



University Management System

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.

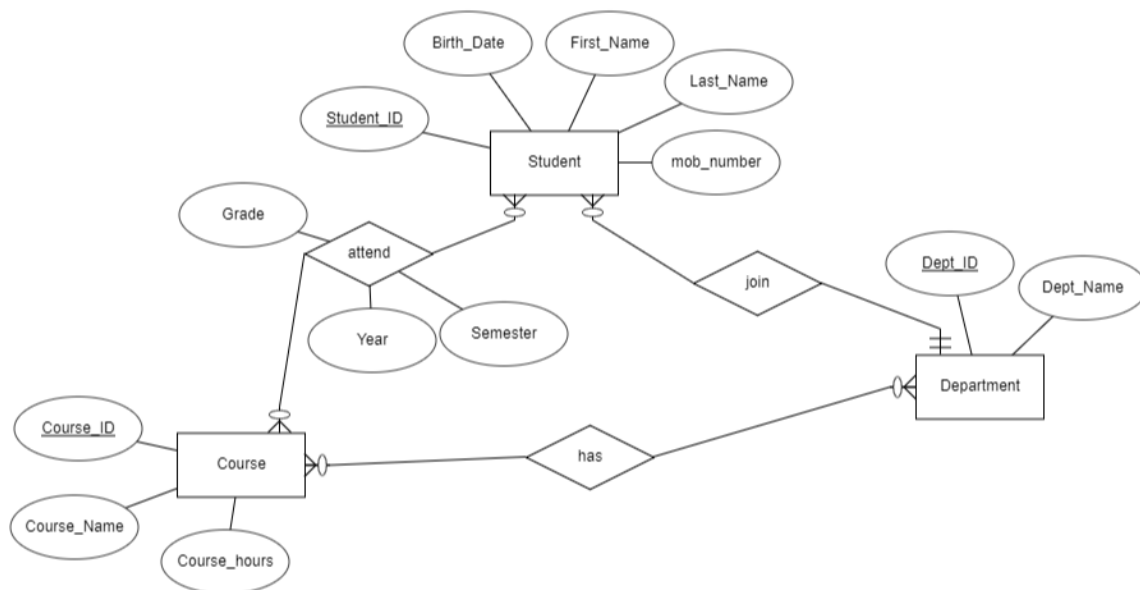
6. JAVA Application Development

- 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.

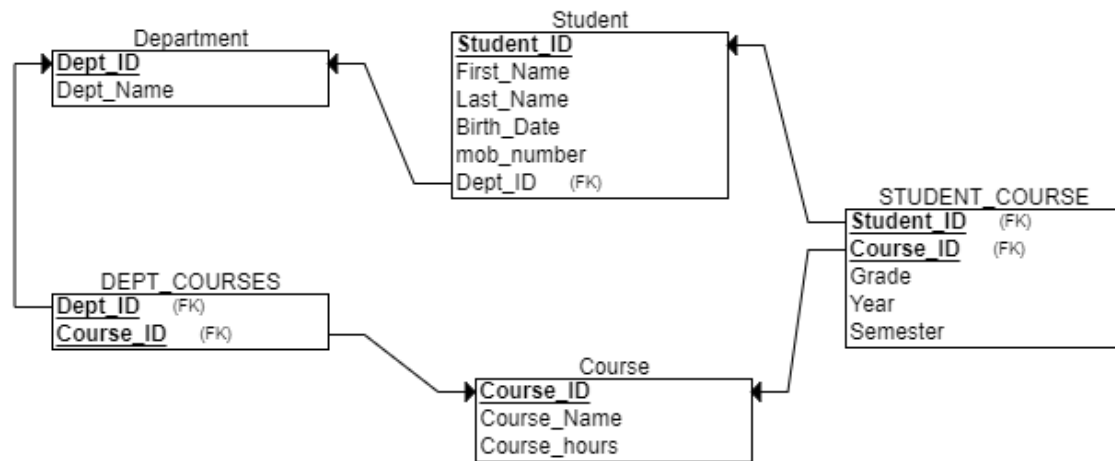
2. Database design



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)
);
  
```

```

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.

```

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,

```

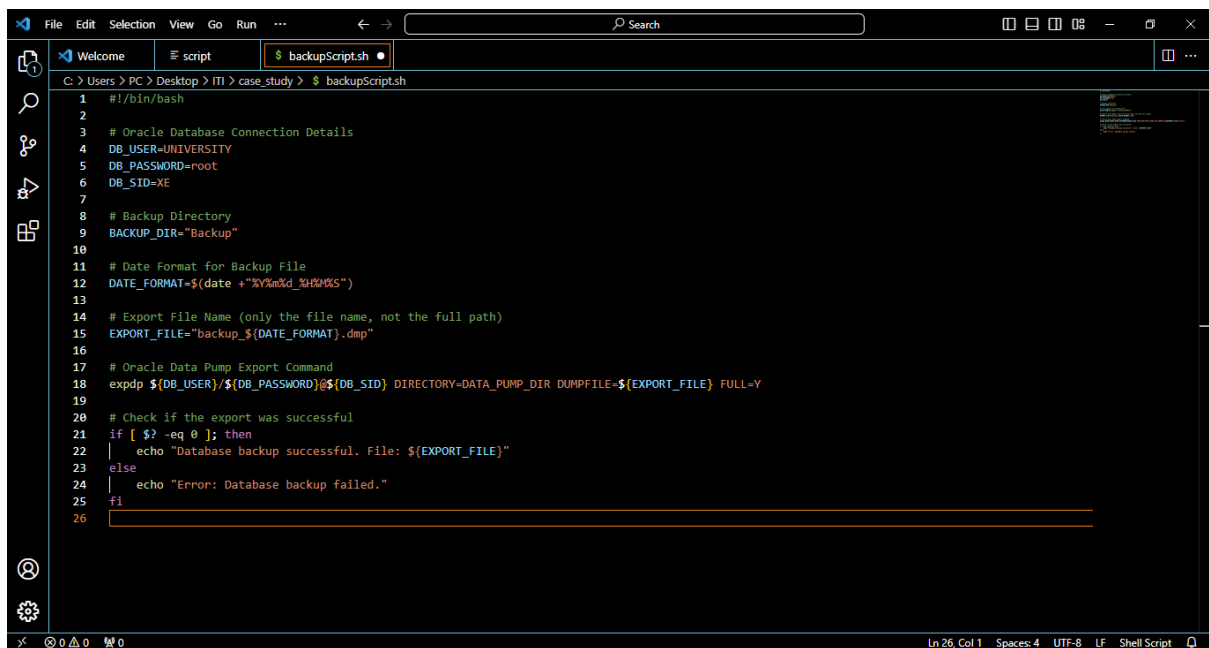
```

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



```

1  #!/bin/bash
2
3  # Oracle Database Connection Details
4  DB_USER=UNIVERSITY
5  DB_PASSWORD=root
6  DB_SID=XE
7
8  # Backup Directory
9  BACKUP_DIR="Backup"
10
11 # Date Format for Backup File
12 DATE_FORMAT=$(date +"%Y%m%d_%H%M%S")
13
14 # Export File Name (only the file name, not the full path)
15 EXPORT_FILE="backup_${DATE_FORMAT}.dmp"
16
17 # Oracle Data Pump Export Command
18 expdp ${DB_USER}/${DB_PASSWORD}@${DB_SID} DIRECTORY=DATA_PUMP_DIR DUMPFILE=${EXPORT_FILE} FULL=Y
19
20 # Check if the export was successful
21 if [ $? -eq 0 ]; then
22 |   echo "Database backup successful. File: ${EXPORT_FILE}"
23 | else
24 |   echo "Error: Database backup failed."
25 | fi
26

```

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:

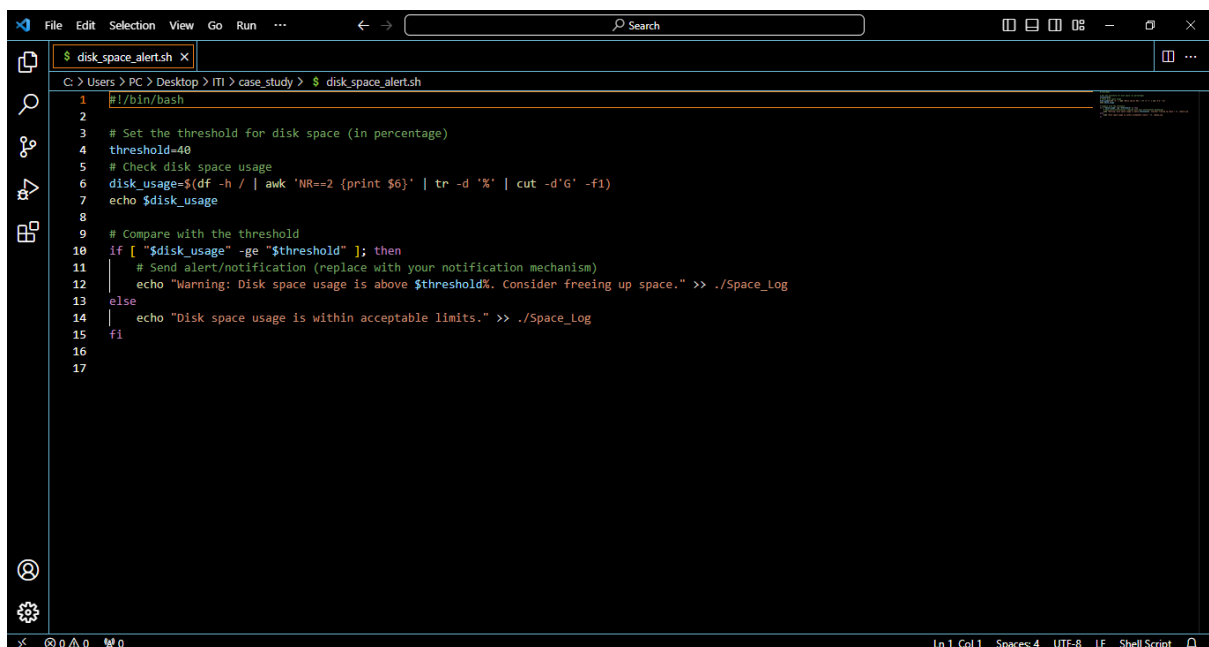
Checks the exit status of the previous command (expdp).

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

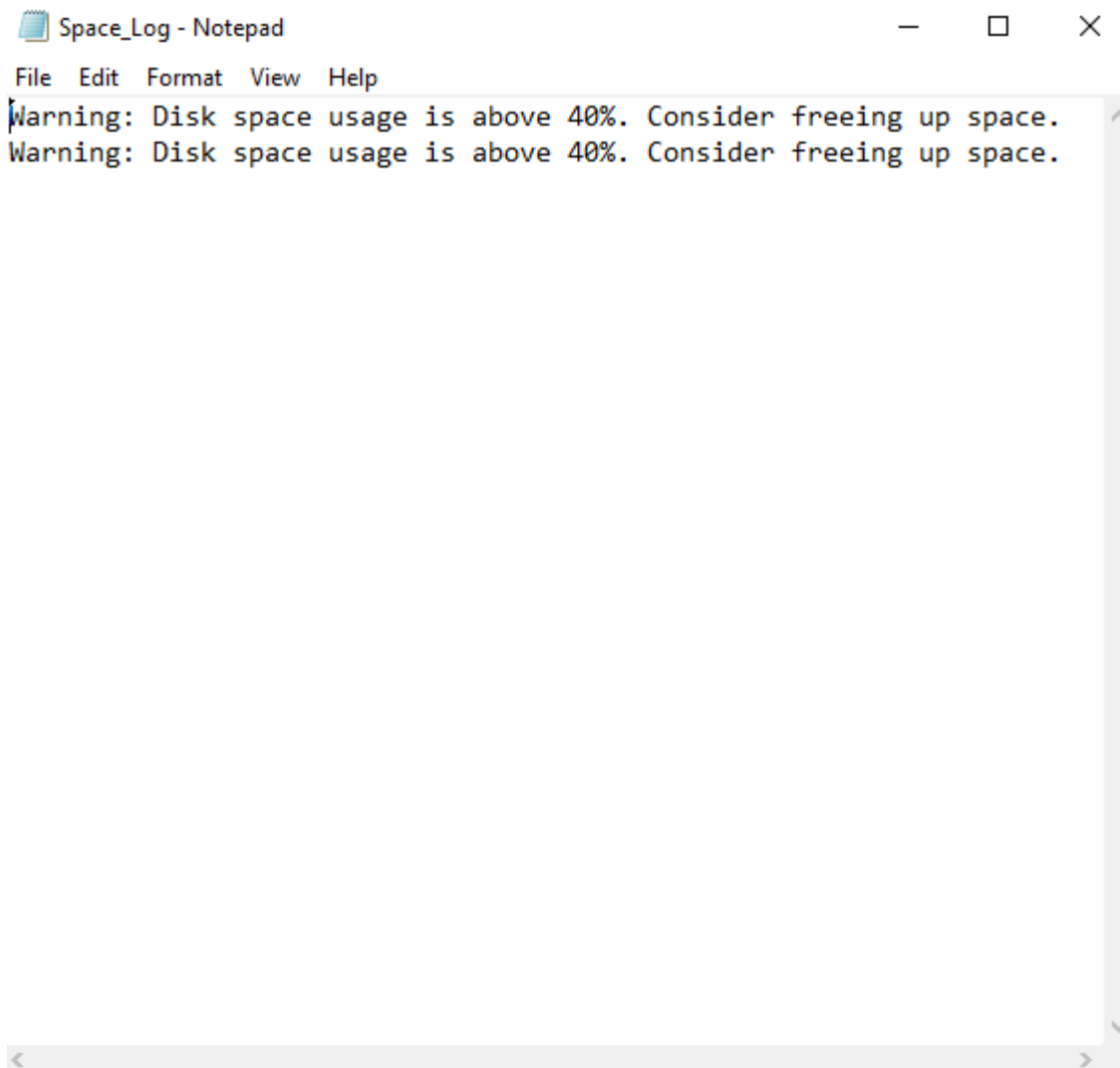
Echo Messages:

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



```
1  #!/bin/bash
2
3  # Set the threshold for disk space (in percentage)
4  threshold=40
5  # Check disk space usage
6  disk_usage=$(df -h / | awk 'NR==2 {print $6}' | tr -d '%' | cut -d'G' -f1)
7  echo $disk_usage
8
9  # Compare with the threshold
10 if [ "$disk_usage" -ge "$threshold" ]; then
11     # Send alert/notification (replace with your notification mechanism)
12     echo "Warning: Disk space usage is above $threshold%. Consider freeing up space." >> ./Space_Log
13 else
14     echo "Disk space usage is within acceptable limits." >> ./Space_Log
15 fi
16
17
```



```
Space_Log - Notepad
File Edit Format View Help
Warning: Disk space usage is above 40%. Consider freeing up space.
Warning: Disk space usage is above 40%. Consider freeing up space.
```

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:

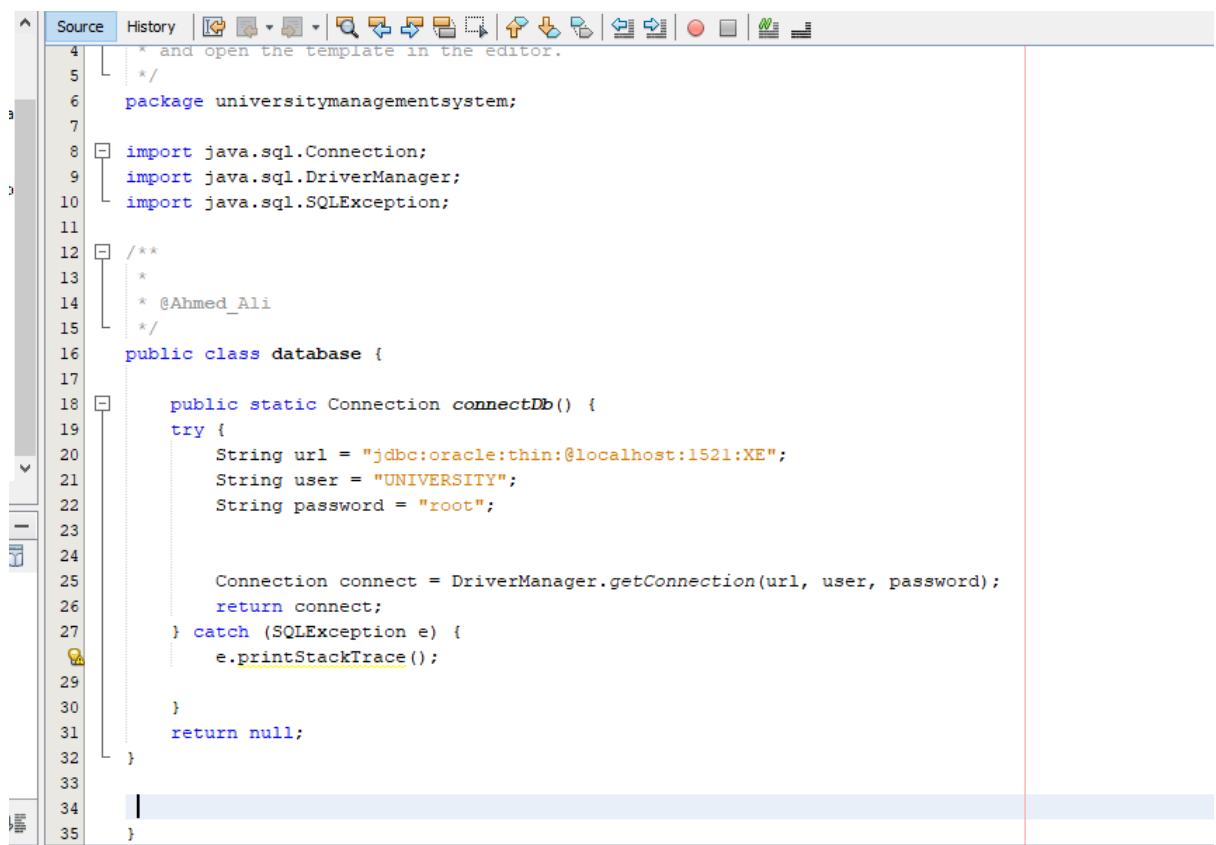
If disk usage is above the threshold, an alert/notification is generated.

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:

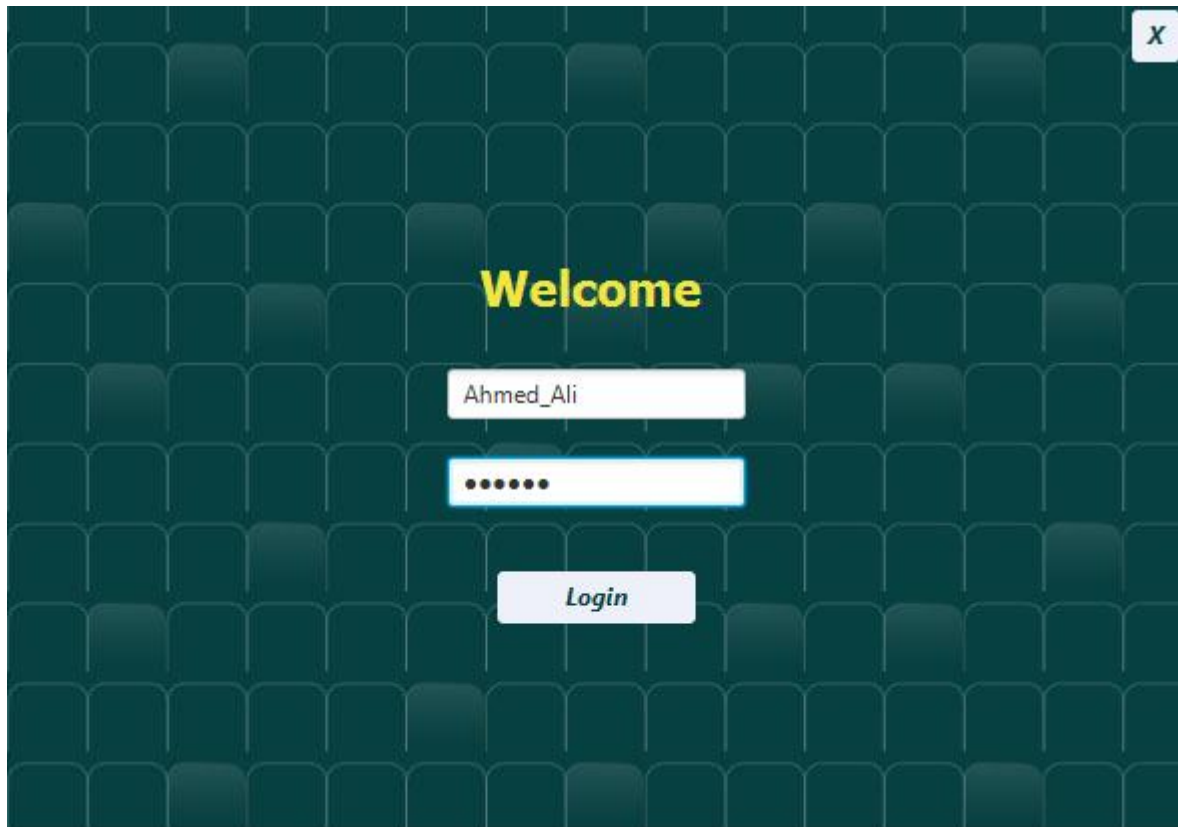
- **Data Access Layer:** to ensure single tone concept which allow us to make a single connection with the database.

A screenshot of a Java IDE window showing a source code editor. The code is for a class named 'database' in the package 'universitymanagementsystem'. It imports 'java.sql.Connection', 'java.sql.DriverManager', and 'java.sql.SQLException'. It contains a static method 'connectDb()' that attempts to establish a database connection using 'jdbc:oracle:thin:@localhost:1521:XE' with user 'UNIVERSITY' and password 'root'. The method returns the connection or null if an exception occurs, which is then printed using 'e.printStackTrace()'.

```
4      * and open the template in the editor.
5      */
6      package universitymanagementsystem;
7
8      import java.sql.Connection;
9      import java.sql.DriverManager;
10     import java.sql.SQLException;
11
12     /**
13      *
14      * @Ahmed_Ali
15      */
16     public class database {
17
18         public static Connection connectDb() {
19             try {
20                 String url = "jdbc:oracle:thin:@localhost:1521:XE";
21                 String user = "UNIVERSITY";
22                 String password = "root";
23
24
25                 Connection connect = DriverManager.getConnection(url, user, password);
26                 return connect;
27             } catch (SQLException e) {
28                 e.printStackTrace();
29             }
30
31             return null;
32         }
33     }
34
35 }
```

- 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.

Welcome

Dashboard
Students
Available Courses
Grades of Students
Departments

Student ID	First name	Last name	Birth Date	mob number	Department ID
10001	ahmed	ali	1998-10-01	01113568468	101
10002	Sara	Khan	1999-05-15	01234567890	101
10003	Omar	Hassan	2000-02-28	01011223344	101
10004	Lina	Mohamed	1997-08-10	01122334455	101
10005	Khaled	Abdullah	1999-11-20	01098765432	101
10006	Rania	Hassan	1998-04-05	01111222333	101

Student ID:

Mob number:

First Name:

Birth Date:

Last Name:

Dept ID:

Clear

Delete

Update

Add

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

Welcome

Dashboard
Students
Available Courses
Grades of Students
Departments

Course ID:
Course Name:
Course hours:

Add

Update

Clear

Delete

Course ID	Course Name	Course hours
1001	Structure analysis	15
1013	Computer Science	16
1002	fluid mechanics	12
1003	thermodynamics	18
1004	electrical circuits	20
1005	programming fundamentals	10
1006	database management	15
1007	networking basics	8
1008	algorithm design	16
1009	artificial intelligence	22
1010	machine learning	18
1011	software engineering principles	14
1012	Medicine fundamentals	14

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.

Welcome

- Dashboard
- Students
- Available Courses
- Grades of Students
- Departments

Student ID:

Course ID:

Grade:

Year:

Semster:

Clear

Update

Delete

Add

Student ID	Course ID	Grade	Year	Semester
10002	1002	78	2	1
10003	1003	92	3	2
10004	1004	65	4	1
10001	1002	60	1	2
10001	1003	65	1	2
10001	1004	71	1	2
10001	1005	83	1	2
10002	1004	60	1	2
10005	1001	65	4	1
10014	1010	65	4	1
10003	1002	65	4	1

```
        || 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 ?");
```

```

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.

