Ex. No. : Date :	MINI-PROJECT

PROJECT DESCRIPTION:

The Student Record Management System is a desktop application developed using Java (with Swing components for GUI) and a relational database (SQLPlus) for data storage. Its primary function is to centralize and manage key academic and administrative information for students.

FUNCTIONAL OVERVIEW:

The system provides Create, Read, Update, and Delete (CRUD) operations across five dedicated tables, categorized into four modules:

1. **Student Core Records**: Manages basic student data (Name, Register Number, DOB, Address).
2. **Academic History**: Stores prior educational details (11th/12th marks, family names).
3. **Student Attendance**: Tracks subject-wise attendance, separated for Year I and Year II.
4. **Student Achievements**: Records extra-curricular awards and accomplishments.

DATABASE SCHEMA:

Table Name	Attributes (Columns)
STUDENT_RECORDS	Sno (Auto-increment), **REG_NO** (Primary Key), S_NAME, S_YEAR, DOB, HOSTELLER, PHONE_NO, ADDRESS
STUDENT_HISTORY	Sno (Auto-increment), **REG_NO** (Foreign Key), S_NAME, FATHER_NAME, MOTHER_NAME, GR_11_GROUP, GR_11_SCHOOL, GR_11_MARKS, GR_12_SCHOOL, GR_12_MARKS
STUDENT_ACHIEVEMENTS	Sno (Auto-increment), **REG_NO** (Foreign Key), S_NAME, A_YEAR, PRIZE, EVENT_NAME, EVENT_DATE, FEEDBACK, TECH_NONTECH
STUDENT_ATTENDANCE_01	Sno (Auto-increment), **REG_NO** (Foreign Key), S_NAME, AI, DBMS, DM, NDSA, OOPS, SYS_DISC, CLUB, AI_LAB, NDSA_LAB, OOP_LAB, DBMS_LAB, ESS, MENTORING, TOTAL_ATTENDANCE
STUDENT_ATTENDANCE_02	Sno (Auto-increment), **REG_NO** (Foreign Key), S_NAME, MATHS, CHEMISTRY, PYTHON, BEEE, MENTORING, TAMIL, ENGLISH, LIFESKILLS, EPL_LAB, PYTHON_LAB, CHEMISTRY_LAB, TOTAL_ATTENDANCE

Database Implementation Script (DDL):

1. User Access and Privileges:

GRANT CREATE TABLE, DROP ANY TABLE, CREATE SEQUENCE, CREATE TRIGGER, UNLIMITED TABLESPACE TO AD7013;

GRANT DBA TO AD7013;

EXEC SETUP_STUDENT_SCHEMA;

2. Schema Definition (DDL Script):

```
CREATE OR REPLACE PROCEDURE SETUP_STUDENT_SCHEMA
IS
BEGIN
  BEGIN
    EXECUTE IMMEDIATE 'DROP TABLE STUDENT_ACHIEVEMENTS CASCADE CONSTRAINTS';
    EXECUTE IMMEDIATE 'DROP TABLE STUDENT_ATTENDANCE_02 CASCADE CONSTRAINTS';
    EXECUTE IMMEDIATE 'DROP TABLE STUDENT_ATTENDANCE_01 CASCADE CONSTRAINTS';
    EXECUTE IMMEDIATE 'DROP TABLE STUDENT_HISTORY CASCADE CONSTRAINTS';
    EXECUTE IMMEDIATE 'DROP TABLE STUDENT_RECORDS CASCADE CONSTRAINTS';

    EXECUTE IMMEDIATE 'DROP SEQUENCE sr_seq';
    EXECUTE IMMEDIATE 'DROP SEQUENCE sh_seq';
    EXECUTE IMMEDIATE 'DROP SEQUENCE sa01_seq';
    EXECUTE IMMEDIATE 'DROP SEQUENCE sa02_seq';
    EXECUTE IMMEDIATE 'DROP SEQUENCE sach_seq';
  EXCEPTION WHEN OTHERS THEN
    NULL; -- Ignore errors if objects don't exist
  END;

  EXECUTE IMMEDIATE 'CREATE SEQUENCE sr_seq START WITH 1 INCREMENT BY 1';
  EXECUTE IMMEDIATE 'CREATE SEQUENCE sh_seq START WITH 1 INCREMENT BY 1';
  EXECUTE IMMEDIATE 'CREATE SEQUENCE sa01_seq START WITH 1 INCREMENT BY 1';
  EXECUTE IMMEDIATE 'CREATE SEQUENCE sa02_seq START WITH 1 INCREMENT BY 1';
  EXECUTE IMMEDIATE 'CREATE SEQUENCE sach_seq START WITH 1 INCREMENT BY 1';
```

```
-- STUDENT_RECORDS Table (Core Student Data)
   EXECUTE IMMEDIATE 'CREATE TABLE STUDENT_RECORDS (
      SNO NUMBER(10) PRIMARY KEY,
      REG_NO VARCHAR2(12) NOT NULL UNIQUE,
      S_NAME VARCHAR2(100) NOT NULL,
      S_YEAR CHAR(1) NOT NULL,
      DOB DATE NOT NULL,
      HOSTELLER CHAR(1) NOT NULL,
      PHONE_NO NUMBER(10) NOT NULL UNIQUE,
      ADDRESS VARCHAR2(250) NOT NULL,
      CONSTRAINT chk_s_year CHECK (S_YEAR IN ("1", "2", "3", "4")),
      CONSTRAINT chk_hosteller CHECK (HOSTELLER IN ("Y", "N"))
   )';
   EXECUTE IMMEDIATE 'CREATE OR REPLACE TRIGGER sr_trigger
   BEFORE INSERT ON STUDENT_RECORDS FOR EACH ROW
   BEGIN SELECT sr_seq.NEXTVAL INTO :NEW.SNO FROM dual; END;';

   -- STUDENT_HISTORY Table (Prior Academic/Family Data)
   EXECUTE IMMEDIATE 'CREATE TABLE STUDENT_HISTORY (
      SNO NUMBER(10) PRIMARY KEY,
      REG_NO VARCHAR2(12) NOT NULL,
      S_NAME VARCHAR2(100) NOT NULL,
      FATHER_NAME VARCHAR2(100) NOT NULL,
      MOTHER_NAME VARCHAR2(100) NOT NULL,
      GR_11_GROUP VARCHAR2(50),
      GR_11_SCHOOL VARCHAR2(100),
      GR_11_MARKS NUMBER(5, 2) CHECK (GR_11_MARKS <= 100),
      GR_12_SCHOOL VARCHAR2(100),
      GR_12_MARKS NUMBER(5, 2) CHECK (GR_12_MARKS <= 100),
      CONSTRAINT fk_history_regno FOREIGN KEY (REG_NO) REFERENCES
STUDENT_RECORDS (REG_NO)
   )';
   EXECUTE IMMEDIATE 'CREATE OR REPLACE TRIGGER sh_trigger
   BEFORE INSERT ON STUDENT_HISTORY FOR EACH ROW
   BEGIN SELECT sh_seq.NEXTVAL INTO :NEW.SNO FROM dual; END;';

   -- STUDENT_ATTENDANCE_01 Table (Year 1 Attendance)
   EXECUTE IMMEDIATE 'CREATE TABLE STUDENT_ATTENDANCE_01 (
      SNO NUMBER(10) PRIMARY KEY, REG_NO VARCHAR2(12) NOT NULL UNIQUE,
S_NAME VARCHAR2(100) NOT NULL,
      AI NUMBER(5, 2) CHECK (AI <= 100), DBMS NUMBER(5, 2) CHECK (DBMS <= 100),
DM NUMBER(5, 2) CHECK (DM <= 100), NDSA NUMBER(5, 2) CHECK (NDSA <= 100),
      OOPS NUMBER(5, 2) CHECK (OOPS <= 100), SYS_DISC NUMBER(5, 2) CHECK
(SYS_DISC <= 100), CLUB NUMBER(5, 2) CHECK (CLUB <= 100),
      AI_LAB NUMBER(5, 2) CHECK (AI_LAB <= 100), NDSA_LAB NUMBER(5, 2) CHECK
(NDSA_LAB <= 100), OOP_LAB NUMBER(5, 2) CHECK (OOP_LAB <= 100),
      DBMS_LAB NUMBER(5, 2) CHECK (DBMS_LAB <= 100), ESS NUMBER(5, 2) CHECK
(ESS <= 100), MENTORING NUMBER(5, 2) CHECK (MENTORING <= 100),
      TOTAL_ATTENDANCE NUMBER(5, 2) NOT NULL CHECK (TOTAL_ATTENDANCE
<= 100),
```

```
CONSTRAINT fk_att01_regno FOREIGN KEY (REG_NO) REFERENCES STUDENT_RECORDS (REG_NO) )';

EXECUTE IMMEDIATE 'CREATE OR REPLACE TRIGGER sa01_trigger BEFORE INSERT ON STUDENT_ATTENDANCE_01 FOR EACH ROW BEGIN SELECT sa01_seq.NEXTVAL INTO :NEW.SNO FROM dual; END;';
```

-- STUDENT_ATTENDANCE_02 Table (Year 2 Attendance)

```
EXECUTE IMMEDIATE 'CREATE TABLE STUDENT_ATTENDANCE_02 (
    SNO NUMBER(10) PRIMARY KEY, REG_NO VARCHAR2(12) NOT NULL UNIQUE, S_NAME VARCHAR2(100) NOT NULL,
    MATHS NUMBER(5, 2) CHECK (MATHS <= 100), CHEMISTRY NUMBER(5, 2) CHECK (CHEMISTRY <= 100), PYTHON NUMBER(5, 2) CHECK (PYTHON <= 100),
    BEEE NUMBER(5, 2) CHECK (BEEE <= 100), MENTORING NUMBER(5, 2) CHECK (MENTORING <= 100), TAMIL NUMBER(5, 2) CHECK (TAMIL <= 100),
    ENGLISH NUMBER(5, 2) CHECK (ENGLISH <= 100), LIFESKILLS NUMBER(5, 2) CHECK (LIFESKILLS <= 100), EPL_LAB NUMBER(5, 2) CHECK (EPL_LAB <= 100),
    PYTHON_LAB NUMBER(5, 2) CHECK (PYTHON_LAB <= 100), CHEMISTRY_LAB NUMBER(5, 2) CHECK (CHEMISTRY_LAB <= 100),
    TOTAL_ATTENDANCE NUMBER(5, 2) NOT NULL CHECK (TOTAL_ATTENDANCE <= 100),
    CONSTRAINT fk_att02_regno FOREIGN KEY (REG_NO) REFERENCES STUDENT_RECORDS (REG_NO)
)';
EXECUTE IMMEDIATE 'CREATE OR REPLACE TRIGGER sa02_trigger BEFORE INSERT ON STUDENT_ATTENDANCE_02 FOR EACH ROW BEGIN SELECT sa02_seq.NEXTVAL INTO :NEW.SNO FROM dual; END;';
```

-- STUDENT_ACHIEVEMENTS Table (Extracurricular Data)

```
EXECUTE IMMEDIATE 'CREATE TABLE STUDENT_ACHIEVEMENTS (
    SNO NUMBER(10) PRIMARY KEY, REG_NO VARCHAR2(12) NOT NULL, S_NAME VARCHAR2(100) NOT NULL,
    A_YEAR VARCHAR2(10) NOT NULL, PRIZE VARCHAR2(50), EVENT_NAME VARCHAR2(100) NOT NULL,
    EVENT_DATE DATE NOT NULL, FEEDBACK VARCHAR2(150), TECH_NONTECH CHAR(1) NOT NULL,
    CONSTRAINT chk_tnt CHECK (TECH_NONTECH IN ("Y", "N")),
    CONSTRAINT fk_ach_regno FOREIGN KEY (REG_NO) REFERENCES STUDENT_RECORDS (REG_NO)
)';
EXECUTE IMMEDIATE 'CREATE OR REPLACE TRIGGER sach_trigger BEFORE INSERT ON STUDENT_ACHIEVEMENTS FOR EACH ROW BEGIN SELECT sach_seq.NEXTVAL INTO :NEW.SNO FROM dual; END;';

END;
/
```

-- Execution command to run the schema setup

```
EXEC SETUP_STUDENT_SCHEMA;
```

3. Test Data Population:

```
CREATE OR REPLACE PROCEDURE LOAD_TEST_DATA
IS
BEGIN

   EXECUTE IMMEDIATE 'TRUNCATE TABLE STUDENT_ACHIEVEMENTS';
   EXECUTE IMMEDIATE 'TRUNCATE TABLE STUDENT_ATTENDANCE_02';
   EXECUTE IMMEDIATE 'TRUNCATE TABLE STUDENT_ATTENDANCE_01';
   EXECUTE IMMEDIATE 'TRUNCATE TABLE STUDENT_HISTORY';
   EXECUTE IMMEDIATE 'TRUNCATE TABLE STUDENT_RECORDS';

   -- A. STUDENT_RECORDS
   INSERT INTO STUDENT_RECORDS (REG_NO, S_NAME, S_YEAR, DOB, HOSTELLER, PHONE_NO, ADDRESS) VALUES ('311124243001', 'Arjun Kumar', '1', TO_DATE('2006-05-15', 'YYYY-MM-DD'), 'Y', 9876543210, 'Hostel Block A, Chennai');
   INSERT INTO STUDENT_RECORDS (REG_NO, S_NAME, S_YEAR, DOB, HOSTELLER, PHONE_NO, ADDRESS) VALUES ('311124243002', 'Priya Sharma', '1', TO_DATE('2005-11-22', 'YYYY-MM-DD'), 'N', 9988776655, '12/A Gandhipuram, Coimbatore');
   -- ... (23 more STUDENT_RECORDS inserts) ...

   -- B. STUDENT_HISTORY
   INSERT INTO STUDENT_HISTORY (REG_NO, S_NAME, FATHER_NAME, MOTHER_NAME, GR_11_GROUP, GR_11_SCHOOL, GR_11_MARKS, GR_12_SCHOOL, GR_12_MARKS) VALUES ('311124243001', 'Arjun Kumar', 'Rajesh Kumar', 'Meena Kumar', 'Maths-Biology', 'HSS School 1', 92.5, 'HSS School 1', 94.0);
   INSERT INTO STUDENT_HISTORY (REG_NO, S_NAME, FATHER_NAME, MOTHER_NAME, GR_11_GROUP, GR_11_SCHOOL, GR_11_MARKS, GR_12_SCHOOL, GR_12_MARKS) VALUES ('311124243002', 'Priya Sharma', 'Anil Sharma', 'Geeta Sharma', 'Computer Science', 'Ideal School', 88.0, 'Ideal School', 91.5);
   -- ... (23 more STUDENT_HISTORY inserts) ...

   -- C. STUDENT_ATTENDANCE_01 (Year 1)
   INSERT INTO STUDENT_ATTENDANCE_01 (REG_NO, S_NAME, AI, DBMS, DM, NDSA, OOPS, SYS_DISC, CLUB, AI_LAB, NDSA_LAB, OOP_LAB, DBMS_LAB, ESS, MENTORING, TOTAL_ATTENDANCE) VALUES
   ('311124243001', 'Arjun Kumar', 75.0, 88.5, 90.0, 78.0, 85.5, 92.0, 70.0, 80.0, 82.0, 79.5, 91.0, 95.0, 80.0, 84.0);
   -- ... (4 more STUDENT_ATTENDANCE_01 inserts) ...

   -- D. STUDENT_ATTENDANCE_02 (Year 2)
   INSERT INTO STUDENT_ATTENDANCE_02 (REG_NO, S_NAME, MATHS, CHEMISTRY, PYTHON, BEEE, MENTORING, TAMIL, ENGLISH, LIFESKILLS, EPL_LAB, PYTHON_LAB, CHEMISTRY_LAB, TOTAL_ATTENDANCE) VALUES
   ('311125243006', 'Meena Krishnan', 85.0, 88.0, 92.0, 80.0, 85.0, 95.0, 90.0, 88.0, 93.0, 90.0, 86.0, 88.8);
   -- ... (4 more STUDENT_ATTENDANCE_02 inserts) ...

   -- E. STUDENT_ACHIEVEMENTS
   INSERT INTO STUDENT_ACHIEVEMENTS (REG_NO, S_NAME, A_YEAR, PRIZE, EVENT_NAME, EVENT_DATE, FEEDBACK, TECH_NONTECH) VALUES
   ('311124243001', 'Arjun Kumar', '2024-25', 'First Place', 'AI Hackathon', TO_DATE('2025-01-20', 'YYYY-MM-DD'), 'Developed a functional chatbot model.', 'Y');
   INSERT INTO STUDENT_ACHIEVEMENTS (REG_NO, S_NAME, A_YEAR, PRIZE, EVENT_NAME, EVENT_DATE, FEEDBACK, TECH_NONTECH) VALUES
```

```
('311124243002', 'Priya Sharma', '2024-25', 'Second Place', 'Essay Writing Contest', TO_DATE('2025-02-15', 'YYYY-MM-DD'), 'Excellent descriptive writing skills.', 'N');
   -- ... (23 more STUDENT_ACHIEVEMENTS inserts) ...

   COMMIT;

END;
/
-- Execution command to load the test data
EXEC LOAD_TEST_DATA;
```

Application Source Code:

1. DBUtil.java (Database Utility):

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

public class DBUtil {

   // --- CONFIGURATION ---
   private static final String DB_URL = "jdbc:oracle:thin:@192.168.1.18:1521:orcl";
   private static final String DB_USER = "ad7013";
   private static final String DB_PASSWORD = "licet";

   public static Connection createDbConnection() {
      Connection con = null;
      try {
         // Load the Oracle Driver
         Class.forName("oracle.jdbc.driver.OracleDriver");
         con = DriverManager.getConnection(DB_URL, DB_USER, DB_PASSWORD);
         con.setAutoCommit(false); // Enable manual transaction control
         System.out.println("Database Connection Successful.");
      } catch (ClassNotFoundException e) {
         System.err.println("JDBC Driver not found: " + e.getMessage());
      } catch (SQLException e) {
         System.err.println("Database Connection Failed: " + e.getMessage());
      }
      return con;
   }

   public static void closeDbConnection(Connection con) {
      if (con != null) {
         try {
            con.close();
            System.out.println("Database Connection Closed.");
         } catch (SQLException e) {
            System.err.println("Error closing connection: " + e.getMessage());
```

```
        }
    }
}

/**
 * Executes a SELECT query.
 * @return A List of String arrays (data) and a String array (column names).
 */
public static Object[] fetchAllData(Connection con, String tableName) {
    String query = "SELECT * FROM " + tableName;
    List<String[]> data = new ArrayList<>();
    String[] columns = null;

    try (Statement stmt = con.createStatement();
         ResultSet rs = stmt.executeQuery(query)) {

        ResultSetMetaData rsmd = rs.getMetaData();
        int columnCount = rsmd.getColumnCount();
        columns = new String[columnCount];

        // Get column names
        for (int i = 1; i <= columnCount; i++) {
            columns[i - 1] = rsmd.getColumnName(i);
        }

        // Get data rows
        while (rs.next()) {
            String[] row = new String[columnCount];
            for (int i = 1; i <= columnCount; i++) {
                // Get data as String for simple GUI display
                row[i - 1] = rs.getString(i);
            }
            data.add(row);
        }

    } catch (SQLException e) {
        System.err.println("Error fetching data from " + tableName + ": " + e.getMessage());
        return new Object[]{new ArrayList<String[]>(), null}; // Return empty on error
    }

    return new Object[]{data, columns};
}

/**
 * Executes INSERT, UPDATE, or DELETE query.
 * @param params The parameters for the PreparedStatement.
 * @return The number of rows affected, or -1 on error.
 */
public static int executeUpdate(Connection con, String query, Object[] params) {
    try (PreparedStatement pstmt = con.prepareStatement(query)) {
        for (int i = 0; i < params.length; i++) {
            pstmt.setObject(i + 1, params[i]);
```

```
            }
            int rows = pstmt.executeUpdate();
            con.commit(); // Commit the transaction on success
            return rows;
        } catch (SQLException e) {
            System.err.println("Error executing update query: " + e.getMessage());
            try {
                if (con != null) con.rollback(); // Rollback on error
            } catch (SQLException rollbackEx) {
                System.err.println("Rollback failed: " + rollbackEx.getMessage());
            }
            return -1;
        }
    }

    /**
     * Executes a SELECT query with parameters to fetch a single record.
     */
    public static Object[] fetchRecord(Connection con, String tableName, String regNo) {
        String query = "SELECT * FROM " + tableName + " WHERE REG_NO = ?";
        String[] recordData = null;
        String[] columns = null;

        try (PreparedStatement pstmt = con.prepareStatement(query)) {
            pstmt.setString(1, regNo);
            try (ResultSet rs = pstmt.executeQuery()) {
                if (rs.next()) {
                    ResultSetMetaData rsmd = rs.getMetaData();
                    int columnCount = rsmd.getColumnCount();
                    columns = new String[columnCount];
                    recordData = new String[columnCount];

                    for (int i = 1; i <= columnCount; i++) {
                        columns[i - 1] = rsmd.getColumnName(i);
                        recordData[i - 1] = rs.getString(i);
                    }
                }
            }
        } catch (SQLException e) {
            System.err.println("Error fetching single record: " + e.getMessage());
        }
        return new Object[]{recordData, columns};
    }
}
```

2. java (Reusable GUI Component):

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionListener;
import java.util.HashMap;
import java.util.Map;

public class MenuPanel extends JPanel {
   private final Map<String, JButton> buttons = new HashMap<>();

   public MenuPanel(String title) {
      setLayout(new BoxLayout(this, BoxLayout.Y_AXIS));
      setBorder(BorderFactory.createEmptyBorder(50, 50, 50, 50));

      JLabel titleLabel = new JLabel(title);
      titleLabel.setFont(new Font("Arial", Font.BOLD, 20));
      titleLabel.setAlignmentX(Component.CENTER_ALIGNMENT);
      add(titleLabel);
      add(Box.createRigidArea(new Dimension(0, 30)));

      String[] options = {"VIEW", "ADD RECORD", "EDIT RECORD", "BACK"};

      for (String text : options) {
         JButton button = new JButton(text);
         button.setFont(new Font("Arial", Font.PLAIN, 16));
         button.setPreferredSize(new Dimension(200, 40));
         button.setMaximumSize(new Dimension(300, 40));
         button.setAlignmentX(Component.CENTER_ALIGNMENT);

         buttons.put(text, button);
         add(button);
         add(Box.createRigidArea(new Dimension(0, 15)));
      }
   }

   public void setButtonAction(String buttonText, ActionListener listener) {
      JButton button = buttons.get(buttonText);
      if (button != null) {
         // Remove previous listeners and add the new one
         for(ActionListener al : button.getActionListeners()) {
            button.removeActionListener(al);
         }
         button.addActionListener(listener);
      }
   }
}
```

3. StudentRecordManager.java (Main Application):

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
```

```
import java.util.Arrays;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.sql.Connection;
import java.text.SimpleDateFormat;
import javax.swing.table.DefaultTableModel;
import javax.swing.border.EmptyBorder;

public class StudentRecordManager extends JFrame {

    private final CardLayout cardLayout = new CardLayout();
    private final JPanel mainPanel = new JPanel(cardLayout);
    private final Connection dbConn;

    // Table definitions for dynamic form generation
    private final Map<String, Object[]> tableFields = new HashMap<>();

    public StudentRecordManager() {
        super("AI & DS STUDENT RECORD MANAGER");

        // 1. Database Connection
        dbConn = DBUtil.createDbConnection();
        if (dbConn == null) {
            JOptionPane.showMessageDialog(this, "Failed to connect to the database. Application will exit.", "DB
Error", JOptionPane.ERROR_MESSAGE);
            System.exit(1);
        }

        // 2. Frame Setup
        setDefaultCloseOperation(DO_NOTHING_ON_CLOSE);
        setSize(800, 600);
        setResizable(false); // Do not allow maximize
        setLocationRelativeTo(null);
        add(mainPanel);

        // 3. Initialize Fields Map (used by RecordFormPanel)
        initializeTableFields();

        // 4. Create Panels (GUI components)
        createOpeningInterface();
        createAllMenuFrames();

        // 5. Initial Display
        cardLayout.show(mainPanel, "OpeningInterface");
        setVisible(true);

        // 6. Window Close Handler (for Case 1.4 equivalent - exit)
        addWindowListener(new WindowAdapter() {
            @Override
            public void windowClosing(WindowEvent e) {
                DBUtil.closeDbConnection(dbConn);
                dispose();
            }
        });
    }

    // --- Helper Methods ---

    private void initializeTableFields() {
        // [COL_NAME, Label Text, Type (text, int, float, date, option), [options_list], isEditableInEditMode]
        tableFields.put("STUDENT_RECORDS", new Object[][]{
            {"REG_NO", "Register Number:", "text", null, false},
            {"S_NAME", "Name:", "text", null, true},
```

```
            {"S_YEAR", "Year (1-4):", "option", new String[]{"1", "2", "3", "4"}, true},
            {"DOB", "DOB (YYYY-MM-DD):", "date", null, true},
            {"HOSTELLER", "Hosteller (Y/N):", "option", new String[]{"Y", "N"}, true},
            {"PHONE_NO", "Phone Number:", "int", null, true},
            {"ADDRESS", "Address (Hint: Enter 'Hostel' if Y):", "text", null, true}
        });

        tableFields.put("STUDENT_HISTORY", new Object[][]{
            {"REG_NO", "Register Number:", "text", null, false},
            {"S_NAME", "Name:", "text", null, true},
            {"FATHER_NAME", "Father Name:", "text", null, true},
            {"MOTHER_NAME", "Mother Name:", "text", null, true},
            {"GR_11_GROUP", "11th Group Studied:", "text", null, true},
            {"GR_11_SCHOOL", "11th School:", "text", null, true},
            {"GR_11_MARKS", "11th Marks (%):", "float", null, true},
            {"GR_12_SCHOOL", "12th School:", "text", null, true},
            {"GR_12_MARKS", "12th Marks (%):", "float", null, true},
        });

        // ATTENDANCE tables have dynamic fields (handled in getFormFieldsForAttendance)
        // ACHIEVEMENTS
        tableFields.put("STUDENT_ACHIEVEMENTS", new Object[][]{
            {"REG_NO", "Register Number:", "text", null, false},
            {"S_NAME", "Name:", "text", null, true},
            {"A_YEAR", "Academic Year:", "text", null, true},
            {"PRIZE", "Prize:", "text", null, true},
            {"EVENT_NAME", "Event Name:", "text", null, true},
            {"EVENT_DATE", "Event Date (YYYY-MM-DD):", "date", null, true},
            {"FEEDBACK", "Feedback (Max 150 chars):", "text", null, true},
            {"TECH_NONTECH", "Tech/Non-Tech (Y/N):", "option", new String[]{"Y", "N"}, true},
        });
    }

    // --- GUI Creation Methods ---

    private void createOpeningInterface() {
        JPanel panel = new JPanel();
        panel.setLayout(new BoxLayout(panel, BoxLayout.Y_AXIS));
        panel.setBorder(new EmptyBorder(80, 0, 0, 0)); // Top padding

        JLabel title = new JLabel("AI & DS STUDENT RECORD MANAGER");
        title.setFont(new Font("Arial", Font.BOLD, 24));
        title.setAlignmentX(Component.CENTER_ALIGNMENT);
        panel.add(title);
        panel.add(Box.createRigidArea(new Dimension(0, 50)));

        String[][] options = {
            {"View Students records", "RecordsMenu"},
            {"View student history", "HistoryMenu"},
            {"View Student Attendace", "AttendanceYearSelect"},
            {"View student Achievements", "AchievementsMenu"}
        };

        for (String[] option : options) {
            JButton button = new JButton(option[0]);
            button.setFont(new Font("Arial", Font.PLAIN, 16));
            button.setPreferredSize(new Dimension(300, 40));
            button.setMaximumSize(new Dimension(300, 40));
            button.setAlignmentX(Component.CENTER_ALIGNMENT);
            button.addActionListener(e -> showFrame(option[1]));
            panel.add(button);
            panel.add(Box.createRigidArea(new Dimension(0, 15)));
        }
```

```
        mainPanel.add(panel, "OpeningInterface");
    }

    private void createAllMenuFrames() {
        // Case 1: STUDENT_RECORDS Menu
        createMenuFrame("RecordsMenu", "STUDENT RECORDS", "STUDENT_RECORDS",
"RecordsMenu");
        // Case 2: STUDENT_HISTORY Menu
        createMenuFrame("HistoryMenu", "STUDENT HISTORY", "STUDENT_HISTORY", "HistoryMenu");
        // Case 4: STUDENT_ACHIEVEMENTS Menu
        createMenuFrame("AchievementsMenu", "STUDENT ACHIEVEMENTS",
"STUDENT_ACHIEVEMENTS", "AchievementsMenu");

        // Case 3: Attendance Year Selection
        createAttendanceYearSelection();

        // Case 3.1: Attendance Year I Menu
        createMenuFrame("Year1Menu", "ATTENDANCE - YEAR I", "STUDENT_ATTENDANCE_01",
"Year1Menu", "AttendanceYearSelect");
        // Case 3.2: Attendance Year II Menu
        createMenuFrame("Year2Menu", "ATTENDANCE - YEAR II", "STUDENT_ATTENDANCE_02",
"Year2Menu", "AttendanceYearSelect");
    }

    private void createMenuFrame(String frameKey, String title, String tableName, String menuFrameKey) {
        // Default back button goes to Opening Interface
        createMenuFrame(frameKey, title, tableName, menuFrameKey, "OpeningInterface");
    }

    private void createMenuFrame(String frameKey, String title, String tableName, String menuFrameKey, String
backFrame) {
        MenuPanel panel = new MenuPanel(title);

        // Case X.1: VIEW
        panel.setButtonAction("VIEW", e -> showDataView(tableName, title, menuFrameKey));
        // Case X.2: ADD RECORD
        panel.setButtonAction("ADD RECORD", e -> showRecordForm("ADD", tableName, menuFrameKey,
null));
        // Case X.3: EDIT RECORD
        panel.setButtonAction("EDIT RECORD", e -> showEditPrompt(tableName, menuFrameKey));
        // Case X.4: BACK
        panel.setButtonAction("BACK", e -> showFrame(backFrame));

        mainPanel.add(panel, frameKey);
    }

    private void createAttendanceYearSelection() {
        JPanel panel = new JPanel();
        panel.setLayout(new BoxLayout(panel, BoxLayout.Y_AXIS));
        panel.setBorder(new EmptyBorder(80, 0, 0, 0));

        JLabel title = new JLabel("VIEW STUDENT ATTENDANCE");
        title.setFont(new Font("Arial", Font.BOLD, 24));
        title.setAlignmentX(Component.CENTER_ALIGNMENT);
        panel.add(title);
        panel.add(Box.createRigidArea(new Dimension(0, 50)));

        String[][] options = {
            {"Year I", "Year1Menu"},
            {"Year II", "Year2Menu"}
        };

        for (String[] option : options) {
            JButton button = new JButton(option[0]);
```

```
            button.setFont(new Font("Arial", Font.PLAIN, 16));
            button.setPreferredSize(new Dimension(300, 40));
            button.setMaximumSize(new Dimension(300, 40));
            button.setAlignmentX(Component.CENTER_ALIGNMENT);
            button.addActionListener(e -> showFrame(option[1]));
            panel.add(button);
            panel.add(Box.createRigidArea(new Dimension(0, 15)));
        }

        JButton backButton = new JButton("BACK");
        backButton.setFont(new Font("Arial", Font.PLAIN, 16));
        backButton.setPreferredSize(new Dimension(300, 40));
        backButton.setMaximumSize(new Dimension(300, 40));
        backButton.setAlignmentX(Component.CENTER_ALIGNMENT);
        backButton.addActionListener(e -> showFrame("OpeningInterface"));
        panel.add(Box.createRigidArea(new Dimension(0, 30)));
        panel.add(backButton);

        mainPanel.add(panel, "AttendanceYearSelect");
    }

    // --- Core Navigation and Display ---

    private void showFrame(String name) {
        cardLayout.show(mainPanel, name);
    }

    private void showDataView(String tableName, String title, String backFrame) {
        String frameKey = tableName + "DataView";
        if (mainPanel.getComponent(frameKey) == null) {
            DataViewPanel panel = new DataViewPanel(this, tableName, title, backFrame);
            mainPanel.add(panel, frameKey);
        }
        ((DataViewPanel) mainPanel.getComponent(frameKey)).updateView(); // Update data before showing
        showFrame(frameKey);
    }

    private void showRecordForm(String mode, String tableName, String backFrame, String[] existingData) {
        String frameKey = tableName + "Form";
        RecordFormPanel formPanel;

        // Reuse the form panel if it exists
        if (mainPanel.getComponent(frameKey) == null) {
            formPanel = new RecordFormPanel(this, tableName, backFrame);
            mainPanel.add(formPanel, frameKey);
        } else {
            formPanel = (RecordFormPanel) mainPanel.getComponent(frameKey);
            formPanel.setBackFrame(backFrame);
        }

        formPanel.loadForm(mode, existingData);
        showFrame(frameKey);
    }

    private void showEditPrompt(String tableName, String backFrame) {
        String regNo = JOptionPane.showInputDialog(this, "Enter Register Number for " + tableName + ":", "Edit
Record", JOptionPane.QUESTION_MESSAGE);

        if (regNo != null && !regNo.trim().isEmpty()) {
            Object[] result = DBUtil.fetchRecord(dbConn, tableName, regNo.trim());
            String[] existingData = (String[]) result[0];

            if (existingData != null) {
                showRecordForm("EDIT", tableName, backFrame, existingData);
```

```
            } else {
                JOptionPane.showMessageDialog(this, "No record found for Register Number: " + regNo, "Error",
JOptionPane.ERROR_MESSAGE);
            }
        }
    }

    // --- Data View Panel Implementation (Case X.1) ---

    class DataViewPanel extends JPanel {
        private final StudentRecordManager controller;
        private String tableName;
        private String titleText;
        private String backFrame;
        private final JTable table;
        private final JLabel titleLabel;

        public DataViewPanel(StudentRecordManager controller, String tableName, String title, String
backFrame) {
            this.controller = controller;
            this.tableName = tableName;
            this.titleText = title;
            this.backFrame = backFrame;

            setLayout(new BorderLayout());

            titleLabel = new JLabel("VIEW: " + titleText, SwingConstants.CENTER);
            titleLabel.setFont(new Font("Arial", Font.BOLD, 20));
            add(titleLabel, BorderLayout.NORTH);

            table = new JTable();
            JScrollPane scrollPane = new JScrollPane(table);
            add(scrollPane, BorderLayout.CENTER);

            JButton backButton = new JButton("BACK");
            backButton.setFont(new Font("Arial", Font.PLAIN, 16));
            backButton.addActionListener(e -> controller.showFrame(this.backFrame));
            add(backButton, BorderLayout.SOUTH);
        }

        public void updateView() {
            titleLabel.setText("VIEW: " + titleText);

            Object[] result = DBUtil.fetchAllData(controller.dbConn, tableName);
            List<String[]> data = (List<String[]>) result[0];
            String[] columns = (String[]) result[1];

            if (columns == null || data.isEmpty()) {
                titleLabel.setText("VIEW: " + titleText + " (No Records Found)");
                table.setModel(new DefaultTableModel()); // Clear table
                return;
            }

            // Convert List<String[]> to Object[][] for DefaultTableModel
            Object[][] dataArray = new Object[data.size()][columns.length];
            for (int i = 0; i < data.size(); i++) {
                dataArray[i] = data.get(i);
            }

            DefaultTableModel model = new DefaultTableModel(dataArray, columns);
            table.setModel(model);
        }
    }
```

```
// --- Record Form Panel Implementation (Case X.2, X.3) ---

class RecordFormPanel extends JPanel {
    private final StudentRecordManager controller;
    private String tableName;
    private String backFrame;
    private String mode; // ADD or EDIT
    private String existingRegNo;
    private final Map<String, JComponent> inputFields = new HashMap<>();
    private final JLabel titleLabel;
    private final JPanel formPanel; // The panel holding all the labels and fields
    private final JButton confirmButton;

    public RecordFormPanel(StudentRecordManager controller, String tableName, String backFrame) {
        this.controller = controller;
        this.tableName = tableName;
        this.backFrame = backFrame;
        this.setLayout(new BorderLayout());

        titleLabel = new JLabel("", SwingConstants.CENTER);
        titleLabel.setFont(new Font("Arial", Font.BOLD, 20));
        this.add(titleLabel, BorderLayout.NORTH);

        formPanel = new JPanel(new GridBagLayout());
        JScrollPane scrollPane = new JScrollPane(formPanel);
        scrollPane.setHorizontalScrollBarPolicy(JScrollPane.HORIZONTAL_SCROLLBAR_NEVER);
        this.add(scrollPane, BorderLayout.CENTER);

        JPanel buttonPanel = new JPanel();
        confirmButton = new JButton("CONFIRM");
        confirmButton.addActionListener(e -> confirmRecord());
        buttonPanel.add(confirmButton);

        JButton backButton = new JButton("BACK");
        backButton.addActionListener(e -> controller.showFrame(this.backFrame));
        buttonPanel.add(backButton);
        this.add(buttonPanel, BorderLayout.SOUTH);
    }

    public void setBackFrame(String backFrame) {
        this.backFrame = backFrame;
    }

    private Object[][] getFormFieldsForAttendance(String tableName) {
        String[] subjects01 = {"AI", "DBMS", "DM", "NDSA", "OOPS", "SYS_DISC", "CLUB", "AI_LAB",
"NDSA_LAB", "OOP_LAB", "DBMS_LAB", "ESS", "MENTORING"};
        String[] subjects02 = {"MATHS", "CHEMISTRY", "PYTHON", "BEEE", "MENTORING", "TAMIL",
"ENGLISH", "LIFESKILLS", "EPL_LAB", "PYTHON_LAB", "CHEMISTRY_LAB"};
        String[] subjects = tableName.equals("STUDENT_ATTENDANCE_01") ? subjects01 : subjects02;

        int initialSize = 2 + subjects.length + 1; // REG_NO, S_NAME, subjects, TOTAL
        Object[][] fields = new Object[initialSize][5];

        fields[0] = new Object[]{"REG_NO", "Register Number:", "text", null, false};
        fields[1] = new Object[]{"S_NAME", "Name:", "text", null, true};

        for (int i = 0; i < subjects.length; i++) {
            fields[i + 2] = new Object[]{subjects[i], subjects[i].replace('_', ' ').toUpperCase() + " Attendance (%):",
"float", null, true};
        }

        fields[fields.length - 1] = new Object[]{"TOTAL_ATTENDANCE", "Total Attendance (%):", "float",
null, true};
        return fields;
```

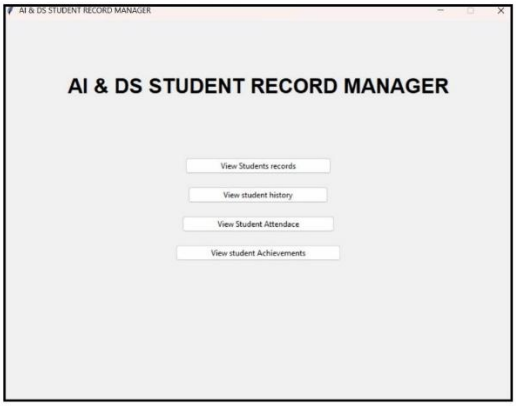
```
    }

    public void loadForm(String mode, String[] existingData) {
        this.mode = mode;
        this.existingRegNo = (mode.equals("EDIT") && existingData.length > 0) ? existingData[0] : null;

        titleLabel.setText(mode + " RECORD - " + tableName.replace('_', ' ').toUpperCase());
        confirmButton.setText(mode.toUpperCase());

        // Clear previous form fields
        formPanel.removeAll();
        inputFields.clear();

        Object[][] fieldDefs;
        if (tableName.startsWith("STUDENT_ATTENDANCE")) {
            fieldDefs = getFormFieldsForAttendance(tableName);
        } else {
            fieldDefs = tableFields.get(tableName);
        }

        if (fieldDefs == null) {
            JOptionPane.showMessageDialog(this, "Configuration for " + tableName + " not found.", "Error",
JOptionPane.ERROR_MESSAGE);
            return;
        }

        GridBagConstraints gbc = new GridBagConstraints();
        gbc.insets = new Insets(5, 10, 5, 10);
        gbc.anchor = GridBagConstraints.WEST;
        gbc.fill = GridBagConstraints.HORIZONTAL;

        for (int i = 0; i < fieldDefs.length; i++) {
            String colName = (String) fieldDefs[i][0];
            String labelText = (String) fieldDefs[i][1];
            String type = (String) fieldDefs[i][2];
            String[] options = (String[]) fieldDefs[i][3];
            boolean isEditable = (boolean) fieldDefs[i][4];

            JLabel label = new JLabel(labelText);
            gbc.gridx = 0;
            gbc.gridy = i;
            formPanel.add(label, gbc);

            JComponent inputComponent;

            if (type.equals("option") && options != null) {
                JComboBox<String> comboBox = new JComboBox<>(options);
                inputComponent = comboBox;
            } else {
                JTextField textField = new JTextField(20);
                inputComponent = textField;
                if (labelText.contains("Hint:")) {
                    labelText = labelText.substring(0, labelText.indexOf("Hint:"));
                }
            }

            // Set initial value in EDIT mode
            if (mode.equals("EDIT") && existingData != null && existingData.length > i) {
                if (inputComponent instanceof JComboBox) {
                    ((JComboBox<String>) inputComponent).setSelectedItem(existingData[i]);
                } else if (inputComponent instanceof JTextField) {
                    ((JTextField) inputComponent).setText(existingData[i]);
                }
            }
```

```
            // Set editability
            if (mode.equals("EDIT") && !isEditable) {
                inputComponent.setEnabled(false);
            }

            gbc.gridx = 1;
            gbc.weightx = 1.0;
            formPanel.add(inputComponent, gbc);
            inputFields.put(colName, inputComponent);

            // Add hint label
            if (labelText.contains("Hint:")) {
                JLabel hintLabel = new JLabel(labelText.substring(labelText.indexOf("Hint:") + 5).trim());
                hintLabel.setForeground(Color.GRAY);
                hintLabel.setFont(new Font("Arial", Font.ITALIC, 10));
                gbc.gridx = 2;
                gbc.weightx = 0.0;
                formPanel.add(hintLabel, gbc);
            }
            gbc.weightx = 0.0;
        }

        formPanel.revalidate();
        formPanel.repaint();
    }

    private Map<String, String> collectData() {
        Map<String, String> data = new HashMap<>();
        for (Map.Entry<String, JComponent> entry : inputFields.entrySet()) {
            String colName = entry.getKey();
            JComponent comp = entry.getValue();
            String value = "";

            if (comp instanceof JTextField) {
                value = ((JTextField) comp).getText().trim();
            } else if (comp instanceof JComboBox) {
                Object selectedItem = ((JComboBox) comp).getSelectedItem();
                if (selectedItem != null) {
                    value = selectedItem.toString().trim();
                }
            }
            data.put(colName, value);
        }
        return data;
    }

    private boolean validateInput(Map<String, String> data) {
        // This is a minimal implementation, proper validation requires referencing tableFields again
        // and mapping them to the collected data.
        // Example:
        String errors = "";

        for (Object[] fieldDef : (tableName.startsWith("STUDENT_ATTENDANCE") ?
getFormFieldsForAttendance(tableName) : tableFields.get(tableName))) {
            String colName = (String) fieldDef[0];
            String label = (String) fieldDef[1];
            String type = (String) fieldDef[2];
            String value = data.get(colName);

            if (value == null || value.isEmpty()) {
                if (colName.equals("S_NAME") || colName.equals("REG_NO") ||
colName.equals("PHONE_NO")) { // Mandatory fields
                    errors += label + " is required.\n";
```

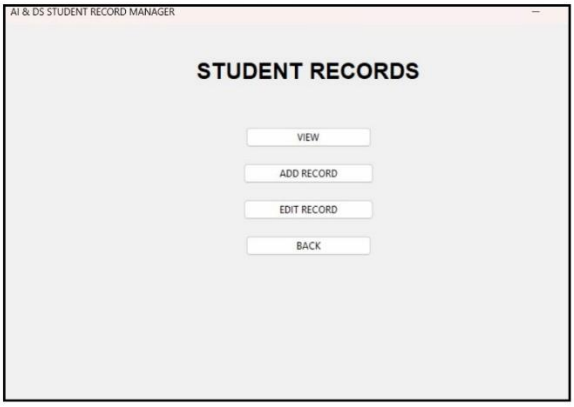
```
                }
                continue;
            }

            try {
                if (type.equals("int")) {
                    long val = Long.parseLong(value);
                    if (colName.equals("PHONE_NO") && (val < 1000000000L || val > 9999999999L)) {
                        errors += "Phone Number must be 10 digits.\n";
                    }
                } else if (type.equals("float")) {
                    float val = Float.parseFloat(value);
                    if (val < 0 || val > 100) {
                        errors += label + " must be between 0 and 100.\n";
                    }
                } else if (type.equals("date")) {
                    new SimpleDateFormat("yyyy-MM-dd").parse(value);
                }
            } catch (Exception ex) {
                errors += "Invalid format for " + label + ". Expected " + type + ".\n";
            }
        }

        if (errors.isEmpty() && tableName.equals("STUDENT_RECORDS")) {
            // REG_NO business logic (Year 1: 3111242430, Year 2: 3111252430)
            String regNo = data.get("REG_NO");
            String year = data.get("S_YEAR");
            if (year != null && regNo != null) {
                String prefix = null;
                if (year.equals("1")) prefix = "3111242430";
                else if (year.equals("2")) prefix = "3111252430";

                if (prefix != null && !regNo.startsWith(prefix)) {
                    errors += "Reg No for Year " + year + " must start with " + prefix + ".\n";
                }
            }
        }


        if (!errors.isEmpty()) {
            JOptionPane.showMessageDialog(controller, errors, "Input Error",
JOptionPane.ERROR_MESSAGE);
            return false;
        }
        return true;
    }

    private void confirmRecord() {
        Map<String, String> data = collectData();
        if (!validateInput(data)) return;

        String query;
        List<Object> params = new ArrayList<>();
        int rowsAffected = -1;

        // Get the field definitions again to maintain order
        Object[][] fieldDefs = tableName.startsWith("STUDENT_ATTENDANCE") ?
getFormFieldsForAttendance(tableName) : tableFields.get(tableName);

        if (mode.equals("ADD")) {
            // INSERT QUERY
            List<String> colNames = new ArrayList<>();
            String placeholders = "";
```

```
            for (Object[] def : fieldDefs) {
                String col = (String) def[0];
                colNames.add(col);
                params.add(data.get(col));
            }

            placeholders = String.join(", ", java.util.Collections.nCopies(colNames.size(), "?"));
            query = String.format("INSERT INTO %s (%s) VALUES (%s)", tableName, String.join(", ", colNames), placeholders);

            // Add Unique Check for STUDENT_RECORDS (optional, but good practice)
            if(tableName.equals("STUDENT_RECORDS")) {
                Object[] check = DBUtil.fetchRecord(dbConn, tableName, data.get("REG_NO"));
                if(check[0] != null) {
                    JOptionPane.showMessageDialog(controller, "Register Number already exists.", "Error", JOptionPane.ERROR_MESSAGE);
                    return;
                }
            }

        } else { // EDIT MODE
            // UPDATE QUERY
            List<String> setClauses = new ArrayList<>();

            for (Object[] def : fieldDefs) {
                String col = (String) def[0];
                boolean isEditable = (boolean) def[4];

                if (isEditable) {
                    setClauses.add(col + " = ?");
                    params.add(data.get(col));
                }
            }

            // Add WHERE clause parameter (existingRegNo)
            params.add(existingRegNo);

            query = String.format("UPDATE %s SET %s WHERE REG_NO = ?", tableName, String.join(", ", setClauses));
        }

        rowsAffected = DBUtil.executeUpdate(controller.dbConn, query, params.toArray());

        if (rowsAffected > 0) {
            JOptionPane.showMessageDialog(controller, "Record " + mode + "ED successfully!", "Success", JOptionPane.INFORMATION_MESSAGE);
            controller.showFrame(backFrame);
        } else {
            JOptionPane.showMessageDialog(controller, "Failed to " + mode + " the record.", "DB Error", JOptionPane.ERROR_MESSAGE);
        }
    }
}

public static void main(String[] args) {
    // Run the GUI creation on the Event Dispatch Thread (standard Java Swing practice)
    SwingUtilities.invokeLater(StudentRecordManager::new);
}
}
```

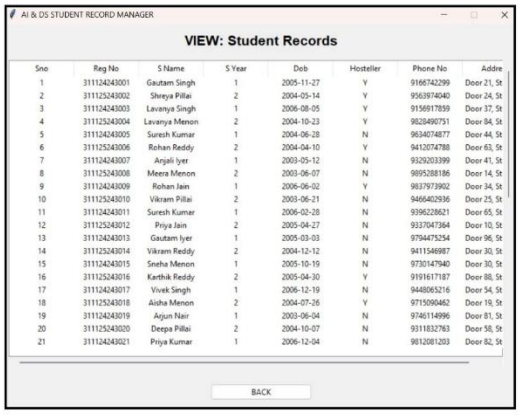
Output :



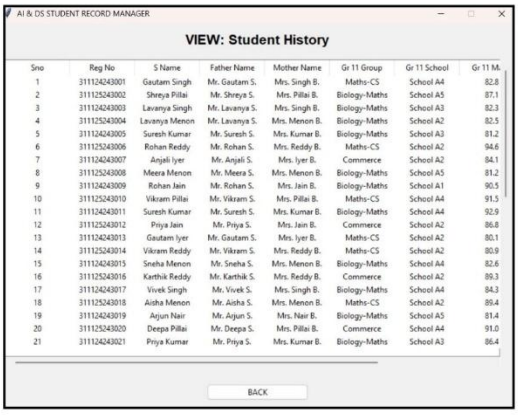
MAIN MENU



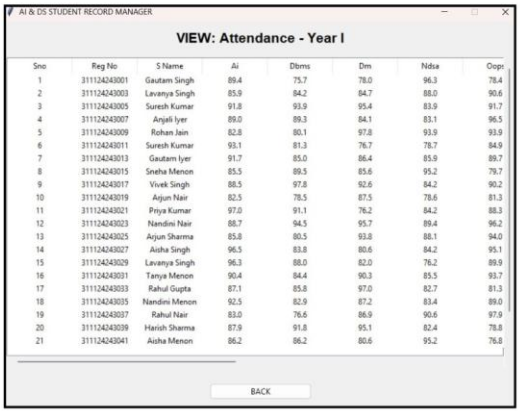
OPTION TAB



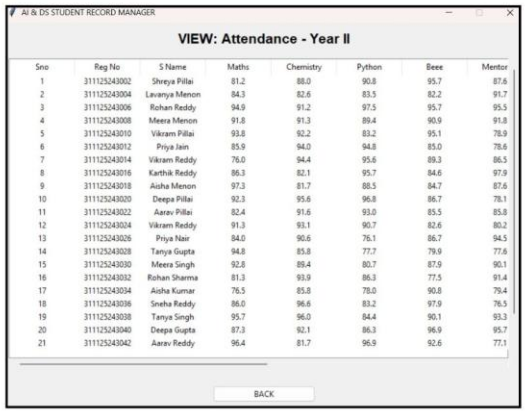
STUDENT RECORDS



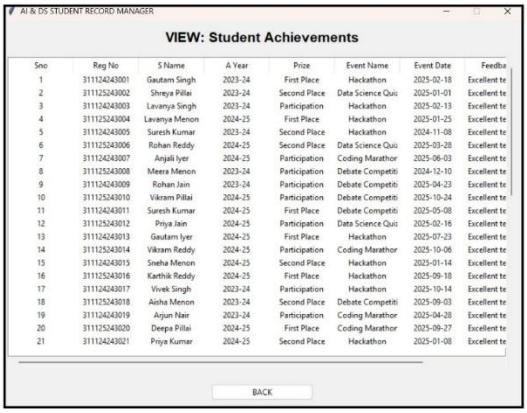
STUDENT HISTORY



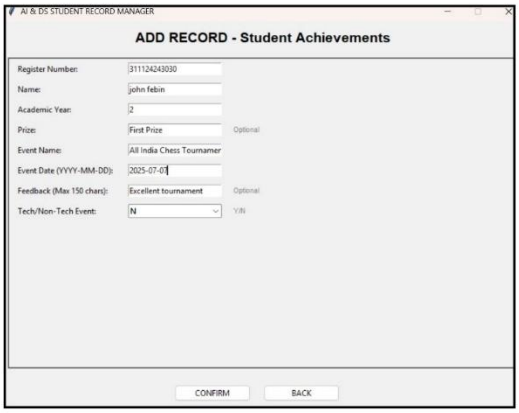
ATTENDANCE - I



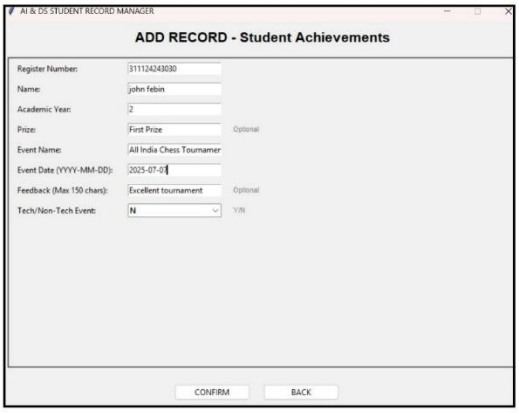
ATTENDANCE - II



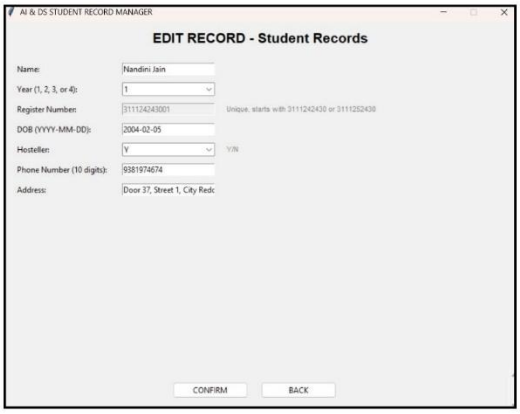
STUDENT ACHIEVEMENTS



ADDING NEW RECORD



ADDING NEW RECORD



EDITING RECORDS

Conclusion :

The Student Record Management System successfully provided a robust Java desktop application with a SQLPlus database backend. The project demonstrated proficiency in JDBC connectivity and database design by implementing a relational schema with Primary/Foreign Keys and Triggers for automatic indexing. It met all functional goals, ensuring efficient, centralized CRUD management of core student, history, attendance, and achievement records.