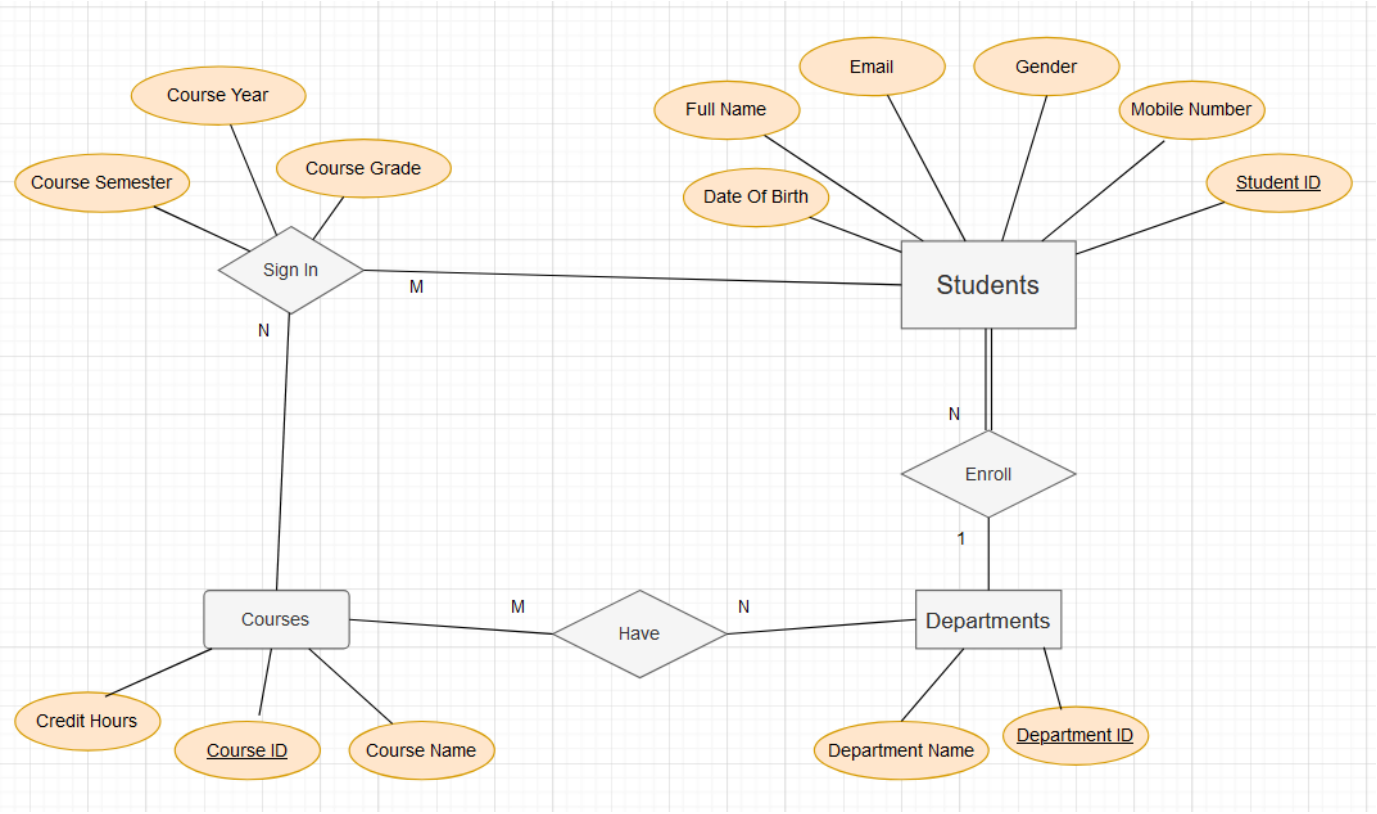
Designed By Mohamed Karam Sharaf

Data Management Track ITI Alexandria Branch

Case Study

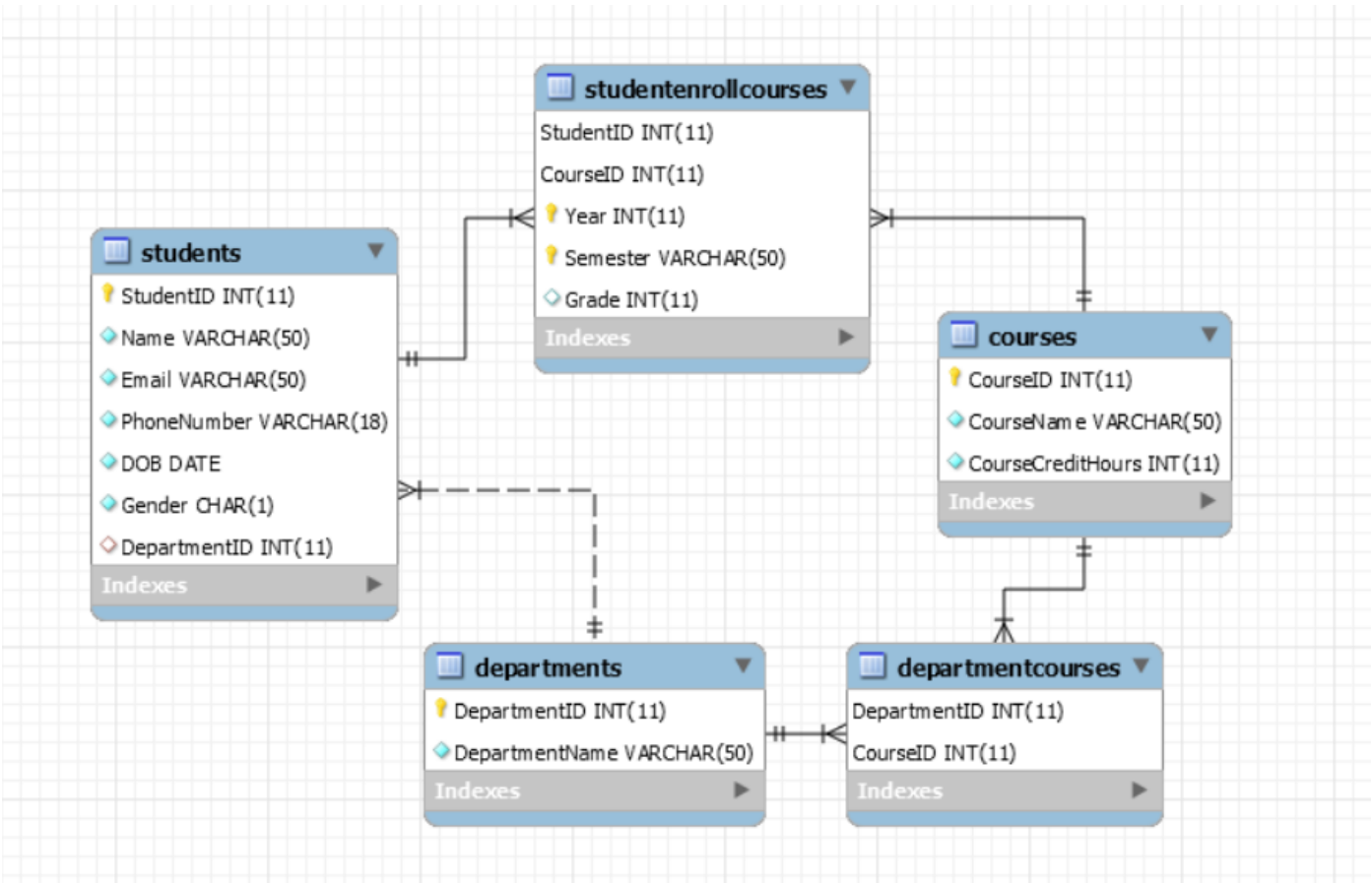
Data Base Design

Generating ERD For the Data Base



The Schema Consists Of 5 Tables & Showing That Data Primary Keys and Relations Between Tables As 1 To Many or Many and The Entities.

Tables View Using MySQL Workbench



The Table View Was Issued by MYSQL Work Bench to Demonstrate the Relations Between Primary Keys, Foreign Keys and Tables in a simple View Along with relations.

Schema Interpretation

1. Departments Table:
   - **Data:**
     - `DepartmentID`: Unique identifier for each department.
     - `DepartmentName`: Name of the department (unique and not null).
   - Features:
     - Retrieve a list of all departments.
     - Find information about a specific department using its ID or name.

2. Students Table:
   - Data:
     - `StudentID`: Unique identifier for each student.
     - `Name`: Name of the student (not null).
     - `Email`: Email address of the student (unique and not null).
     - `PhoneNumber`: Phone number of the student (not null).
     - `DOB`: Date of birth of the student (not null).
     - `Gender`: Gender of the student (not null).
     - `DepartmentID`: Foreign key referencing the department to which the student belongs (nullable).
   - Features:
     - Retrieve a list of all students.
     - Get information about a specific student using their ID or email.
     - Find students belonging to a specific department.

3. Courses Table:

- Data:
  - `CourseID`: Unique identifier for each course.
  - `CourseName`: Name of the course (unique and not null).
  - `CourseCreditHours`: Number of credit hours for the course (not null).
- Features:
  - Retrieve a list of all courses.
  - Find information about a specific course using its ID or name.

4. DepartmentCourses Table:

- Data:
  - `DepartmentID`: Foreign key referencing the department.
  - `CourseID`: Foreign key referencing the course.
- Features:
  - Establish relationships between departments and the courses they offer.
  - Retrieve a list of courses offered by a specific department.

5. StudentEnrollCourses Table:

- Data:
  - `StudentID`: Foreign key referencing the student.
  - `CourseID`: Foreign key referencing the course.
  - `Year`: Academic year for enrollment (not null).
  - `Semester`: Semester for enrollment (not null).
  - `Grade`: Grade obtained in the course (nullable).
- Features:
  - Track the courses in which a student is enrolled.

- Retrieve enrollment details for a specific student or course.
- Calculate statistics like average grades or GPA.

SQL Code Scripts

```
create table departments(
DepartmentID number primary key,
DepartmentName varchar2(50) unique not null
);

CREATE TABLE students (
   StudentID NUMBER PRIMARY KEY,
   Name VARCHAR2(50) NOT NULL,
   Email VARCHAR2(50) NOT NULL UNIQUE,
   PhoneNumber VARCHAR2(18) NOT NULL,
   DOB DATE NOT NULL,
   Gender CHAR(1) NOT NULL,
   DepartmentID  NUMBER  REFERENCES  Departments(DepartmentID)
ON DELETE SET NULL
);

create table courses(
CourseID number primary key,
CourseName varchar2(50) unique not null,
CourseCreditHours number not null
);

create table DepartmentCourses(
```

```
DepartmentID number references Departments(DepartmentID) on delete
cascade,
CourseID number references Courses(CourseID) on delete cascade,
primary key (DepartmentID,CourseID)
);

create table StudentEnrollCourses(
StudentID number references Students(StudentID) on delete cascade,
CourseID number references Courses(CourseID) on delete cascade,
Year number not null,
Semester varchar2(50) not null,
Grade number default null,
primary key(StudentID,CourseID,Year,Semester)
);
```

Examples On PL/SQL Codes

```
-- Procedure to Enroll a Student in a Course:
CREATE OR REPLACE PROCEDURE enroll_student_in_course(
  student_id IN NUMBER,
  course_id IN NUMBER,
  enroll_year IN NUMBER,
  enroll_semester IN VARCHAR2
)
AS
BEGIN
  INSERT INTO StudentEnrollCourses(StudentID, CourseID, Year, Semester)
  VALUES (student_id, course_id, enroll_year, enroll_semester);
  COMMIT;
END;

--------------------------------------------------------------------------------


--Function to Calculate GPA for a Student:
CREATE OR REPLACE FUNCTION calculate_student_gpa(student_id IN NUMBER)
RETURN NUMBER
AS
  total_credits NUMBER := 0;                    total_points NUMBER := 0;
BEGIN
  FOR course_record IN (SELECT * FROM StudentEnrollCourses WHERE StudentID =
student_id)
  LOOP
    total_credits := total_credits + (SELECT CourseCreditHours FROM courses WHERE
CourseID = course_record.CourseID);
```

```
    total_points := total_points + COALESCE(course_record.Grade, 0) * (SELECT
CourseCreditHours FROM courses WHERE CourseID = course_record.CourseID);
  END LOOP;
  IF total_credits = 0 THEN
    RETURN NULL; -- To avoid division by zero
  END IF;
  RETURN total_points / total_credits;
END;

--------------------------------------------------------------------------------

--Trigger to Update Course Credit Hours when a New Course is Inserted:
CREATE OR REPLACE TRIGGER update_credit_hours
BEFORE INSERT ON courses
FOR EACH ROW
BEGIN
  -- Assuming there is a constant for default credit hours, e.g., 3
  IF :NEW.CourseCreditHours IS NULL THEN
    :NEW.CourseCreditHours := 3;
  END IF;
END;
```

Bash Script for Database Manipulation

Data Base backups

```
db_owner= Karam
pass=159357
db=XE
direct="E:/DM44-Alex/Backups"
d=$(date +"%Y%m%d_%H%M%S")
file_name="backup_${d}.dmp"

expdp ${db_owner}/${pass}@${db} DUMPFILE=${file_name}  DIRECTORY=my_directory;

if [ $? -eq 0 ]; then
    echo " Make Backup Successfully " >> "${direct}/backupLogs.log"
else
    echo " ERROR : Make Backup Failed " >> "${direct}/backupLogs.log"
fi
```

This script is designed to automate the process of creating backups for an Oracle database using the Data Pump utility (expdp). It sets several variables, such as the database owner, password, database name, backup directory, and current date and time, to construct a unique backup file name. The script then executes the expdp command, specifying the user credentials, dump file name, and the Oracle directory where the backup will be stored. It checks the exit status of the expdp command to

determine whether the backup was successful. If successful, it appends a success message to a log file; otherwise, it logs an error message.

Disk Space Monitoring

```
#!/bin/bash

log_file="E:/DM44-Alex/Backups/disk_logs.log"
th=55
d=$(date +"%Y-%m-%d %H:%M:%S")
used_space=$(df -h | awk 'NR==2 {print $6}' | sed 's/%//')

if [ "$used_space" -ge "$th" ]; then
    echo "Warning!! The Disk Space Exceeds $th% | Date: ${d}" >> "${log_file}"
else
    echo "The Disk Space Is Safe | Date: ${d}" >> "${log_file}"
fi
```

this script serves as a disk space monitoring tool, providing a warning in the log file if the disk space usage surpasses the specified threshold. It can be useful for proactive monitoring and maintenance of system resources.

Java Application & GUI Implementation

1. Data Transfer Object (DTO):

- Purpose: DTO is used to transfer data between layers of the application, especially between the business logic and the data access layers.

- Structure: DTOs are simple Java objects that typically contain fields for data and may have getters and setters.

- Immutability: DTOs are often designed to be immutable to ensure that once the data is set, it cannot be changed. This helps in maintaining consistency and avoiding unintended modifications.

2. Data Access Object (DAO):

- Purpose: DAO is responsible for abstracting and encapsulating the interaction with the database. It provides a set of methods for

performing CRUD (Create, Read, Update, Delete) operations on the database.

- Interface-based: DAOs are often defined as interfaces, allowing for multiple implementations (e.g., different database vendors) while maintaining a consistent API.

- Transaction Management: DAOs may handle transaction management to ensure data consistency and integrity when multiple operations need to be executed atomically.

3. Database:

- Type: The application interacts with a relational database (e.g., MySQL, PostgreSQL, Oracle) or a NoSQL database (e.g., MongoDB) depending on the requirements.

- Schema: The database schema defines the structure of the tables and relationships between them. It reflects the data model of the application.

- Connection Pooling: To enhance performance, connection pooling is often used to efficiently manage and reuse database connections.

Controllers

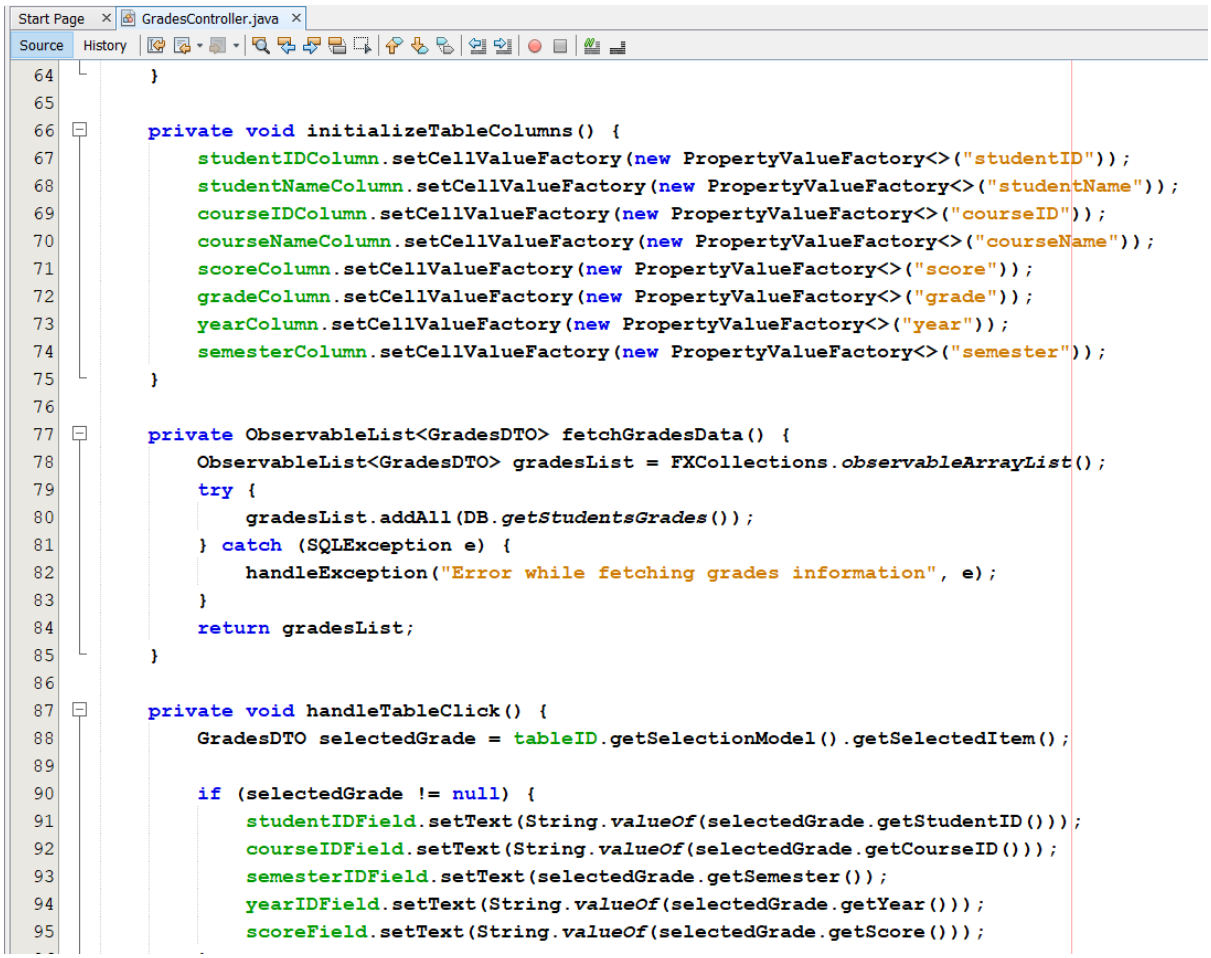
Purpose:

Controllers handle user input, interpret requests from the client (typically a web or API request), and invoke the appropriate business logic.

Structure:

Controllers are often organized into classes or packages based on the application's features or modules.

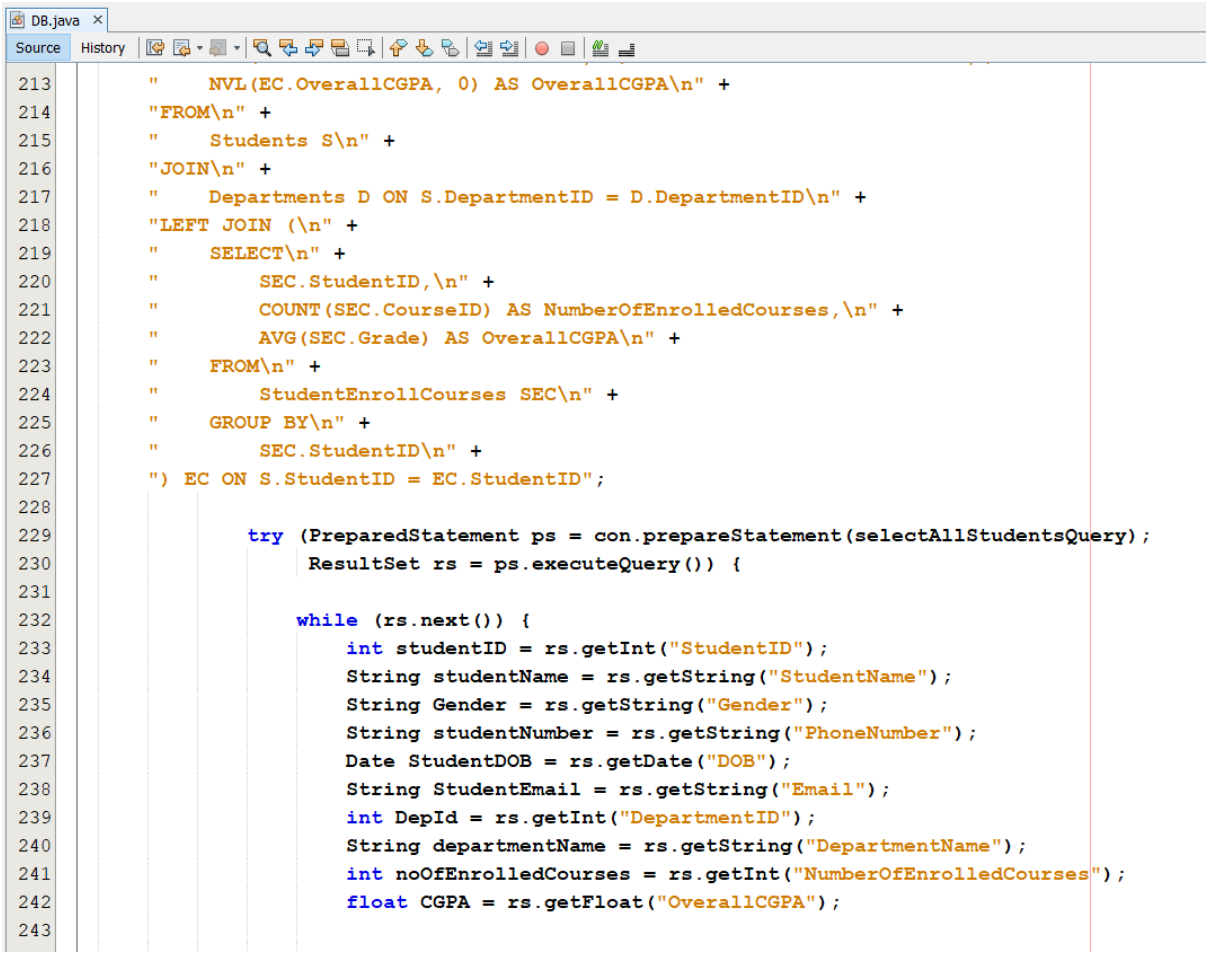
Each controller method corresponds to a specific endpoint or user action.


Start Page
GradesController.java
Source
History
64 }
65
66 private void initializeTableColumns() {
67 studentIDColumn.setCellValueFactory(new PropertyValueFactory<>("studentID"));
68 studentNameColumn.setCellValueFactory(new PropertyValueFactory<>("studentName"));
69 courseIDColumn.setCellValueFactory(new PropertyValueFactory<>("courseID"));
70 courseNameColumn.setCellValueFactory(new PropertyValueFactory<>("courseName"));
71 scoreColumn.setCellValueFactory(new PropertyValueFactory<>("score"));
72 gradeColumn.setCellValueFactory(new PropertyValueFactory<>("grade"));
73 yearColumn.setCellValueFactory(new PropertyValueFactory<>("year"));
74 semesterColumn.setCellValueFactory(new PropertyValueFactory<>("semester"));
75 }
76
77 private ObservableList<GradesDTO> fetchGradesData() {
78 ObservableList<GradesDTO> gradesList = FXCollections.observableArrayList();
79 try {
80 gradesList.addAll(DB.getStudentsGrades());
81 } catch (SQLException e) {
82 handleException("Error while fetching grades information", e);
83 }
84 return gradesList;
85 }
86
87 private void handleTableClick() {
88 GradesDTO selectedGrade = tableID.getSelectionModel().getSelectedItem();
89
90 if (selectedGrade != null) {
91 studentIDField.setText(String.valueOf(selectedGrade.getStudentID()));
92 courseIDField.setText(String.valueOf(selectedGrade.getCourseID()));
93 semesterIDField.setText(selectedGrade.getSemester());
94 yearIDField.setText(String.valueOf(selectedGrade.getYear()));
95 scoreField.setText(String.valueOf(selectedGrade.getScore()));

GradesDTO.java

Source | History

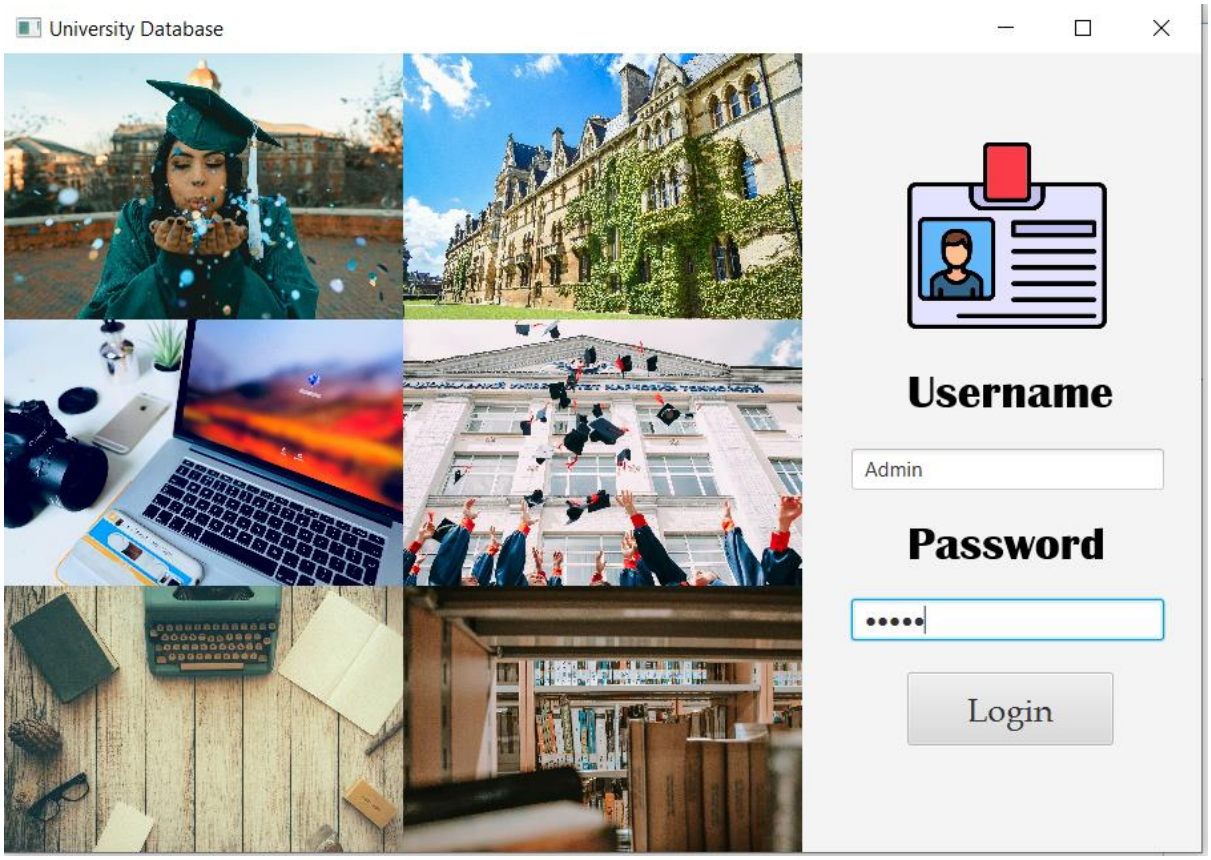
```
13  public class GradesDTO {
14      private int StudentID;
15      private String StudentName;
16      private int CourseID;
17      private String CourseName;
18      private int Score;
19      private String Grade;
20      private int Year;
21      private String Semester;
22
23      public GradesDTO(int StudentID, String StudentName, int CourseID, String CourseName,
24              int Score, String Grade,int Year, String Semester) {
25          this.StudentID = StudentID;
26          this.StudentName = StudentName;
27          this.CourseID = CourseID;
28          this.CourseName = CourseName;
29          this.Score = Score;
30          this.Grade = Grade;
31          this.Year = Year;
32          this.Semester = Semester;
33      }
34
35      public int getStudentID() {
36          return StudentID;
37      }
38
39      public void setStudentID(int StudentID) {
40          this.StudentID = StudentID;
41      }
42
43      public String getStudentName() {
```


DB.java
Source
History
213 "   NVL(EC.OverallCGPA, 0) AS OverallCGPA\n" +
214 "FROM\n" +
215 "   Students S\n" +
216 "JOIN\n" +
217 "   Departments D ON S.DepartmentID = D.DepartmentID\n" +
218 "LEFT JOIN (\n" +
219 "   SELECT\n" +
220 "   SEC.StudentID,\n" +
221 "   COUNT(SEC.CourseID) AS NumberOfEnrolledCourses,\n" +
222 "   AVG(SEC.Grade) AS OverallCGPA\n" +
223 "   FROM\n" +
224 "   StudentEnrollCourses SEC\n" +
225 "   GROUP BY\n" +
226 "   SEC.StudentID\n" +
227 ") EC ON S.StudentID = EC.StudentID";
228
229 try (PreparedStatement ps = con.prepareStatement(selectAllStudentsQuery);
230 ResultSet rs = ps.executeQuery()) {
231
232 while (rs.next()) {
233 int studentID = rs.getInt("StudentID");
234 String studentName = rs.getString("StudentName");
235 String Gender = rs.getString("Gender");
236 String studentNumber = rs.getString("PhoneNumber");
237 Date StudentDOB = rs.getDate("DOB");
238 String StudentEmail = rs.getString("Email");
239 int DepId = rs.getInt("DepartmentID");
240 String departmentName = rs.getString("DepartmentName");
241 int noOfEnrolledCourses = rs.getInt("NumberOfEnrolledCourses");
242 float CGPA = rs.getFloat("OverallCGPA");
243

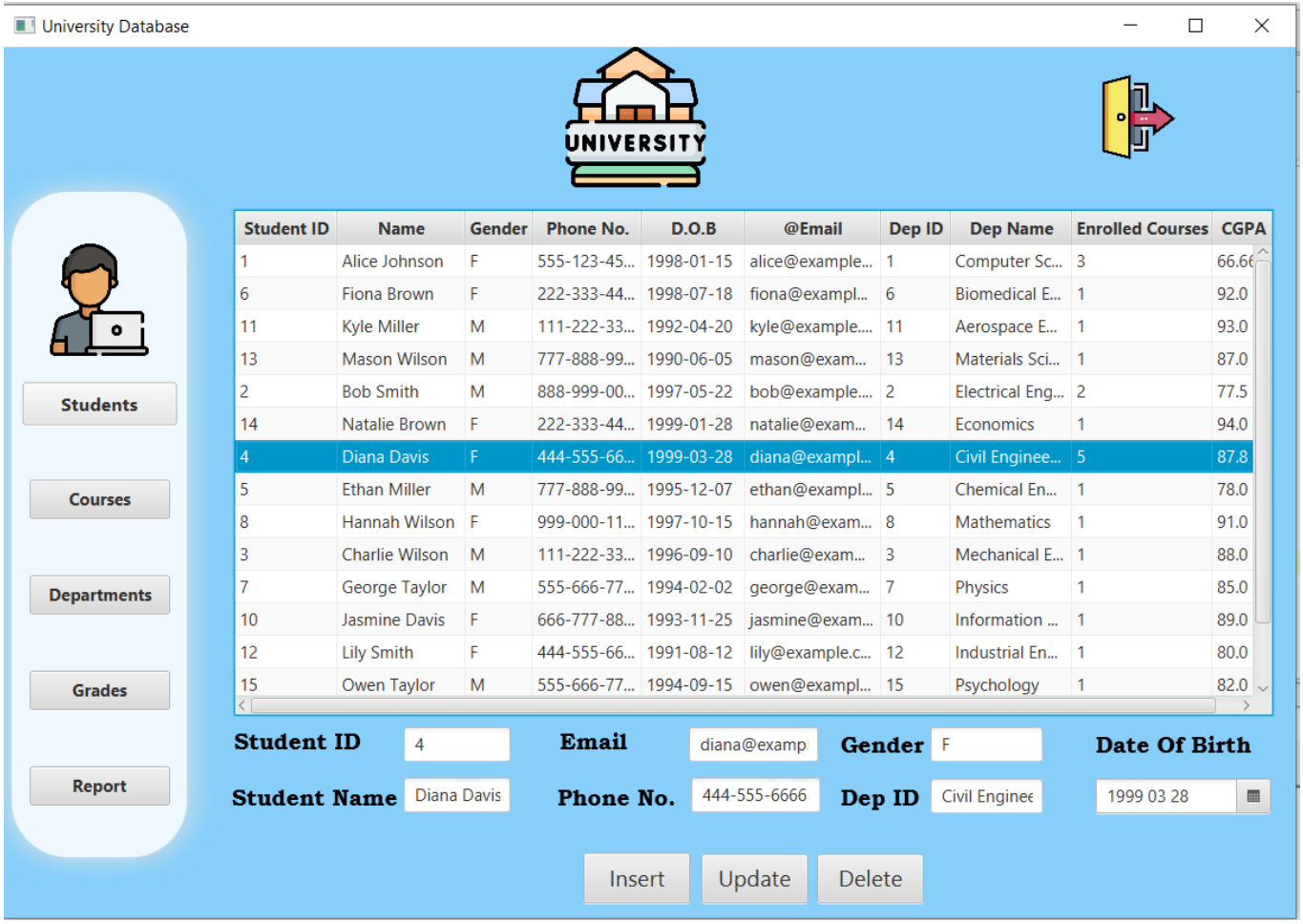
Login Scene

Username: Admin

Password: Admin

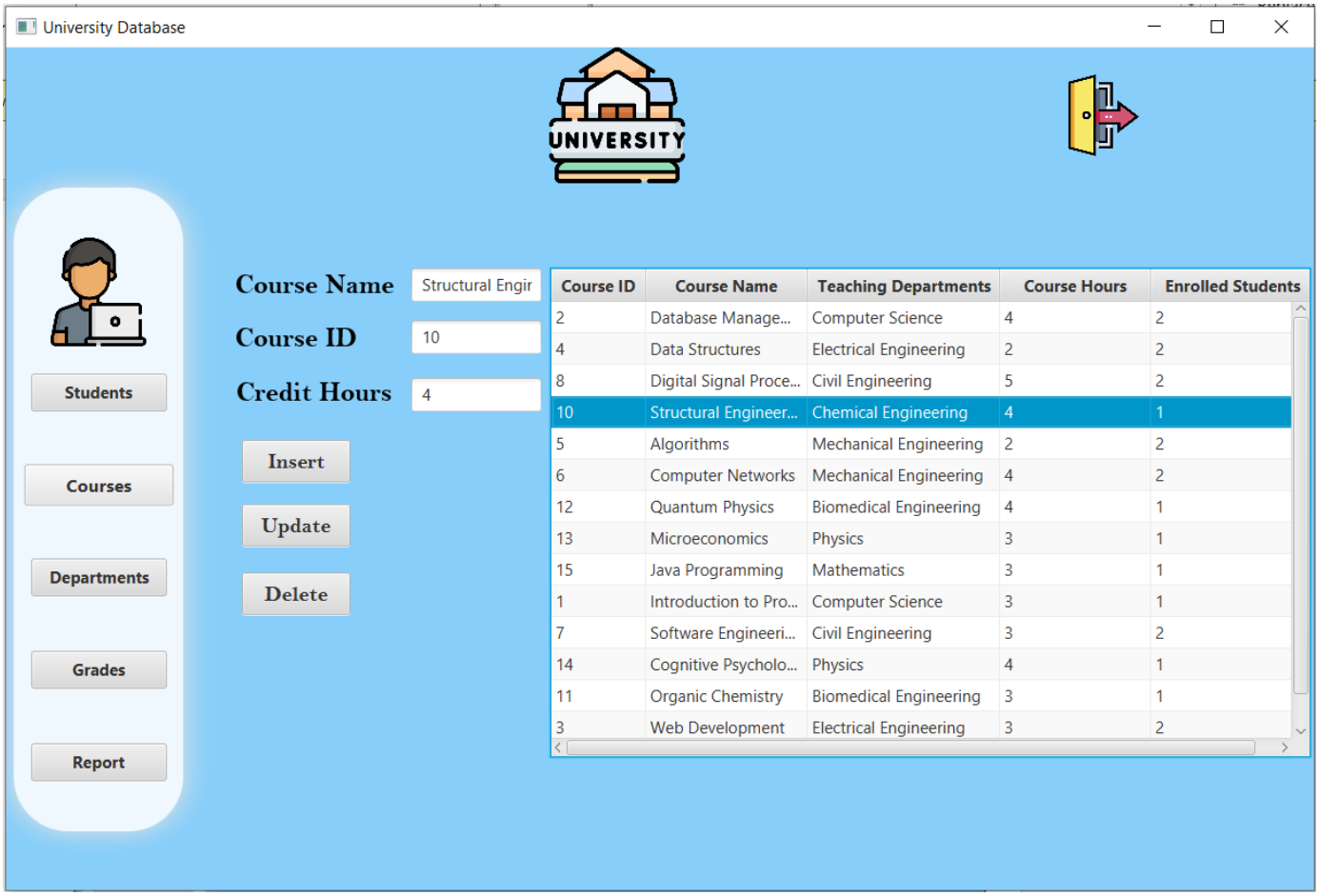


Student Scene



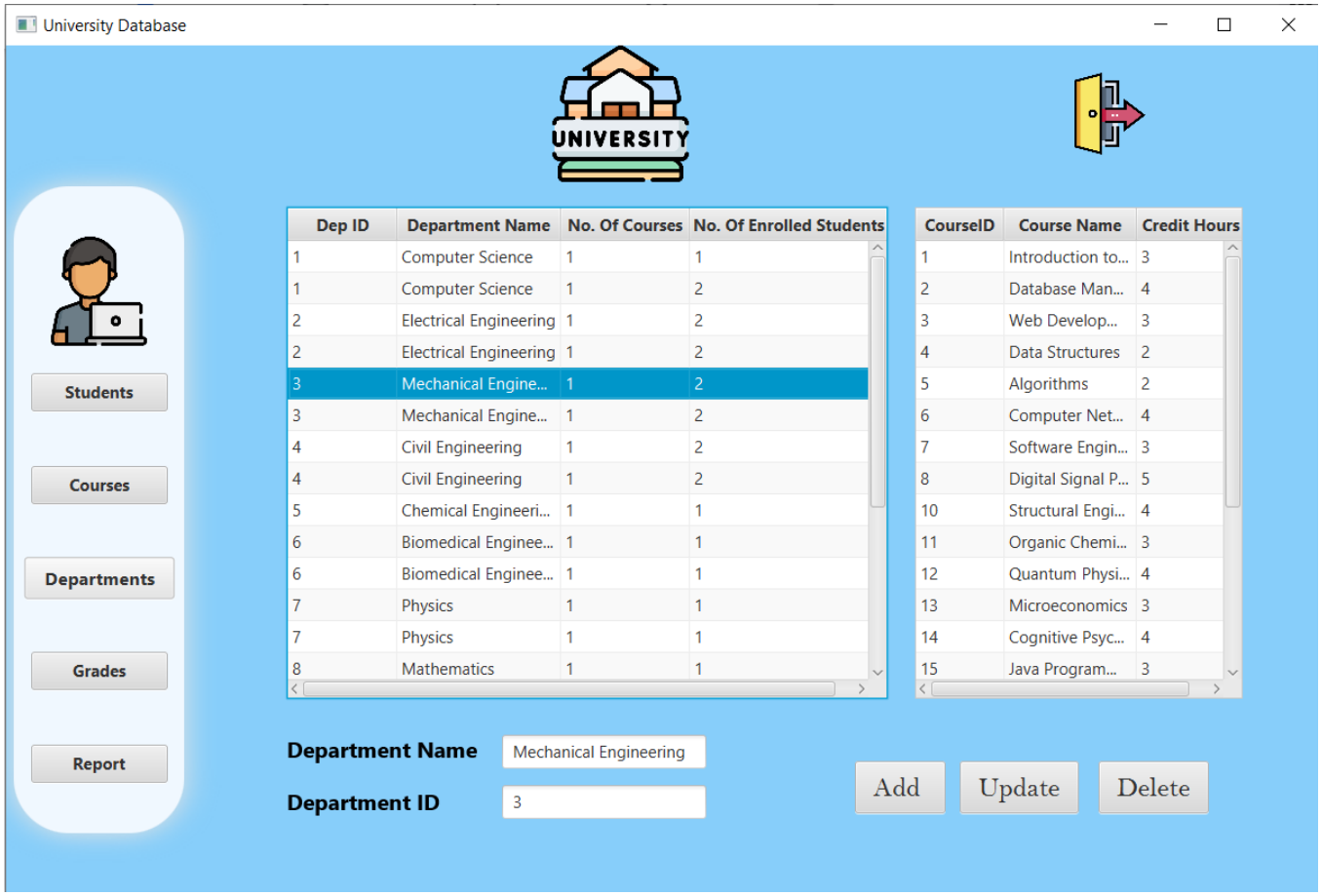
The User Can Insert, Update & Delete Students With all Information Related To them and Can See any Student Overall CGPA for all Semester's During the Last 4 Years. The Pane have Many Buttons to Guide the User to Check All Information He Needs and On Top Left the User Can See the Logout Button Also.

Courses Scene



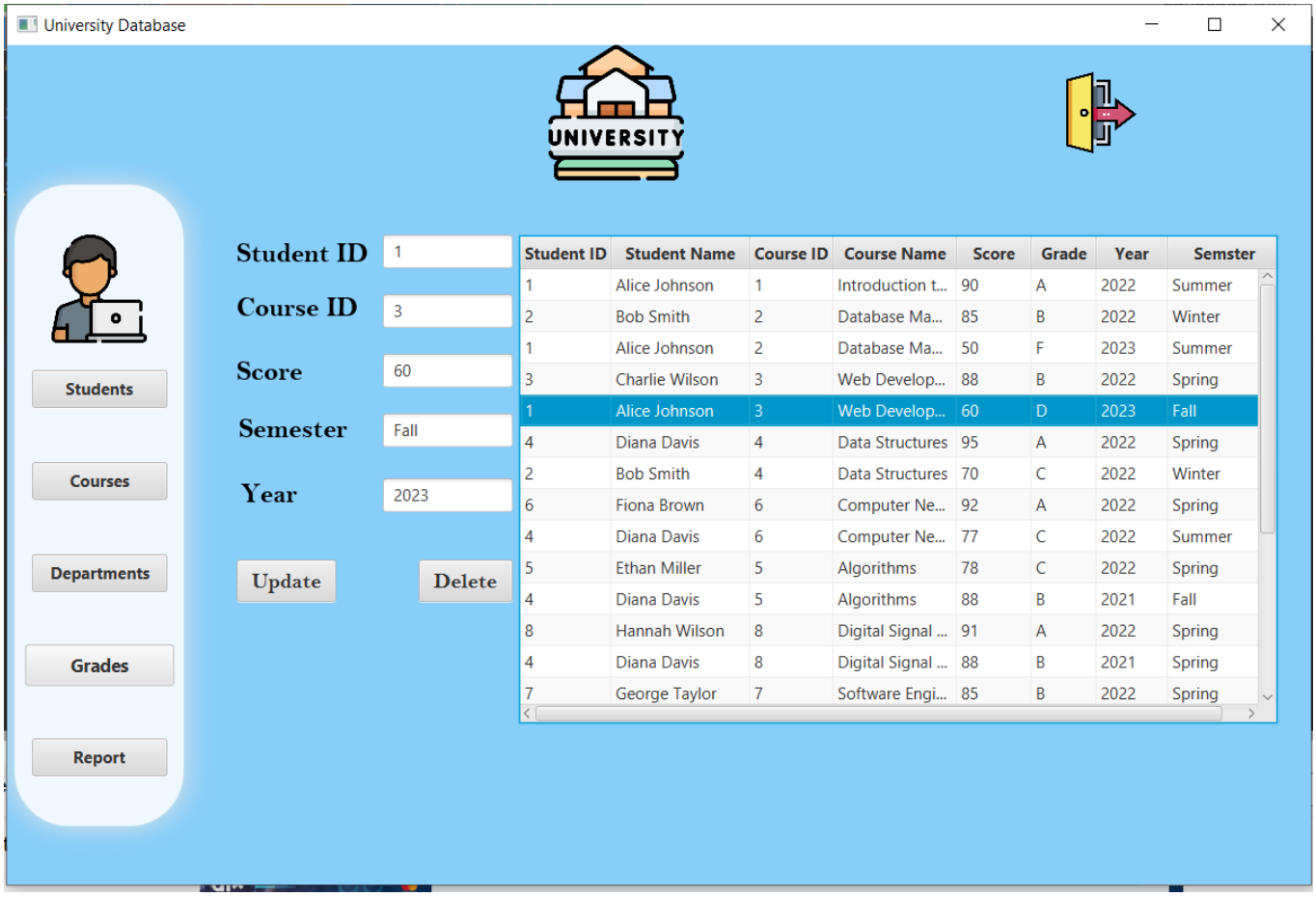
The User/ Admin Can See All the Courses in The Database with Their Teaching Departments and Course Credit Hours Also and The Number of Enrolled Students in The Course and Insert, Update and Delete any Course he Wants.

Departments Scene



The User/Admin Can Check All Departments Information from Number of Teaching Courses Up to The Number of Enrolled Students in The Department and The Courses That Department Teach and Also the Update, Delete and Insert New Departments.

Grades Scene



The Admin/ User Can Update and Delete Students Grades in A Course and Check all Grades Information when it comes to Score, year and the Semester that Grade have been issued.