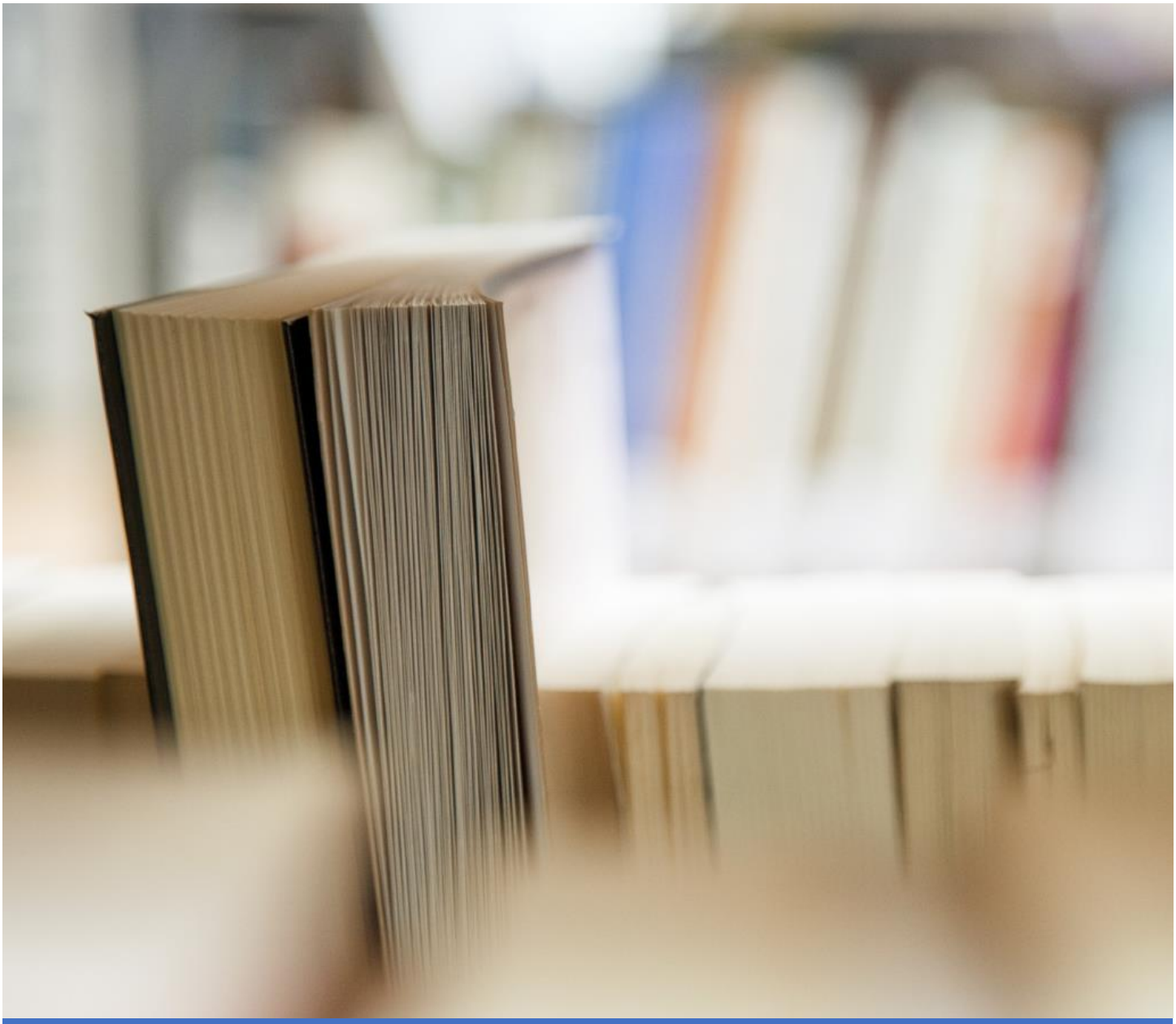


University Management System

Ahmed Ali

3/2/24

ITI Data Management Intake 44

1. Introduction

2. Database design
    - a relational database schema including the tables.
    - Normalize the schema to ensure data integrity.

3. SQL Implementation
    - Create SQL scripts to build the database schema.
    - make suitable constraints to ensure data consistency and integrity.

4. PLSQL Implementation
    - Creating some procedures and functions.
    - Creating Triggers to ensure that we'll never lose any data.

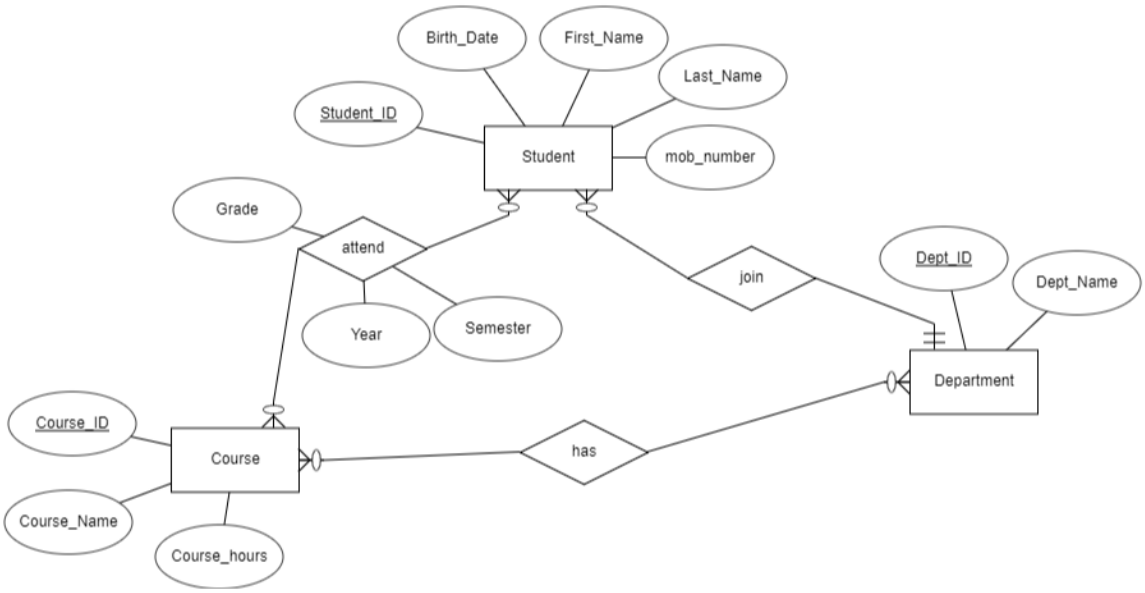
5. Automation Scripts
    - Creating Bash scripts for database Backup.
    - Bash script for monitoring disk space and sending alerts and schedule it.

6. JAVA Application Development
    - Implement CRUD operations in the application.
    - Providing Reports and Statistics about students.
    - Integrate with the database to ensure consistency.

1.Introduction

The objective of this project is to design and implement a comprehensive data management system for a university using SQL, PLSQL, Advanced PLSQL, Red Hat, Bash scripting, Java SE, and OOP principles. The project encompasses various aspects, including database design, SQL and PLSQL implementation, automation scripts, Java application development, and integration.

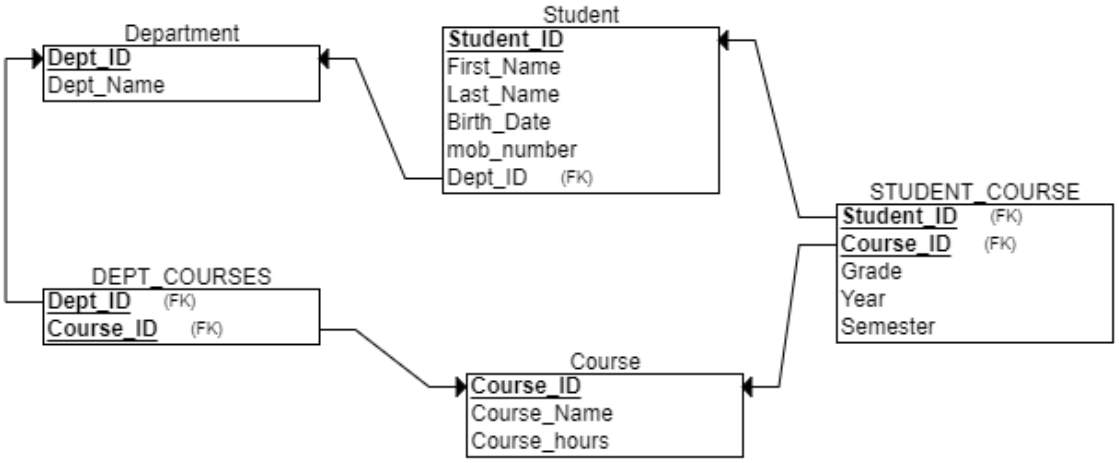
2. Database design



Our ERD illustrate that:

1. Every student attends many courses and courses are offered to many students.
2. Every Department has many students, but the students join one department.
3. The same course exists in many departments and departments have many courses.

In the next stage we map this ERD to create our relational schema and get our tables to be ready in the **3rd normalization form** to implement it using SQL in the database.



As we see we have 5 tables to include our needs in the database:

1. **Many to many relationship** generate three tables (Student – Course – Student Course).
2. The same is true with Department and Course generate (Department – Course – Dept Courses).
3. **One to many relationship** convert to table (Department) and Dept_ID as foreign key in Student table.

4. SQL Implementation

Here's the SQL script to create our database tables.

```
Dept_Name VARCHAR2(100) NOT NULL,
  PRIMARY KEY (Dept_ID)
);

CREATE TABLE Course
(
  Course_ID INT NOT NULL,
  Course_Name INT VARCHAR2(100) NOT NULL,
  Course_hours INT NOT NULL,
  PRIMARY KEY (Course_ID)
);
```

```sql
CREATE TABLE DEPT_COURSES
(
  Dept_ID INT NOT NULL,
  Course_ID INT NOT NULL,
  PRIMARY KEY (Dept_ID, Course_ID),
  FOREIGN KEY (Dept_ID) REFERENCES Department(Dept_ID),
  FOREIGN KEY (Course_ID) REFERENCES Course(Course_ID)
);

CREATE TABLE Student
(
  Student_ID INT NOT NULL,
  First_Name VARCHAR2(100) NOT NULL,
  Last_Name INT VARCHAR2(100) NOT NULL,
  Birth_Date DATE NOT NULL,
  mob_number INT NOT NULL,
  Dept_ID INT NOT NULL,
  PRIMARY KEY (Student_ID),
  FOREIGN KEY (Dept_ID) REFERENCES Department(Dept_ID)
);

CREATE TABLE STUDENT_COURSE
(
  Student_ID INT NOT NULL,
  Course_ID INT NOT NULL,
  Grade INT NOT NULL,
  Year INT NOT NULL,
  Semester INT NOT NULL,
  PRIMARY KEY (Student_ID, Course_ID),
  FOREIGN KEY (Student_ID) REFERENCES Student(Student_ID),
  FOREIGN KEY (Course_ID) REFERENCES Course(Course_ID)
);
```

Here's the last table as an Archive table to receive the deleted data as the data is a valuable thing we've and can't let it deleted without having a backup.

```sql
CREATE TABLE UNIVERSITY.Grades_Archive
(
  STUDENT_ID    INTEGER   ,
  COURSE_ID     INTEGER  ,
  GRADE         NUMBER(4,2)  ,
  YEAR          INTEGER   ,
  SEMESTER      INTEGER
);
```

Also, we've some constraints on the tables to ensure integrity and consistency.

Constraints on delete from table DEPARTMENT :

```
ALTER TABLE university.Dept_courses
ADD CONSTRAINT fk_dept_courses_dept_id
FOREIGN KEY (dept_ID)
REFERENCES university.Department(dept_ID)
ON DELETE CASCADE;


ALTER TABLE university.Student
ADD CONSTRAINT fk_student_dept_id
FOREIGN KEY (dept_ID)
REFERENCES university.Department(dept_ID)
ON DELETE SET NULL;
```

Constraints on delete from table STUDENT_COURSE :

```
ALTER TABLE UNIVERSITY.STUDENT_COURSE ADD (
  CONSTRAINT CONSTRAINT_DELETE
 FOREIGN KEY (STUDENT_ID)
 REFERENCES UNIVERSITY.STUDENT
    ON DELETE CASCADE);


ALTER TABLE UNIVERSITY.STUDENT_COURSE
ADD CONSTRAINT CONSTRAINT_DELETE_Course
FOREIGN KEY (COURSE_ID)
REFERENCES UNIVERSITY.COURSE(COURSE_ID)
ON DELETE CASCADE;
```

5. PLSQL Implementation

Here's a function to calculate the total GPA of the students.

```
CREATE OR REPLACE function UNIVERSITY.calc_total_GPA(v_student_id number )
return number
is
       V_SUM number(8, 2);        v_total number(8, 2);
       V_COUNT number(8, 2);       V_GPA number(8, 2);
begin
    select sum(GRADE) ,count(GRADE)
    into V_SUM , V_COUNT
    from UNIVERSITY.STUDENT_COURSE
    where STUDENT_ID = v_student_id
    Group by STUDENT_ID ;
     v_total := V_COUNT * 100;
```

```
   V_GPA := V_SUM / v_total * 100;
   return V_GPA;
end;
```

Also, another procedure to update any information about the student.

```
CREATE OR REPLACE procedure UNIVERSITY.update_student_info
   (v_student_id INTEGER, V_fname VARCHAR2, V_lname VARCHAR2, B_DATE Date,
V_MOB_NUMBER  VARCHAR2, V_DEPT_ID INTEGER)
is
begin
       UPDATE UNIVERSITY.Student
       SET
         FIRST_NAME = V_fname,
         LAST_NAME = V_lname,
         BIRTH_DATE = B_DATE,
         MOB_NUMBER = V_MOB_NUMBER,
         DEPT_ID = V_DEPT_ID
       WHERE STUDENT_ID = v_student_id;
end;
```

And one of the most crucial things to protect our data from loss, we've a trigger contains a procedure to fire when we delete anything from table STUDENT_COURSE or any other related table as STUDENT or COURSE.

The trigger runs a procedure to insert the data from STUDENT_COURSE table to Archive table we made to save the grades of the student even if we delete the course or the student.

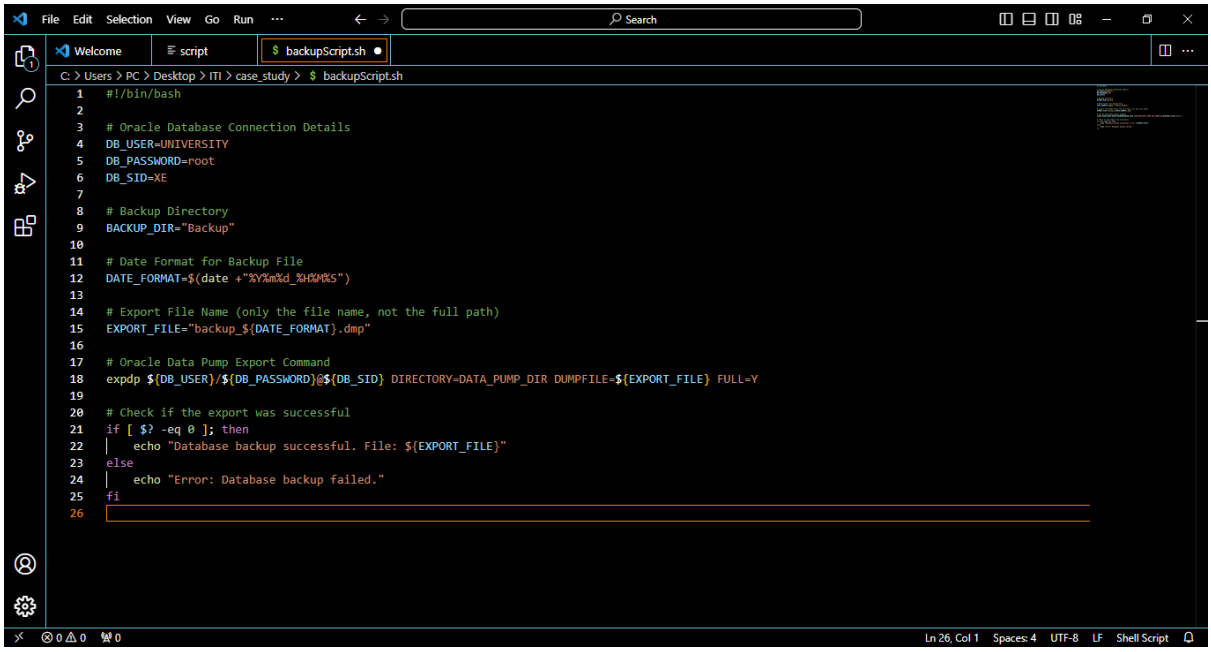
```
CREATE OR REPLACE TRIGGER UNIVERSITY.STUDENT_COURSE_TRG
Before Delete ON UNIVERSITY.STUDENT_COURSE
FOR EACH ROW
BEGIN
   UNIVERSITY.ADD_To_Archive(:old.STUDENT_ID, :old.COURSE_ID, :old.GRADE, :old.YEAR,
:old.SEMESTER);
END;
```

```
CREATE OR REPLACE PROCEDURE UNIVERSITY.ADD_To_Archive
  (  p_std_id          Grades_Archive.STUDENT_ID%type
   , p_course_id       Grades_Archive.COURSE_ID%type
   , p_grade         Grades_Archive.GRADE%type
   , p_year          Grades_Archive.YEAR%type
   , p_semster    Grades_Archive.SEMESTER%type
   )
IS
BEGIN
  INSERT INTO Grades_Archive (STUDENT_ID, COURSE_ID, GRADE,
```

```
                    YEAR, SEMESTER)
  VALUES(p_std_id, p_course_id, p_grade, p_year, p_semster);
END;
```

6. Automation scripts

We've this script to backup the data



Shebang (#!/bin/bash):

Indicates that this script should be executed using the Bash shell.

Database Connection Details:

DB_USER: Oracle database username.

DB_PASSWORD: Oracle database password.

DB_SID: Oracle database System Identifier (SID).

Backup Directory:

BACKUP_DIR: Directory where the backup file will be stored.

Date Format for Backup File:

DATE_FORMAT: Current date and time formatted as YYYYMMDD_HHMMSS.

Export File Name:

EXPORT_FILE: Name of the export file using the specified date format.

Oracle Data Pump Export Command (expdp):

Exports the entire database (FULL=Y) using Oracle Data Pump.

The export file is named according to the specified format.

Check Export Status:

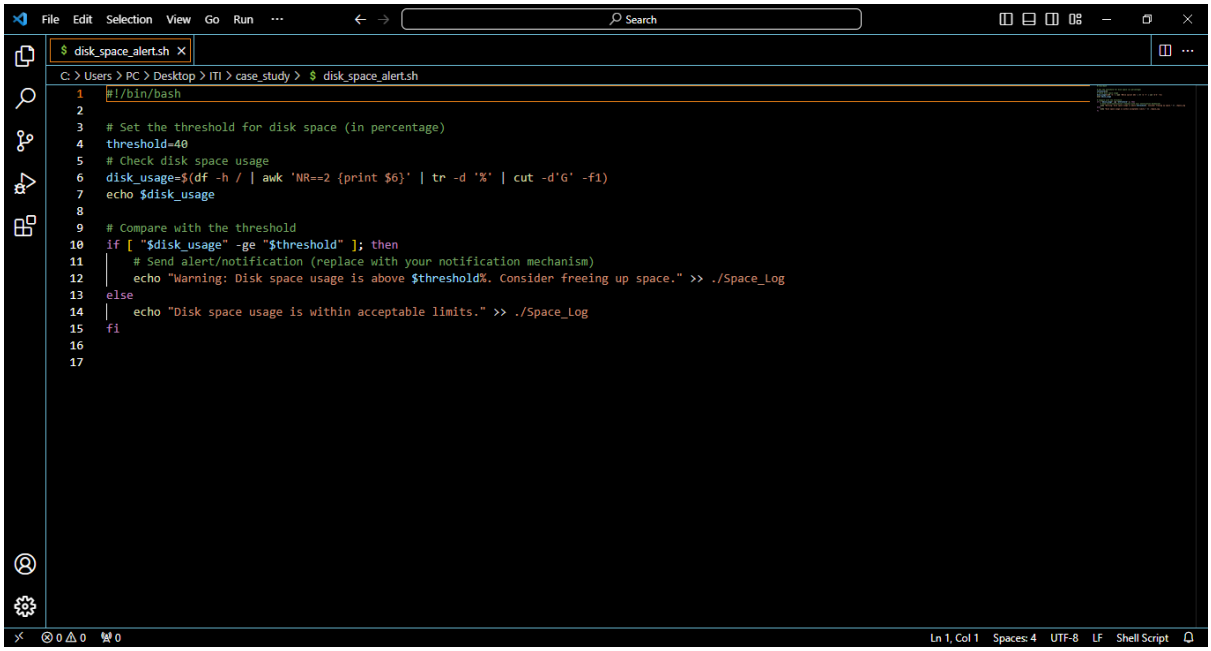
Checks the exit status of the previous command (expdp).

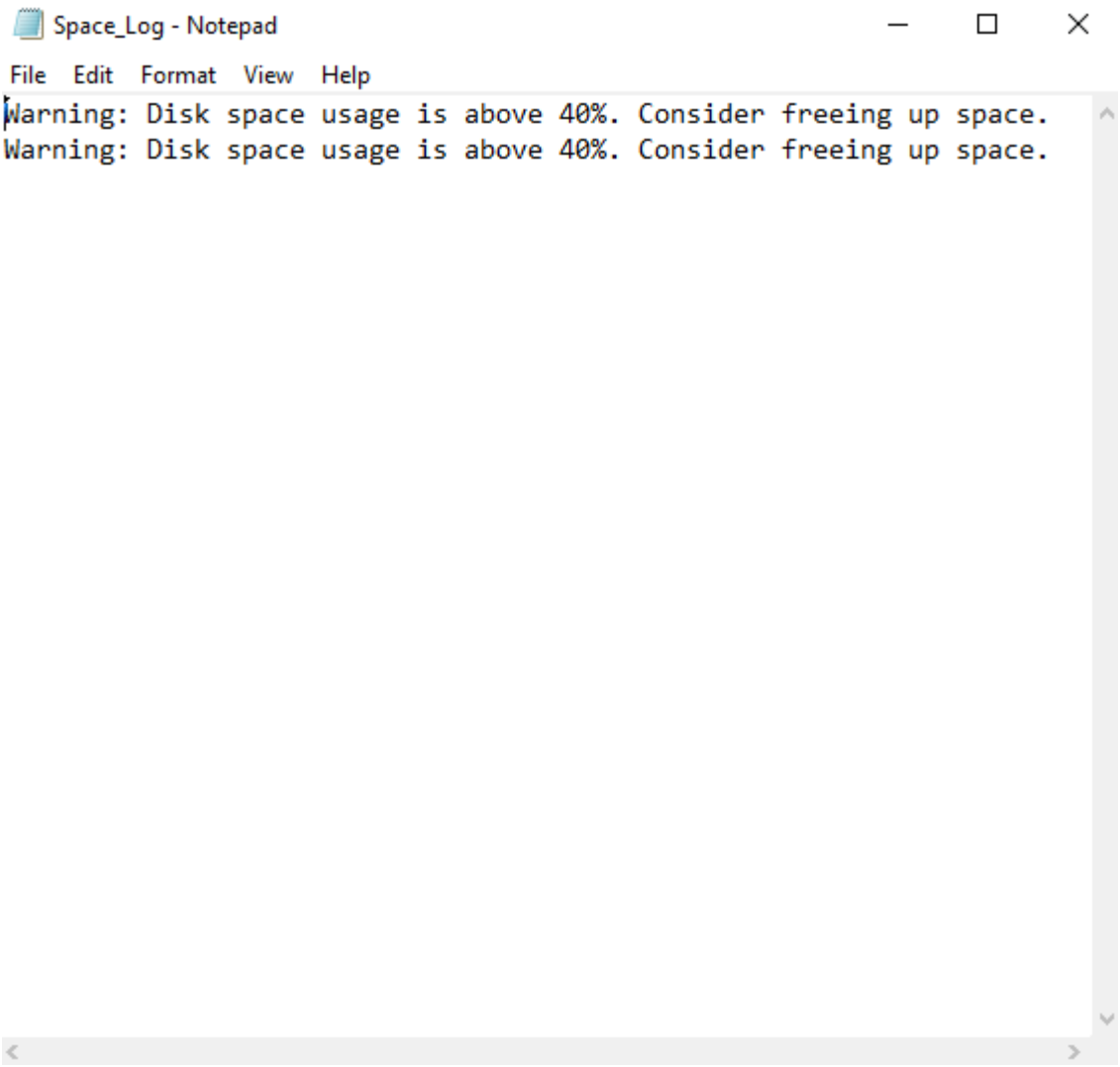
If the exit status is 0, the export was successful; otherwise, it indicates an error.

Echo Messages:

Prints a message indicating whether the database backup was successful or if there was an error.

We've this script for monitoring disk space and sending alerts





Shebang (#!/bin/bash):

Indicates that this script should be executed using the Bash shell.

Set Threshold for Disk Space:

threshold: The threshold percentage for disk space usage. If the disk space usage exceeds this threshold, a warning will be generated.

Check Disk Space Usage:

disk_usage: Uses the df command to check the disk space usage of the root file system ("/").

The awk command extracts the usage percentage (e.g., "75%").

tr -d '%' removes the percentage symbol.

cut -d'G' -f1 extracts the numeric value (e.g., "75").

Compare with the Threshold:

Compares the disk usage with the defined threshold using an if statement.

Generate Alert/Notification:

If disk usage is above the threshold, an alert/notification is generated.

The alert is appended to a file named **Space_Log** in the current directory.

7. JAVA Application Development

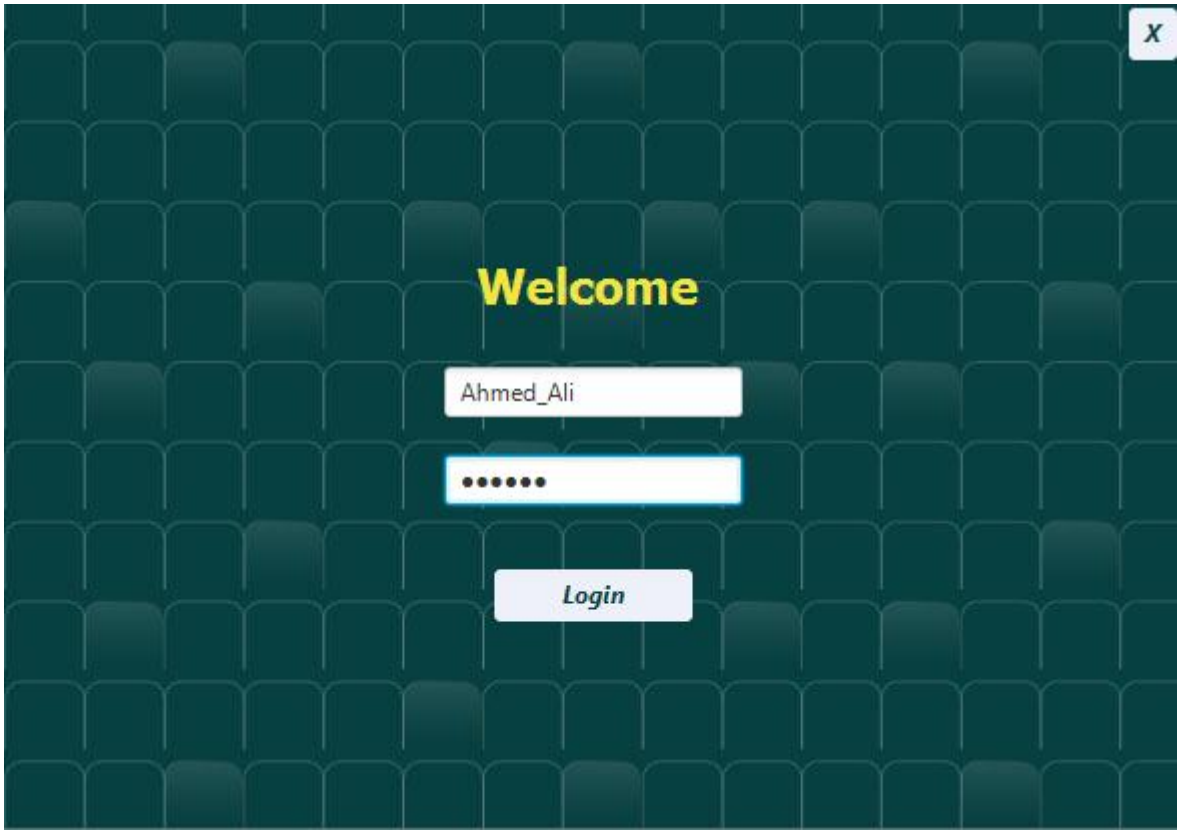
In the Java application we've integrated the database with the application which consists of components as:

- **Data Access Layer**: to ensure single tone concept which allow us to make a single connection with the database.

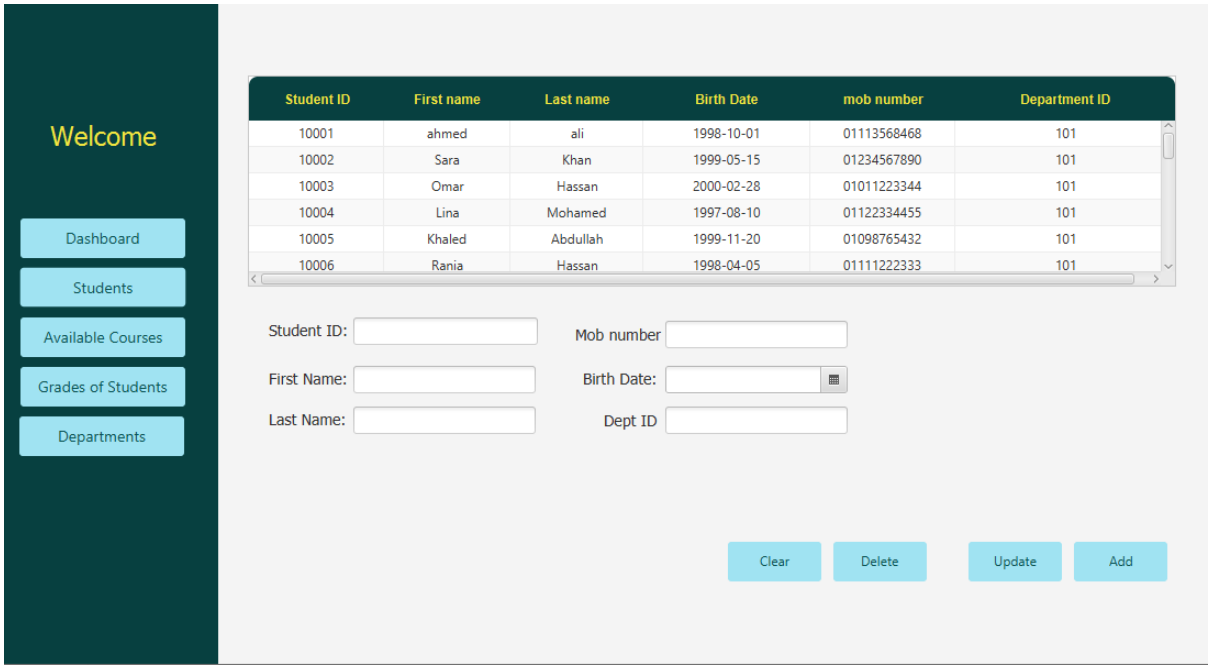
```java
 * and open the template in the editor.
 */
package universitymanagementsystem;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

/**
 *
 * @Ahmed_Ali
 */
public class database {

    public static Connection connectDb() {
    try {
        String url = "jdbc:oracle:thin:@localhost:1521:XE";
        String user = "UNIVERSITY";
        String password = "root";


        Connection connect = DriverManager.getConnection(url, user, password);
        return connect;
    } catch (SQLException e) {
        e.printStackTrace();

    }
    return null;
}

}
```

- Java classes **DTO's** for database tables.
- **Controllers** to make the functions and backend we need in the application.
- **FXML** to make the Scenes UI and integrate it with the application.

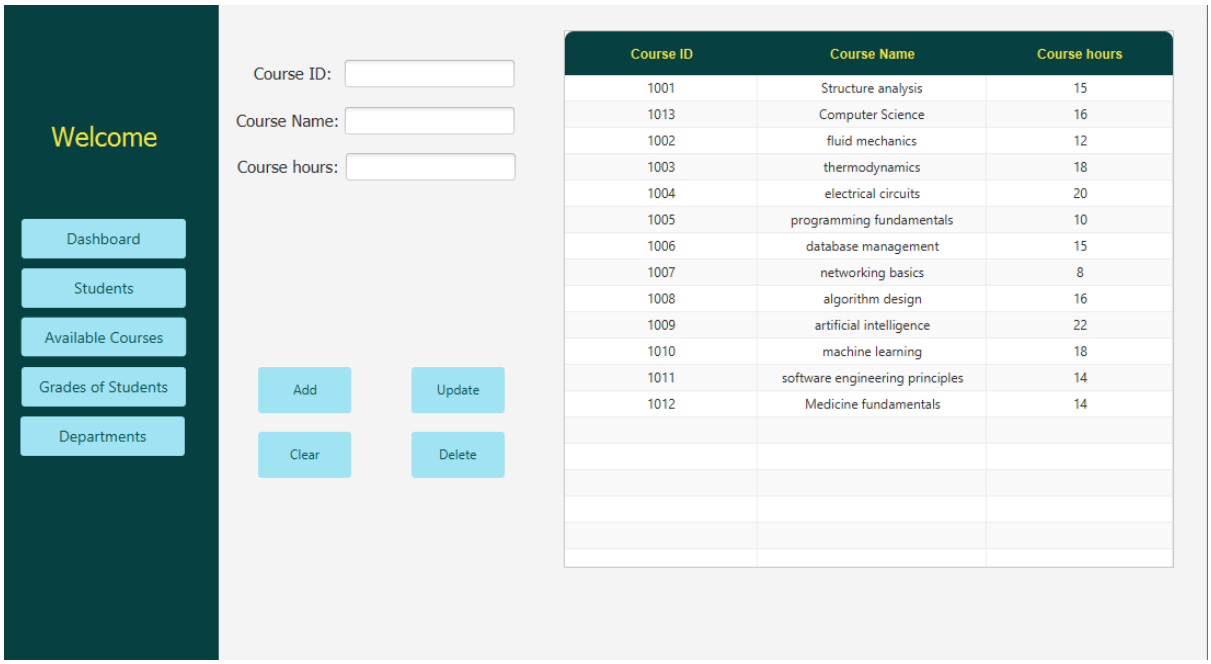
First, we've the login Scene



Here's the Student's Scene to fill their personal information and assign them for their department and it allows us to (Update, Add, Delete) anything about them and clear to clear the text fields if we want.



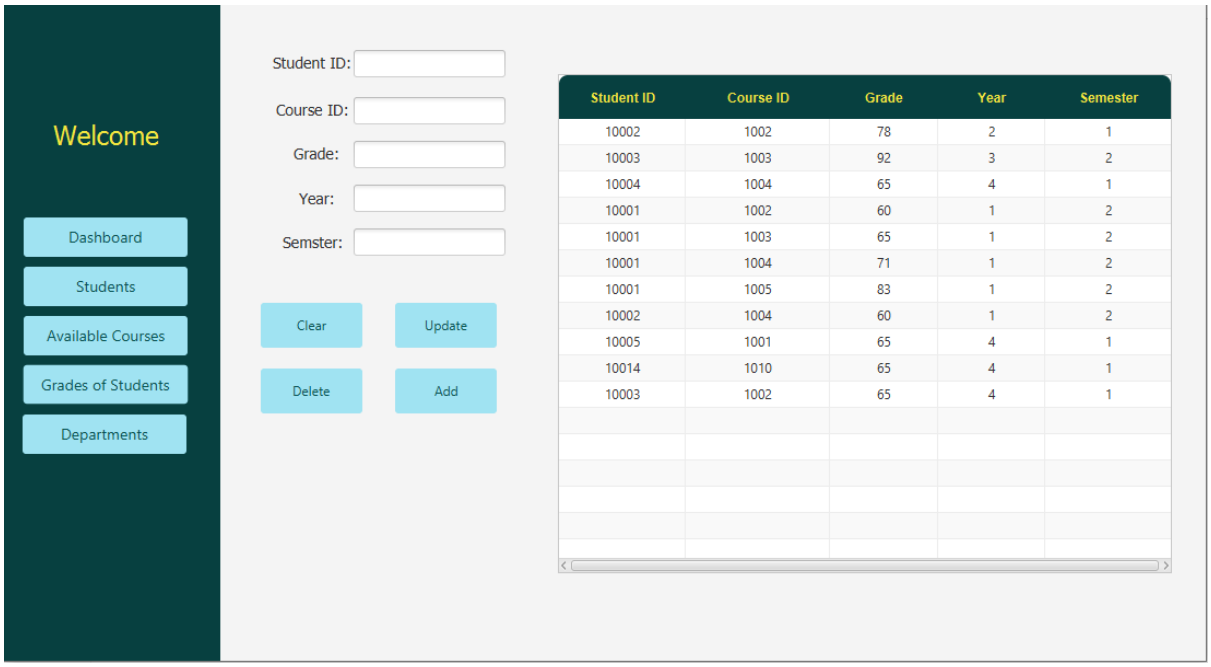
Available Courses Scene to (add, update, delete) any course we want.



Grade of Students Scene to add courses to student or update their Grades or even if we want to delete anything.

We ensure that you can't add a course to a student if his department courses doesn't contain the course.

Also, the range of Grades you can add between (0-100) or it'll display an Error Alert.

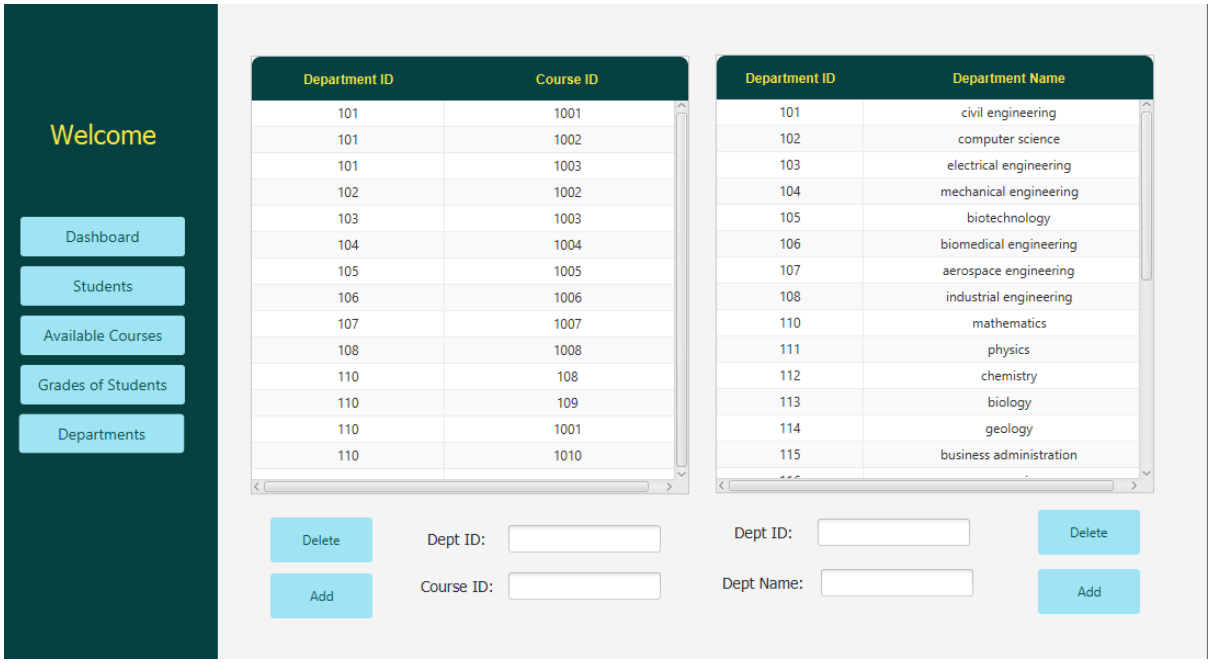



```
            || studentGrade_Sem.getText().isEmpty() ) {
        alert = new Alert(AlertType.ERROR);
        alert.setTitle("Error Message");
        alert.setHeaderText(null);
        alert.setContentText("Please fill all blank fields");
        alert.showAndWait();
    } else {

        try {
        int gradeValue = Integer.parseInt(studentGrade_Grade.getText());
        if (gradeValue < 0 || gradeValue > 100) {
            throw new NumberFormatException();
        }
    } catch (NumberFormatException ex) {
        alert = new Alert(Alert.AlertType.ERROR);
        alert.setTitle("Error Message");
        alert.setHeaderText(null);
        alert.setContentText("Please enter a valid grade value in Range (0-100)");
        alert.showAndWait();
        return;
    }

        alert = new Alert(AlertType.CONFIRMATION);
        alert.setTitle("Confirmation Message");
        alert.setHeaderText(null);
        alert.setContentText("Are you sure you want to UPDATE ?");
```

```java
            return;
        }
            String checkProgram = "SELECT COUNT(*) AS Count \n" +
                                  "FROM UNIVERSITY.STUDENT S JOIN UNIVERSITY.DEPT_COURSES D\n" +
                                  "ON S.DEPT_ID = D.DEPT_ID \n" +
                                  "WHERE S.STUDENT_ID = ? AND D.COURSE_ID = ? ";

                prepare = connect.prepareStatement(checkProgram);
                prepare.setString(1, studentGrade_studentID.getText());
                prepare.setString(2, studentGrade_studentCourseID.getText());

                result = prepare.executeQuery();


            if (result.next() && result.getInt("Count") > 0) {
            prepare = connect.prepareStatement(insertData);
            prepare.setString(1, studentGrade_studentID.getText());
            prepare.setString(2, studentGrade_studentCourseID.getText());

            String gradeValue = studentGrade_Grade.getText();
                if (gradeValue.isEmpty()) {
                    prepare.setNull(3, Types.INTEGER); // Setting GRADE to null
             } else {
                    int grade = Integer.parseInt(gradeValue);
                    prepare.setInt(3, grade);
             }
            prepare.setString(4, studentGrade_Year.getText());
            prepare.setString(5, studentGrade_Sem.getText());

            prepare.executeUpdate();
```

Departments Scene where you can add or delete department.

Also, you can add courses to department or delete them.



Dashboard Scene where you can get a **summary** of students Average GPA in a specific course or the number of students that have enrolled in this course.