University System Documentation

Prepared by
Ahmed Jumaa

FEB. 2024

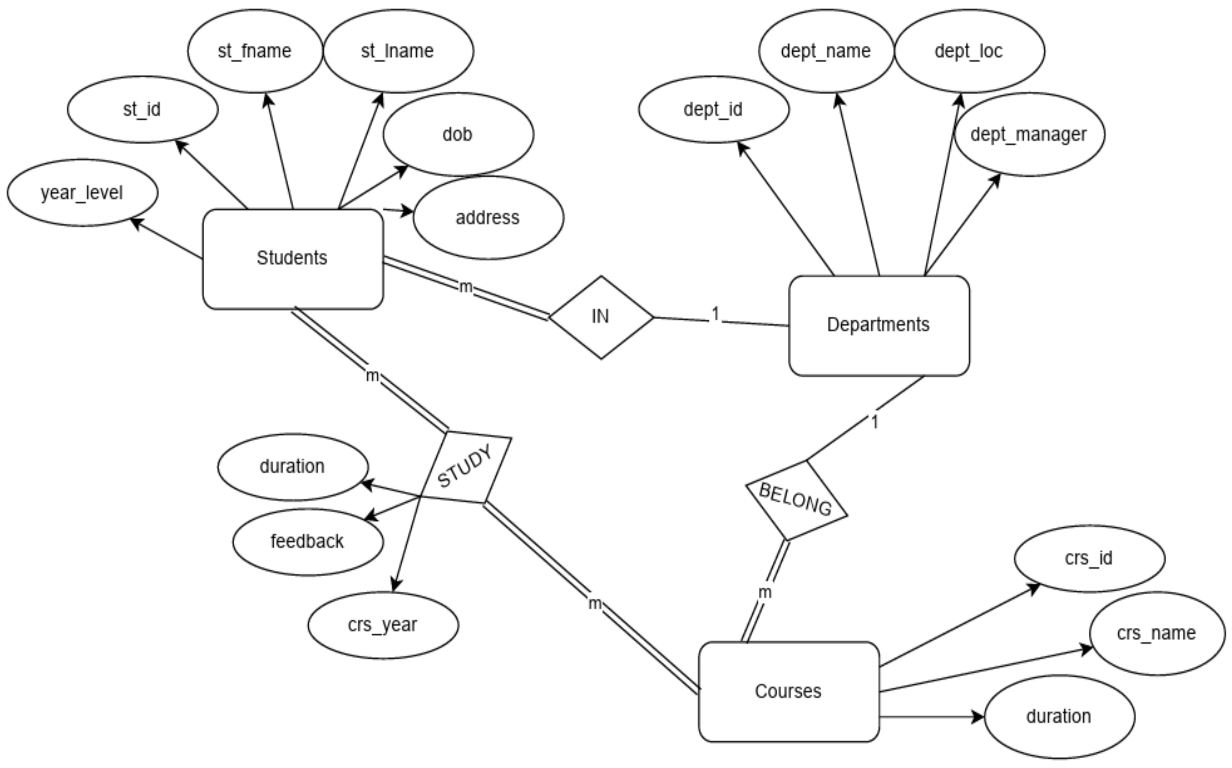
Table of Contents

1- Introduction:

This comprehensive documentation outlines the design, implementation, and functionality of a University System, covering key aspects such as database design, SQL implementation, PL/SQL procedures, an automation script, and a Java application. Each section is meticulously detailed to provide a clear understanding of the system's architecture, facilitating ease of development, maintenance, and troubleshooting.

2- Database Design:

This document has provided a comprehensive overview of the relational database schema for managing student, course, department, and grade information. The design adheres to normalization principles to ensure data integrity. The outlined schema will serve as a foundation for subsequent steps, including SQL script creation, data population, and the development of PL/SQL procedures and Java applications.



Students Table

The **Students** table stores information about students.

st_id: Unique identifier for each student.

st_fname: First name of the student.

st_lname: Last name of the student.

dob: Date of birth of the student.

address: Address of the student.

year_level: Current academic year of the student.

dept_id: Foreign key referencing the Departments table.

Departments Table

The Departments table contains details about academic departments.

dept_id: Unique identifier for each department.

dept_name: Name of the department.

dept_loc: Location of the department.

dept_manager: Manager of the department.

Courses Table

The **Courses** table holds information about academic courses.

crs_id: Unique identifier for each course.

crs_name: Name of the course.

credits: Credits associated with the course.

Grades Table

The **Grades** table records student grades for each course.

g_id: Unique identifier for each grade entry.

st_id: Foreign key referencing the Students table.

crs_id: Foreign key referencing the Courses table.

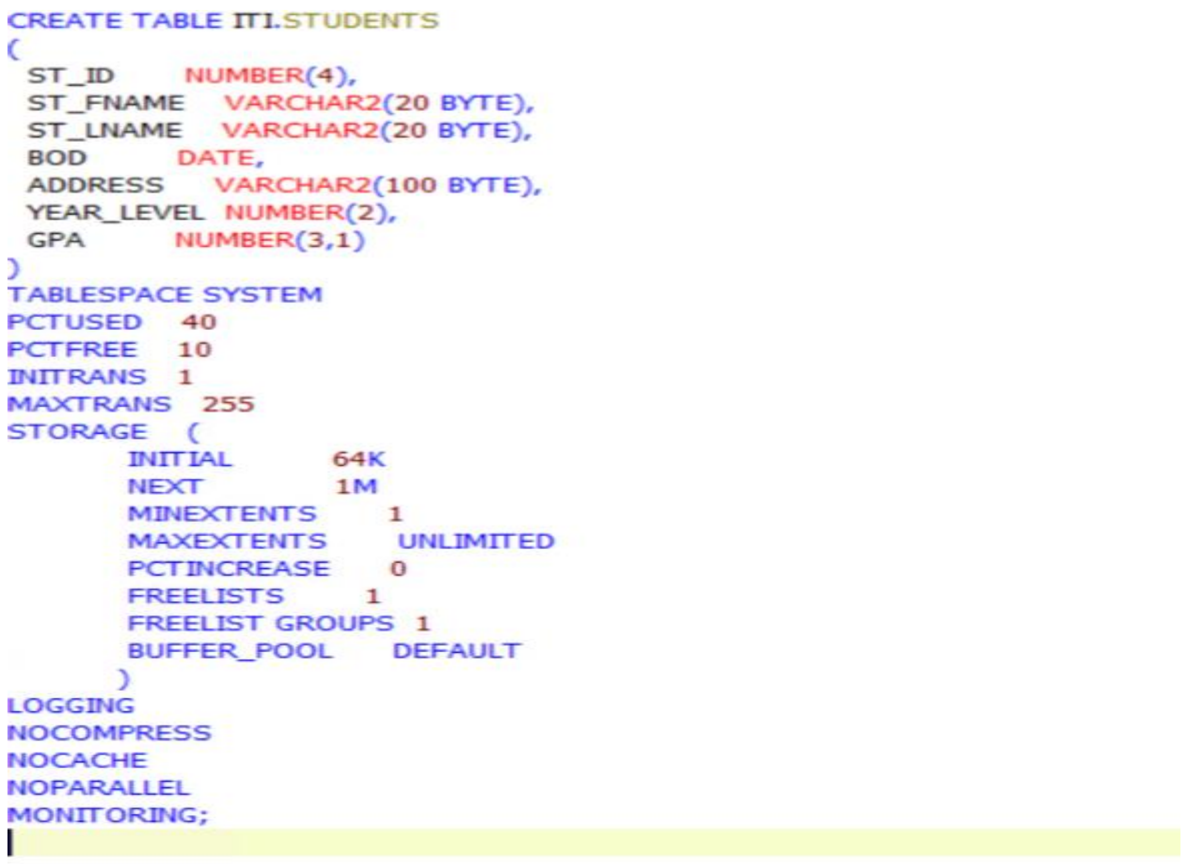
grade: Grade obtained by the student.

feedback: Feedback related to the student's performance.

crs_year: Academic year in which the course was taken.

3- SQL Script Implementation:

The code for creating the database schema is in a file named **metadata.sql** This SQL script is designed to create tables for a university-related database, specifically focusing on students, departments, courses, and grades. Below is an example for table creation :



```
CREATE TABLE ITI.STUDENTS
(
  ST_ID       NUMBER(4),
  ST_FNAME    VARCHAR2(20 BYTE),
  ST_LNAME    VARCHAR2(20 BYTE),
  BOD         DATE,
  ADDRESS     VARCHAR2(100 BYTE),
  YEAR_LEVEL  NUMBER(2),
  GPA         NUMBER(3,1)
)
TABLESPACE SYSTEM
PCTUSED    40
PCTFREE    10
INITRANS   1
MAXTRANS   255
STORAGE    (
            INITIAL          64K
            NEXT             1M
            MINEXTENTS       1
            MAXEXTENTS       UNLIMITED
            PCTINCREASE      0
            FREELISTS        1
            FREELIST GROUPS  1
            BUFFER_POOL      DEFAULT
           )
LOGGING
NOCOMPRESS
NOCACHE
NOPARALLEL
MONITORING;
```

4- PL/SQL Functions & Procedures:

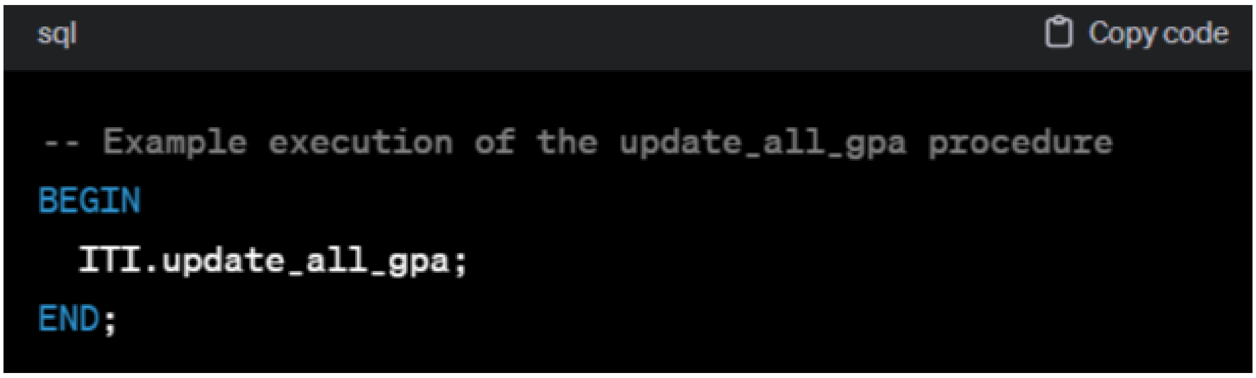
Creating some functions and procedures to help organize code by grouping related tasks together. This enhances code readability and makes it easier to maintain and troubleshoot.

- The **calculate_gpa** PL/SQL function is designed to convert numerical grades into the corresponding GPA (Grade Point Average) according to a standard grading scale.

- **97 and above:** 4.0
- **93 to 96:** 4.0
- **90 to 92:** 3.7
- **87 to 89:** 3.3
- **83 to 86:** 3.0
- **80 to 82:** 2.7
- **77 to 79:** 2.3
- **73 to 76:** 2.0
- **70 to 72:** 1.7
- **67 to 69:** 1.3
- **65 to 66:** 1.0
- **Below 65:** 0.0

- The **update_all_gpa** PL/SQL procedure is designed to update the GPA of all students based on the average grades they have achieved. The purpose of this procedure is to recalculate and update the GPA of all students in the database based on the average grades they have obtained in their courses.

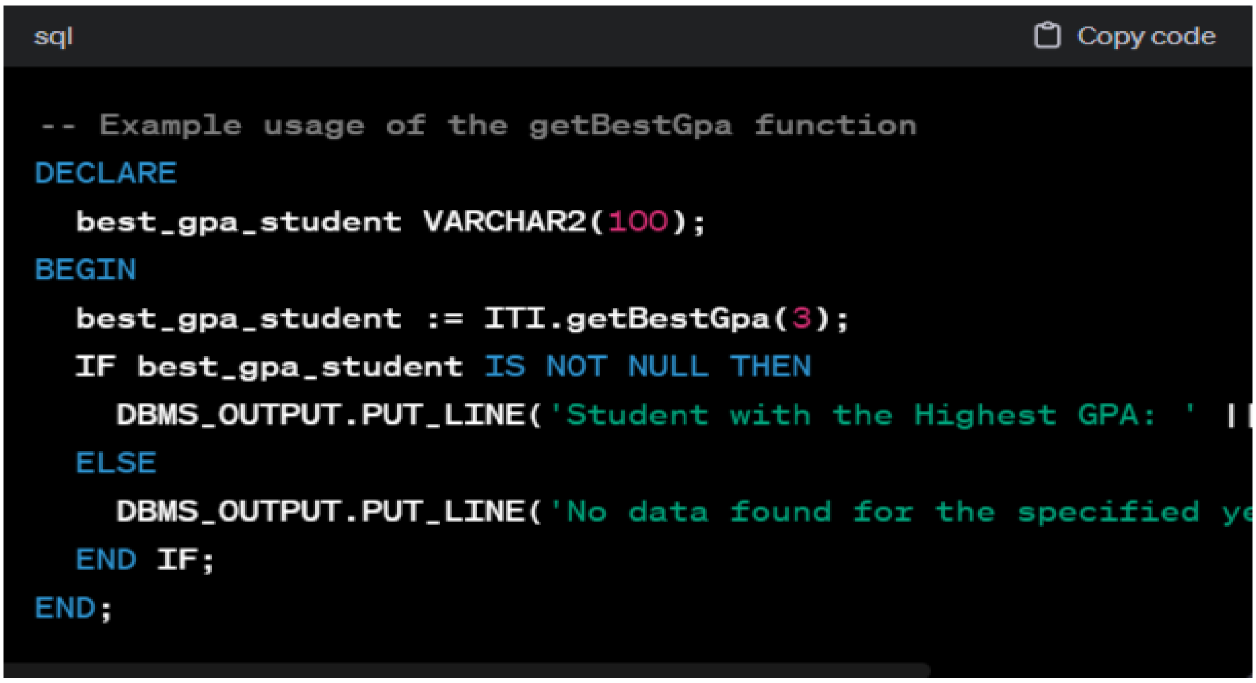
Example Usage:



```sql
-- Example execution of the update_all_gpa procedure
BEGIN
  ITI.update_all_gpa;
END;
```

- The **getBestGpa** PL/SQL function is designed to retrieve the first name of the student with the highest GPA in a specific year level. The purpose of this function is to find and return the first name of the student with the highest GPA in a given academic year level.

Example Usage:



```sql
-- Example usage of the getBestGpa function
DECLARE
  best_gpa_student VARCHAR2(100);
BEGIN
  best_gpa_student := ITI.getBestGpa(3);
  IF best_gpa_student IS NOT NULL THEN
    DBMS_OUTPUT.PUT_LINE('Student with the Highest GPA: ' ||
  ELSE
    DBMS_OUTPUT.PUT_LINE('No data found for the specified ye
  END IF;
END;
```

Seq-trigger pair:

Sequence: ITI.GRADES_SEQ

The sequence **ITI.GRADES_SEQ** is designed to generate unique values for the primary key column **G_ID** in the **ITI.GRADES** table.

Details:

- **Start Value:** The sequence starts with the value 131.
- **Maximum Value:** The maximum value is set to a very high number to accommodate a large number of unique values.
- **Minimum Value:** The minimum value is set to 1.
- **No Cycle:** The sequence does not cycle back to its start value.
- **Cache:** The sequence preallocates and caches 10 values for better performance.
- **No Order:** The sequence may not generate values in order of request.

Trigger: ITI.GRADES_TRG

The trigger **ITI.GRADES_TRG** is associated with the **BEFORE INSERT** event on the **ITI.GRADES** table. Its purpose is to automatically populate the **G_ID** column with the next value from the **ITI.GRADES_SEQ** sequence.

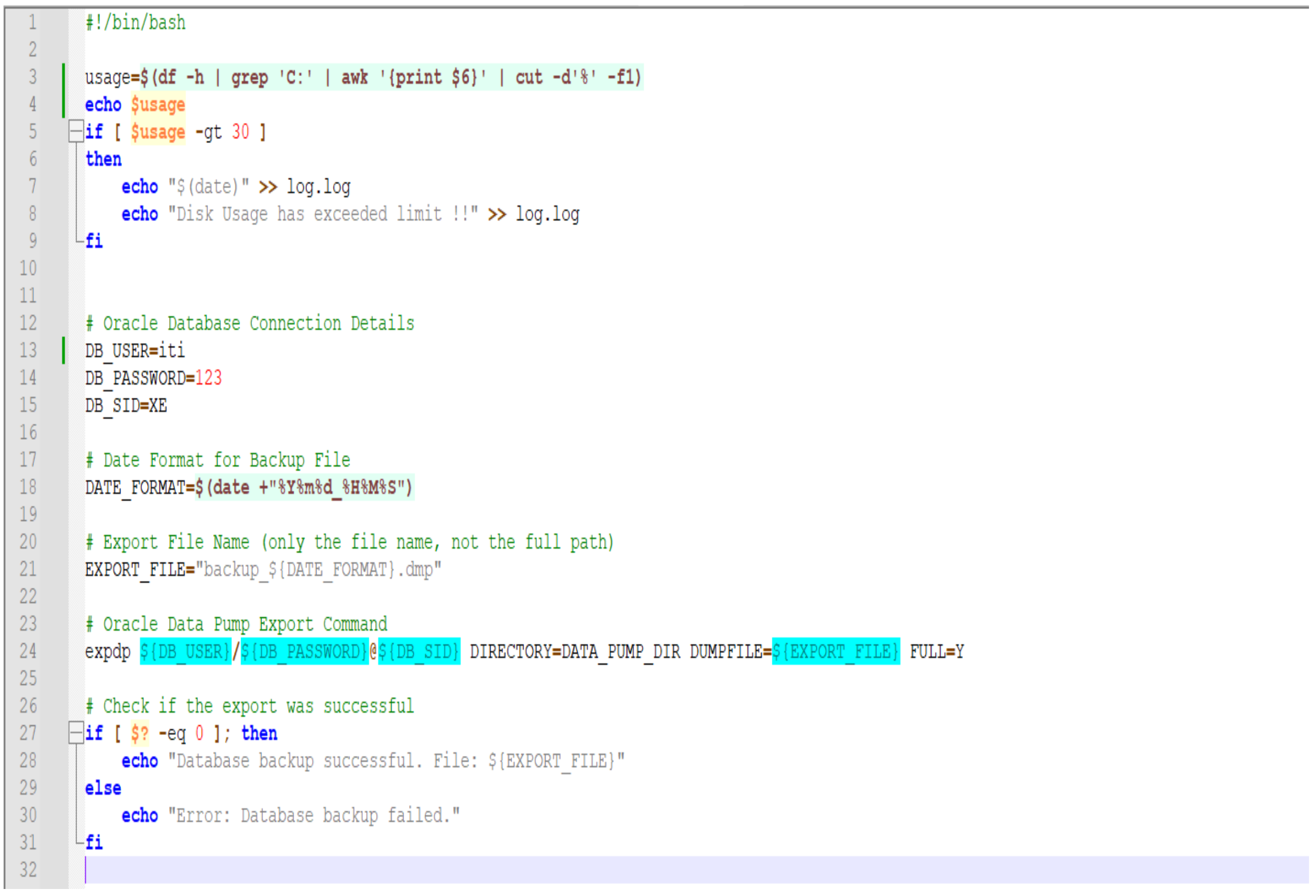
Details:

- **Event:** The trigger is fired before an **INSERT** operation on the **ITI.GRADES** table.
- **Referencing Clause:** The **REFERENCING** clause is used to reference the new and old values of the rows being affected.
- **Trigger Body:** In the trigger body, the **:new.G_ID** is assigned the next value from the **ITI.GRADES_SEQ** sequence.

5- Automation Script (BASH):

DISK MONITORING - Bash Script to monitor the Hard Disk usage and send alerts in case a specific threshold is exceeded => in a file named backup.sh. - The log.log file contains the disk monitoring and the backup logs.

DATABAS BACKUP - Perform a full backup of the database => the script is in a file named backup.sh. - The script.bat file is used by windows task scheduler to run the bash script.



```bash
#!/bin/bash

usage=$(df -h | grep 'C:' | awk '{print $6}' | cut -d'%' -f1)
echo $usage
if [ $usage -gt 30 ]
then
    echo "$(date)" >> log.log
    echo "Disk Usage has exceeded limit !!" >> log.log
fi


# Oracle Database Connection Details
DB_USER=iti
DB_PASSWORD=123
DB_SID=XE

# Date Format for Backup File
DATE_FORMAT=$(date +"%Y%m%d_%H%M%S")

# Export File Name (only the file name, not the full path)
EXPORT_FILE="backup_${DATE_FORMAT}.dmp"

# Oracle Data Pump Export Command
expdp ${DB_USER}/${DB_PASSWORD}@${DB_SID} DIRECTORY=DATA_PUMP_DIR DUMPFILE=${EXPORT_FILE} FULL=Y

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

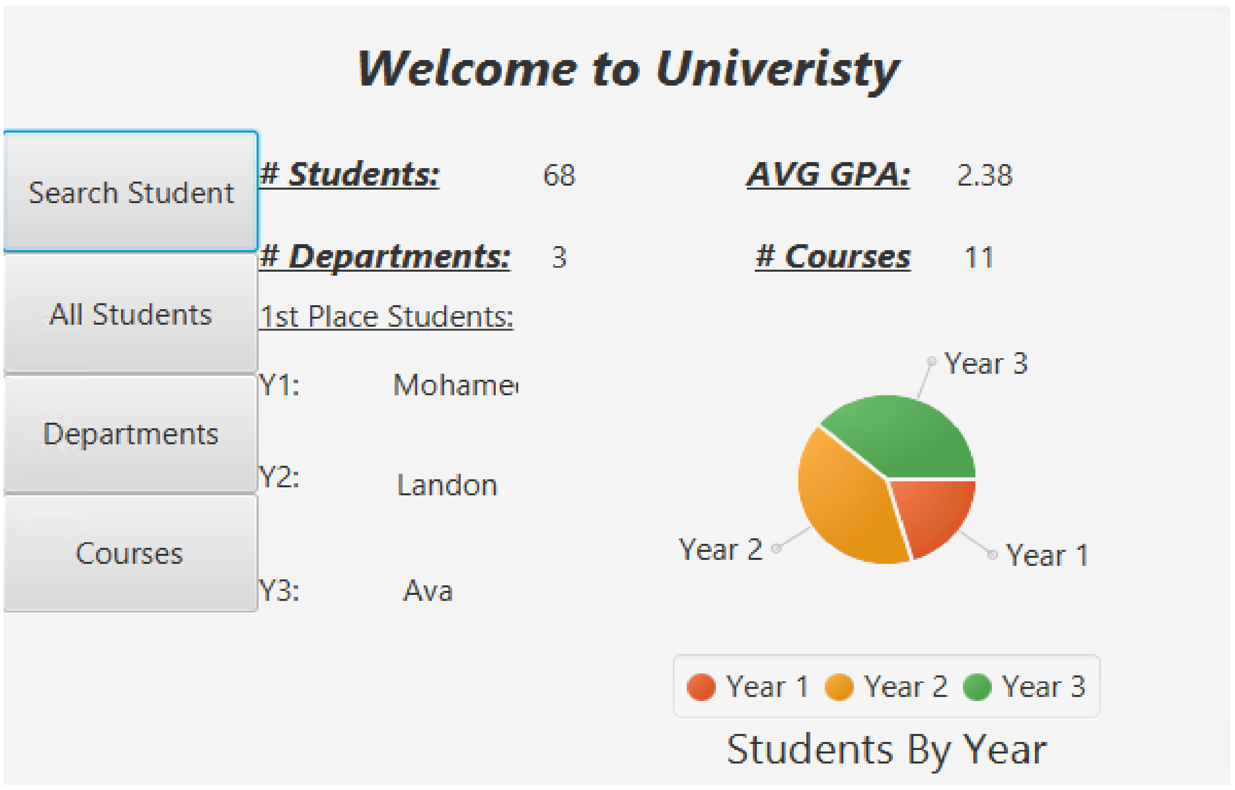
6- Java Application:

In my project I provided the **Java Source Code:**
This folder contains the source code for the Java application, enabling you to use any code editor for building and running the application.

- **Client:** Contains classes shared across applications (DTOs).
- **database:** Houses the Singleton class managing the Database Connection.
- **gui:** Encompasses code for all front-end (GUI) classes, including the source code for the application's backend.
- **Images:** Stores assets (pictures) used by the application.

The Application contain 5 main scenes:

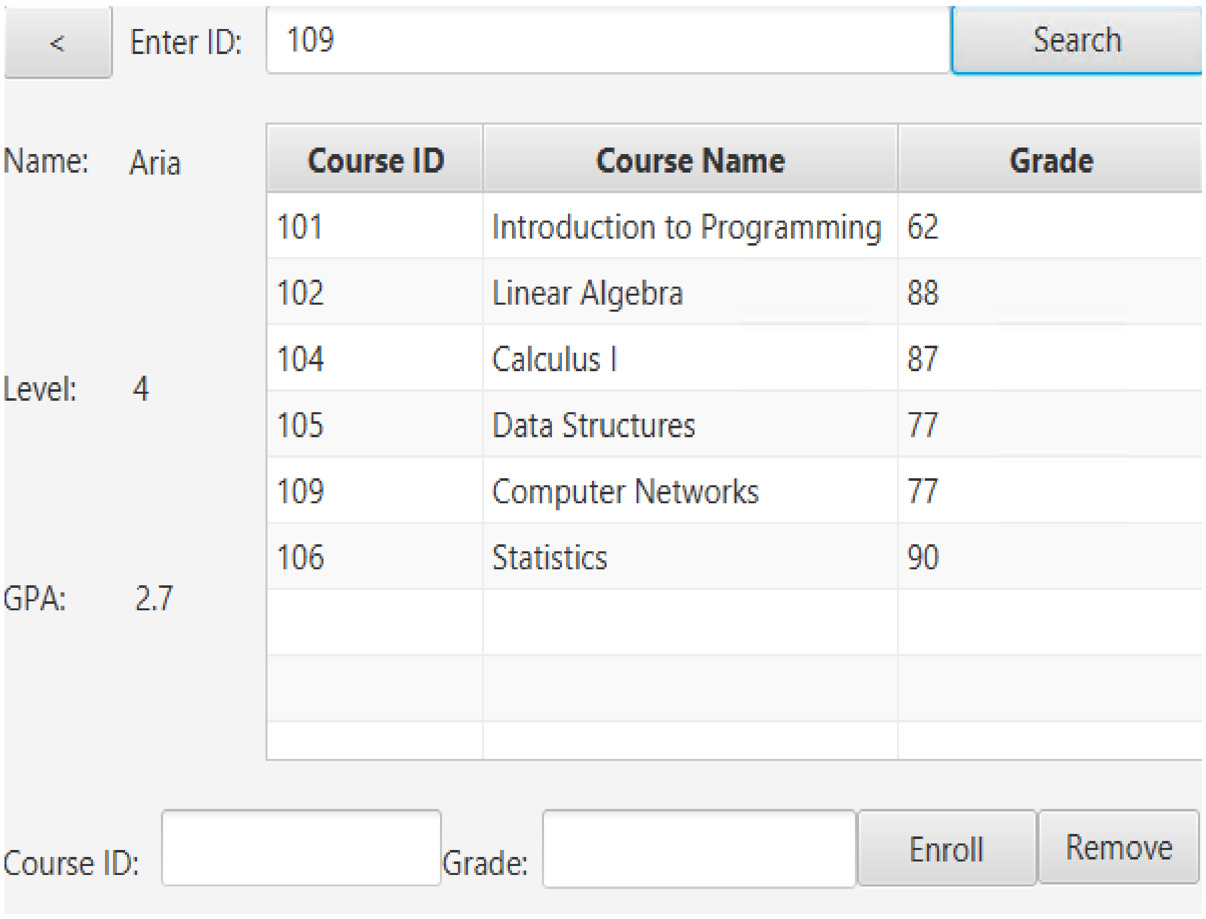
1- Report & Homepage:
This section provides comprehensive information and analysis concerning students, courses, departments, and student GPAs within the application.


Welcome to Univeristy

Students: 68
AVG GPA: 2.38
Departments: 3
Courses 11
1st Place Students:
Y1: Mohame
Y2: Landon
Y3: Ava

2- Search Students:

This scene offers detailed information on specific students, allowing users to view their complete profiles. Additionally, users can seamlessly enroll students in courses and grade those courses, all within the same interactive interface.



Enter ID: 109

Name: Aria

Level: 4

GPA: 2.7

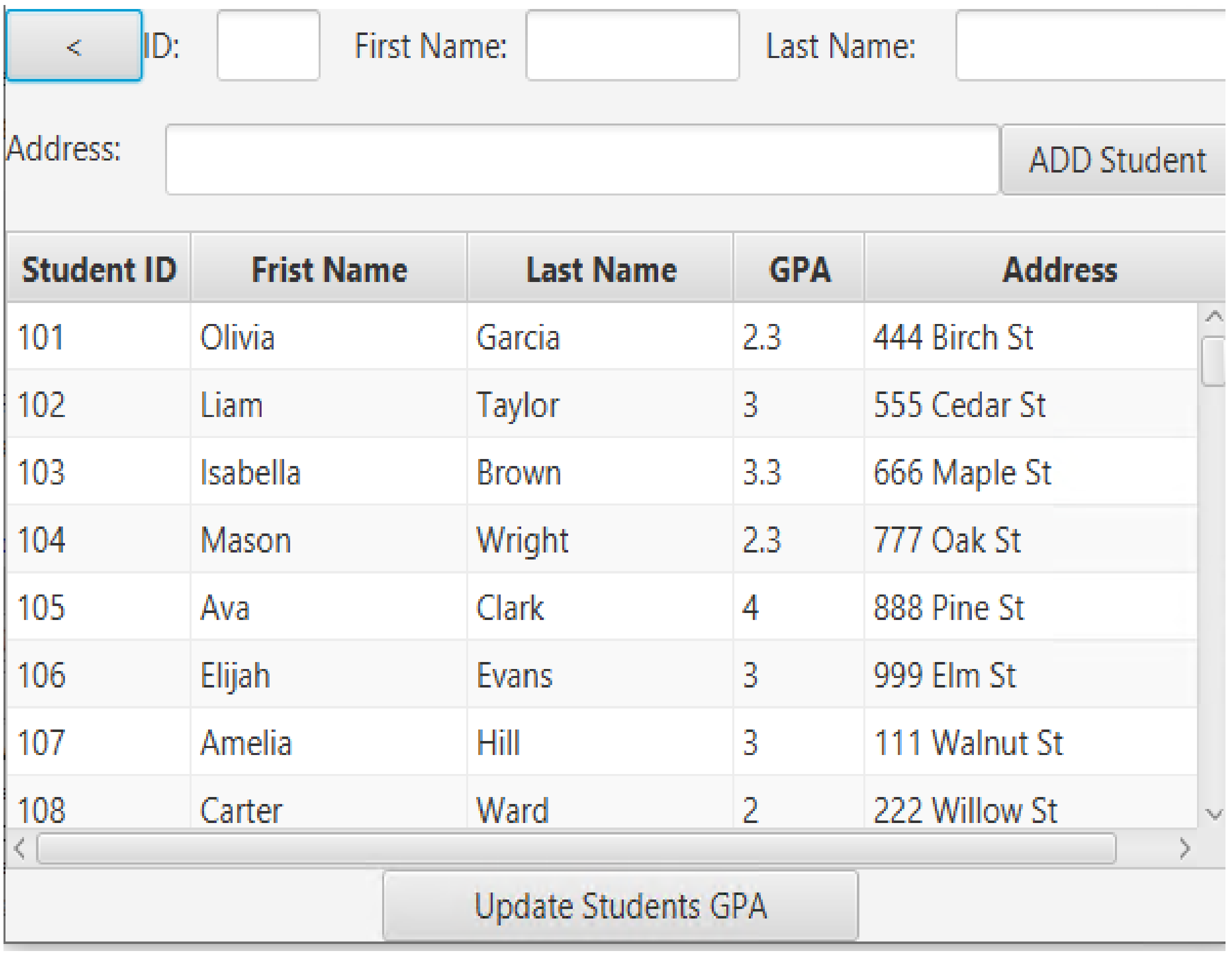
Course ID	Course Name	Grade
101	Introduction to Programming	62
102	Linear Algebra	88
104	Calculus I	87
105	Data Structures	77
109	Computer Networks	77
106	Statistics	90

Course ID:

Grade:

3- All Students:

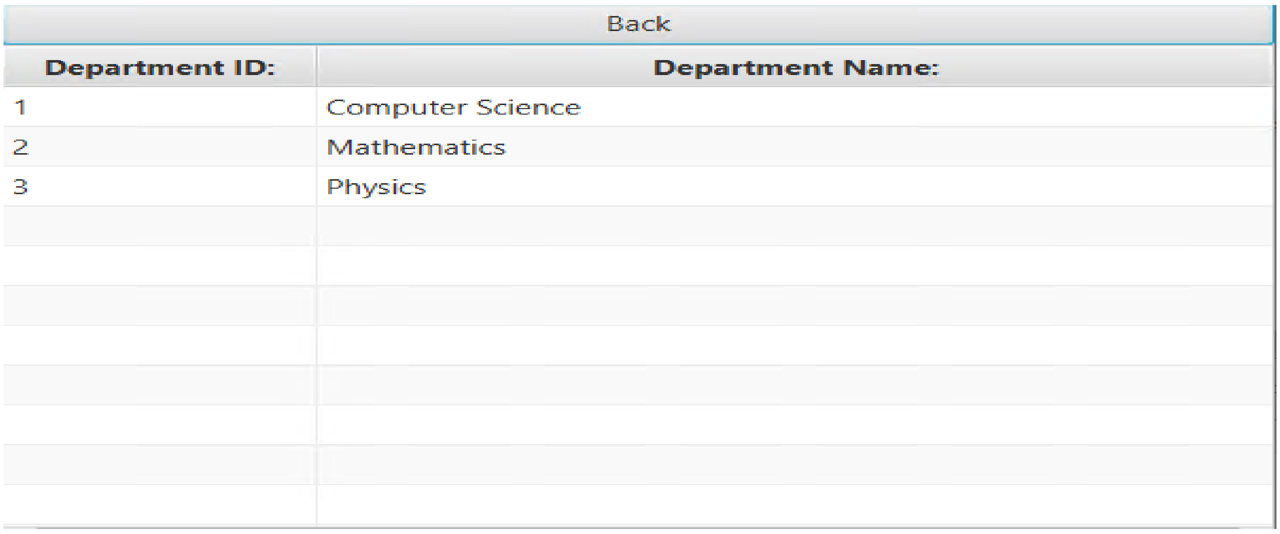
This scene serves as a central hub for accessing information on all students. Users can not only view existing data but also conveniently insert new students directly within the same interface. Additionally, it features the "Update Students GPA" function, seamlessly connecting with a database procedure for maintaining accurate student records.



Student ID	Frist Name	Last Name	GPA	Address
101	Olivia	Garcia	2.3	444 Birch St
102	Liam	Taylor	3	555 Cedar St
103	Isabella	Brown	3.3	666 Maple St
104	Mason	Wright	2.3	777 Oak St
105	Ava	Clark	4	888 Pine St
106	Elijah	Evans	3	999 Elm St
107	Amelia	Hill	3	111 Walnut St
108	Carter	Ward	2	222 Willow St

4- Departments:

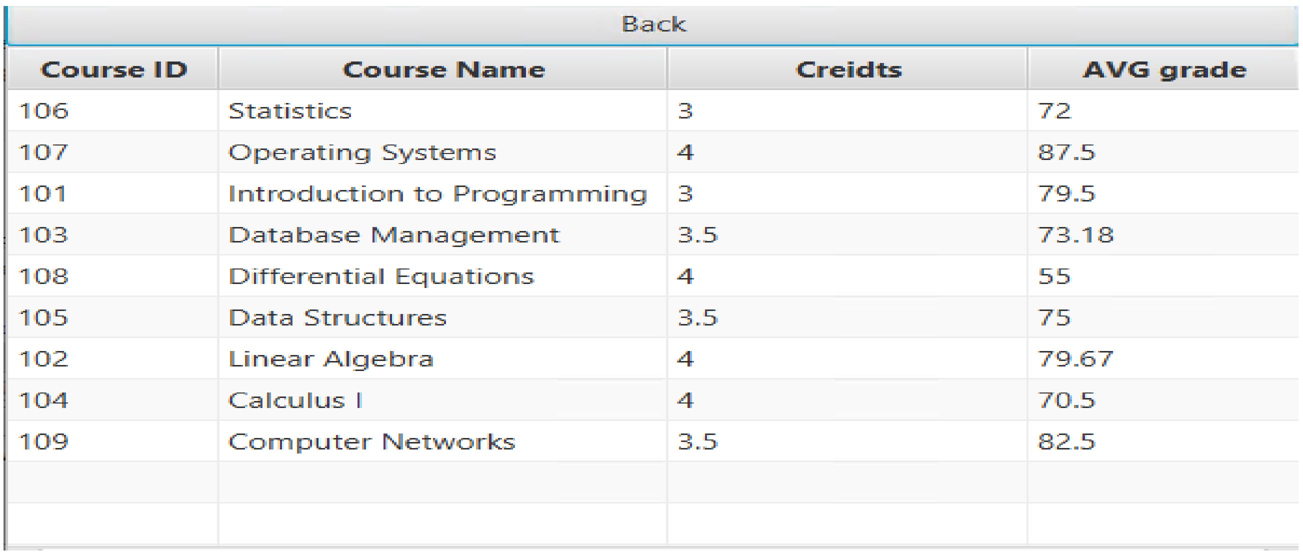
This scene provides a comprehensive display of departmental data, offering insights into various aspects related to each department within the application.



Department ID:	Department Name:
1	Computer Science
2	Mathematics
3	Physics

5- Courses:

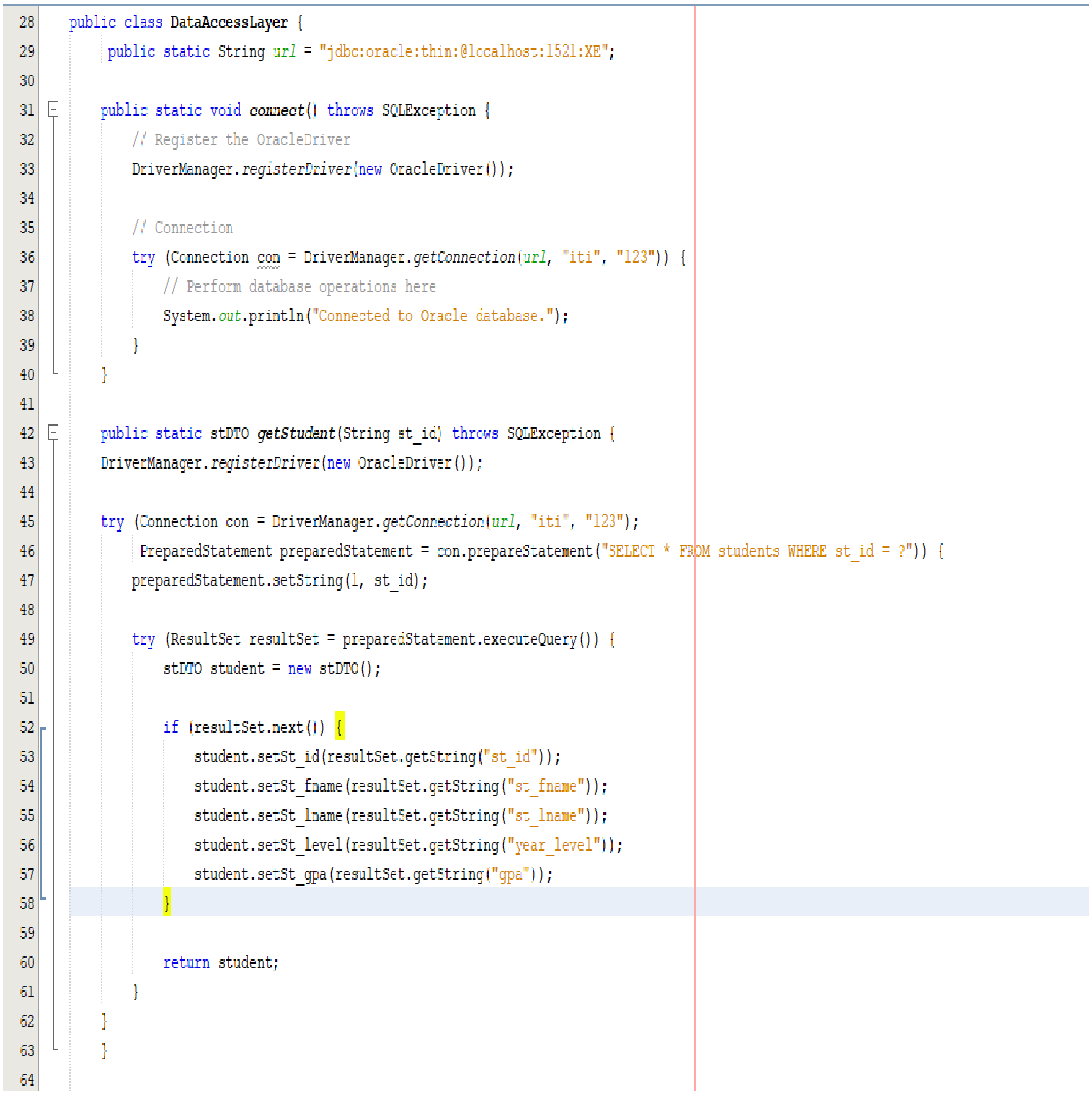
Within this scene, you can explore detailed information about courses, including relevant course data and the average grade associated with each course. This feature provides a quick overview of the performance metrics for individual courses within the application.



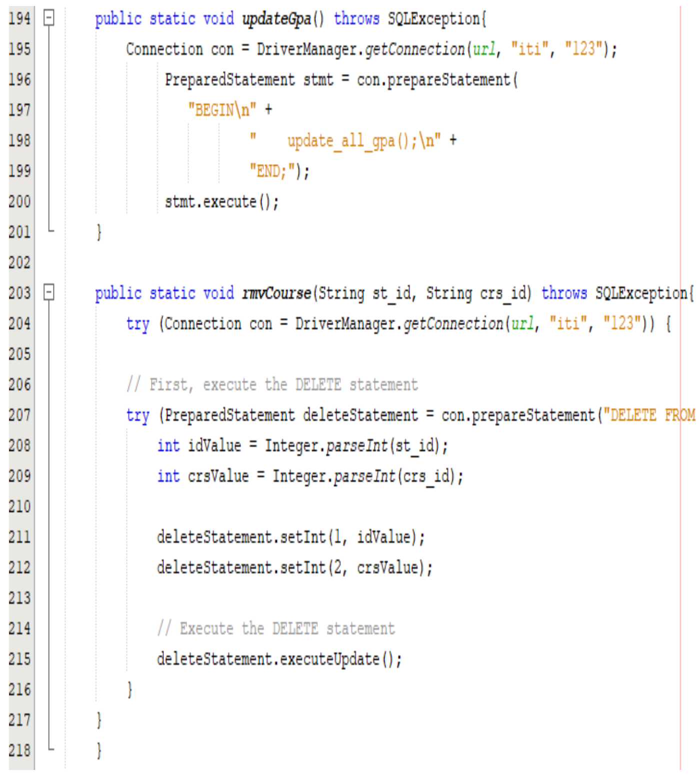
Course ID	Course Name	Creidts	AVG grade
106	Statistics	3	72
107	Operating Systems	4	87.5
101	Introduction to Programming	3	79.5
103	Database Management	3.5	73.18
108	Differential Equations	4	55
105	Data Structures	3.5	75
102	Linear Algebra	4	79.67
104	Calculus I	4	70.5
109	Computer Networks	3.5	82.5

DataAccessLayer:

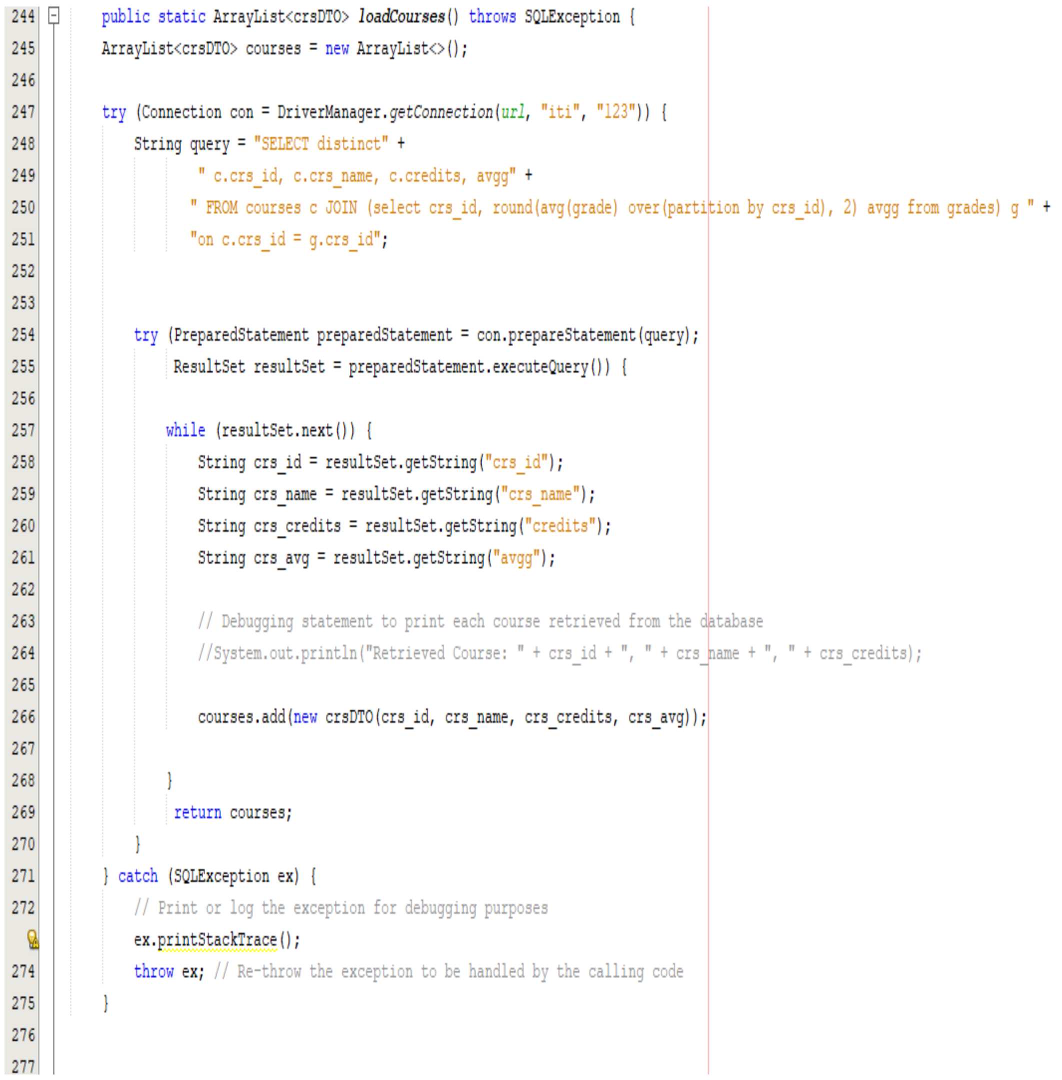
Having explored the Scenes in detail, let's now delve into the implementation by providing code snippets from the **Data access layer.** This will facilitate a deeper understanding of the integration between the scene and the underlying data functionality.



```java
public class DataAccessLayer {
     public static String url = "jdbc:oracle:thin:@localhost:1521:XE";

    public static void connect() throws SQLException {
        // Register the OracleDriver
        DriverManager.registerDriver(new OracleDriver());

        // Connection
        try (Connection con = DriverManager.getConnection(url, "iti", "123")) {
            // Perform database operations here
            System.out.println("Connected to Oracle database.");
        }
    }

    public static stDTO getStudent(String st_id) throws SQLException {
    DriverManager.registerDriver(new OracleDriver());

    try (Connection con = DriverManager.getConnection(url, "iti", "123");
         PreparedStatement preparedStatement = con.prepareStatement("SELECT * FROM students WHERE st_id = ?")) {
        preparedStatement.setString(1, st_id);

        try (ResultSet resultSet = preparedStatement.executeQuery()) {
            stDTO student = new stDTO();

            if (resultSet.next()) {
                student.setSt_id(resultSet.getString("st_id"));
                student.setSt_fname(resultSet.getString("st_fname"));
                student.setSt_lname(resultSet.getString("st_lname"));
                student.setSt_level(resultSet.getString("year_level"));
                student.setSt_gpa(resultSet.getString("gpa"));
            }

            return student;
        }
    }
    }
```



```java
    public static void updateGpa() throws SQLException{
        Connection con = DriverManager.getConnection(url, "iti", "123");
             PreparedStatement stmt = con.prepareStatement(
                "BEGIN\n" +
                        "    update_all_gpa();\n" +
                        "END;");
             stmt.execute();
    }

    public static void rmvCourse(String st_id, String crs_id) throws SQLException{
        try (Connection con = DriverManager.getConnection(url, "iti", "123")) {

        // First, execute the DELETE statement
        try (PreparedStatement deleteStatement = con.prepareStatement("DELETE FROM grades WHERE st_id = ? AND crs_id = ?")) {
            int idValue = Integer.parseInt(st_id);
            int crsValue = Integer.parseInt(crs_id);

            deleteStatement.setInt(1, idValue);
            deleteStatement.setInt(2, crsValue);

            // Execute the DELETE statement
            deleteStatement.executeUpdate();
        }
    }
    }
```



```java
    public static ArrayList<crsDTO> loadCourses() throws SQLException {
    ArrayList<crsDTO> courses = new ArrayList<>();

    try (Connection con = DriverManager.getConnection(url, "iti", "123")) {
        String query = "SELECT distinct" +
                " c.crs_id, c.crs_name, c.credits, avgg" +
               " FROM courses c JOIN (select crs_id, round(avg(grade) over(partition by crs_id), 2) avgg from grades) g " +
               "on c.crs_id = g.crs_id";


        try (PreparedStatement preparedStatement = con.prepareStatement(query);
             ResultSet resultSet = preparedStatement.executeQuery()) {

            while (resultSet.next()) {
                String crs_id = resultSet.getString("crs_id");
                String crs_name = resultSet.getString("crs_name");
                String crs_credits = resultSet.getString("credits");
                String crs_avg = resultSet.getString("avgg");

                // Debugging statement to print each course retrieved from the database
                //System.out.println("Retrieved Course: " + crs_id + ", " + crs_name + ", " + crs_credits);

                courses.add(new crsDTO(crs_id, crs_name, crs_credits, crs_avg));

            }
             return courses;
        }
    } catch (SQLException ex) {
        // Print or log the exception for debugging purposes
        ex.printStackTrace();
        throw ex; // Re-throw the exception to be handled by the calling code
    }

```