Science
```