Contents

[**Introduction** 2](#_Toc158558466)

[**Database Design** 2](#_Toc158558467)

[**Departments Table:** 3](#_Toc158558468)

[**Students Table:** 3](#_Toc158558469)

[**Courses Table:** 3](#_Toc158558470)

[**Relationships** 4](#_Toc158558471)

[**SQL Implementation** 5](#_Toc158558472)

[**PLSQL Implementation** 7](#_Toc158558473)

[**Automation Scripts** 11](#_Toc158558474)

[**Java Application Development** 12](#_Toc158558475)

[1. Overview 12](#_Toc158558476)

[2. Features : 12](#_Toc158558477)

[3. Code structure 12](#_Toc158558478)

[4. Functionality : 13](#_Toc158558479)

[**Snippets from Project** 14](#_Toc158558480)

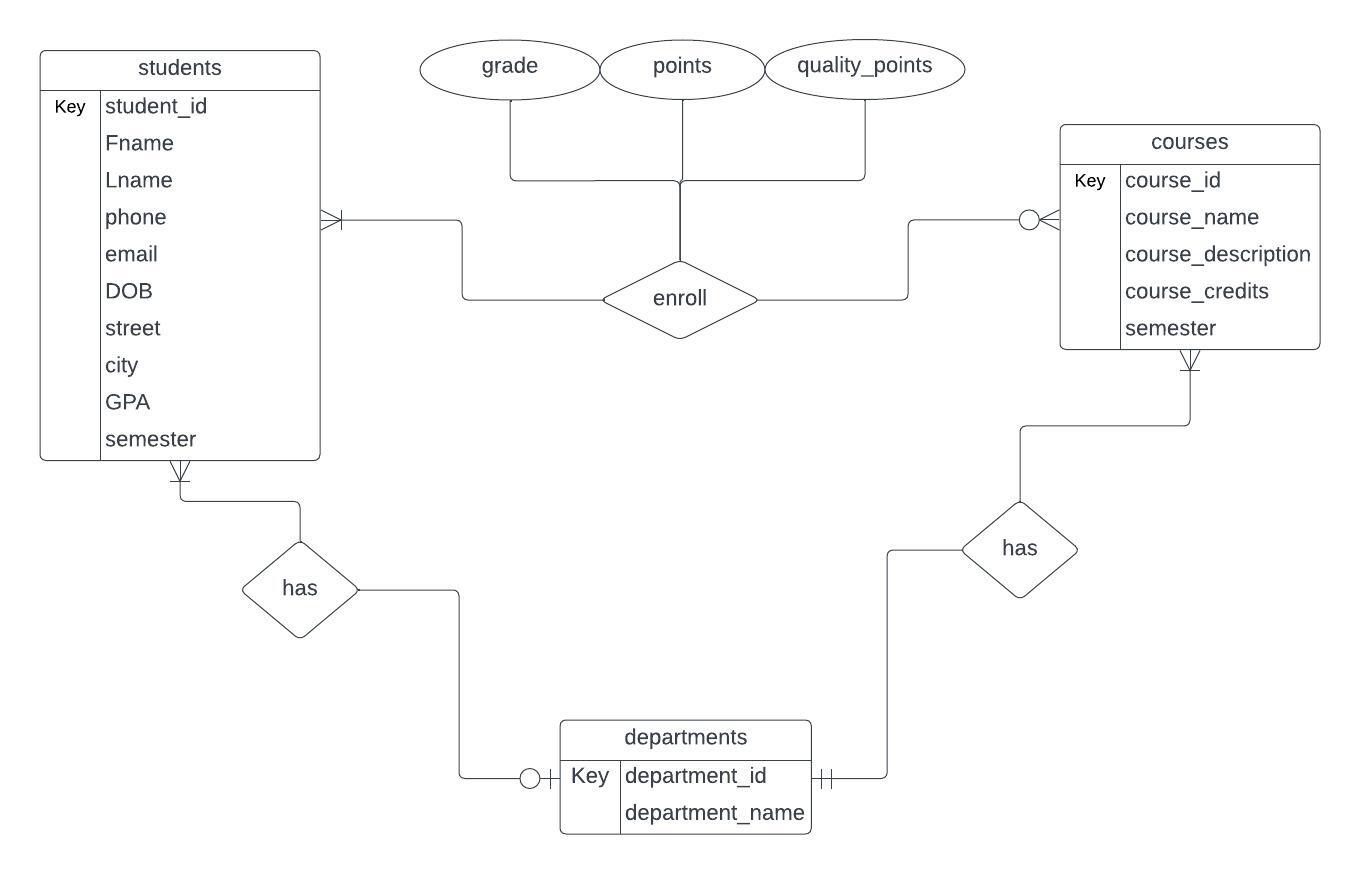
**By Hassan Hosny**

# **Introduction**

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

# **Database Design**

**ERD**



**Database schema**

**Mapping**

* departments(department\_id (pk), department\_name)
* students(studend\_id (pk), Fname, Lname, DOB, phone, street, city, gpa, department\_id (fk), semester)
* courses(course\_id (pk), course\_name, course\_description, department\_id (fk), course\_credits, semester)
* enrollment((studend\_id (fk), course\_id (fk))(pk), grade, points, quality\_points)

# **Departments Table:**

* Purpose: The Departments table serves to store information related to academic departments within an educational institution.
* Columns:
  + department\_id: A unique numerical identifier for each department (Primary Key).
  + department\_name: A non-null string representing the name of the department.
* Key Constraints:
  + department\_id is a primary key, ensuring each department has a unique identifier.

# **Students Table:**

* Purpose: The Students table stores comprehensive information about individual students.
* Columns:
  + student\_id: A unique numerical identifier for each student (Primary Key).
  + fname: First name of the student.
  + lname: Last name of the student.
  + phone: Contact phone number of the student.
  + email: Email address of the student.
  + DOB: Date of birth of the student.
  + street: Street address of the student.
  + city: City of residence of the student.
  + gpa: Grade Point Average of the student.
  + department\_id: Foreign key referencing the Departments table.
  + semester: Current semester of the student.
* Key Constraints:
  + student\_id is a primary key, ensuring each student has a unique identifier.
  + department\_id is a foreign key, linking students to their respective departments.

# **Courses Table:**

* Purpose: The Courses table manages information about different courses offered by the institution.
* Columns:
  + course\_id: A unique numerical identifier for each course (Primary Key).
  + course\_name: The name/title of the course.
  + course\_description: A detailed description of the course.
  + department\_id: Foreign key referencing the Departments table, indicating the department offering the course.
  + course\_credits: The number of credits assigned to the course.
  + semester: The semester in which the course is offered.
* Key Constraints:
  + course\_id is a primary key, ensuring each course has a unique identifier.
  + department\_id is a foreign key, linking courses to their respective departments.

# **Relationships**

* Student:
  + Has one: Department (a student belongs to one department)
  + Enrolls in: Many Courses (a student can enroll in many courses)
* Course:
  + Offered by: One Department (a course is offered by one department)
  + Has: Many Students (a course can have many students enrolled)
* Department:
  + Offers: Many Courses (a department can offer many courses)
  + Has: Many Students (a department can have many students enrolled in its courses)
* Additional Notes:
* The ERD also shows that each entity has a primary key attribute:
  + Student: student\_id
  + Course: course\_id
  + Department: department\_id
* The "enrolls" relationship between Student and Course likely has additional attributes to capture details such as the semester and grade earned.

# **SQL Implementation**

**Departments Table:**

* Columns:
  + department\_id (NUMBER, PRIMARY KEY): Unique identifier for each department.
  + department\_name (VARCHAR(255), NOT NULL): Name of the department.
* Constraints:
  + Primary Key: Ensures uniqueness of department\_id.
  + NOT NULL: Ensures department\_name is always provided.

**Students Table:**

* Columns:
  + student\_id (NUMBER, PRIMARY KEY): Unique identifier for each student.
  + fname (VARCHAR(255), NOT NULL): First name of the student.
  + lname (VARCHAR(255), NOT NULL): Last name of the student.
  + phone (VARCHAR(20)): Phone number of the student.
  + email (VARCHAR(255)): Email address of the student.
  + DOB (DATE): Date of birth of the student.
  + street (VARCHAR(255)): Street address of the student.
  + city (VARCHAR(255)): City of residence of the student.
  + gpa (DECIMAL(5,2), DEFAULT 0): Grade Point Average of the student with a default value of 0.
  + department\_id (NUMBER, FOREIGN KEY): References the department\_id in the Departments table.
  + semester (NUMBER(1), NOT NULL): Semester in which the student is enrolled.
* Constraints:
  + Primary Key: Ensures uniqueness of student\_id.
  + Foreign Key: Ensures that department\_id references a valid department.

**Courses Table:**

* Columns:
  + course\_id (NUMBER, PRIMARY KEY): Unique identifier for each course.
  + course\_name (VARCHAR(255), NOT NULL): Name of the course.
  + course\_description (VARCHAR(255)): Description of the course.
  + department\_id (NUMBER, FOREIGN KEY): References the department\_id in the Departments table.
  + course\_credits (NUMBER, NOT NULL): Number of credits assigned to the course.
  + semester (NUMBER(1), NOT NULL): Semester in which the course is offered.
* Constraints:
  + Primary Key: Ensures uniqueness of course\_id.
  + Foreign Key: Ensures that department\_id references a valid department.

**Enroll Table:**

* Columns:
  + student\_id (NUMBER, FOREIGN KEY): References the student\_id in the Students table.
  + course\_id (NUMBER, FOREIGN KEY): References the course\_id in the Courses table.
  + grade (VARCHAR(10), DEFAULT 0): Grade obtained by the student in the course with a default value of 0.
  + points (NUMBER(5,2), DEFAULT 0): Numeric points corresponding to the grade, with a default value of 0.
  + quality\_points (DECIMAL(5,2), DEFAULT 0): Quality points calculated based on the grade and course, with a default value of 0.
* Constraints:
  + Primary Key: Composite key using student\_id and course\_id.
  + Foreign Keys: Ensure that student\_id and course\_id reference valid records in the Students and Courses tables, respectively.

# **PLSQL Implementation**

**CALCULATE\_POINTS\_QUALITY TRIGGER**

* Purpose:
  + Automated Grade Calculation: The trigger automates the calculation of the points and quality\_points fields based on the grade assigned to a student in a particular course.
* Key Components:
* Grade Points Lookup: The trigger looks up the corresponding grade points for a given grade using a CASE statement.
* Maximum GPA: The maximum GPA value is set to 4.00, and the trigger ensures that
* calculated GPA values do not exceed this maximum.
* Operation:
  + Context: The trigger operates in the context of an ENROLL table insert or update operation.
  + Input Data: The NEW pseudorecord represents the new data being inserted or updated in the ENROLL table.
* Detailed Steps:
  + Fetch Course Credits: The trigger fetches the course\_credits for the given course\_id from the COURSES table.
  + Grade Points Calculation: Based on the provided grade, the trigger calculates the corresponding grade points.
  + Calculate GPA: The trigger calculates the GPA by taking the minimum of the calculated grade points and the maximum GPA value.
  + Calculate Quality Points: The trigger calculates the quality points by multiplying the GPA with the course\_credits.
  + Update ENROLL Table: The trigger updates the points and quality\_points fields in the ENROLL table with the calculated values.
* Use Cases:
  + Automated GPA Calculation: This trigger is useful for ensuring that GPA calculations are consistently and automatically applied whenever new enrollment data is added or existing data is updated.
  + Consistency: By centralizing the grade point calculation logic in a trigger, it helps maintain consistency and avoids manual errors in GPA calculations during enrollment.
* Example:
  + Assuming a new enrollment record is inserted into the ENROLL table with a grade of 'A' for a course with 3 credits, the trigger will calculate the GPA as 4.00 and update the points and quality\_points fields accordingly.

**UPDATE\_GPA\_TRIGGER**

* Trigger Type:
  + Compound Trigger: This trigger is a compound trigger, combining multiple timing points (BEFORE and AFTER) and sections (BEFORE STATEMENT, BEFORE EACH ROW, and AFTER STATEMENT) into a single structure.
* Timing Points:
  + BEFORE STATEMENT: Executed once before the entire SQL statement.
  + BEFORE EACH ROW: Executed once for each affected row.
  + AFTER STATEMENT: Executed once after the entire SQL statement.
* Trigger Components:
  + Collection (enroll\_tab): A collection is used to store relevant data for each row affected by the trigger, specifically student\_id, quality\_points, and course\_credits.
  + BEFORE STATEMENT Section: Empties the collection before the execution of the SQL statement.
  + BEFORE EACH ROW Section: Populates the collection with data for each row being inserted, updated, or deleted.
  + AFTER STATEMENT Section: Processes the data in the collection after the SQL statement is executed.
* Processing Logic:
  + Data Collection: Before each row is processed, relevant data (student\_id, quality\_points, course\_credits) is collected in the collection (enroll\_tab).
  + AFTER STATEMENT Processing: After the SQL statement (insert, update, or delete on ENROLL) is executed, the trigger calculates the total quality points and total credits for each student affected by the changes.
  + GPA Calculation: Using the formula total\_quality\_points / total\_credits, the trigger calculates the GPA for each student. It avoids division by zero and sets GPA to zero if total credits are zero.
  + GPA Update: The calculated GPA is then updated in the STUDENTS table for the corresponding student.
* Purpose and Use:
  + Dynamic GPA Update: The trigger ensures that whenever enrollments are inserted, updated, or deleted, the GPA of affected students is dynamically recalculated and updated in the STUDENTS table.
  + Consistency: By centralizing GPA calculation logic in a trigger, consistency is maintained, and manual errors in GPA updates are avoided.
  + Efficiency: The trigger processes multiple rows in a single SQL statement, improving efficiency compared to row-by-row processing.

**ON\_COURSE\_DELETE TRIGGER**

* Trigger Type:
  + BEFORE DELETE Trigger: This trigger is executed before a row is deleted from the COURSES table.
* Trigger Logic:
  + Trigger Body (BEGIN to END): Contains the logic to be executed when a course is deleted.
  + DELETE Statement: Deletes records from the ENROLL table where the course\_id matches the :OLD.course\_id (the course\_id value of the course being deleted).
* Purpose and Use:
  + Cascade Deletion: The trigger enforces a cascading effect, ensuring that when a course is deleted, all associated enrollments in the ENROLL table are also deleted automatically.
* Benefits:
  + Automated Maintenance: The trigger automates the process of handling related records when a course is deleted, reducing the need for manual intervention and potential errors.
  + Consistency: Ensures that the data remains consistent between the COURSES and ENROLL tables by keeping enrollments in sync with course deletions.

**ON\_DEPARTMENT\_DELETE TRIGGER**

* Trigger Type:
  + BEFORE DELETE Trigger: This trigger is executed before a row is deleted from the DEPARTMENTS table.
* Trigger Logic:
  + Trigger Body (BEGIN to END): Contains the logic to be executed when a department is deleted.
* DELETE Statements:
  + Deletes records from the COURSES table where the department\_id matches the :OLD.department\_id (the department\_id value of the department being deleted).
  + Deletes records from the STUDENTS table where the department\_id matches the :OLD.department\_id.
* Purpose and Use:
  + Cascade Deletion: The trigger enforces a cascading effect, ensuring that when a department is deleted, all associated courses and students in the COURSES and STUDENTS tables are also deleted automatically.
* Benefits:
  + Automated Maintenance: The trigger automates the process of handling related records when a department is deleted, reducing the need for manual intervention and potential errors.
  + Consistency: Ensures that the data remains consistent between the DEPARTMENTS, COURSES, and STUDENTS tables by keeping courses and students in sync with department deletions.

**ON\_STUDENT\_DELETE TRIGGER**

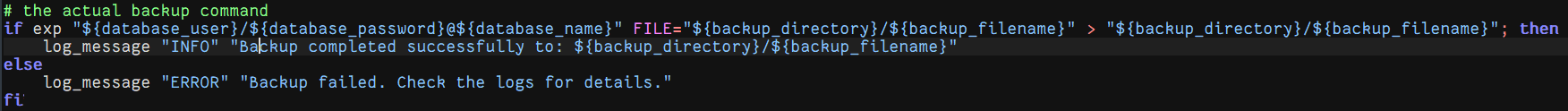
* Trigger Type:
  + BEFORE DELETE Trigger: This trigger is executed before a row is deleted from the STUDENTS table.
* Trigger Logic:
  + Trigger Body (BEGIN to END): Contains the logic to be executed when a student is deleted.
  + DELETE Statement: Deletes records from the ENROLL table where the student\_id matches the :OLD.student\_id (the student\_id value of the student being deleted).
* Purpose and Use:
  + Cascade Deletion: The trigger enforces a cascading effect, ensuring that when a student is deleted, all associated enrollments in the ENROLL table are also deleted automatically.
* Benefits:
* Automated Maintenance: The trigger automates the process of handling related records when a student is deleted, reducing the need for manual intervention and potential errors.
* Consistency: Ensures that the data remains consistent between the STUDENTS and ENROLL tables by keeping enrollments in sync with student deletions.

# **Automation Scripts**

**Database Backup:**

* Sets up database credentials and backup directory.
* Creates logs directory and logs file.
* Generates a timestamp and constructs a backup filename.
* Uses exp command to backup the database to a SQL file.
* Logs the backup status (success/failure) with a timestamp.

A computer screen shot of text

Description automatically generated 

**Disk Space Monitoring:**

* Sets a disk usage threshold.
* Extracts disk usage information using df -h.
* Compares disk usage with the threshold.
* Displays a warning if disk usage exceeds the threshold.
* Logs disk space status with a timestamp.

A screenshot of a computer

Description automatically generated

A screenshot of a computer program

Description automatically generated

**Task scheduling:**

* + - **Command to create a task on windows**



* + - **And here is the task created successfully**

A screenshot of a computer

Description automatically generated

# **Java Application Development**

# Overview

The Student Management System facilitates users in executing various operations, including adding, selecting, and managing information pertaining to students, courses, grades, and departments. Additionally, it supports the allocation of courses to students.

# Features :

* Student: Add and view student information.
* Course: Add, delete and view course information.
* Grade: Add and view student grades for courses.
* Department: Add, delete, and view department information.
* Course Assignment: Assign courses to students.
* Graphical User Interface(GUI): Intuitive UI for easy navigation.

# Code structure

* A screenshot of a computer program

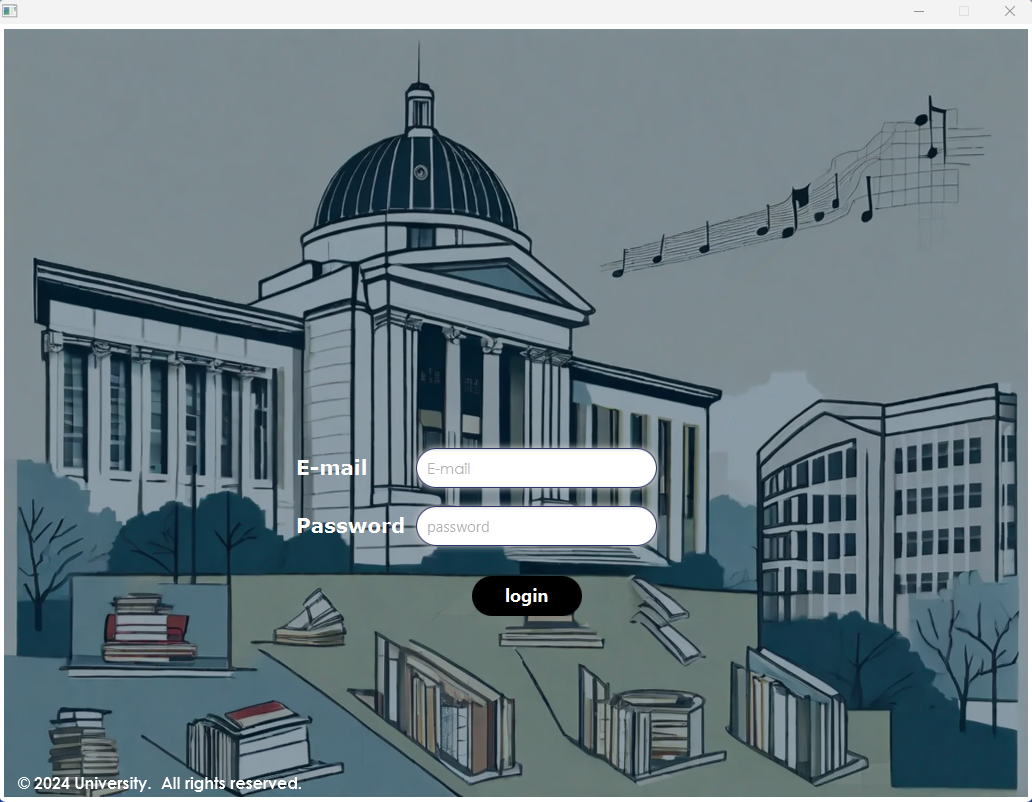
  Description automatically generatedData Access Layer (DAO):
  + The Data Access Layer (DAO) is like the bridge between our software and the database. It handles all the behind-the-scenes work, such as connecting to the database and running queries to get the information we need. This keeps our code clean and organized, making it easier to manage and maintain.
* Data transfer object (DTO):
  + A Data Transfer Object (DTO) is like a courier for our data. It efficiently moves information between different parts of our software, maintaining a tidy and organized flow of data across the application.
* Scenes and controllers
  + Scenes:
    - departments.fxml
    - departmentsCourses.fxml
    - enroll.fxml
    - login.fxml
    - main.fxml
    - report.fxml
    - students.fxml
  + Controllers:
    - DepartmentsCoursesController: responsible for handling interactions with the departments and courses data.
    - EnrollController: responsible for handling student enrollment.
    - LoginController: responsible for handling user login.
    - MainController: the main controller for the application.
    - ReportController: responsible for generating reports.
    - StudentsController: responsible for handling student data.

# Functionality :

* Student:
  + Add student: Allows the addition of new student records with all informations.
  + View students: Displays a list of all students.
  + Delete students: Removes selected student records from the system.
* Course:
  + Add Course: Adds new courses with name, code, and duration.
  + View Courses: Displays a list of all courses.
  + Delete courses: Eliminates specific courses, including their name, code, and duration.
* Grade:
  + Add Grade: Records student grades for specific courses.
  + View Grades: Displays a list of all recorded grades.
* Department:
  + Add Department: Adds new departments with a unique code and name.
  + View Departments: Displays a list of all departments.
  + Delete departments: Erases departments along with their unique code and name.
* Course Assignment
  + Assign Course: Assigns courses to specific students.

# **Snippets from Project**

**Login page**



**Main page**

A blue square with white text

Description automatically generated

**Students page**

A screenshot of a computer

Description automatically generated

**Departments and courses page**

A screenshot of a computer

Description automatically generated

**Enrollment page**

A screenshot of a computer

Description automatically generated

**Report page**

To view some reports about students age and grades per course

A screenshot of a computer

Description automatically generated