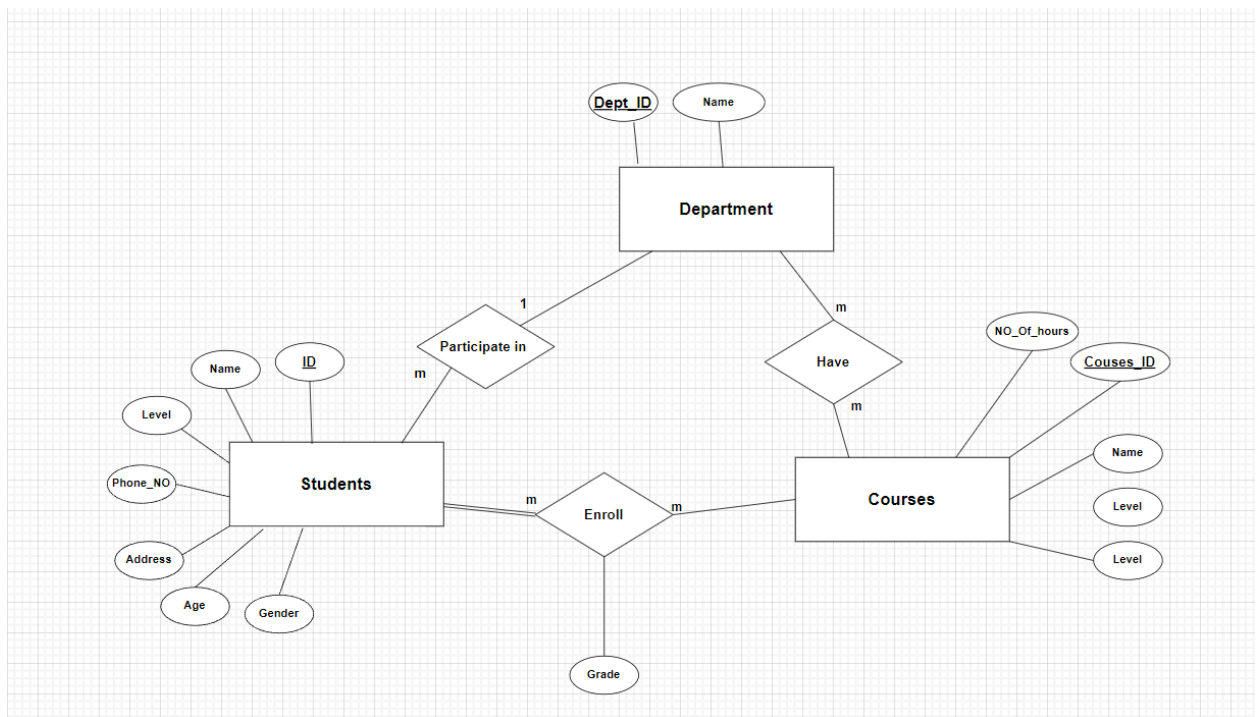


Introduction

The Student Management System is a user-friendly, technologically advanced platform that facilitates efficient management of student records, enrollment, grades, attendance, and other crucial information. This system not only centralizes student records but also offers a user-friendly Graphical User Interface (GUI) for seamless navigation.

1) Database Design

- ERD



2) Database Schema

- The "**student**" table is a pivotal component of our database schema, containing crucial information about each student in our educational institution. It comprises several attributes:
 - The **id** attribute serves as a unique identifier for each student.
 - The **name** attribute captures the full name of the student.
 - The **level** attribute indicates the academic level at which the student is enrolled.

- The **phone_number** attribute stores the contact number of the student or their guardian.
 - The **address** attribute records the residential address of the student.
 - The **age** attribute denotes the age of the student.
 - The **gender** attribute specifies the gender of the student.
 - students can enroll in many courses and have grades for each course but a student can only join one department.
- The "**department**" table is a fundamental component of our database schema, containing essential information about various academic departments within our institution. It comprises two key attributes:
- The **department id** attribute serves as a unique identifier for each department.
 - The **department name** attribute captures the name or title of each academic department.
 - A department can have many students and also can teach multiple courses.
- The "**courses**" table is a vital component of our database schema, encompassing key information about the various courses offered within our educational institution. It includes the following attributes:
- The **course id** attribute acts as a unique identifier for each course.
 - The **name** attribute represents the name or title of the course.
 - The **level** attribute denotes the academic level or class associated with the course.
 - The **number of hours** attribute indicates the duration of the course in terms of hours.
 - The **enrollment date** attribute captures the date when the course is enrolled.
 - A course can be taught by many departments and can be enrolled by many students.
-

Ensured Methods

Referential Integrity: Foreign key constraints play a pivotal role in establishing relationships between tables, guaranteeing referential integrity. In this context, the foreign keys in the "Student_Course" table point to the "Student" and "Course" tables, ensuring consistency in data across the database.

Streamlined Query Performance: The database structure is optimized for efficient querying of data. This allows for seamless retrieval of information related to students, courses, grades, and their interconnections.

Data Integrity Assurance: Various constraints, such as unique constraints and check constraints, are in place to validate that the data stored in the tables adheres to specific criteria. This proactive approach promotes and maintains data integrity throughout the database.

SQL Implementation

Courses table

```
CREATE TABLE ROOT.COURSES
(
    COURSE_ID          NUMBER(10),
    COURSE_NAME        VARCHAR2(100 BYTE),
    COURSE_LEVEL       NUMBER(10),
    NO_OF_HOURS        NUMBER(10),
    CONSTRAINT PK_COURSES PRIMARY KEY (COURSE_ID)
);
```

Departments Table

```
CREATE TABLE ROOT.DEPARTMENTS
(
    DEPARTMNET_ID      NUMBER(8),
    DEPARTMENT_NAME    VARCHAR2(20 BYTE),
    CONSTRAINT PK_DEPARTMENTS PRIMARY KEY (DEPARTMNET_ID)
);
```

Student Table

```
CREATE TABLE ROOT.STUDENTS
(
  STUDENT_ID      NUMBER(10),
  STUDENT_NAME    VARCHAR2(100 BYTE),
  STUDENT_LEVEL   NUMBER(10),
  PHONE_NO        NUMBER(10),
  ADDRESS         VARCHAR2(100 BYTE),
  AGE             NUMBER(10),
  CLASS_ID        NUMBER(10),
  GENDER          VARCHAR2(20 BYTE),
  DEPARTMENT_ID   NUMBER(8),
  CONSTRAINT PK_STUDENTS PRIMARY KEY (STUDENT_ID),
  CONSTRAINT DEPT_FK
  FOREIGN KEY (DEPARTMENT_ID)
  REFERENCES ROOT.DEPARTMENTS (DEPARTMNET_ID)
);
```

Department_courses Table

```
CREATE TABLE ROOT.DEPARTMENT_COURSES
(
  DEPARTMENT_ID   NUMBER(10),
  COURSE_ID       NUMBER(10),
  CONSTRAINT DEPT_COURSE_COM_PK
  PRIMARY KEY (DEPARTMENT_ID, COURSE_ID),
  CONSTRAINT DEPT_C_FK
  FOREIGN KEY (DEPARTMENT_ID)
  REFERENCES ROOT.DEPARTMENTS (DEPARTMNET_ID)
);
```

Student_courses Table

```
CREATE TABLE ROOT.STUDENT_COURSES
(
  STUDENT_ID      NUMBER(10),
  COURSE_ID       NUMBER(10),
  GRADE           NUMBER(4),
  DESCRIPTION     VARCHAR2(2000 BYTE),
  ENROLLMENT_DATE VARCHAR2(50 BYTE),
  CONSTRAINT GRADE_CHK
  CHECK (GRADE <= 100),
  CONSTRAINT COURSE_STUDENT_FK
  FOREIGN KEY (COURSE_ID)
  REFERENCES ROOT.COURSES (COURSE_ID),
  CONSTRAINT STUDENT_ID_FK
  FOREIGN KEY (STUDENT_ID)
  REFERENCES ROOT.STUDENTS (STUDENT_ID)
);
```

PL/Sql Functions and Triggers

1- **calc_courses_count**: function takes a department ID (v_id) as input and returns the count of courses for that department.

```
CREATE OR REPLACE FUNCTION ROOT.calc_courses_count(v_id NUMBER)
RETURN NUMBER
IS
    v_result NUMBER(10);
BEGIN
    SELECT COUNT(*) INTO v_result
    FROM department_courses
    WHERE DEPARTMENT_ID = v_id;

    RETURN v_result;
END;
```

2- **calc_student_count**: function calculates the count of students in a specific department.

```
CREATE OR REPLACE FUNCTION ROOT.calc_student_count(v_id NUMBER)
RETURN NUMBER
IS
    v_result NUMBER(10);
BEGIN
    SELECT COUNT(s.STUDENT_NAME) INTO v_result
    FROM students s
    JOIN departments d ON s.DEPARTMENT_ID = d.DEPARTMENT_ID
    WHERE d.DEPARTMENT_ID = v_id;

    RETURN v_result;
END;
```

3- **calc_success_rate**: function calculates the success rate for a specific course based on the grades in the student_courses table.

```
CREATE OR REPLACE FUNCTION ROOT.calc_success_rate(v_id NUMBER)
RETURN NUMBER
IS
    v_success NUMBER(10);
    v_all NUMBER(10);
    v_result NUMBER(10, 2);
BEGIN
    SELECT COUNT(*) INTO v_success
    FROM student_courses
    WHERE DESCRIPTION != 'F' AND course_id = v_id;

    SELECT COUNT(*) INTO v_all
    FROM student_courses
    WHERE course_id = v_id;

    IF v_all <> 0 THEN
        v_result := (v_success / v_all) * 100;
    ELSE
        v_result := 0; -- or handle it differently based on your
        requirements
    END IF;

    RETURN v_result;
END;
```

4- **calc_total_courses**: function calculates the total number of courses available.

```
CREATE OR REPLACE FUNCTION ROOT.calc_total_courses
RETURN NUMBER
IS
    v_result NUMBER(10);
BEGIN
    SELECT COUNT(*) INTO v_result
    FROM courses

    RETURN v_result;
END;
```

5- **calc_total_depts**: function calculates the total number of departments based on the records in the departments table.

```
CREATE OR REPLACE FUNCTION ROOT.calc_total_depts
RETURN NUMBER
IS
    v_result NUMBER(10);
BEGIN
    SELECT COUNT(*) INTO v_result
    FROM departments;

    RETURN v_result;
END;
```

6- **calc_total_students** : function calculates the total number of students based on the records in the students table

```
CREATE OR REPLACE FUNCTION ROOT.calc_total_students
RETURN NUMBER
IS
    v_result NUMBER(10);
BEGIN
    SELECT COUNT(*) INTO v_result
    FROM students;

    RETURN v_result;
END;
```

7- **count_student_number** : function calculates the number of students enrolled in a specific course based on the records in the student_courses table.

```
CREATE OR REPLACE FUNCTION ROOT.count_student_number(v_id NUMBER)
RETURN NUMBER
IS
    v_result NUMBER(10);
BEGIN
    SELECT COUNT(*) INTO v_result
    FROM student_courses
    WHERE course_id = v_id;

    RETURN v_result;
END;
```

STUDENT_COURSES_TRG: a BEFORE INSERT OR UPDATE trigger for the STUDENT_COURSES table. This trigger automatically populates the DESCRIPTION column based on the grade values.

```
CREATE OR REPLACE TRIGGER ROOT.STUDENT_COURSES_TRG
BEFORE INSERT OR UPDATE
ON ROOT.STUDENT_COURSES REFERENCING NEW AS New OLD AS Old
FOR EACH ROW
BEGIN
    IF :new.grade >= 97 THEN
        :new.DESCRPTION := 'A+';
    ELSIF :new.grade >= 93 THEN
        :new.DESCRPTION := 'A';
    ELSIF :new.grade >= 90 THEN
        :new.DESCRPTION := 'A-';
    ELSIF :new.grade >= 87 THEN
        :new.DESCRPTION := 'B+';
    ELSIF :new.grade >= 83 THEN
        :new.DESCRPTION := 'B';
    ELSIF :new.grade >= 80 THEN
        :new.DESCRPTION := 'B-';
    ELSIF :new.grade >= 77 THEN
        :new.DESCRPTION := 'C+';
    ELSIF :new.grade >= 73 THEN
        :new.DESCRPTION := 'C';
    ELSIF :new.grade >= 70 THEN
        :new.DESCRPTION := 'C-';
    ELSIF :new.grade >= 67 THEN
        :new.DESCRPTION := 'D+';
    ELSIF :new.grade >= 63 THEN
        :new.DESCRPTION := 'D';
    ELSIF :new.grade >= 60 THEN
        :new.DESCRPTION := 'D-';
    ELSE
        :new.DESCRPTION := 'F';
    END IF;
END;
```

1- Bash Script for monitoring disk space and sending alerts.

```
#!/bin/bash
# Bash Script for Disk Space Monitoring and Logging

# Set the path for the log file
log_file="/D/case_study/Bash/log_disk.txt"

# Set the threshold for disk space in percentage
threshold=60

# Get the current disk space usage and extract the percentage
disk_space=$(df -h | awk 'NR==2 {print $6}' | sed 's/%//')

# Display the current disk space
echo "Current Disk Space: $disk_space%"

# Check if disk space is above the threshold
if [[ $disk_space -gt $threshold ]]; then
    # Log a warning if disk space exceeds the threshold
    echo "Warning: Disk Space Is Above $threshold%" >> "$log_file"
else
    # Log a message if disk space is below or equal to the threshold
    echo "Disk Space Is Under $threshold%" >> "$log_file"
fi
```

1. Shebang (#!/bin/bash):
 - a. Indicates that the script should be executed using the Bash shell.
2. Log File Path (log_file="/D/case_study/Bash/log_disk.txt"):
 - a. Specifies the path for the log file where disk space information will be logged.
3. Threshold Setting (threshold=60):
 - a. Sets the threshold for disk space usage in percentage. If the disk space exceeds this threshold, a warning will be logged.
4. Disk Space Retrieval (disk_space=\$(df -h | awk 'NR==2 {print \$6}' | sed 's/%//')):
 - a. Uses the `df` command to retrieve disk space information.
 - b. Utilizes `awk` to extract the percentage usage from the second line of the `df` output.
 - c. Employs `sed` to remove the percentage sign from the extracted value.

5. Display Current Disk Space (`echo "Current Disk Space: $disk_space%"`):
 - a. Prints the current disk space to the console.
6. Disk Space Comparison and Logging:
 - a. Compares the current disk space with the specified threshold.
 - b. If the disk space exceeds the threshold, logs a warning message in the designated log file
 - c. If the disk space is below or equal to the threshold, logs a message indicating that the disk space is within an acceptable range.

2- Bash Script for database backup.

```
#!/bin/bash
# Bash Script for Oracle Database Export using Data Pump

# Oracle Database Connection Details
DB_USER=ROOT                # Oracle database username
DB_PASSWORD=root            # Oracle database password
DB_SID=XE                   # Oracle System Identifier (SID)

# Date Format for Backup File
DATE_FORMAT=$(date +"%Y%m%d_%H%M%S") # Format for the timestamp in the
backup file name
# Export File Name (only the file name, not the full path)
EXPORT_FILE="backup_${DATE_FORMAT}.dmp" # Final export file name with
timestamp

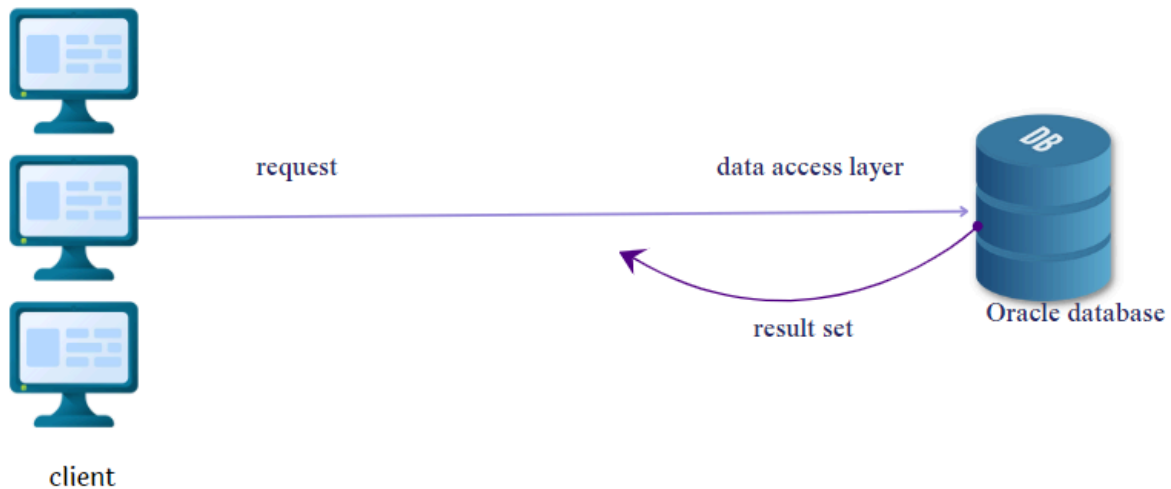
# Oracle Data Pump Export Command
expdp ${DB_USER} ${DB_PASSWORD}@${DB_SID} DIRECTORY=DATA_PUMP_DIR
DUMPFIL= ${EXPORT_FILE} FULL=Y
# Uses Oracle Data Pump to export the entire database.
# ${DB_USER}/${DB_PASSWORD}@${DB_SID}: Specifies the database connection
details.
# DIRECTORY=DATA_PUMP_DIR: Specifies the Oracle directory object for the
export.
# DUMPFIL= ${EXPORT_FILE}: Specifies the name of the export file.
# FULL=Y: Performs a full database export.

# Check if the export was successful
if [ $? -eq 0 ]; then
    echo "Database backup successful. File: ${EXPORT_FILE}"
else
    echo "Error: Database backup failed."
fi
```

1. **Oracle Database Connection Details:**
 - a. DB_USER=ROOT: Specifies the Oracle database username.
 - b. DB_PASSWORD=root: Specifies the Oracle database password
 - c. DB_SID=XE: Specifies the Oracle System Identifier (SID).
 2. **Date Format for Backup File:**
 - a. DATE_FORMAT=\$(date +"%Y%m%d_%H%M%S"): Defines the format for the timestamp to be included in the backup file name.
 3. **Export File Name:**
 - a. EXPORT_FILE="backup_\${DATE_FORMAT}.dmp": Generates the export file name by appending the timestamp to a base name.
 4. **Oracle Data Pump Export Command:**
 - a. expdp \${DB_USER}/\${DB_PASSWORD}@\${DB_SID} DIRECTORY=DATA_PUMP_DIR DUMPFILE=\${EXPORT_FILE} FULL=Y: Executes the Oracle Data Pump export command.
 - b. \${DB_USER}/\${DB_PASSWORD}@\${DB_SID}: Specifies the database connection details
 - c. DIRECTORY=DATA_PUMP_DIR: Specifies the Oracle directory object for the export.
 - d. DUMPFILE=\${EXPORT_FILE}: Specifies the name of the export file.
 - e. FULL=Y: Specifies a full database export
 5. **Export Status Check:**
 - a. if [\$? -eq 0]; then: Checks the exit status of the last executed command.
 - b. echo "Database backup successful. File: \${EXPORT_FILE}": Prints a success message if the exit status is 0.
 - c. else: Indicates the beginning of the block for handling errors.
 - d. echo "Error: Database backup failed.": Prints an error message if the exit status is non-zero.
-

System Architecture

System architecture



- Data Access Layer

1. **Introduction:** The `DataAccessLayer` class is part of a Java application designed to manage and interact with a database related to a university or educational institution. It provides methods to connect to the database, retrieve and manipulate data related to students, departments, courses, and grades.
2. **Class structure :** The class is organized into several sections, each handling a specific aspect of the application's data management. These sections include methods for managing student data, department data, course data, grade data, and generating various reports.
3. **Connection management :** `connect()`: Establishes a connection to the database using the `DatabaseConnectionManager` singleton class.
4. **Student operations:**
 - a. `getStudentsData()`: Retrieves a list of student data from the database.
 - b. `insertIntoStudent(StudentsDto std)`: Inserts a new student record into the database.
 - c. `updateStudent(StudentsDto std)`: Updates an existing student record in the database.
 - d. `deleteStudent(String id)`: Deletes a student record from the database.

5. Department Operations :

- a. `getDepartmentsData()`: Retrieves a list of department data from the database.
- b. `insertIntoDepartment(DepartmentsDto dept)`: Inserts a new department record into the database.
- c. `updateDepartment(DepartmentsDto dept)`: Updates an existing department record in the database.
- d. `deleteDepartment(String deptId)`: Deletes a department record from the database.

6. Courses Operations:

- a. `getCoursesData()`: Retrieves a list of course data from the database.
- b. `insertIntoCourse(CourseDto course)`: Inserts a new course record into the database.
- c. `updateCourse(CourseDto course)`: Updates an existing course record in the database.
- d. `deleteCourse(String courseId)`: Deletes a course record from the database.

7. Grades Operations:

- a. `getGradesData()`: Retrieves a list of grade data from the database.
- b. `insertIntoGrades(GradesDto grade)`: Inserts a new grade record into the database.
- c. `updateGrade(GradesDto grade)`: Updates an existing grade record in the database.
- d. `deleteGrade(String studentId, String courseId)`: Deletes a grade record from the database.

8. Report Generation:

- a. `getReportIntialsData(String studentId)`: Retrieves initial data for generating a student report.
- b. `getReportTableData(String studentId)`: Retrieves tabular data for generating a student report.
- c. `getCourseReport(String courseId)`: Retrieves data for generating a course report.
- d. `getCourseReportTableData(String courseId)`: Retrieves tabular data for generating a course report.
- e. `getDepartmentTopReport(String departmentId)`: Retrieves top-level data for generating a department report.

- f. `getDepartmentTableReportData(String departmentId):`
Retrieves tabular data for generating a department report.
 - g. `getHomeTopData():` Retrieves top-level data for generating a home page report.
 - h. `getHomeTableData():` Retrieves tabular data for generating a home page report.
 - i. `getHomeChartData(String courseName):` Retrieves chart data for generating a home page report.
- 9. Miscellaneous operation:**
- a. `InsertIntoDeptCourse(String deptID, String courseId):`
Inserts a new department-course relationship into the database.
- 10. Conclusion :** The `DataAccessLayer` class serves as a crucial component of the application, facilitating the interaction between the Java code and the underlying database. The organized structure allows for efficient data retrieval, manipulation, and reporting within an educational context.

- Home Scene

1. The `HomeScene` class is a JavaFX implementation representing the home page of a dashboard in a client application. It consists of various UI elements organized in an `AnchorPane` layout. The scene includes navigation buttons (`studentsBtn`, `coursessBtn`, `departmentsBtn`, `gradesBtn`) and displays essential statistics such as the total number of students, departments, and courses. Additionally, a line chart and a table are used to present dynamic data related to courses.
2. **Navigation:**
 - a. Navigation buttons (`studentsBtn`, `coursessBtn`, `departmentsBtn`, `gradesBtn`) facilitate switching between different sections of the application.
3. **Statistics:**
 - a. The total number of students, departments, and courses are displayed using labeled images and corresponding `TextField` elements.
4. **Dynamic Data Presentation:**
 - a. The scene dynamically fetches and displays data from the database:
 - i. The total number of students, departments, and courses are fetched from the database and displayed.

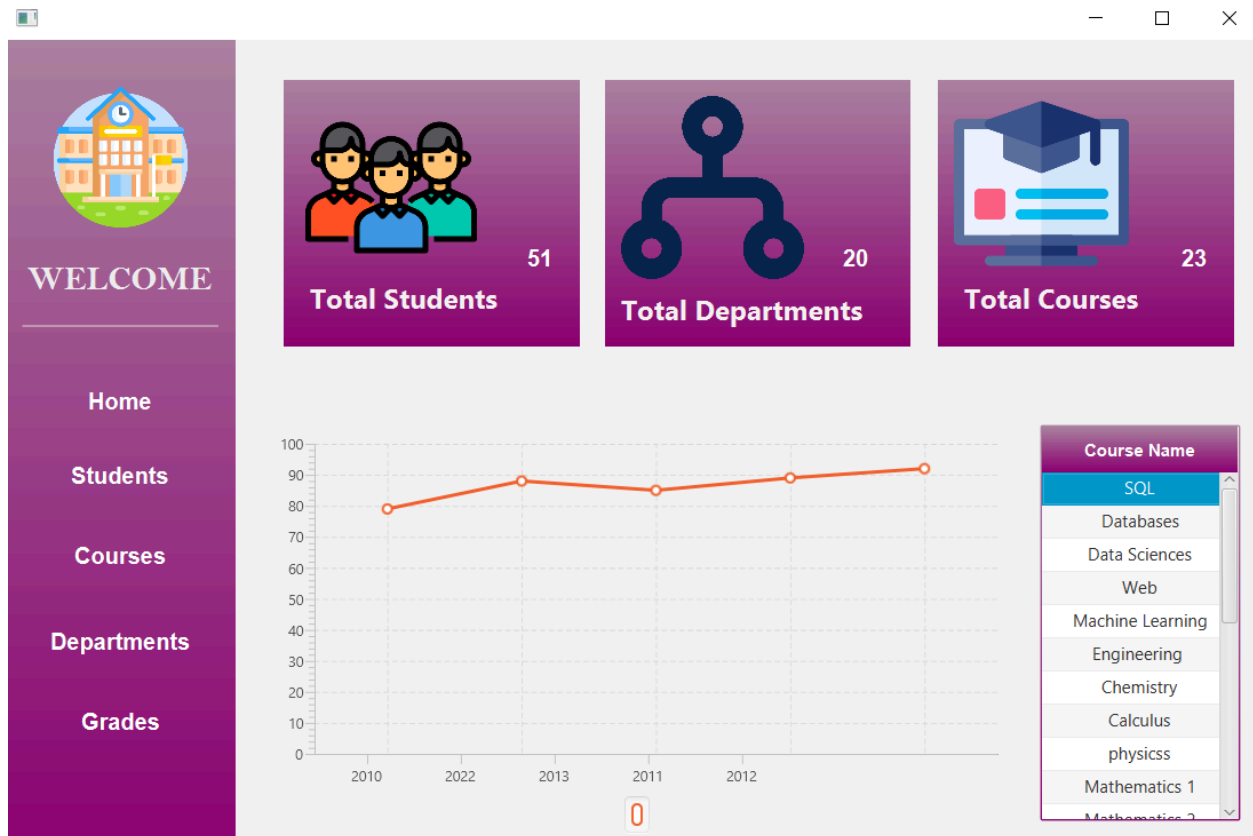
- ii. A table (`coursesTable`) is populated with course names.
- iii. A line chart (`lineChart`) visualizes historical data for a selected course, such as the average grade over time.

5. Navigation Actions:

- a. Each navigation button is associated with an `ActionEvent` listener, redirecting the user to different scenes (Students, Courses, Departments, Grades) upon button click.

6. Integration with Database:

- a. Database interactions are handled through the `DataAccessLayer` class, fetching data for statistics, table, and chart.



- The StudentScene is a JavaFX application designed for managing student information. It features the following components:

1. Navigation Sidebar:

- a. Home, Students, Courses, Departments, and Grades buttons for navigation.
- b. Visual indicators and icons for better user experience.

2. Student Table View:

- a. Displays a table with columns for student information (ID, Name, Level, Phone No, Address, Age, Class No, Gender, Department ID).
- b. Allows selection of a row for detailed view and modification.

3. Student Information Form:

- a. Text fields for entering/modifying student information.
- b. Add, Update, Delete, and Refresh buttons for data manipulation.
- c. **Show Report button to generate a student report.**

4. Styling and Layout:

- a. Consistent styling using CSS for a clean and professional appearance.
- b. Responsive layout for optimal viewing on different screen sizes.

5. Integration with Database:

- a. Utilizes a `DataAccessLayer` class for connecting to a database.
- b. Fetches and displays student data in the table.
- c. Supports adding, updating, and deleting student records.

6. Navigation to Other Scenes:

- a. Buttons to navigate to other scenes for managing courses, departments, and grades.
- b. Home button for returning to the main dashboard.

7. Report Generation:

- a. Generates a student report based on the selected student ID.

8. Error Handling:


- a. Provides error messages for empty ID fields and other potential issues.

9. Refresh Functionality:

- a. Refresh button to reload and display the latest student data.

10. Integration with Other Scenes:

- a. Seamless navigation to scenes for managing courses, departments, and grades.



WELCOME

Home

Students

Courses

Departments

Grades

ID	Name	Level	Phone NO	Address	Age	Class NO	Gender	Dept ID
1	Omar Asharf	1	1234567890	123 Main St	20	101	Male	1
2	ahmed atef	2	9876543210	456 Oak St	21	102	Female	2
3	Mohamed Elfeky	2	9876543210	456 Oak St	21	102	Female	2
4	Hany Shaker	2	1272501501	987 Maple St	22	104	Female	4
5	mark	3	4445556666	654 Birch St	20	105	Male	5
6	Olivia Davis	1	7778889999	321 Cedar St	21	106	Female	6
7	William Johnson	2	1112223333	555 Elm St	19	107	Male	7
8	Sophia Lee	3	8889990000	987 Pine St	22	108	Female	8
10	Emma Wilson	2	9998887777	456 Cedar St	21	110	Female	10
11	Ethan Harris	3	7776665555	789 Oak St	19	111	Male	1
12	Ava Turner	1	2223334444	321 Main St	20	112	Female	2

ID

6

Phone No

7778889999

Class No

106

Name

Olivia Davis

Address

321 Cedar St

Gender

Female

Level

1

Age

21

Dept ID

6

Add

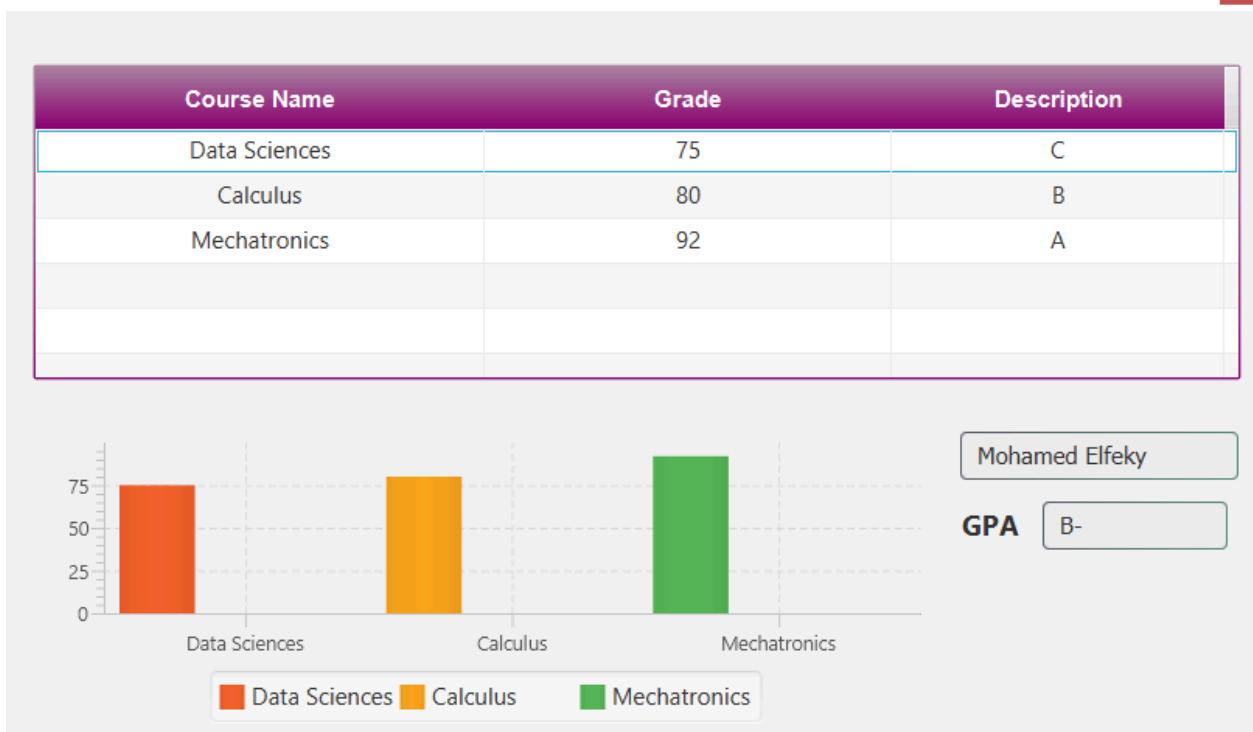
Update

Delete

Refresh

Show Report

Student Report



- **CourseScene::**The `CourseScene` class represents the user interface for managing courses in a Student Management System. It includes navigation buttons, a table displaying course information, and input fields for adding, updating, and deleting courses.

- Key Components:

- 1. Navigation Sidebar:**

- a. Buttons for Home, Students, Courses, Departments, and Grades.
- b. Consistent styling using CSS for a unified appearance.

- 2. Course Table View:**

- a. Displays a table with columns for Course ID, Name, Level, and Number of Hours.
- b. Retrieves data from the database and populates the table.
- c. Allows single-click selection of a course for detailed view and modification.

- 3. Input Fields for Course Information:**

- a. Text fields for Course ID, Name, Level, Number of Hours.
- b. Additional text field for assigning a course to a department (optional).

- 4. Action Buttons:**

- a. Add, Update, Delete, Show Report, and Refresh buttons.
- b. Buttons are styled for a cohesive look and feel.

- 5. Integration with Database:**

- a. Utilizes a `DataAccessLayer` class for database interaction.
- b. Fetches course data and displays it in the table.
- c. Supports adding, updating, and deleting course records.

- 6. Department Assignment:**

- a. Optional text field for assigning a course to a department.

- 7. Report Generation:**

- a. Generates a report for the selected course when the "Show Report" button is clicked.
- b. Opens a new stage for the report using the `CourseReport` class.

- 8. Navigation to Other Scenes:**

- a. Seamless navigation to scenes for managing students, departments, grades, and returning home.

- 9. Styling and Layout:**

- a. Consistent use of CSS for styling to maintain a professional and organized UI.
- b. Responsive layout for varying screen sizes.

10.Event Handling:

- Event handlers for various buttons (Add, Update, Delete, Show Report, Refresh) to perform corresponding actions.



WELCOME

- Home
- Students
- Courses
- Departments
- Grades

Course ID	Course Name	Course Level	Number of Hours
1	SQL	100	40
2	Databases	200	60
3	Data Sciences	300	80
4	Web	200	45
5	Machine Learning	300	70
6	Engineering	200	55
7	Chemistry	300	60
8	Calculus	100	50
9	physicss	300	75
10	Mathematics 1	200	65
11	Mathematics 2	400	90

Course ID:

Course Name:

Course Level:

Number of Hours:

Assign to a Department (optional):

Add

Update

Delete


Refresh

Show Report

Course Report



Total Students 7



Success Rate % 85.7

Student Name	Grade
Ava Turner	88
Madison Turner	85
Mason Baker	86
Lucas Baker	88
Owen Baker	79

Success Rate

Failed Rate

- **DepartmentsScene::**The `DepartmentsScene` class represents the user interface for managing Departments in a Student Management System. It includes navigation buttons, a table displaying department information, and input fields for adding, updating, and deleting courses.
- Key Components:

1. GUI Layout:

- a. The code defines a JavaFX `StackPane` that contains an `AnchorPane`. The `AnchorPane` is used to organize different UI components.

2. Navigation Buttons:

- a. Navigation buttons (`homeBtn`, `studentsBtn`, `coursesBtn`, `departmentsBtn`, `gradesBtn`) are created and added to the left side of the dashboard. They are styled with CSS classes.

3. Department Table:

- a. `TableView` named `departmentTable` is created to display department information. It has two columns (`idCol` and `nameCol`) for department ID and department name.
- b. The data for the table is fetched from a database using a `DataAccessLayer` class and displayed in the table.

4. Text Fields and Buttons:

- a. Text fields (`deptIdTf` and `deptNameTf`) are used for entering department ID and name.
- b. Buttons (`addBtn`, `updateBtn`, `deleteBtn`, `refreshBtn`, `reportBtn`) are provided for performing actions like adding, updating, deleting, refreshing, and showing reports.

5. Event Handling:

- a. Event handlers are attached to buttons to perform actions like adding, updating, deleting, refreshing, and showing reports.
- b. The table has an `OnMouseClicked` event that allows selecting a department and populating the text fields with its information.

6. Styling:

- a. CSS stylesheets (`dashboardDesign.css`) are applied to style the UI components.

7. Navigation:

- a. Each navigation button has an associated event handler that switches the main content area to different scenes (e.g., `students`, `courses`, `grades`) when clicked.

8. Error Handling:

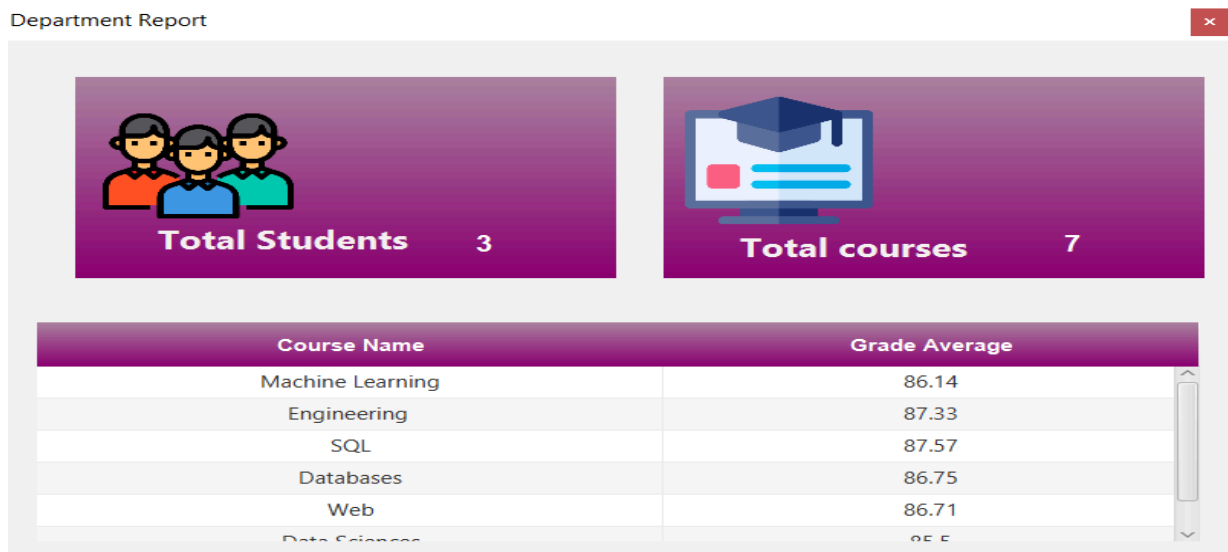
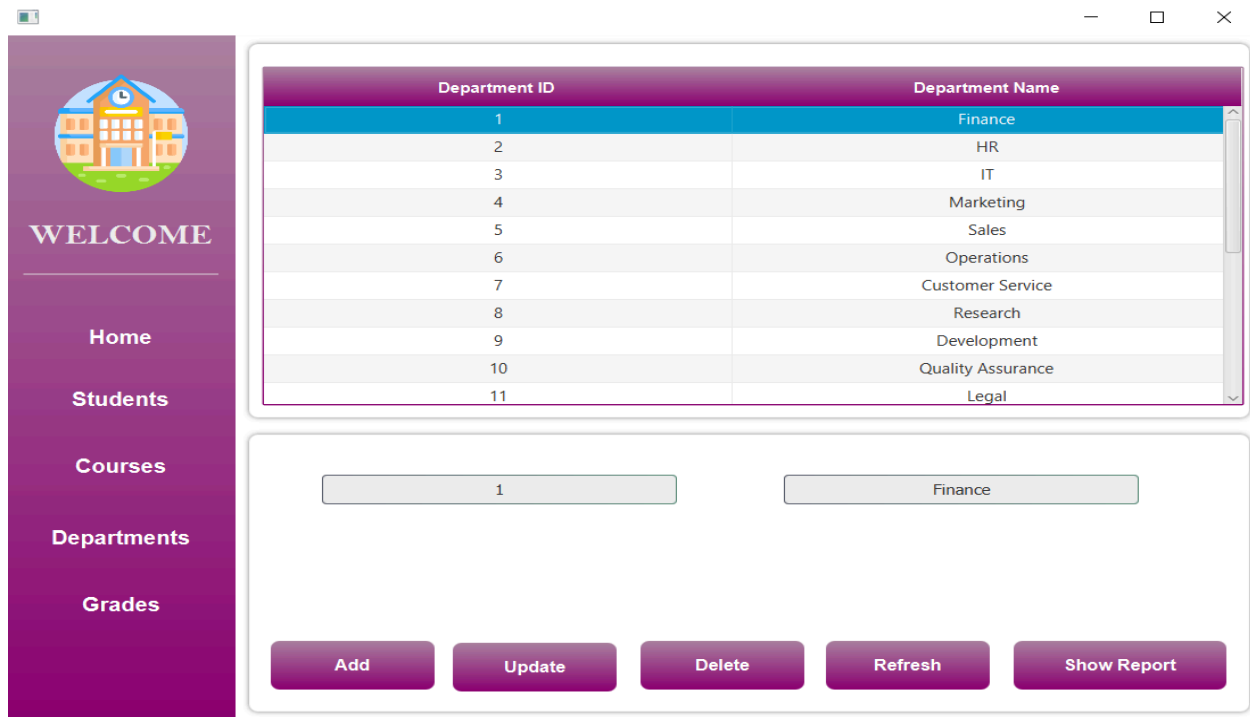
- Exception handling is implemented for database-related operations (SQLException).

9. Report Generation:

- The reportBtn triggers the creation of a DepartmentReport scene and displays it in a new stage.

10. Integration with Other Scenes:

- Navigation buttons allow switching between different scenes (students, courses, grades, home).



- **GradesScene::**The `GradesScene` class represents the user interface for managing Grades in a Student Management System. It includes navigation buttons, a table displaying Grades information, and input fields for adding, updating, and deleting Grades.
- Key Components:

1. UI Elements:

- a. The application uses JavaFX for the user interface.
- b. It has buttons for navigation (e.g., Students, Courses, Departments) and features (e.g., Add, Update, Delete).
- c. A TableView is used to display a list of grades with columns for Student ID, Course ID, Grade, Description, and Enrollment Date.
- d. TextFields for input and Labels for descriptions are present.

2. Functionality:

- a. The TableView is populated with data retrieved from a database using a `DataAccessLayer` class.
- b. Clicking on a row in the table fills the corresponding data into the input TextFields for editing.
- c. Buttons like Add, Update, Delete, Refresh trigger corresponding actions in the database via the `DataAccessLayer`.
- d. Navigation buttons switch between different scenes.

3. Styling:

- a. CSS styles are applied to enhance the visual appeal.
- b. Shadow and white background styles are used for different panes.

4. Event Handling:

- a. Event handlers are used for button clicks (e.g., AddBtn, UpdateBtn, DeleteBtn, RefreshBtn).
- b. Clicking on a row in the table triggers an event to fill the input fields with the selected grade.

5. Scene Navigation:

- a. Buttons for Students, Courses, Departments, and Home are set to switch scenes when clicked.



WELCOME

Home

Students

Courses

Departments

Grades

Student ID	Course ID	Grade	Description	Enrollment Date
12	1	88	F	2022
13	10	70	C	2012
1	10	85	B	2012
2	2	90	A	2013
2	7	88	B	2013
2	13	95	A	2013
3	3	75	C	2010
3	8	80	B	2010
3	15	92	A	2010
4	4	87	B	2010
4	9	94	A	2010

Student ID Course ID Grade
Description Enrollment Date

Add

Update

Delete

Refresh

- Project Summary: The provided JavaFX project is a comprehensive application for managing student grades. It includes navigation features between different scenes, such as Students, Courses, Departments, and a Home screen. The primary focus is on the "Grades" scene, where users can view, add, update, and delete grade records. Below are the key components and features:
- Main components:
 - 1. GradeScene:**
 - a. Represents the main scene for managing grades.
 - b. Contains navigation buttons, a TableView for displaying grades, and input fields for adding or updating grades.
 - c. Utilizes external CSS for styling.
 - 2. DataAccessLayer:**
 - a. Implements database interactions for grades.
 - b. Includes methods for retrieving, inserting, updating, and deleting grade records.
 - c. Implements the Singleton pattern to ensure a single instance of the class.
- Key functionality:
 - 1. Scene Navigation:**
 - a. Buttons like Students, Courses, Departments, and Home enable navigation between different scenes.
 - b. Each scene is represented by a dedicated class.
 - 2. Grade Management:**
 - a. TableView displays a list of grades with columns for essential information.
 - b. Input fields allow users to add or update grades, triggered by corresponding buttons.
 - c. Deleting grades is also supported.
 - 3. Data Integrity and Security:**
 - a. Singleton pattern ensures a single instance of DataAccessLayer for efficient database access.
 - b. GradesDto acts as a structured data object, enhancing code readability and maintainability.