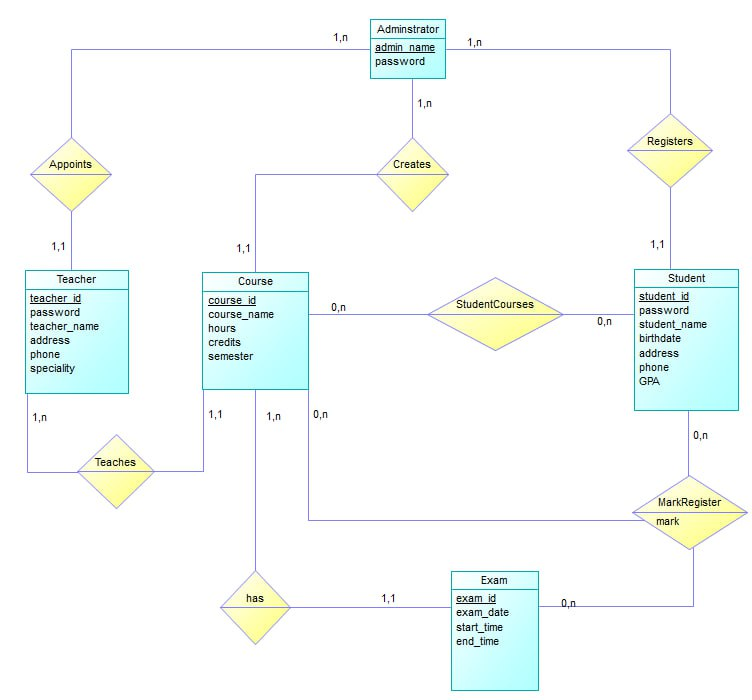
Database UnivDB Project

By Ali Sweidan (100285) & Habib Khalaf (103720)

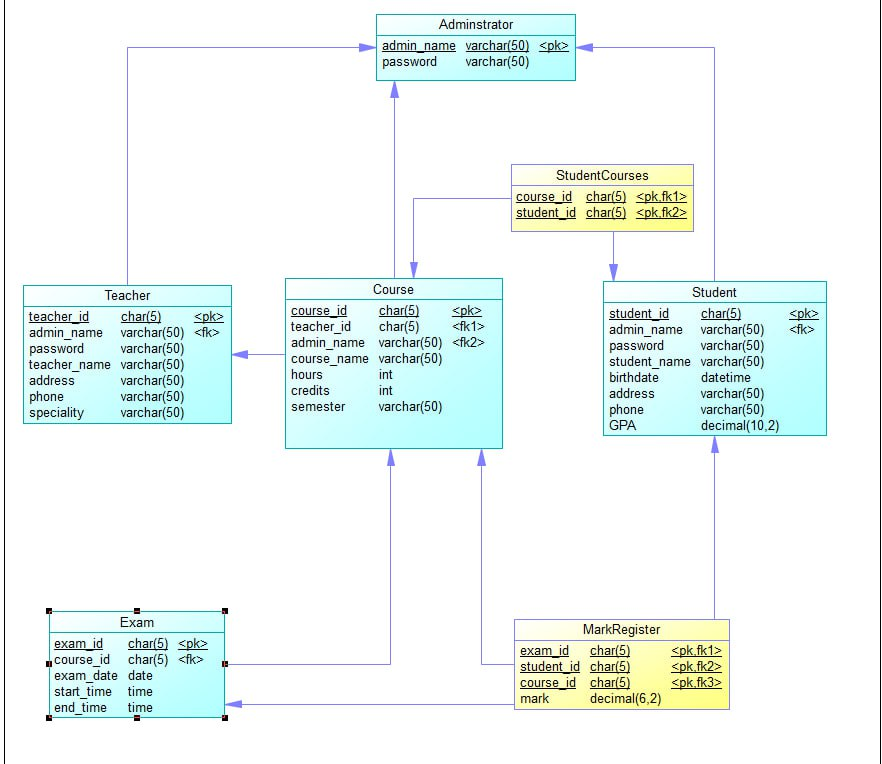
We are building a university database management system designated for a major. that serves the interests of 3 kinds of users over the curse of 4 semesters. The 3 kinds of users are: The Students who will be able to view their marks related to each course, which courses they are currently enrolled in, and which courses they have and haven’t passed with associated details, as well as enrolling and unenrolling in courses, and check their overall GPA. The Teachers who will be able view which courses they are giving, and exams sat by the administrator, and submit or update their students’ marks. Lastly, the administrator, who can add new courses with exam dates, in addition to registering new students and teachers. Each major shall have one adminstrator, several teachers, and many students.

PART A:

**1-Corresponding ER Diagram:**



**2-The Physical Data Model (PDM):**



**3-Code Generation:**

We have 5 tables: Adminstrator, Teacher, Student, and Exam. In addition to establishing two tables (MarkRegister, and StudentCourses) as a result of them being two many-to-many relationships in the ERD diagram.

And we conducted indexing for all the foreign keys existing in each table.

As well as, creating 4 triggers that ensure the referential and data integrity of the database, with the most important two being **“StudentCourseInsertIntoMarkRegister”** which upon having a student enrolling in a course, and having the record saved in “StudentCourses” table, the trigger will insert the student\_id, with the course\_id, and exam\_id related to the course\_id in the MarkRegisterTable. The other trigger **“TriggerDeleteStudent”** which acts when the administrator wants to delete a student by: firstly, instead of deleting the student (directly), the student’s with the desired student\_id will be deleted from the tables “StudentCourses”, “MarkRegister”, and finally from the table “Student”.

As for security, the administrator is granted full security to conduct select, delete, insert, and update operations on all tables in the database.

PART B:

* 1.create-univdb: Creates the database.
* 2.create-univdb-tables: Creates the tables of the

database.

* 3.create-univdb-insertdata: Inserts sample data.
* function1-CalculateTotalStudentCredits: Calculates

the total amount of credits held by the student for

courses' marks that are either null or below 50.

* function2-GetRegisteredTeachersFunction
* function3-getRegisteredStudents
* storedprocedure1-

GetStudentCoursesAndMarksWithSemesterAndCredit

Gets the courses of a student with their affiliated

marks, semester, and credit.

* storedprocedure2(withIransaction)-

EnrollStudentInSemester1: Enrolls a student in

semester one as a transaction and uses function1 to

either rollback or commit.

* storedprocedure3(withTransaction)-

EnrollStudentlnSemester; Enrolls a student in a

semester (different than semester one), as a

transaction by checking logical condition by the help

of function1, like having the student taken all courses

of the previous semester and having total creditscurrently held less than 30 to commit, or else

rollback.

* storedprocedure4(withTransaction)-UnenrollStudentfromCourse
* storedprocedure5-GetTeacherCourses
* storedProcedure6(ReturnPassedCourses)
* storedProcedure7 (ReturnFailedCourses)
* storedProcedure8-GetTeacherExams
* storedProcedure9-AddMarksForExam
* storedProcedure10-GetCourseForMark
* storedProcedure11-AddANDEditTeacher
* storedProcedure12-AddStudent
* storedProcedure13-AddCourse
* storedProcedurel4-AddExam
* storedProcedurel5-ViewCoursesAndExams
* storedProcedure16-DeleteStudentByld
* trigger1-GPA: A student's GPA is calculated

automatically upon any update on mark in the table

MarkRegister.

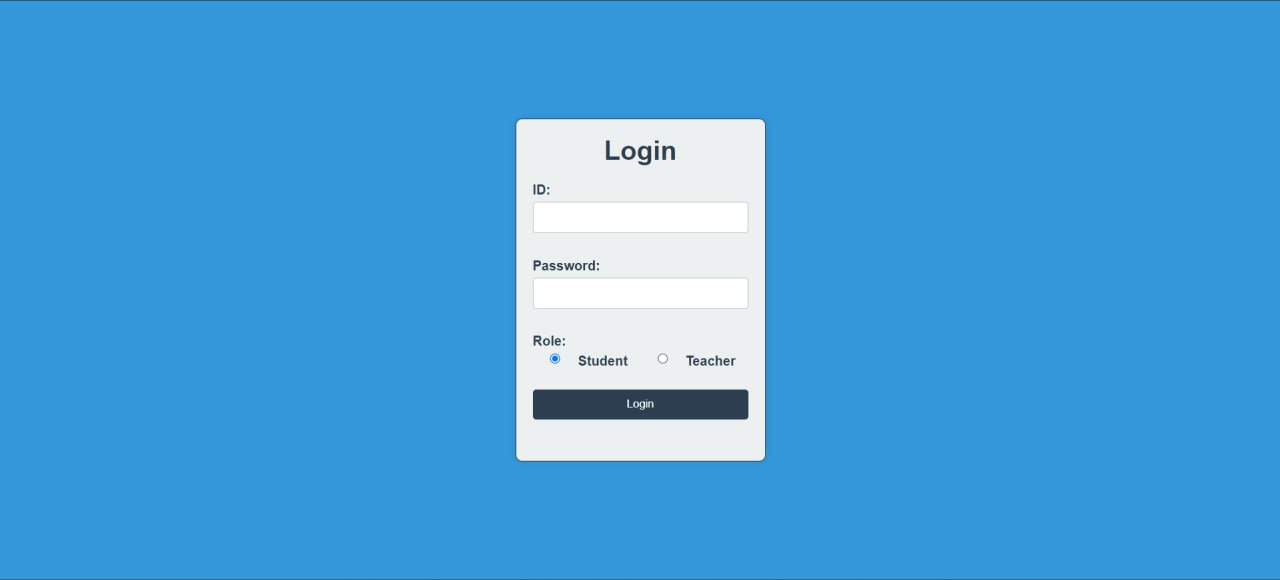
* trigger2-StudentCoursesInsertToMarkRegister
* trigger3-StudentCoursesDeleteToMarkRegister
* trigger4-TriggerDeleteStudent
* Cursor1-UpdateMarksForCourse: Iterates through the

marks of MarkRegister one by one and updates the mark of a course falling in the range between 40 and 49, to 50.

Part C: Interface design & implementation

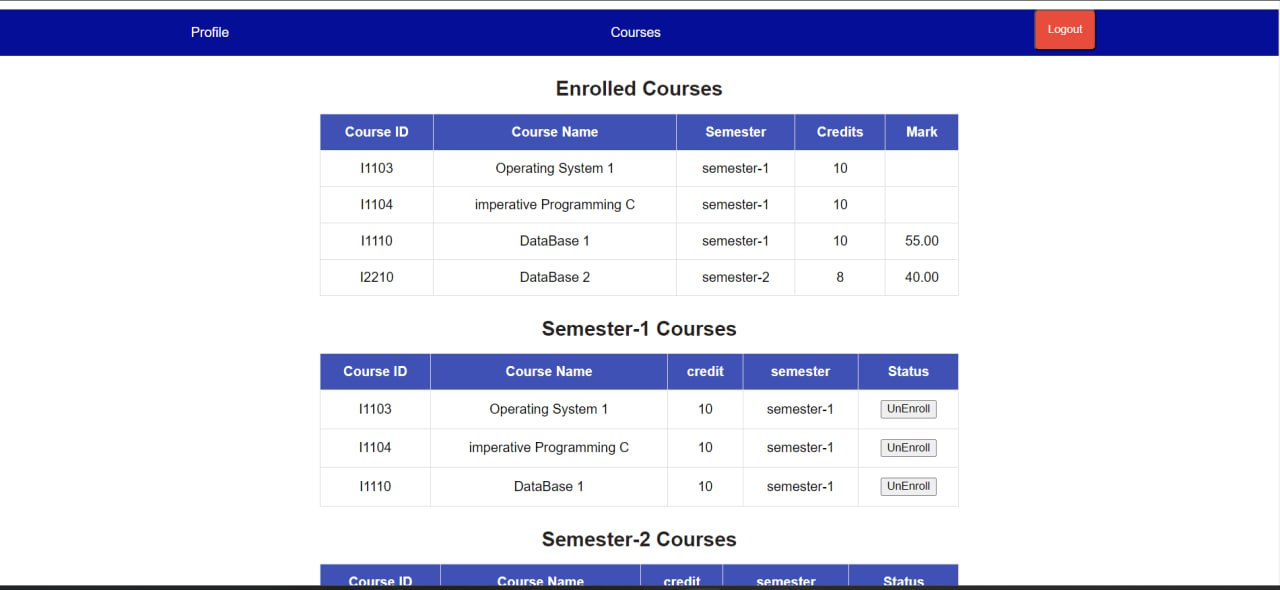
**1-A student’s Interface:**

**1.1-The login site for both a teacher and a student is shared:**



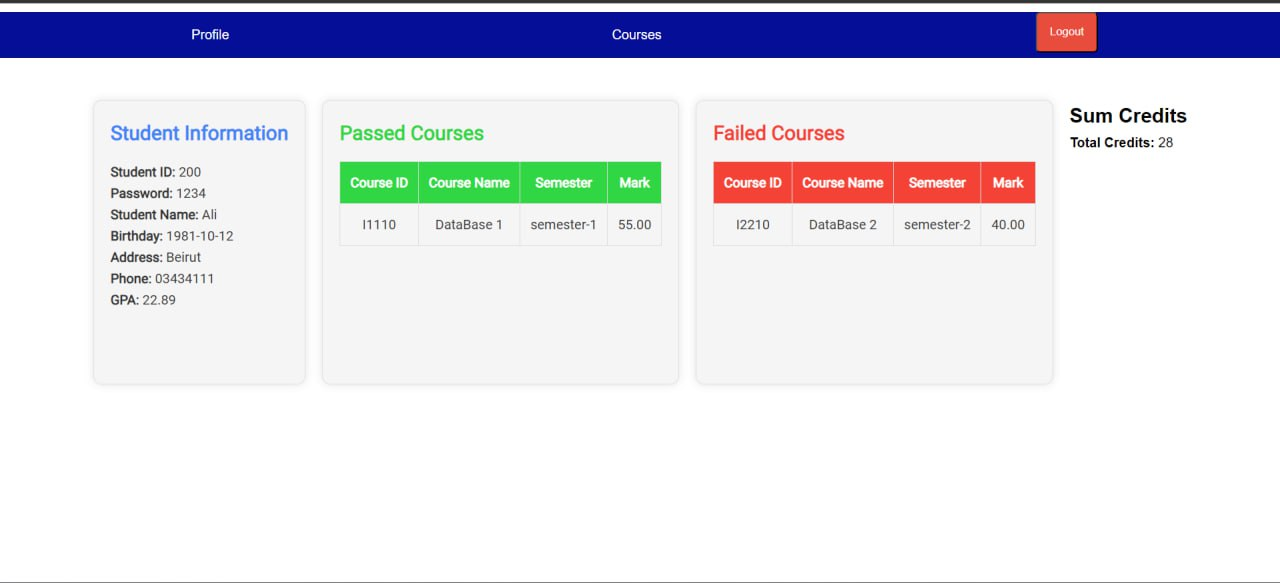
Here, the user should choose what role he belongs to; either a teacher or a student, write his ID and Password which will be examined and checked for validity in the database upon clicking login. In case the student’s or teacher’s ID or Password don’t match with any existing data in the database, the user won’t be permitted an entry, and an error message will be displayed instead.

**1.2-A student’s courses page:**



After a student’s id and password get validated in the login page, he is allowed further access, and is directed towards the Student’s Courses page which displays the courses which the Student has been enrolled in throughout his up to date studying in the university. As well as displaying the upcoming courses which the student will have to take, with each course being affiliated to its respective semester. The Student will be able to enroll in a new course only after he has taken the courses of a previous semester (each semester has 30 credits) and has passed enough amounts of them (after passing a course’s exam, the credits of that course get freed from the total credits) so that his total credits + the new credits of the course he wants to enroll in don’t exceed 30, or else the student won’t be able to enroll in a new course.

**1.3-A student’s profile page:**

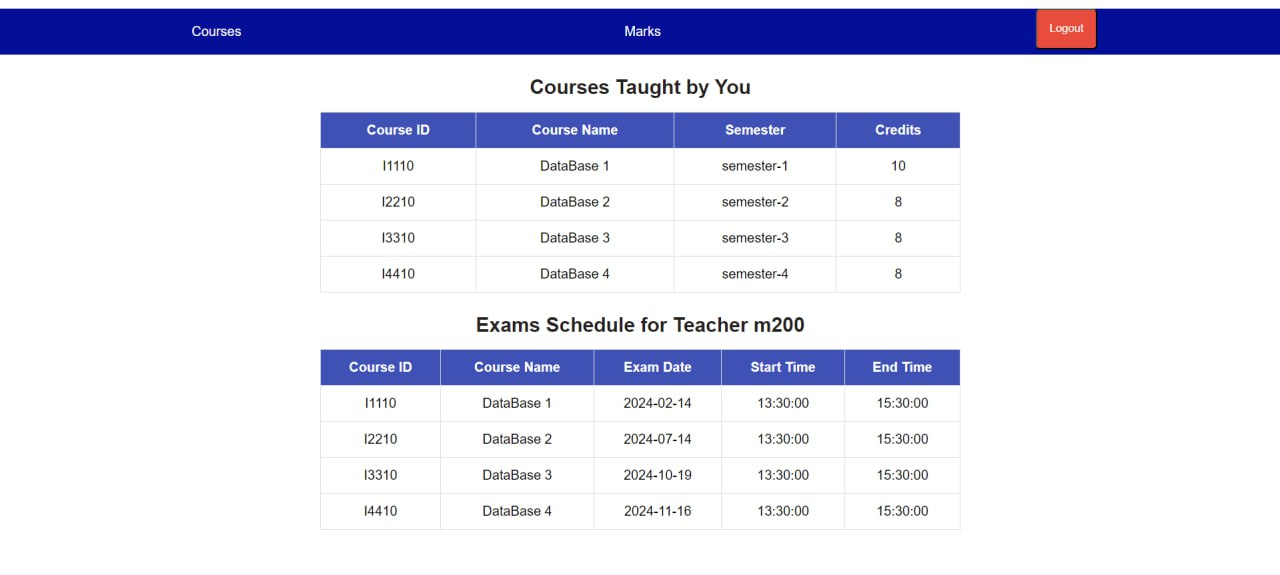


A student can’t navigate himself to a page called Profile as it appears in the navigation bar atop the site. This page contains information about the Student (As can be seen on the Student Information box on the left side of the page), as well as the courses which the Student has passed, and failed, in two other boxes. And the total credits being held currently by the student.

**2-A Teacher’s Interface:**

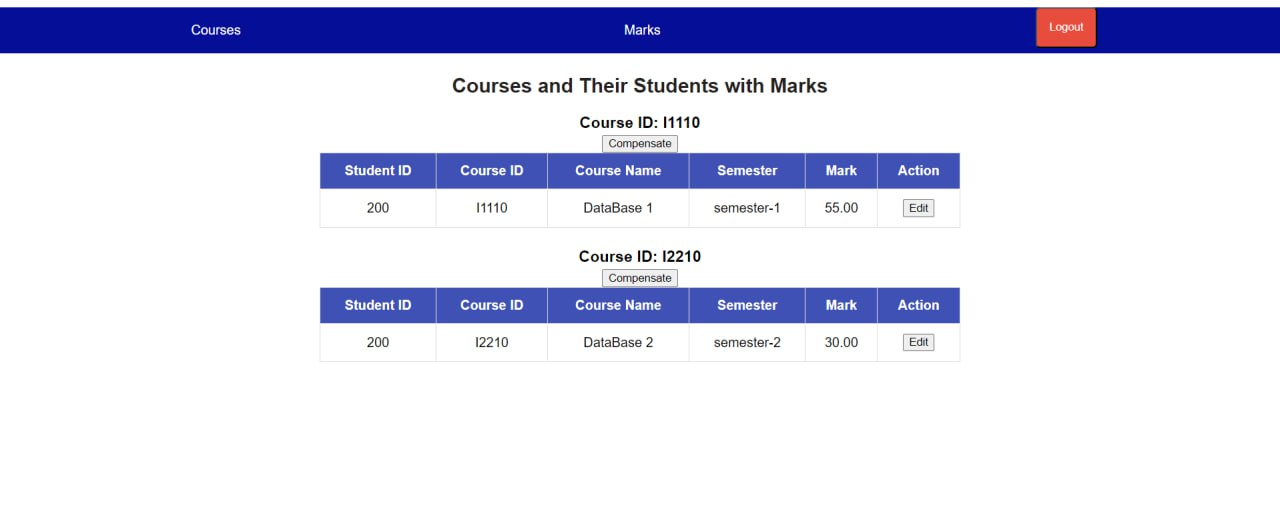
**1.1-A teacher’s login page (already mentioned above)**

**1.2-A teacher’s course page:**



Upon validation for a teacher’s inserted id and password in the login page, the teacher is taken to a “Courses” page, which shows all the courses being taught by the teacher, in addition to an Exams Schedule table which displays the exams’ date and other details of courses being taught by the teacher, these are entered by the administrator on his page which will be discusses soon later.

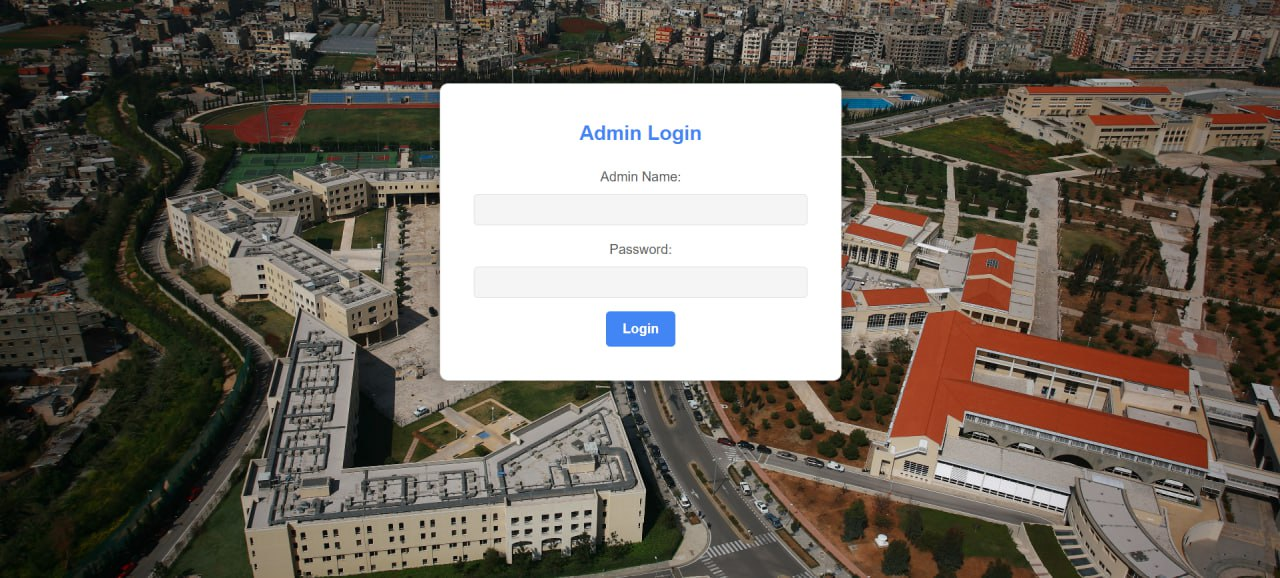
**1.3-Marks page:**



On the “Marks” page, a teacher will be able to insert a mark for a certain student after correcting his/her exam, through searching for the student’s id in the page, and then locating the row of the course which the mark belongs to, and entering the mark by clicking on the edit button at the rightmost column of the row. Furthermore, the teacher can click the “Compensate” button above every course, which compensates the marks of students who have marks ranging between 40 and 49, to 50.

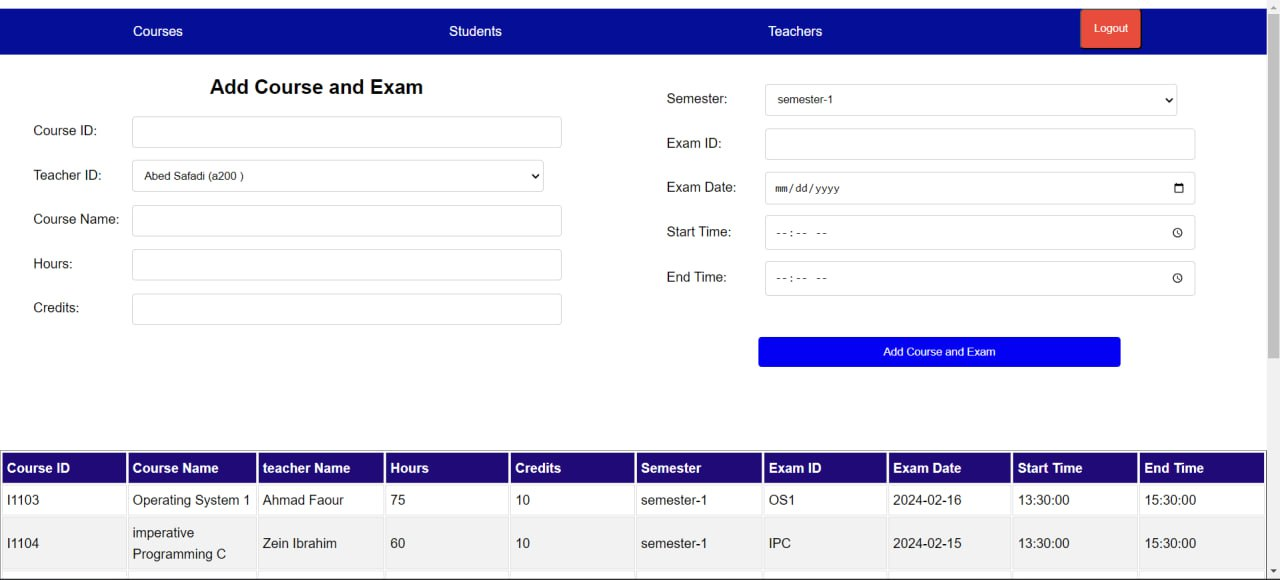
**3-Adminstartor’s Interface:**

**3.1-Admin login page:**



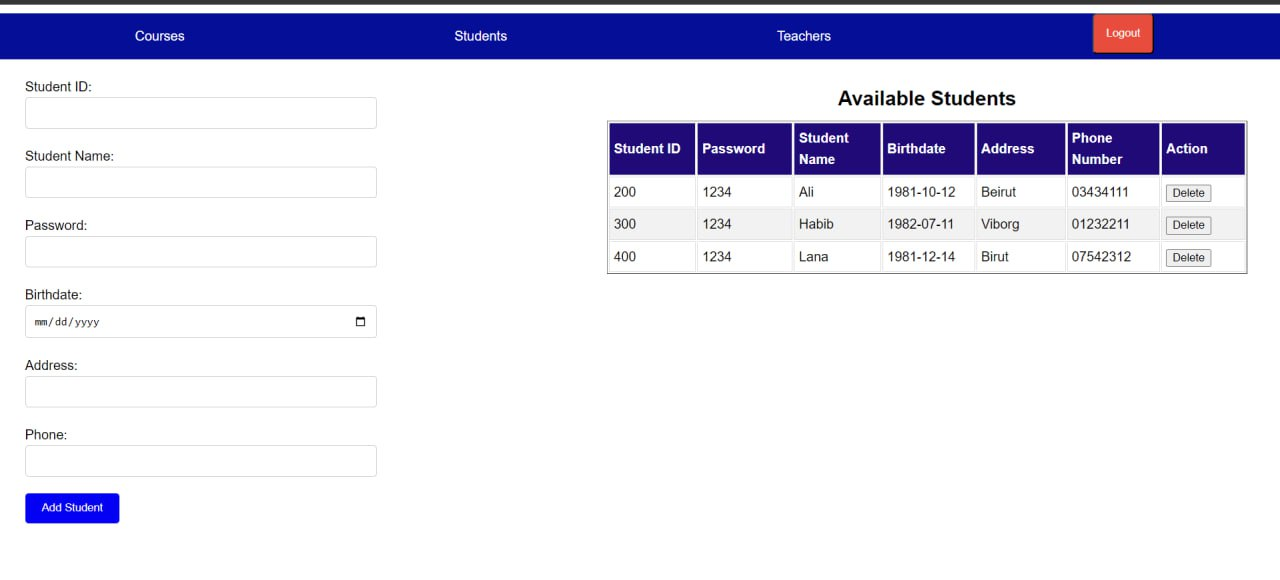
The admin has a separate login page, which only he has access to on his device. The admin’s name and password will be compared with existing data in the database for validation. In case of validation, the admin will be able to login, otherwise an error message will be displayed.

**3.2-Courses page:**



Upon validation in the login page, the administrator will be directed to the “Courses” page where he can add a new course, appoint a teacher to it, and set an exam by filling exam’s data to the new course. Also, the administrator can view the existing courses with all of their affiliated details and information in the table appearing at the bottom of the page.

**3.3-Students’ page:**



The administrator can navigate to the ”Students” page by clicking the “Students” reference in the navigation bar atop the screen. In this page, the administrator will be able to view the available students enrolled in the major in the table appearing on the right side of the page, and delete any enrolled student via the delete button appearing in the last column of every row in the table. Upon deleting the student, all of the student’s record will be deleted from the database. In addition to adding a new student by filling the student’s details in the forms present on the left side of the page.

IMPLEMENTATION AND VERIFICATION

The initiation of the University database application's execution involves systematically executing a series of planned steps. To begin, suitable technologies are carefully chosen to align with the application's specific requirements.

Establishing the Database Framework

The first phase revolves around establishing the database framework. This encompasses the creation of essential tables and relationships, referring to the previously designed Entity-Relationship (ER) model from the database modeling phase.

Crafting User Interfaces

Following the database setup, the subsequent step involves designing user interfaces for the Admin, Teacher, and Student. These interfaces prioritize user-friendliness and intuitive navigation.

Seamless Interface-Database Integration

Once the interfaces are constructed, they seamlessly integrate with the database, ensuring that actions performed on the interfaces accurately reflect in the database, and vice versa.

Driver Setup

To facilitate communication between the application and the database, drivers are obtained from the Microsoft website. Subsequently, these drivers are configured in the php.ini file, a crucial configuration file for PHP.

IDE Customization

Visual Studio Code serves as the Integrated Development Environment (IDE) for this project. Specific project-related settings are configured in the settings.json file to tailor the development environment to the project's unique needs.

Efficiency through Ready Functions

Streamlining development is achieved by employing prebuilt functions, which handle various tasks like server connection, database querying, parameter transmission, and table row count verification. The use of these functions enhances development efficiency and maintains code consistency.

Validation via Testing

Upon the completion of implementation, the application undergoes comprehensive testing to validate its functionality and reliability. A range of testing procedures, including unit testing, integration testing, and system testing, are conducted to identify and resolve any issues or bugs within the application.

Annex

* 1.create-univdb
* USE master;
* GO
* CREATE DATABASE univdb
* ON
* ( NAME = 'univdb',
* FILENAME = 'E:\AliSweidan\_HabibKhalaf\univdb\univdb.mdf')
* LOG ON
* ( NAME = 'univdb\_log',
* FILENAME = 'E:\AliSweidan\_HabibKhalaf\univdb\univdb.ldf');
* 2.create-univdb-tables
* use univdb
* go
* if exists (select 1
* from sysobjects
* where id = object\_id('ADMINSTRATOR')
* and type = 'U')
* drop table Adminstrator
* go
* if exists (select 1
* from sysobjects
* where id = object\_id('TEACHER')
* and type = 'U')
* drop table Teacher
* go
* if exists (select 1
* from sysobjects
* where id = object\_id('COURSE')
* and type = 'U')
* drop table COURSE
* go
* if exists (select 1
* from sysobjects
* where id = object\_id('STUDENT')
* and type = 'U')
* drop table STUDENT
* go
* if exists (select 1
* from sysobjects
* where id = object\_id('EXAM')
* and type = 'U')
* drop table EXAM
* go
* if exists (select 1
* from sysobjects
* where id = object\_id('MARKREGISTER')
* and type = 'U')
* drop table MARKREGISTER
* go
* if exists (select 1
* from sysobjects
* where id = object\_id('STUDENTCOURSES')
* and type = 'U')
* drop table STUDENTCOURSES
* go
* /\*==============================================================\*/
* /\* Table: ADMINSTRATOR \*/
* /\*==============================================================\*/
* create table Adminstrator (
* admin\_name varchar(50) not null,
* password varchar(50) not null,
* primary key (admin\_name)
* )
* go
* /\*==============================================================\*/
* /\* Table: TEACHER \*/
* /\*==============================================================\*/
* create table Teacher (
* teacher\_id char(5) not null,
* admin\_name varchar(50) not null,
* password varchar(50) not null,
* teacher\_name varchar(50) not null,
* address varchar(50) null,
* phone varchar(50) null,
* speciality varchar(50) null,
* primary key (teacher\_id),
* foreign key (admin\_name) references Adminstrator (admin\_name)
* )
* go
* /\*==============================================================\*/
* /\* Table: COURSE \*/
* /\*==============================================================\*/
* create table Course (
* course\_id char(5) not null,
* teacher\_id char(5) not null,
* admin\_name varchar(50) not null,
* course\_name varchar(50) null,
* hours int null,
* credits int null,
* semester varchar(50) null,
* primary key (course\_id),
* foreign key (teacher\_id) references Teacher (teacher\_id),
* foreign key (admin\_name) references Adminstrator (admin\_name)
* )
* go
* /\*==============================================================\*/
* /\* Table: EXAM \*/
* /\*==============================================================\*/
* create table Exam (
* exam\_id char(5) not null,
* course\_id char(5) not null,
* exam\_date date null,
* start\_time time null,
* end\_time time null,
* primary key (exam\_id),
* foreign key (course\_id) references Course (course\_id)
* )
* go
* /\*==============================================================\*/
* /\* Table: STUDENT \*/
* /\*==============================================================\*/
* create table Student (
* student\_id char(5) not null,
* admin\_name varchar(50) not null,
* password varchar(50) null,
* student\_name varchar(50) null,
* birthdate datetime null,
* address varchar(50) null,
* phone varchar(50) null,
* GPA decimal(10,2) null,
* primary key (student\_id),
* foreign key (admin\_name) references Adminstrator (admin\_name)
* )
* go
* /\*==============================================================\*/
* /\* Table: MARKREGISTER \*/
* /\*==============================================================\*/
* create table MarkRegister (
* exam\_id char(5) not null,
* student\_id char(5) not null,
* course\_id char(5) not null,
* mark decimal(6,2) null,
* foreign key (exam\_id) references Exam (exam\_id),
* foreign key (student\_id) references Student (student\_id),
* foreign key (course\_id) references Course (course\_id)
* )
* go
* /\*==============================================================\*/
* /\* Table: STUDENTCOURSES \*/
* /\*==============================================================\*/
* create table StudentCourses (
* course\_id char(5) not null,
* student\_id char(5) not null,
* foreign key (student\_id) references Student (student\_id),
* foreign key (course\_id) references Course (course\_id)
* )
* go
* 3.create-univdb-insertdata
* use univdb
* go
* --insert data to Adminstrator
* insert into Adminstrator values ('zeinab Saghir','CS1234')
* go
* --insert data to Teacher
* insert into Teacher values ('m200','zeinab Saghir', 'dbouk', 'Mohamed Dbouk', 'Beirut', '76000000', 'CS-DB')
* insert into Teacher values ('z200','zeinab Saghir', 'zein', 'Zein Ibrahim', 'Beirut', '71000000', 'CS-C')
* insert into Teacher values ('f200','zeinab Saghir', 'faour', 'Ahmad Faour', 'Beirut', '03000000', 'CS-OS')
* insert into Teacher values ('a200','zeinab Saghir', 'abed', 'Abed Safadi', 'Beirut', '81000000', 'CS-GUI')
* go
* --insert data to Course
* insert into Course values ('I1110', 'm200','zeinab Saghir', 'DataBase 1', 72, 10 , 'semester-1')
* insert into Course values ('I2210', 'm200','zeinab Saghir', 'DataBase 2', 72, 8 , 'semester-2')
* insert into Course values ('I3310','m200','zeinab Saghir' , 'DataBase 3', 50, 8 , 'semester-3')
* insert into Course values ('I4410','m200','zeinab Saghir' , 'DataBase 4', 50, 8 , 'semester-4')
* insert into Course values ('I1104','z200','zeinab Saghir' , 'imperative Programming C', 60, 10 , 'semester-1')
* insert into Course values ('I2204','z200','zeinab Saghir' , 'Data Structures', 60, 8 , 'semester-2')
* insert into Course values ('I1103','f200','zeinab Saghir' , 'Operating System 1', 75, 10 , 'semester-1')
* insert into Course values ('I2203','f200','zeinab Saghir' , 'Operating System 2', 75, 8 , 'semester-2')
* insert into Course values ('I3303','f200','zeinab Saghir' , 'Operating System 3', 75, 8 , 'semester-3')
* insert into Course values ('I4403','f200','zeinab Saghir' , 'Operating System 4', 75, 8 , 'semester-4')
* insert into Course values ('I2205','a200','zeinab Saghir' , 'GUI-1', 50, 6 , 'semester-2')
* insert into Course values ('I3305','a200','zeinab Saghir' , 'GUI-2', 50, 6 , 'semester-3')
* insert into Course values ('I4405','a200','zeinab Saghir' , 'GUI-3', 50, 6 , 'semester-4')
* go
* -- insert data to Student
* insert into Student values ('200','zeinab Saghir', '1234', 'Ali', '10-12-81', 'Beirut', '03434111', NULL)
* insert into Student values ('300','zeinab Saghir', '1234', 'Habib', '7/11/82', 'Viborg', '01232211', NULL)
* insert into Student values ('400','zeinab Saghir', '1234', 'Lana', '12/14/81', 'Birut', '07542312', NULL)
* go
* --insert Exam data
* insert into Exam values ('DB1', 'I1110', '02/14/2024', '13:30:00' ,'15:30:00')
* insert into Exam values ('DB2', 'I2210', '07/14/2024', '13:30:00' ,'15:30:00')
* insert into Exam values ('DB3', 'I3310', '10/19/2024', '13:30:00' ,'15:30:00')
* insert into Exam values ('DB4', 'I4410', '11/16/2024', '13:30:00' ,'15:30:00')
* insert into Exam values ('IPC','I1104', '02/15/2024', '13:30:00' ,'15:30:00')
* insert into Exam values ('DS', 'I2204', '07/14/2024', '13:30:00' ,'15:30:00')
* insert into Exam values ('OS1', 'I1103', '02/16/2024', '13:30:00' ,'15:30:00')
* insert into Exam values ('OS2', 'I2203', '07/15/2024', '13:30:00' ,'15:30:00')
* insert into Exam values ('OS3', 'I3303', '10/14/2024', '13:30:00' ,'15:30:00')
* insert into Exam values ('OS4', 'I4403', '11/14/2024', '13:30:00' ,'15:30:00')
* insert into Exam values ('GUI-1', 'I2205', '07/17/2024', '13:30:00' ,'15:30:00')
* insert into Exam values ('GUI-2', 'I3305', '10/14/2024', '13:30:00' ,'15:30:00')
* insert into Exam values ('GUI-3', 'I4405', '11/14/2024', '13:30:00' ,'15:30:00')
* go
* function1-CalculateTotalStudentCredits
* USE univdb;
* GO
* DROP FUNCTION [dbo].[CalculateTotalStudentCredits];
* GO
* -- Create a scalar-valued function
* CREATE FUNCTION CalculateTotalStudentCredits
* (
* @student\_id CHAR(5)
* )
* RETURNS INT
* AS
* BEGIN
* DECLARE @totalCredits INT;
* -- Calculate the total credits for the given student, including only courses with marks below 50 or NULL
* SELECT @totalCredits = COALESCE(SUM(C.credits), 0)
* FROM Course C
* WHERE C.course\_id IN (SELECT course\_id FROM StudentCourses WHERE student\_id = @student\_id)
* AND (NOT EXISTS (SELECT 1 FROM MarkRegister MR WHERE MR.course\_id = C.course\_id AND MR.student\_id = @student\_id) OR
* COALESCE((SELECT MAX(MR.mark) FROM MarkRegister MR WHERE MR.course\_id = C.course\_id AND MR.student\_id = @student\_id), 0) < 50);
* RETURN @totalCredits;
* END;
* function2-GetRegisteredTeachersFunction
* use univdb
* go
* CREATE FUNCTION dbo.GetRegisteredTeachersFunction
* (@adminName VARCHAR(50))
* RETURNS TABLE
* AS
* RETURN
* (
* SELECT teacher\_id, password, teacher\_name, address, phone, speciality
* FROM Teacher
* WHERE admin\_name = @adminName
* );
* function3-getRegisteredStudents
* use univdb
* go
* CREATE FUNCTION dbo.GetRegisteredStudentsFunction
* (@adminName VARCHAR(50))
* RETURNS TABLE
* AS
* RETURN
* (
* SELECT student\_id, password, student\_name, birthdate, address, phone, GPA
* FROM Student
* WHERE admin\_name = @adminName
* );
* storedprocedure1-

GetStudentCoursesAndMarksWithSemesterAndCredit

USE univdb;

GO

-- Create a stored procedure to retrieve a student's courses and marks with semester and credit information

CREATE PROCEDURE GetStudentCoursesAndMarksWithSemesterAndCredit

@studentID CHAR(5)

AS

BEGIN

-- Main query to retrieve student's courses and marks with semester and credit information

SELECT

SC.course\_id,

C.course\_name,

C.credits,

MR.mark,

C.semester

FROM

StudentCourses SC

INNER JOIN

MarkRegister MR ON SC.student\_id = MR.student\_id AND SC.course\_id = MR.course\_id

INNER JOIN

Course C ON SC.course\_id = C.course\_id

WHERE

SC.student\_id = @studentID;

END;

* storedprocedure2(withIransaction)-EnrollStudentInSemester1
* USE univdb;
* GO
* -- Drop the existing stored procedure if it exists
* IF EXISTS (SELECT \* FROM sys.objects WHERE object\_id = OBJECT\_ID(N'[dbo].[EnrollStudentInSemester1]') AND type in (N'P', N'PC'))
* DROP PROCEDURE [dbo].[EnrollStudentInSemester1];
* GO
* -- Create the modified stored procedure
* CREATE PROCEDURE EnrollStudentInSemester1
* @student\_id CHAR(5),
* @course\_id CHAR(5),
* @semester varchar(50)
* AS
* BEGIN
* BEGIN TRANSACTION;
* -- Calculate the total credits of the student using the function
* DECLARE @totalCredits INT = dbo.CalculateTotalStudentCredits(@student\_id);
* -- Retrieve the credits of the new course
* DECLARE @courseCredits INT = ISNULL((SELECT credits FROM Course WHERE course\_id = @course\_id), 0);
* -- Check if adding the new course will not exceed 30 credits
* IF (@totalCredits + @courseCredits) <= 30
* BEGIN
* -- Enroll the student in the new course
* INSERT INTO StudentCourses (student\_id, course\_id)
* VALUES (@student\_id, @course\_id);
* COMMIT;
* END
* ELSE
* BEGIN
* ROLLBACK;
* END;
* END;
* storedprocedure3(withTransaction)-EnrollStudentlnSemester
* USE univdb;
* GO
* -- Drop the existing stored procedure if it exists
* IF EXISTS (SELECT \* FROM sys.objects WHERE object\_id = OBJECT\_ID(N'[dbo].[EnrollStudentInSemester]') AND type in (N'P', N'PC'))
* DROP PROCEDURE [dbo].[EnrollStudentInSemester];
* GO
* -- Create the stored procedure for Semester enrollment
* CREATE PROCEDURE EnrollStudentInSemester
* @student\_id CHAR(5),
* @course\_id CHAR(5),
* @semester VARCHAR(50)
* AS
* BEGIN
* BEGIN TRANSACTION;
* DECLARE @totalCredits INT;
* DECLARE @currentSemester VARCHAR(50) = @semester;
* DECLARE @previousSemester VARCHAR(50);
* -- Extract the numeric part of the current semester
* DECLARE @currentSemesterNumber INT;
* SET @currentSemesterNumber = TRY\_CAST(RIGHT(@currentSemester, CHARINDEX('-', REVERSE(@currentSemester)) - 1) AS INT);
* -- Check if the result is not negative
* IF @currentSemesterNumber IS NOT NULL AND @currentSemesterNumber > 1
* BEGIN
* -- Calculate the previous semester
* SET @previousSemester = 'Semester-' + CAST(@currentSemesterNumber - 1 AS VARCHAR(10));
* END
* ELSE
* BEGIN
* PRINT 'Cannot calculate previous semester for the first semester or invalid format.';
* ROLLBACK;
* RETURN;
* END;
* -- Check if the student is enrolled in all previous semester courses
* IF NOT EXISTS (
* SELECT 1
* FROM Course
* WHERE semester = @previousSemester
* AND NOT EXISTS (
* SELECT 1
* FROM StudentCourses SC
* WHERE SC.student\_id = @student\_id
* AND SC.course\_id = Course.course\_id
* )
* )
* BEGIN
* -- Calculate the total credits for the student
* SELECT @totalCredits = dbo.CalculateTotalStudentCredits(@student\_id);
* -- Retrieve the credits of the new Semester 2 course
* DECLARE @courseCredits INT;
* SELECT @courseCredits = ISNULL(credits, 0)
* FROM Course
* WHERE course\_id = @course\_id;
* -- Check if adding the new Semester course will not exceed 30 credits
* IF (@totalCredits + @courseCredits) <= 30
* BEGIN
* -- Enroll the student in the new Semester course
* INSERT INTO StudentCourses (student\_id, course\_id)
* VALUES (@student\_id, @course\_id);
* COMMIT;
* PRINT 'Student successfully enrolled in the ' + @semester + ' course.';
* END
* ELSE
* BEGIN
* PRINT 'Enrollment exceeds the credit limit of 30 or student not enrolled in all previous Semester courses.';
* ROLLBACK;
* END;
* END
* ELSE
* BEGIN
* PRINT 'Student is not enrolled in all previous Semester courses.';
* ROLLBACK;
* END;
* END;
* storedprocedure4(withTransaction)-UnenrollStudentfromCourse
* USE univdb;
* GO
* -- Drop the existing stored procedure if it exists
* IF EXISTS (SELECT \* FROM sys.objects WHERE object\_id = OBJECT\_ID(N'[dbo].[UnenrollStudentFromCourse]') AND type in (N'P', N'PC'))
* DROP PROCEDURE [dbo].[UnenrollStudentFromCourse];
* GO
* CREATE PROCEDURE UnenrollStudentFromCourse
* @student\_id CHAR(5),
* @course\_id CHAR(5)
* AS
* BEGIN
* BEGIN TRANSACTION;
* -- Check if the student is enrolled in the course
* IF EXISTS (SELECT 1 FROM StudentCourses WHERE student\_id = @student\_id AND course\_id = @course\_id)
* BEGIN
* -- Check the mark for the student and course
* DECLARE @studentMark INT;
* SELECT @studentMark = mark
* FROM MarkRegister
* WHERE student\_id = @student\_id AND course\_id = @course\_id;
* -- Check if the mark is below 50
* IF @studentMark IS NULL OR @studentMark < 50
* BEGIN
* -- Unenroll the student from the course
* DELETE FROM StudentCourses WHERE student\_id = @student\_id AND course\_id = @course\_id;
* COMMIT;
* PRINT 'Student successfully unenrolled from the course.';
* END
* ELSE
* BEGIN
* -- Rollback if the student has a mark above or equal to 50
* ROLLBACK;
* PRINT 'The student cannot be unenrolled because the mark is 50 or above.';
* END;
* END
* ELSE
* BEGIN
* -- Rollback if the student is not enrolled in the course
* ROLLBACK;
* PRINT 'The student is not currently enrolled in the specified course.';
* END;
* END;
* storedprocedure5-GetTeacherCourses
* use univdb
* go
* CREATE PROCEDURE GetTeacherCourses
* @teacher\_id char(5)
* AS
* BEGIN
* SELECT
* C.course\_id,
* C.course\_name,
* C.hours,
* C.credits,
* C.semester
* FROM
* Teacher T
* INNER JOIN
* Course C ON T.teacher\_id = C.teacher\_id
* WHERE
* T.teacher\_id = @teacher\_id;
* END;
* storedProcedure6(ReturnPassedCourses)

use univdb

go

CREATE PROCEDURE GetPassedCourses

@studentID CHAR(5)

AS

BEGIN

SELECT c.course\_id, c.course\_name, c.semester, mr.mark

FROM Student s

INNER JOIN MarkRegister mr ON s.student\_id = mr.student\_id

INNER JOIN Course c ON mr.course\_id = c.course\_id

WHERE s.student\_id = @studentID AND mr.mark >= 50

ORDER BY c.semester DESC;

END;

* storedProcedure7 (ReturnFailedCourses)
* use univdb
* go
* CREATE PROCEDURE GetFailedCourses
* @studentID CHAR(5)
* AS
* BEGIN
* SELECT c.course\_id, c.course\_name, c.semester, mr.mark
* FROM Student s
* INNER JOIN MarkRegister mr ON s.student\_id = mr.student\_id
* INNER JOIN Course c ON mr.course\_id = c.course\_id
* WHERE s.student\_id = @studentID AND mr.mark < 50
* ORDER BY c.semester DESC;
* END;
* storedProcedure8-GetTeacherExams
* use univdb
* go
* CREATE PROCEDURE GetTeacherExams
* @teacher\_id char(5)
* AS
* BEGIN
* SELECT E.exam\_id, E.course\_id, C.course\_name, E.exam\_date, E.start\_time, E.end\_time
* FROM Exam E
* INNER JOIN Course C ON E.course\_id = C.course\_id
* WHERE C.teacher\_id = @teacher\_id;
* END;
* storedProcedure9-AddMarksForExam
* USE univdb;
* GO
* -- Drop the existing stored procedure
* IF EXISTS (SELECT \* FROM sys.objects WHERE type = 'P' AND name = 'AddMarksForExam')
* DROP PROCEDURE AddMarksForExam;
* GO
* -- Recreate the stored procedure with the desired modifications
* CREATE PROCEDURE AddMarksForExam
* @course\_id char(5),
* @student\_id char(5),
* @mark decimal(6, 2)
* AS
* BEGIN
* UPDATE MarkRegister
* SET mark = @mark
* WHERE course\_id = @course\_id AND student\_id = @student\_id;
* END;
* storedProcedure10-GetCourseForMark
* USE univdb;
* GO
* -- Drop the existing stored procedure
* IF EXISTS (SELECT \* FROM sys.objects WHERE type = 'P' AND name = 'GetCourseForMark')
* DROP PROCEDURE GetCourseForMark;
* GO
* CREATE PROCEDURE GetCourseForMark
* @courseID CHAR(5),
* @teacherID CHAR(5)
* AS
* BEGIN
* SELECT
* SC.course\_id,
* C.course\_name,
* SC.student\_id,
* MR.mark,
* C.semester
* FROM
* StudentCourses SC
* INNER JOIN
* MarkRegister MR ON SC.student\_id = MR.student\_id AND SC.course\_id = MR.course\_id
* INNER JOIN
* Course C ON SC.course\_id = C.course\_id AND C.teacher\_id = @teacherID
* WHERE
* SC.course\_id = @courseID;
* END;
* storedProcedure11-AddANDEditTeacher
* use univdb
* go
* CREATE PROCEDURE AddTeacher
* @teacherID CHAR(5),
* @adminName VARCHAR(50),
* @password VARCHAR(50),
* @teacherName VARCHAR(50),
* @address VARCHAR(50) = NULL,
* @phone VARCHAR(50) = NULL,
* @speciality VARCHAR(50) = NULL
* AS
* BEGIN
* INSERT INTO Teacher (teacher\_id, admin\_name, password, teacher\_name, address, phone, speciality)
* VALUES (@teacherID, @adminName, @password, @teacherName, @address, @phone, @speciality);
* END;
* GO
* CREATE PROCEDURE EditTeacher
* @teacherId CHAR(5),
* @admin\_name VARCHAR(50),
* @teacher\_name VARCHAR(50),
* @password VARCHAR(50),
* @phone VARCHAR(50),
* @address VARCHAR(50),
* @speciality VARCHAR(50)
* AS
* BEGIN
* UPDATE Teacher
* SET teacher\_name = @teacher\_name,
* password = @password,
* phone = @phone,
* address = @address,
* speciality = @speciality
* WHERE teacher\_id = @teacherId;
* END;
* GO
* storedProcedure12-AddStudent

use univdb

go

CREATE PROCEDURE AddStudent

@studentID CHAR(5),

@adminName VARCHAR(50),

@password VARCHAR(50) = NULL,

@studentName VARCHAR(50) = NULL,

@birthdate DATETIME = NULL,

@address VARCHAR(50) = NULL,

@phone VARCHAR(50) = NULL,

@GPA DECIMAL(10,2) = NULL

AS

BEGIN

INSERT INTO Student (student\_id, admin\_name, password, student\_name, birthdate, address, phone)

VALUES (@studentID, @adminName, @password, @studentName, @birthdate, @address, @phone);

END;

* storedProcedure13-AddCourse
* use univdb
* go
* CREATE PROCEDURE AddCourse
* @course\_id CHAR(5),
* @teacher\_id CHAR(5),
* @admin\_name VARCHAR(50),
* @course\_name VARCHAR(50),
* @hours INT,
* @credits INT,
* @semester VARCHAR(50)
* AS
* BEGIN
* -- Check if the course already exists
* IF NOT EXISTS (SELECT 1 FROM Course WHERE course\_id = @course\_id)
* BEGIN
* -- Insert the course into the Course table
* INSERT INTO Course (course\_id, teacher\_id, admin\_name, course\_name, hours, credits, semester)
* VALUES (@course\_id, @teacher\_id, @admin\_name, @course\_name, @hours, @credits, @semester);
* -- Return success message
* SELECT 'Course added successfully.' AS Result;
* END
* ELSE
* BEGIN
* -- Return error message if the course already exists
* SELECT 'Error: Course with ID ' + @course\_id + ' already exists.' AS Result;
* END
* END;
* storedProcedurel4-AddExam
* use univdb
* go
* CREATE PROCEDURE AddExam
* @exam\_id CHAR(5),
* @course\_id CHAR(5),
* @exam\_date DATE,
* @start\_time TIME,
* @end\_time TIME
* AS
* BEGIN
* -- Check if the exam already exists
* IF NOT EXISTS (SELECT 1 FROM Exam WHERE exam\_id = @exam\_id)
* BEGIN
* -- Insert the exam into the Exam table
* INSERT INTO Exam (exam\_id, course\_id, exam\_date, start\_time, end\_time)
* VALUES (@exam\_id, @course\_id, @exam\_date, @start\_time, @end\_time);
* -- Return success message
* SELECT 'Exam added successfully.' AS Result;
* END
* ELSE
* BEGIN
* -- Return error message if the exam already exists
* SELECT 'Error: Exam with ID ' + @exam\_id + ' already exists.' AS Result;
* END
* END;
* storedProcedurel5-ViewCoursesAndExams
* use univdb
* go
* CREATE FUNCTION ViewCoursesAndExams
* (
* @admin\_name VARCHAR(50)
* )
* RETURNS TABLE
* AS
* RETURN
* (
* SELECT
* C.course\_id,
* C.course\_name,
* C.teacher\_id,
* T.teacher\_name,
* C.hours,
* C.credits,
* C.semester,
* E.exam\_id,
* E.exam\_date,
* E.start\_time,
* E.end\_time
* FROM
* Course AS C
* LEFT JOIN
* Exam AS E ON C.course\_id = E.course\_id
* LEFT JOIN
* Teacher AS T ON C.teacher\_id = T.teacher\_id -- Join with Teacher table
* WHERE
* C.admin\_name = @admin\_name
* );
* storedProcedure16-DeleteStudentByld
* use univdb
* go
* CREATE PROCEDURE DeleteStudentById
* @studentID CHAR(5)
* AS
* BEGIN
* DELETE FROM Student WHERE student\_id = @studentID;
* END
* trigger1-GPA
* USE univdb;
* GO
* -- Drop the existing trigger named UpdateGPA
* DROP TRIGGER UpdateGPA;
* USE univdb;
* GO
* CREATE TRIGGER UpdateStudentGPA
* ON MarkRegister
* AFTER INSERT, UPDATE
* AS
* BEGIN
* DECLARE @studentID CHAR(5);
* -- Get the student ID from the affected rows (inserted or updated)
* SELECT @studentID = student\_id
* FROM inserted;
* -- Update the student's GPA
* UPDATE Student
* SET GPA = (
* SELECT ISNULL(SUM(m.mark \* c.credits) / NULLIF(SUM(c.credits), 0), 0)
* FROM MarkRegister m
* INNER JOIN Course c ON m.course\_id = c.course\_id
* WHERE m.student\_id = @studentID
* )
* WHERE student\_id = @studentID;
* END;
* trigger2-StudentCoursesInsertToMarkRegister
* USE univdb;
* GO
* -- Create a trigger
* CREATE TRIGGER trg\_StudentCoursesInsert
* ON StudentCourses
* AFTER INSERT
* AS
* BEGIN
* INSERT INTO MarkRegister (student\_id, course\_id, exam\_id, mark)
* SELECT i.student\_id, i.course\_id, e.exam\_id, NULL
* FROM inserted i
* JOIN Course c ON i.course\_id = c.course\_id
* JOIN Exam e ON e.course\_id = c.course\_id;
* END;
* trigger3-StudentCoursesDeleteToMarkRegister
* USE univdb;
* GO
* -- Create a trigger
* CREATE TRIGGER trg\_StudentCoursesDelete
* ON StudentCourses
* AFTER DELETE
* AS
* BEGIN
* -- Delete records from MarkRegister for each deleted entry in StudentCourses
* DELETE FROM MarkRegister
* WHERE EXISTS (
* SELECT 1
* FROM deleted d
* WHERE MarkRegister.course\_id = d.course\_id
* );
* END;
* trigger4-TriggerDeleteStudent
* USE univdb
* GO
* CREATE TRIGGER TriggerDeleteStudent
* ON Student
* INSTEAD OF DELETE
* AS
* BEGIN
* DELETE FROM StudentCourses WHERE student\_id IN (SELECT student\_id FROM deleted);
* DELETE FROM MarkRegister WHERE student\_id IN (SELECT student\_id FROM deleted);
* DELETE FROM Student WHERE student\_id IN (SELECT student\_id FROM deleted);
* END
* GO
* Cursor1-UpdateMarksForCourse
* use univdb
* go
* CREATE PROCEDURE UpdateMarksForCourse
* @courseID CHAR(5)
* AS
* BEGIN
* DECLARE @examID CHAR(5);
* DECLARE @studentID CHAR(5);
* DECLARE @currentMark DECIMAL(6,2);
* -- Declare and open the cursor
* DECLARE MarkCursor CURSOR FOR
* SELECT exam\_id, student\_id, mark
* FROM MarkRegister
* WHERE course\_id = @courseID AND mark >= 40 AND mark <= 49;
* OPEN MarkCursor;
* -- Fetch the first row
* FETCH NEXT FROM MarkCursor INTO @examID, @studentID, @currentMark;
* -- Loop through the cursor
* WHILE @@FETCH\_STATUS = 0
* BEGIN
* -- Update the mark to 50
* UPDATE MarkRegister
* SET mark = 50
* WHERE exam\_id = @examID AND student\_id = @studentID;
* -- Fetch the next row
* FETCH NEXT FROM MarkCursor INTO @examID, @studentID, @currentMark;
* END
* -- Close and deallocate the cursor
* CLOSE MarkCursor;
* DEALLOCATE MarkCursor;
* END
* GO
* Index
* use univdb
* go
* ---
* /\*==============================================================\*/
* /\* Index: Teaches\_FK \*/
* /\*==============================================================\*/
* create index Teaches\_FK on Course (
* teacher\_id ASC
* )
* go
* /\*==============================================================\*/
* /\* Index: AdminC\_FK \*/
* /\*==============================================================\*/
* create index AdminC\_FK on Course (
* admin\_name ASC
* )
* go
* /\*==============================================================\*/
* /\* Index: AdminT\_FK \*/
* /\*==============================================================\*/
* create index AdminT\_FK on Teacher (
* admin\_name ASC
* )
* go
* /\*==============================================================\*/
* /\* Index: AdminS\_FK \*/
* /\*==============================================================\*/
* create index AdminS\_FK on Student (
* admin\_name ASC
* )
* go
* /\*==============================================================\*/
* /\* Index: MarkRegister1\_FK \*/
* /\*==============================================================\*/
* create index MarkRegister1\_FK on MarkRegister (
* course\_id ASC
* )
* go
* /\*==============================================================\*/
* /\* Index: MarkRegister2\_FK \*/
* /\*==============================================================\*/
* create index MarkRegister2\_FK on MarkRegister (
* exam\_id ASC
* )
* go
* /\*==============================================================\*/
* /\* Index: MarkRegister3\_FK \*/
* /\*==============================================================\*/
* create index MarkRegister3\_FK on MarkRegister (
* student\_id ASC
* )
* go
* /\*==============================================================\*/
* /\* Index: StudentCourses1\_FK \*/
* /\*==============================================================\*/
* create index StudentCourses1\_FK on StudentCourses (
* course\_id ASC
* )
* go
* /\*==============================================================\*/
* /\* Index: StudentCourses2\_FK \*/
* /\*==============================================================\*/
* create index StudentCourses2\_FK on StudentCourses (
* student\_id ASC
* )
* go
* /\*==============================================================\*/
* /\* Index: Exam\_FK \*/
* /\*==============================================================\*/
* create index Exam\_FK on StudentCourses (
* course\_id ASC
* )
* go
* security
* use univdb
* go
* sp\_addlogin 'zeinab Saghir', 'CS1234', 'univdb', 'us\_english'
* select name from master..syslogins
* go
* use univdb
* go
* sp\_addrole 'Univ\_Managers'
* go
* sp\_adduser 'zeinab Saghir', 'mark\_manag', 'Univ\_Managers'
* go
* grant select, insert, update, delete on Course to Univ\_Managers
* grant select, insert, update, delete on Student to Univ\_Managers
* grant select, insert, update, delete on Teacher to Univ\_Managers
* grant select, insert, update, delete on MarkRegister to Univ\_Managers
* grant select, insert, update, delete on StudentCourses to Univ\_Managers
* grant select, insert, update, delete on Exam to Univ\_Managers
* go

BAT

@echo off

set projloc=E:\AliSweidan\_HabibKhalaf

set conn=-S ALI-SWEIDAN -U sa -P 3li@admin -w 300

cls

echo Begining on top of MS Sqlserver DBMS engine...

sqlcmd %conn% -i "%projloc%\sql.scripts\1.create-univdb.sql" -o "%projloc%\univdb\log\1.create-univdb.log"

echo database created...

echo Begining on top of MS Sqlserver DBMS engine...

echo Security

sqlcmd %conn% -i "%projloc%\sql.scripts\Security.sql" -o "%projloc%\univdb\log\Security.log"

if %ERRORLEVEL% neq 0 (

echo Error occurred during script execution.

exit /b %ERRORLEVEL%

)

echo create tables...

sqlcmd %conn% -i "%projloc%\sql.scripts\2.create-univdb-tables.sql" -o "%projloc%\univdb\log\2.create-univdb-tables.log"

if %ERRORLEVEL% neq 0 (

echo Error occurred during script execution.

exit /b %ERRORLEVEL%

)

echo insert data....

sqlcmd %conn% -i "%projloc%\sql.scripts\3.create-univdb-insertdata.sql" -o "%projloc%\univdb\log\3.create-univdb-insertdata.log"

if %ERRORLEVEL% neq 0 (

echo Error occurred during script execution.

exit /b %ERRORLEVEL%

)

echo index

sqlcmd %conn% -i "%projloc%\sql.scripts\index.sql" -o "%projloc%\univdb\log\index.log"

if %ERRORLEVEL% neq 0 (

echo Error occurred during script execution.

exit /b %ERRORLEVEL%

)

echo Cursor1-UpdateMarksForCourse

sqlcmd %conn% -i "%projloc%\sql.scripts\Cursor1-UpdateMarksForCourse.sql" -o "%projloc%\univdb\log\univdb-Cursor1-UpdateMarksForCourse.log"

echo function1-CalculateTotalStudentCredits

sqlcmd %conn% -i "%projloc%\sql.scripts\function1-CalculateTotalStudentCredits.sql" -o "%projloc%\univdb\log\univdb-function1-CalculateTotalStudentCredits.log"

echo function2-GetRegisteredTeachersFunction

sqlcmd %conn% -i "%projloc%\sql.scripts\function2-GetRegisteredTeachersFunction.sql" -o "%projloc%\univdb\log\univdb-function2-GetRegisteredTeachersFunction.log"

echo function3-getRegisteredStudents

sqlcmd %conn% -i "%projloc%\sql.scripts\function3-getRegisteredStudents.sql" -o "%projloc%\univdb\log\univdb-function3-getRegisteredStudents.log"

echo storedprocedure1-GetStudentCoursesAndMarksWithSemesterAndCredit

sqlcmd %conn% -i "%projloc%\sql.scripts\storedprocedure1-GetStudentCoursesAndMarksWithSemesterAndCredit.sql" -o "%projloc%\univdb\log\storedprocedure1-GetStudentCoursesAndMarksWithSemesterAndCredit.log"

echo storedprocedure2(withTransaction)-EnrollStudentInSemester1

sqlcmd %conn% -i "%projloc%\sql.scripts\storedprocedure2(withTransaction)-EnrollStudentInSemester1.sql" -o "%projloc%\univdb\log\storedprocedure2(withTransaction)-EnrollStudentInSemester1.log"

echo storedprocedure3(withTransaction)-EnrollStudentInSemester

sqlcmd %conn% -i "%projloc%\sql.scripts\storedprocedure3(withTransaction)-EnrollStudentInSemester.sql" -o "%projloc%\univdb\log\storedprocedure3(withTransaction)-EnrollStudentInSemester.log"

echo storedprocedure4(withTransaction)-UnenrollStudentfromCourse

sqlcmd %conn% -i "%projloc%\sql.scripts\storedprocedure4(withTransaction)-UnenrollStudentfromCourse.sql" -o "%projloc%\univdb\log\storedprocedure4(withTransaction)-UnenrollStudentfromCourse.log"

echo storedprocedure5-GetTeacherCourses

sqlcmd %conn% -i "%projloc%\sql.scripts\storedprocedure5-GetTeacherCourses.sql" -o "%projloc%\univdb\log\storedprocedure5-GetTeacherCourses.log"

echo storedProcedure6(ReturnPassedCourses)

sqlcmd %conn% -i "%projloc%\sql.scripts\storedProcedure6(ReturnPassedCourses).sql" -o "%projloc%\univdb\log\storedProcedure6(ReturnPassedCourses).log"

echo storedProcedure7(ReturnFailedCourses)

sqlcmd %conn% -i "%projloc%\sql.scripts\storedProcedure7(ReturnFailedCourses).sql" -o "%projloc%\univdb\log\storedProcedure7(ReturnFailedCourses).log"

echo storedProcedure8-GetTeacherExams

sqlcmd %conn% -i "%projloc%\sql.scripts\storedProcedure8-GetTeacherExams.sql" -o "%projloc%\univdb\log\storedProcedure8-GetTeacherExams.log"

echo storedProcedure9-AddMarksForExam

sqlcmd %conn% -i "%projloc%\sql.scripts\storedProcedure9-AddMarksForExam.sql" -o "%projloc%\univdb\log\storedProcedure9-AddMarksForExam.log"

echo storedProcedure10-GetCourseForMark

sqlcmd %conn% -i "%projloc%\sql.scripts\storedProcedure10-GetCourseForMark.sql" -o "%projloc%\univdb\log\storedProcedure10-GetCourseForMark.log"

echo storedProcedure11-AddANDEditTeacher

sqlcmd %conn% -i "%projloc%\sql.scripts\storedProcedure11-AddANDEditTeacher.sql" -o "%projloc%\univdb\log\storedProcedure11-AddANDEditTeacher.log"

echo storedProcedure12-AddStudent

sqlcmd %conn% -i "%projloc%\sql.scripts\storedProcedure12-AddStudent.sql" -o "%projloc%\univdb\log\storedProcedure12-AddStudent.log"

echo storedProcedure13-AddCourse

sqlcmd %conn% -i "%projloc%\sql.scripts\storedProcedure13-AddCourse.sql" -o "%projloc%\univdb\log\storedProcedure13-AddCourse.log"

echo storedProcedure14-AddExam

sqlcmd %conn% -i "%projloc%\sql.scripts\storedProcedure14-AddExam.sql" -o "%projloc%\univdb\log\storedProcedure14-AddExam.log"

echo storedProcedure15-ViewCoursesAndExams

sqlcmd %conn% -i "%projloc%\sql.scripts\storedProcedure15-ViewCoursesAndExams.sql" -o "%projloc%\univdb\log\storedProcedure15-ViewCoursesAndExams.log"

echo storedProcedure16-DeleteStudentById

sqlcmd %conn% -i "%projloc%\sql.scripts\storedProcedure16-DeleteStudentById.sql" -o "%projloc%\univdb\log\storedProcedure16-DeleteStudentById.log"

echo trigger1-GPA

sqlcmd %conn% -i "%projloc%\sql.scripts\trigger1-GPA.sql" -o "%projloc%\univdb\log\univdb-trigger1-GPA.log"

echo trigger2-StudentCoursesInsertToMarkRegister

sqlcmd %conn% -i "%projloc%\sql.scripts\trigger2-StudentCoursesInsertToMarkRegister.sql" -o "%projloc%\univdb\log\trigger2-StudentCoursesInsertToMarkRegister.log"

echo trigger3-StudentCoursesDeleteToMarkRegister

sqlcmd %conn% -i "%projloc%\sql.scripts\trigger3-StudentCoursesDeleteToMarkRegister.sql" -o "%projloc%\univdb\log\trigger3-StudentCoursesDeleteToMarkRegister.log"

echo trigger4-TriggerDeleteStudent

sqlcmd %conn% -i "%projloc%\sql.scripts\trigger4-TriggerDeleteStudent.sql" -o "%projloc%\univdb\log\trigger4-TriggerDeleteStudent.log"

echo End of batch file....