Student Management System Documentation

By Ahmed Atef

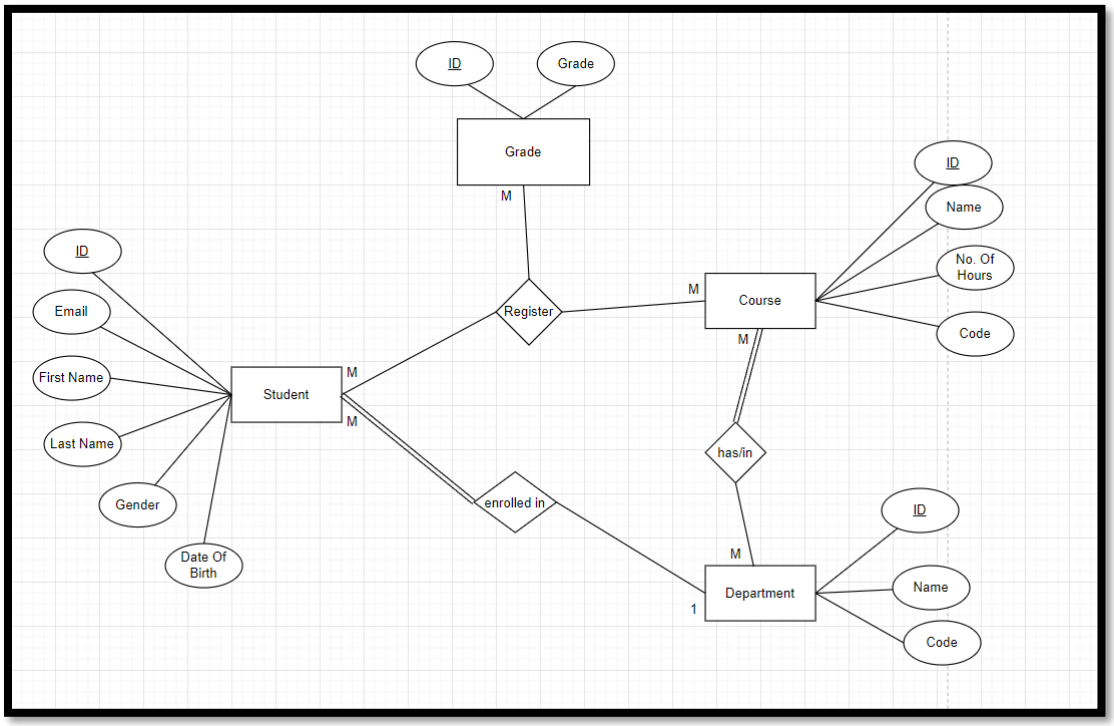
Student Management System Documentation

Introduction

The Student Management System is a Java-based application designed to manage information related to students, courses, grades, departments, and course assignments. The system provides a graphical user interface (GUI) to perform various tasks related to student management.

Database Design

ERD :



Database Schema :

1. **Department Table:**
    - **Purpose:** Stores information about university departments.
    - **Need:** Allows tracking and organizing various academic departments within the university.
    - **Columns**:
        - id (Primary Key, Auto Increment)
        - name (Not Null)
        - code
2. **Student Table:**
    - **Purpose:** Represents individual students in the university.
    - **Need:** Captures essential details such as name, email, gender, and department affiliation for each student.

- **Columns**:
  - id (Primary Key, Auto Increment)
  - fname (Not Null)
  - lname (Not Null)
  - email (Not Null, Unique)
  - gender (Enum: 'male', 'female', Not Null)
  - department_id (Foreign Key: department.id)
  - dob (Date)
- **Constraints**:
  - Unique Email Constraint

3. **Course Table:**
   - **Purpose:** Contains information about academic courses offered by the university.
   - **Need:** Helps in organizing and managing course details, including name, code, and duration (in hours).
   - **Columns**:
     - id (Primary Key, Auto Increment)
     - name (Not Null)
     - code (Not Null, Unique)
     - hours (Not Null, Check: hours IN (2, 3, 6))
   - Constraints:
     - Unique Code Constraint
4. **Student_Course Table:**
   - **Purpose:** Establishes a many-to-many relationship between students and courses.
   - **Need:** Enables tracking of which students are enrolled in which courses, creating an association between students and their selected courses.
   - **Columns**:
     - student_id (Foreign Key: student.id)
     - course_id (Foreign Key: course.id)
   - **Constraints**:
     - Primary Key (student_id, course_id)
     - Foreign Key (student_id) References student(id)
     - Foreign Key (course_id) References course(id) (On Delete Cascade)
5. **Department_Course Table:**
   - **Purpose:** Defines the relationship between departments and the courses they offer.
   - **Need:** Allows mapping of courses to specific departments, providing information on which departments are responsible for teaching certain courses.
   - **Columns**:
     - department_id (Foreign Key: department.id)

        - course_id (Foreign Key: course.id)
    - **Constraints**:
        - Primary Key (department_id, course_id)
        - Foreign Key (department_id) References department(id)
        - Foreign Key (course_id) References course(id) (On Delete Cascade)

6. **Grade Table:**
    - **Purpose:** Records grades achieved by students in specific courses.
    - **Need:** Facilitates the storage of student grades, associated with both the student and course through foreign key relationships.
    - **Columns**:
        - id (Primary Key, Auto Increment)
        - student_id (Foreign Key: student.id)
        - course_id (Foreign Key: course.id)
        - grade (Not Null, Check: grade >= 0 AND grade <= 100)
    - **Constraints**:
        - Primary Key (id)
        - Foreign Key (student_id, course_id) References student_course(student_id, course_id) (On Delete Cascade)

Applied Concepts:

- **Normalization:** The database design is normalized to avoid redundancy and ensure data integrity. For example, student information is in the "Student" table, and the relationship between students and courses is handled in the "Student_Course" table.
- **Referential Integrity:** Foreign key constraints are used to establish relationships between tables, ensuring referential integrity. For instance, the "Student_Course" table's foreign keys reference the "Student" and "Course" tables, maintaining consistency in data.
- **Efficient Querying:** The structure allows for efficient querying of data. For instance, you can easily retrieve information about students, courses, grades, and their relationships.
- **Flexibility:** The design accommodates changes or additions to data without significant modifications. It supports the dynamic nature of university-related data.
- **Data Integrity:** Constraints such as unique constraints and check constraints ensure that the data stored in the tables meets certain criteria, promoting data integrity.

SQL Implementation

```sql
CREATE DATABASE dm_case_study;
USE dm_case_study;
CREATE TABLE department (
  id INT PRIMARY KEY AUTO_INCREMENT NOT NULL,
  name VARCHAR(40) NOT NULL,
  code VARCHAR(10)
);
CREATE TABLE student (
  id INT PRIMARY KEY AUTO_INCREMENT NOT NULL,
  fname VARCHAR(40) NOT NULL,
  lname VARCHAR(40) NOT NULL,
  email VARCHAR(40) NOT NULL,
  gender ENUM('male', 'female') NOT NULL,
  department_id INT,
  dob DATE,
  FOREIGN KEY (department_id) REFERENCES department(id),
  CONSTRAINT unique_email_constraint UNIQUE (email)
);
CREATE TABLE course (
  id INT PRIMARY KEY AUTO_INCREMENT NOT NULL,
  name VARCHAR(40) NOT NULL,
  code VARCHAR(10) NOT NULL,
  hours INT CHECK (
    hours IN (2, 3, 6)
  ) NOT NULL,
  CONSTRAINT unique_code_constraint UNIQUE (code)
);
CREATE TABLE student_course (
  student_id INT NOT NULL,
  course_id INT NOT NULL,
  PRIMARY KEY (student_id, course_id),
  FOREIGN KEY (student_id) REFERENCES student(id),
  FOREIGN KEY (course_id) REFERENCES course(id) on delete cascade
);
CREATE TABLE department_course (
  department_id INT NOT NULL,
  course_id INT NOT NULL,
  PRIMARY KEY (department_id, course_id),
  FOREIGN KEY (department_id) REFERENCES department(id),
  FOREIGN KEY (course_id) REFERENCES course(id) on delete cascade
);
CREATE TABLE grade (
  id INT PRIMARY KEY AUTO_INCREMENT NOT NULL,
  student_id INT NOT NULL,
  course_id INT NOT NULL,
  grade INT NOT NULL CHECK (
    grade >= 0
    AND grade <= 100
  ),
  FOREIGN KEY (student_id, course_id) REFERENCES student_course(student_id,
course_id) on delete cascade
);
```

PLSQL Implementation

1. **Stored Procedure: update_student**
   - **Purpose:** Updates information for a specific student based on the provided parameters.
   - **Parameters:**
     - student_id (INT): ID of the student to be updated.
     - new_fname (VARCHAR(40)): New first name for the student.
     - new_lname (VARCHAR(40)): New last name for the student.
     - new_email (VARCHAR(40)): New email for the student.
     - new_gender (ENUM('male', 'female')): New gender for the student.
     - new_department_id (INT): New department ID for the student.
     - new_dob (DATE): New date of birth for the student.

```
DELIMITER //
CREATE PROCEDURE update_student(
    IN student_id INT,
    IN new_fname VARCHAR(40),
    IN new_lname VARCHAR(40),
    IN new_email VARCHAR(40),
    IN new_gender ENUM('male', 'female'),
    IN new_department_id INT,
    IN new_dob DATE
)
BEGIN
    UPDATE student
    SET fname = new_fname,
        lname = new_lname,
        email = new_email,
        gender = new_gender,
        department_id = new_department_id,
        dob = new_dob
    WHERE id = student_id;
END //
DELIMITER ;
```

2. **Function: calculate_student_gpa**
   - **Purpose:** Calculates the GPA for a specific student based on their grades in courses.
   - **Parameters:**
     - student_id (INT): ID of the student for whom GPA is calculated.
   - **Returns:** DECIMAL(5, 2) - The calculated GPA for the student.

```
DELIMITER //
CREATE FUNCTION calculate_student_gpa(student_id INT) RETURNS DECIMAL(5, 2) DETERMINISTIC
BEGIN
    DECLARE total_credits INT DEFAULT 0;
    DECLARE total_grade_points DECIMAL(5, 2) DEFAULT 0;
    DECLARE student_gpa DECIMAL(5, 2) DEFAULT 0;
    SELECT SUM(CASE
                    WHEN g.grade >= 90 THEN 4.0
                    WHEN g.grade >= 85 THEN 3.7
                    WHEN g.grade >= 80 THEN 3.3
                    WHEN g.grade >= 75 THEN 3.0
                    WHEN g.grade >= 70 THEN 2.7
                    WHEN g.grade >= 65 THEN 2.3
                    WHEN g.grade >= 60 THEN 2.0
                    WHEN g.grade >= 55 THEN 1.7
                    WHEN g.grade >= 50 THEN 1.3
                    ELSE 0.0
                END * c.hours),
            SUM(c.hours) INTO total_grade_points, total_credits
    FROM grade g
    JOIN course c ON g.course_id = c.id
    WHERE g.student_id = student_id;
    IF total_credits > 0 THEN
        SET student_gpa = total_grade_points / total_credits;
    END IF;
    RETURN student_gpa;
END //
DELIMITER ;
```

3. **Function: calculate_course_avg_gpa**
   - **Purpose:** Calculates the average GPA for a specific course based on the grades of enrolled students.
   - **Parameters:**
     - course_id (INT): ID of the course for which the average GPA is calculated.
   - **Returns:** DECIMAL(5, 2) - The calculated average GPA for the course.

```
DELIMITER //
CREATE FUNCTION calculate_course_avg_gpa(course_id INT)
RETURNS DECIMAL(5, 2) DETERMINISTIC
BEGIN
    DECLARE avg_gpa DECIMAL(5, 2) DEFAULT 0;

    SELECT AVG(CASE
                    WHEN g.grade >= 90 THEN 4.0
                    WHEN g.grade >= 85 THEN 3.7
                    WHEN g.grade >= 80 THEN 3.3
                    WHEN g.grade >= 75 THEN 3.0
                    WHEN g.grade >= 70 THEN 2.7
                    WHEN g.grade >= 65 THEN 2.3
                    WHEN g.grade >= 60 THEN 2.0
                    WHEN g.grade >= 55 THEN 1.7
                    WHEN g.grade >= 50 THEN 1.3
                    ELSE 0.0
                END) INTO avg_gpa
    FROM grade g
    WHERE g.course_id = course_id;

    RETURN avg_gpa;
END //
DELIMITER ;
```

4. **Function: calculate_level**
   - **Purpose:** Determines the level of a student based on the total hours of courses they have passed.
   - **Parameters:**
     - studentId (INT): ID of the student for whom the level is determined.
   - **Returns:** VARCHAR(10) - The calculated level for the student (e.g., 'Level 1', 'Level 2', etc.).

```
CREATE FUNCTION calculate_level(studentId INT)
RETURNS VARCHAR(10) DETERMINISTIC
BEGIN
    DECLARE totalHours INT;
    DECLARE studentLevel VARCHAR(10);
    SELECT SUM(c.hours) INTO totalHours
    FROM grade g
    JOIN course c ON g.course_id = c.id
    WHERE g.student_id = studentId
    and g.grade >= 50;
    IF totalHours < 10 THEN
        SET studentLevel = 'Level 1';
    ELSEIF totalHours < 20 THEN
        SET studentLevel = 'Level 2';
    ELSEIF totalHours < 30 THEN
        SET studentLevel = 'Level 3';
    ELSEIF totalHours < 40 THEN
        SET studentLevel = 'Level 4';
    END IF;
    RETURN studentLevel;
END //

DELIMITER ;
```

5. **Trigger: before_insert_grade**
   - **Purpose:** Prevents the insertion of a new grade if the student has already succeeded in the course (grade >= 50).
   - **Event:** Before each insertion into the "grade" table.
   - **Conditions:** Checks if a grade with a value greater than or equal to 50 already exists for the same student and course.
   - **Action:** Raises an error if the condition is met, preventing the new grade insertion.

```
DELIMITER //
CREATE TRIGGER before_insert_grade
BEFORE INSERT ON grade
FOR EACH ROW
BEGIN
    DECLARE existing_grade INT;

    SELECT grade INTO existing_grade
    FROM grade
    WHERE student_id = NEW.student_id AND course_id = NEW.course_id and grade >= 50 ;

    IF existing_grade IS NOT NULL THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Cannot insert a new grade. Student has already succeeded in the course.';
    END IF;
END;
//
DELIMITER ;
```

Automation Scripts

1. **Bash Script: Disk Space Monitoring**

   Script Overview:

   - **Purpose:** Monitors disk space usage and sends an alert if it exceeds a specified threshold.
   - **Threshold:** 40%
   - **Log Directory:** H:/iTi/Casestudy/Backup
   - **Log File:** disk_space_alert.log

```bash
#!/bin/bash

threshold=40
disk_usage=$(df -h / | awk 'NR==2 {print $6}' | tr -d '%' | cut -d'G' -f1)
echo $disk_usage

log_dir="H:/iTi/Casestudy/Backup"
log_file="$log_dir/disk_space_alert.log"

if [ "$disk_usage" -ge "$threshold" ]; then
    echo "Warning: Disk space usage is above $threshold%. Consider freeing up space." >> "$log_file"
else
    echo "Disk space usage is within acceptable limits." >> "$log_file"
fi
```

   Script Functionality:

   1. **Set Threshold:** The script sets the threshold for disk space usage to 40%.
   2. **Get Disk Usage:** Retrieves the current disk usage percentage for the root directory.
   3. **Log Directory and File:** Defines the log directory and log file path for recording alerts.
   4. **Check Threshold:** Compares the current disk usage with the defined threshold.
   5. **Generate Alert:** If the disk usage exceeds the threshold, it generates a warning message and appends it to the log file.
   6. **Log Within Limits:** If the disk usage is within acceptable limits, it records a message in the log file indicating normal conditions.

   Usage:

   - The script can be scheduled to run periodically using tools like cron.
   - It provides a log of disk space usage trends and alerts administrators when usage is high.

2. **Bash Script: MySQL Database Backup**

   Script Overview:

   - **Purpose**: Performs a backup of a MySQL database.
   - **Database Details**:
     - User: ahmed

- Password: 123
- Database Name: dm_case_study
- Backup Directory: ./ (Current directory)

- **Backup Filename Format**: dm_case_study-YYYYMMDDHHMMSS.sql

```
#!/bin/bash

DB_USER="ahmed"
DB_PASSWORD="123"
DB_NAME="dm_case_study"
BACKUP_DIR="./"
TIMESTAMP=$(date +%Y%m%d%H%M%S)
BACKUP_FILE="$BACKUP_DIR/$DB_NAME-$TIMESTAMP.sql"
MYSQL_DUMP_COMMAND="mysqldump -u$DB_USER -p'$DB_PASSWORD' $DB_NAME > $BACKUP_FILE"
eval $MYSQL_DUMP_COMMAND
if [ $? -eq 0 ]; then
    echo "Backup completed successfully. File: $BACKUP_FILE"
else
    echo "Error: Backup failed."
fi
```

Script Functionality:

- **Database Connection Details:** The script sets variables for the MySQL database user, password, and database name.
- **Backup Directory and Filename:** Defines the backup directory and generates a timestamped filename for the backup file in the format dm_case_study-YYYYMMDDHHMMSS.sql.
- **MySQL Dump Command:** Constructs the **mysqldump** command using the provided database details and backup filename.
- **Execute MySQL Dump:** Executes the MySQL dump command to create a backup of the specified database.
- **Check Exit Status:** Checks the exit status of the MySQL dump command. If successful (exit status 0), it displays a success message along with the backup file name. If unsuccessful, it displays an error message.

Usage:

- The script can be scheduled to run periodically using tools like cron.
- Ensure that the script has appropriate execution permissions.
- Modify the database connection details and backup directory as needed.

Student Management System Documentation

Java Application Development

Overview

The Student Management System allows users to perform operations such as adding, updating, and selecting information related to students, courses, grades, and departments. It also supports the assignment of courses to students.

Features :

- **Student Management:** Add, update, and view student information.
- **Course Management:** Add, update, and view course information.
- **Grade Management:** Add, update, and view student grades for courses.
- **Department Management:** Add, update, and view department information.
- **Course Assignment:** Assign courses to students.
- **Graphical User Interface:** Intuitive UI for easy navigation.
- **Charts:** Visual representation of student age, average grades per course, and average grades per department.

Code Structure :

The code is structured into various methods and functions, each responsible for specific tasks. The main sections include:

- **StudentScene:** Manages student-related functionalities.
- **CourseScene:** Manages course-related functionalities.
- **GradeScene:** Manages grade-related functionalities.
- **DepartmentScene:** Manages department-related functionalities.
- **MainController:** Controls the main functionality and GUI switching.
- **DatabaseAccessLayer:** Provides database connectivity and SQL operations.
- **Data Transfer Object (DTO) For Each Entity Showed In the app**

Functionality :

Student Management

- **Add Student:** Allows the addition of new student records with name, email, gender, and date of birth.
- **Update Student:** Enables updating existing student records.
- **View Students:** Displays a list of all students.

Course Management

- **Add Course:** Adds new courses with name, code, and duration.
- **Update Course:** Updates existing course information.
- **View Courses:** Displays a list of all courses.

Grade Management

- **Add Grade:** Records student grades for specific courses.
- **Update Grade:** Allows modification of existing grades.
- **View Grades:** Displays a list of all recorded grades.

Department Management

- **Add Department:** Adds new departments with a unique code and name.
- **Update Department:** Updates existing department information.
- **View Departments:** Displays a list of all departments.

Course Assignment

- **Assign Course:** Assigns courses to specific students.

Graphical User Interface :

The GUI consists of different forms for student, course, grade, department, and course assignment management. Users can navigate between these forms using the menu buttons.

Charts :

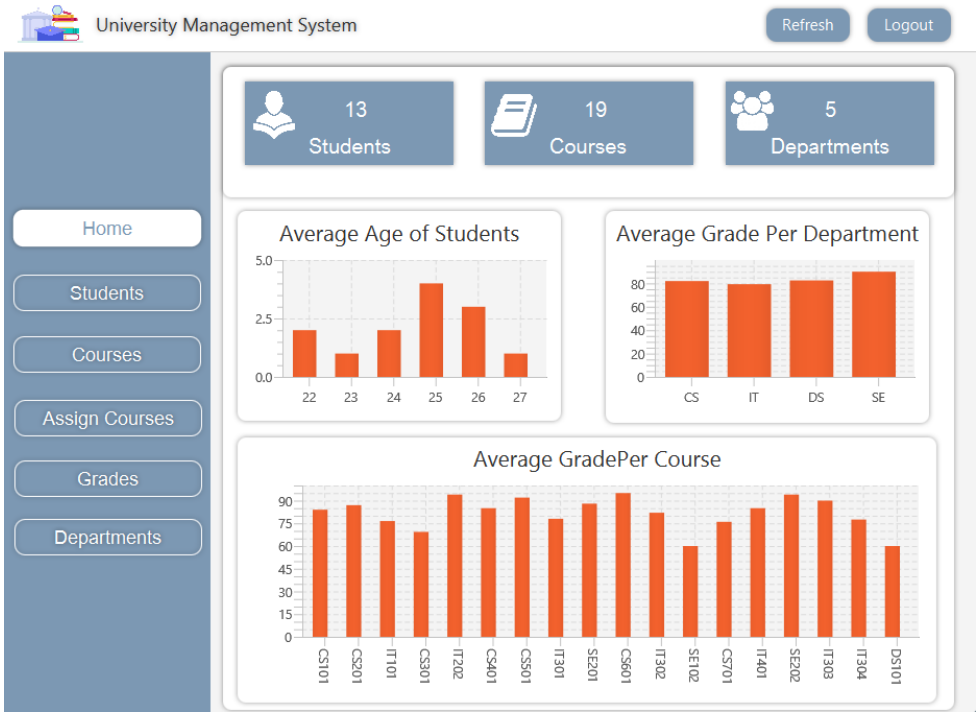
The application generates charts to visualize data:

- **Student Age Chart:** Displays the distribution of students based on their age.
- **Average Grade per Course Chart:** Shows the average grades for each course.
- **Average Grade per Department Chart:** Illustrates the average grades for each department.

Project Screenshots

Home Screen :

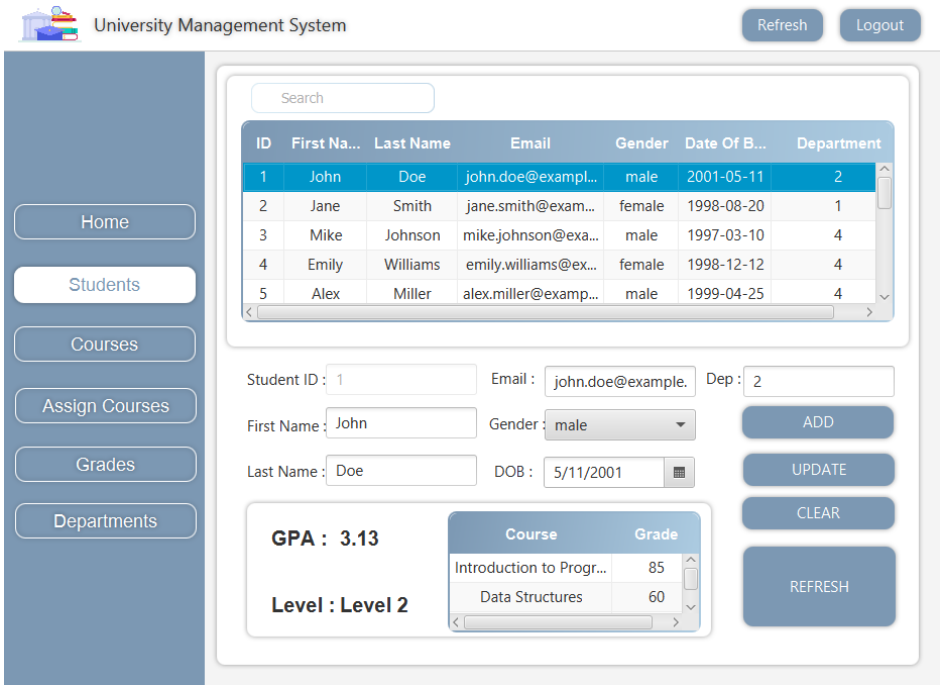
To view some reports about students age and grades per department or per course



You Can refresh to show the updated data

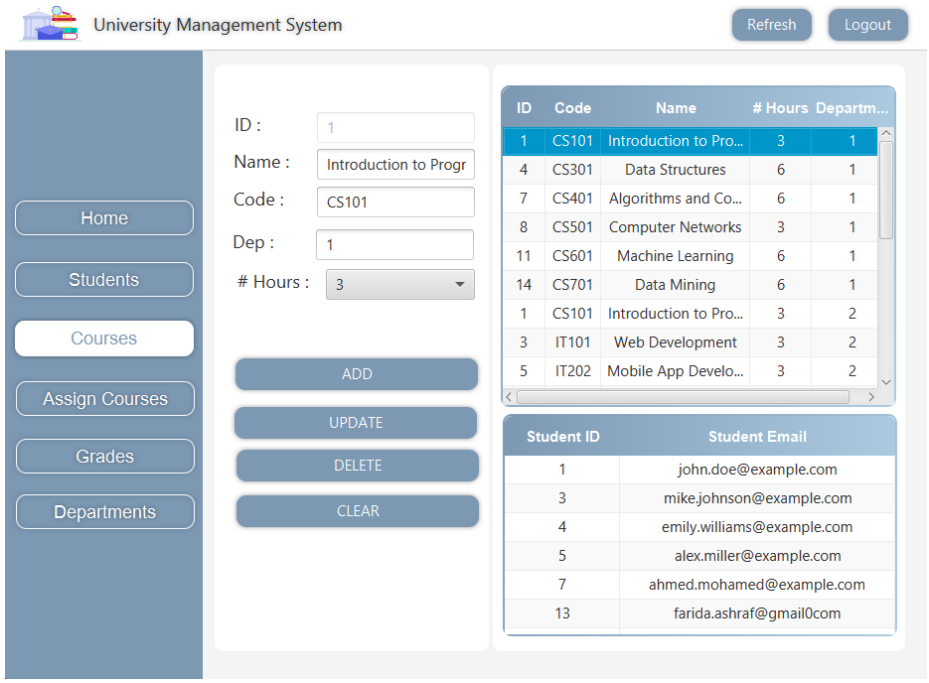
Students Screen :

To add a student , update a student's data and to view his gpa and level based on the courses he was graded in

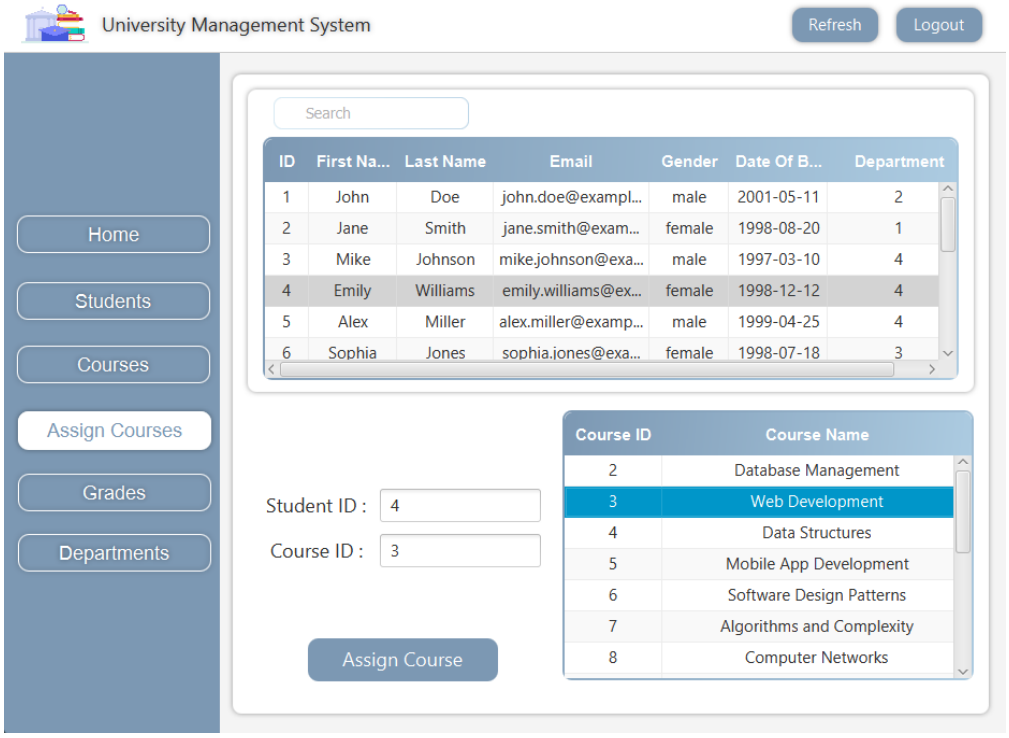


Courses Screen :

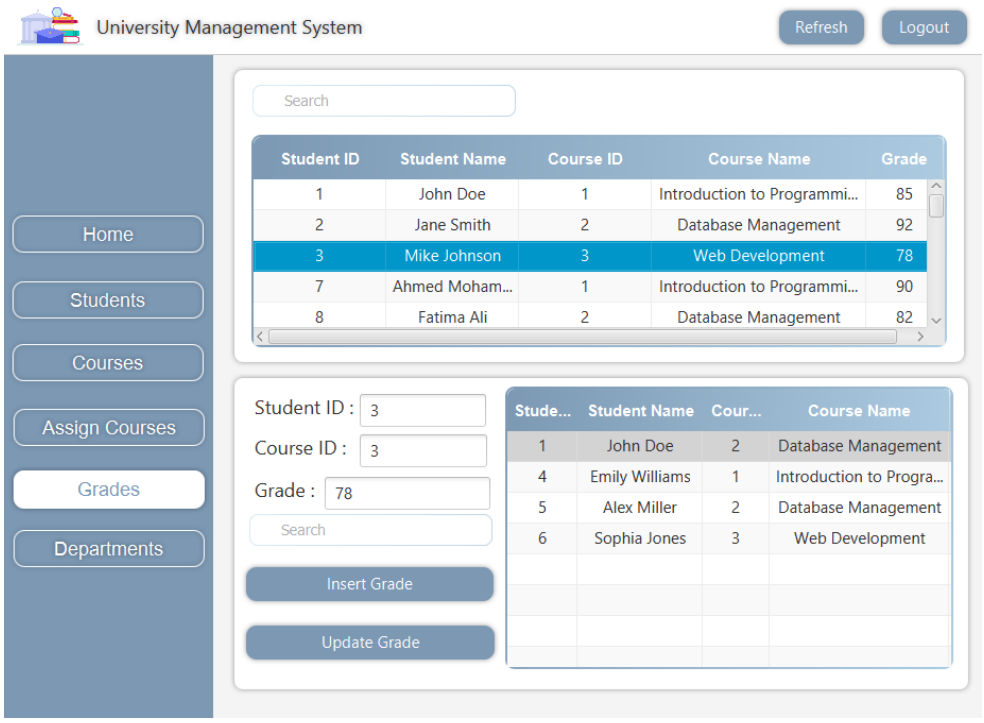
To view the available courses , add new course or update an existing course data , view the enrolled students in each course , delete a course



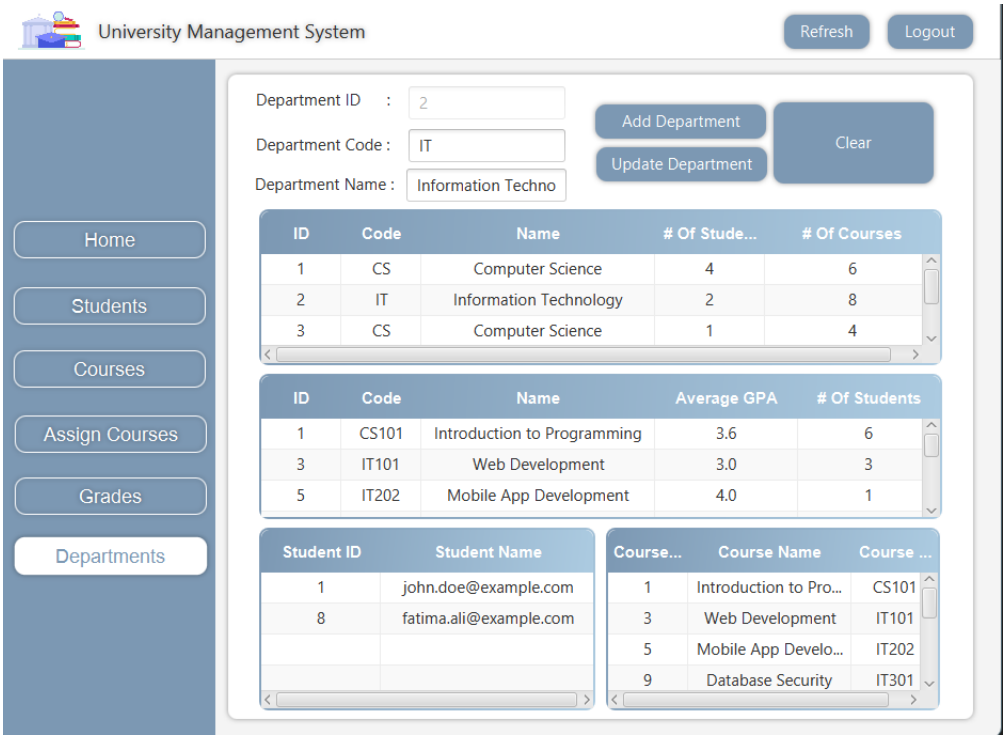
Assign Courses Screen : To assign a student to a course



Grades Screen : To assign a grade to a student enrolled course , to update a student grade ,To view the ungraded student courses



Departments Screen : to add department , update department , view department graded courses and the average gpa for each course , to view department's students and courses



Future Work

1. **Exception Handling Enhancement:**
    - Implement comprehensive exception handling mechanisms to provide detailed error messages to users.
    - Identify specific error scenarios (e.g., database connection issues, input validation failures) and create user-friendly error messages.
2. **Instructors for Courses:**
    - Integrate functionality to associate instructors with courses.
    - Allow the addition, update, and viewing of instructor information.
    - Associate instructors with specific courses to enhance course management.
3. **Course Hierarchy with Prerequisites:**
    - Implement a course hierarchy system that enforces prerequisites for each course.
    - Specify prerequisite courses for each course, ensuring students cannot enroll in a course without completing its prerequisites.
4. **Search Option in All Menus:**
    - Add a search functionality in all menus to facilitate quick and efficient data retrieval.
    - Implement a search bar or filter options to allow users to search for specific students, courses, grades, departments, and other relevant information.
5. **Enhanced User Authentication and Authorization:**
    - Strengthen user authentication mechanisms, potentially incorporating multi-factor authentication.
    - Implement role-based access control to restrict certain functionalities based on user roles (e.g., admin, instructor, student).
