

ONLINE COURSES SYSTEM



ENG: DOAA



Overview

• Team Members	03
• Introduction	04
• ERD Design	05
• Mapping	06
• Create DB	07 - 08
• DB Diagram	09
• Create Procedure	10 - 12
• Create Views	11 - 14
• Create DB Functions	15 - 21
• Testing DB Functions	22
• Create Index	23
• Create Triggers	24 - 25
• Create Schema	26 - 27
• Create Cursor	28 - 29
• Create Constraints & Rules	30
• Thanks	31



Team Members



Kerolos Romany



Moaz Wahed



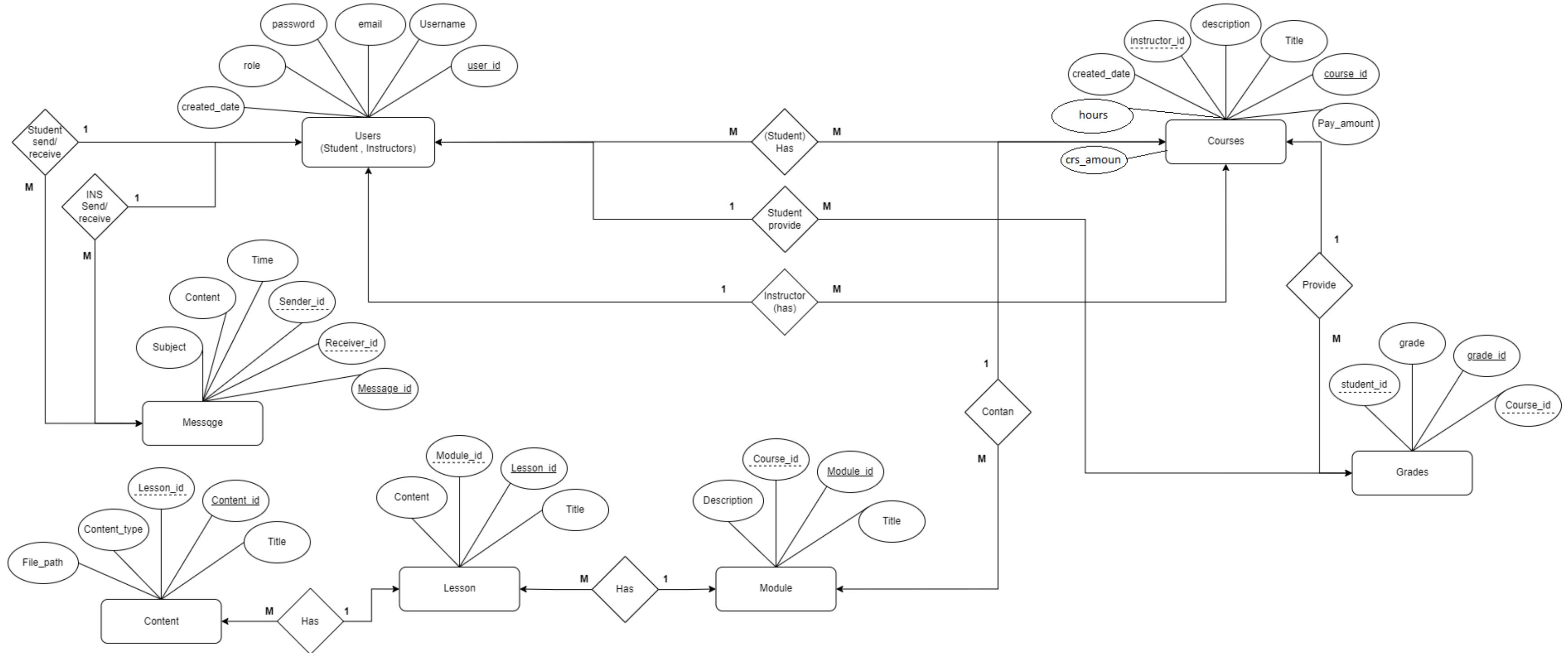
Mohamed Ahmed



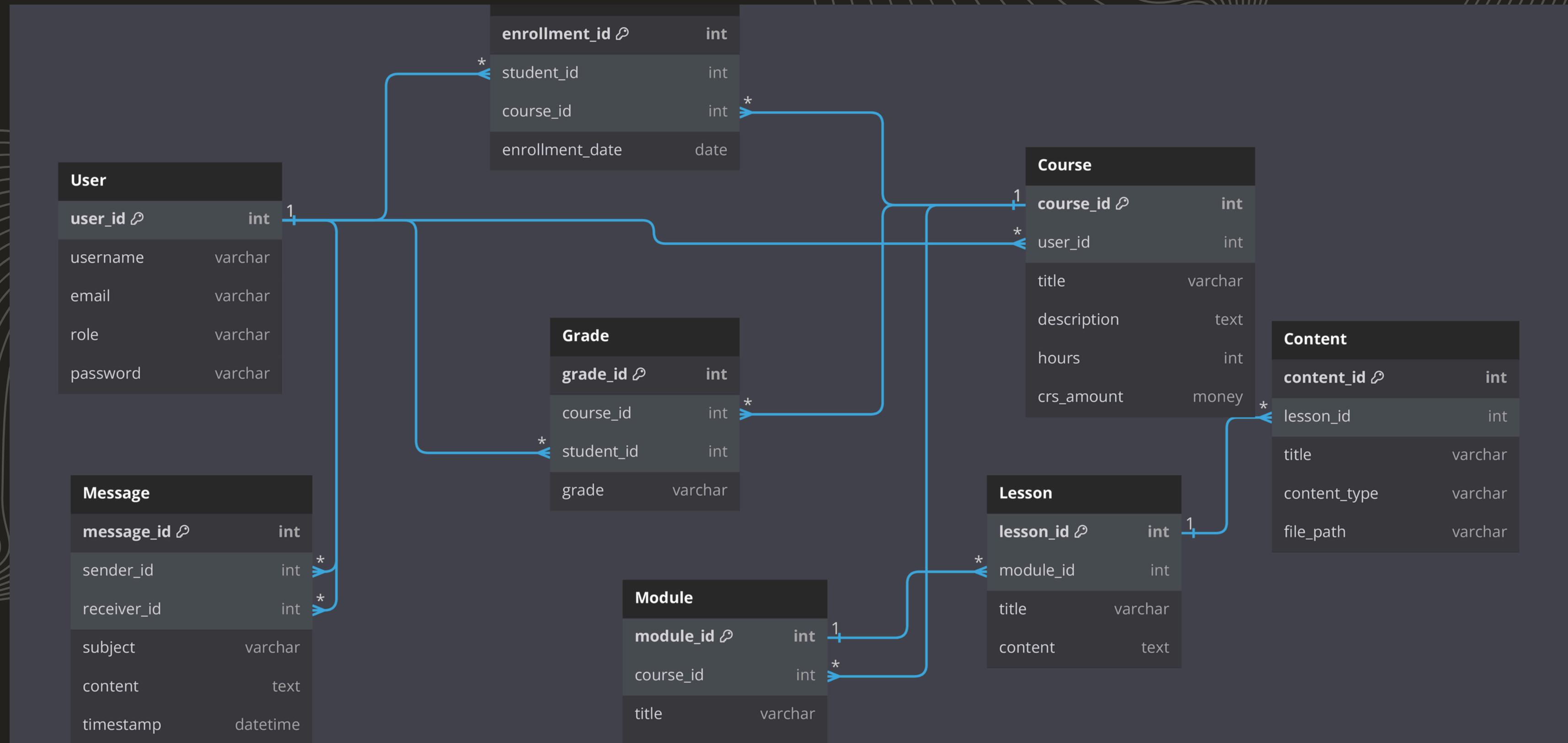
Introduction

Welcome to our Educational Management System, a comprehensive platform designed to streamline and enhance the learning experience for both students and instructors. This system encompasses a robust database structure that efficiently manages users, courses, modules, lessons, and related content. By facilitating seamless enrollment processes and grade management, our system ensures a smooth and interactive educational journey. Our goal is to provide an intuitive and user-friendly environment that promotes effective learning and teaching practices. Join us as we explore the capabilities and benefits of this innovative solution.

ERD Design



Mapping



Create DB

```
use Online_Courses_Sys

CREATE TABLE [User] (
    [user_id] int PRIMARY KEY,
    [username] nvarchar(255),
    [email] nvarchar(255),
    [role] nvarchar(255),
    [password] nvarchar(255)
)
GO

CREATE TABLE [Message] (
    [message_id] int PRIMARY KEY,
    [sender_id] int,
    [receiver_id] int,
    [subject] nvarchar(255),
    [content] text,
    [timestamp] datetime
)
```

```
- CREATE TABLE [Course] (
    [course_id] int PRIMARY KEY,
    [user_id] int,
    [title] nvarchar(255),
    [description] text
)
GO

- CREATE TABLE [Enrollment] (
    [enrollment_id] int PRIMARY KEY,
    [student_id] int,
    [course_id] int,
    [enrollment_date] date
)
GO
```

Create DB

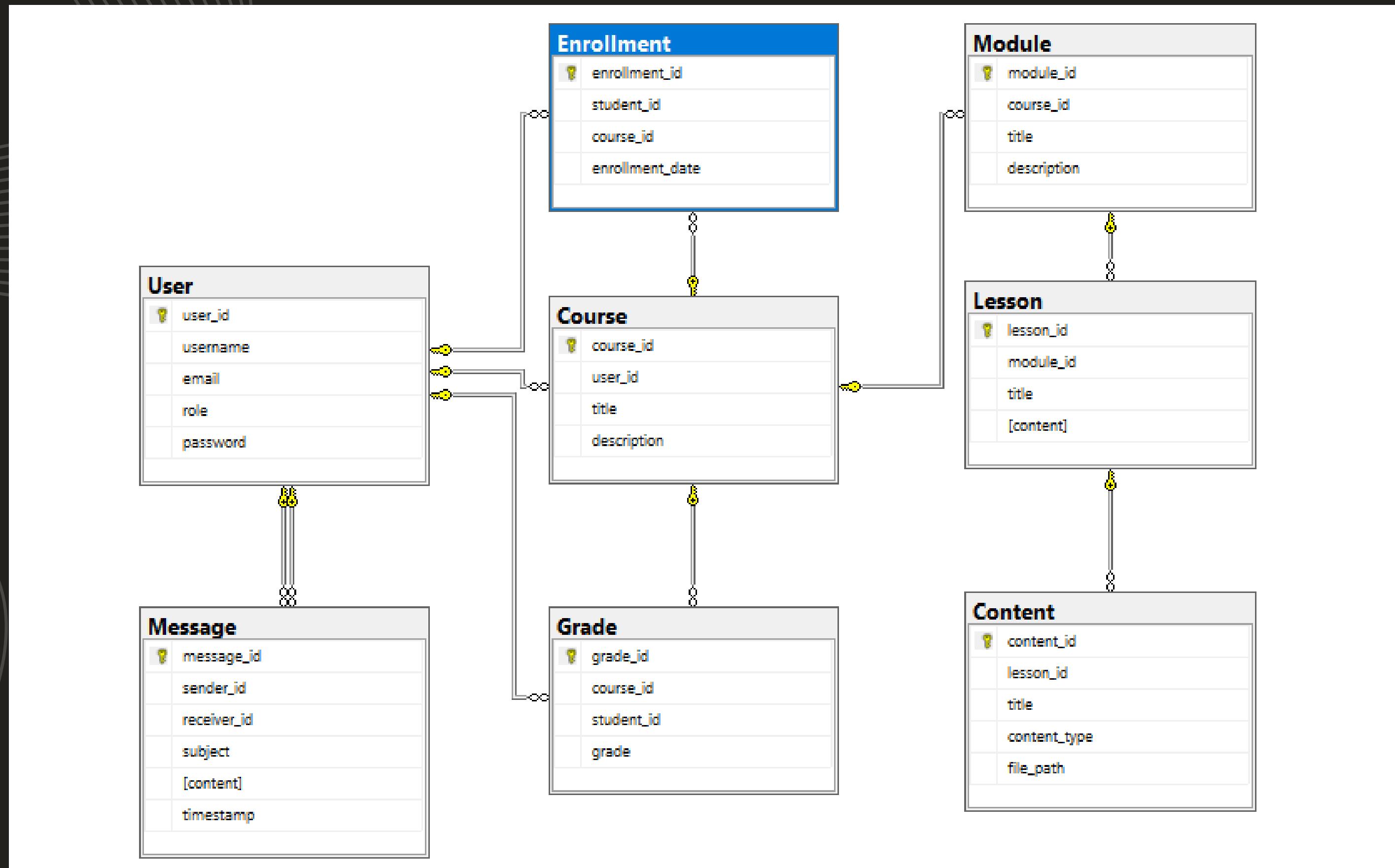
```
- CREATE TABLE [Grade] (
    [grade_id] int PRIMARY KEY,
    [course_id] int,
    [student_id] int,
    [grade] nvarchar(255)
)
GO
```

```
- CREATE TABLE [Module] (
    [module_id] int PRIMARY KEY,
    [course_id] int,
    [title] nvarchar(255),
    [description] text
)
GO
```

```
- CREATE TABLE [Lesson] (
    [lesson_id] int PRIMARY KEY,
    [module_id] int,
    [title] nvarchar(255),
    [content] text
)
GO
```

```
- CREATE TABLE [Content] (
    [content_id] int PRIMARY KEY,
    [lesson_id] int,
    [title] nvarchar(255),
    [content_type] nvarchar(255),
    [file_path] nvarchar(255)
)
GO
```

DB Diagram



Stored Procedure

----- User Table =>

----- insert

```
CREATE PROCEDURE InsertUser
    @Username NVARCHAR(255),
    @Email NVARCHAR(255),
    @Role NVARCHAR(50),
    @Password NVARCHAR(255)
AS
BEGIN
    INSERT INTO [User] (username, email, role, password)
    VALUES (@Username, @Email, @Role, @Password);
END;
```

----- Delete

```
CREATE PROCEDURE DeleteUser
    @UserID INT
AS
BEGIN
    DELETE FROM [User]
    WHERE user_id = @UserID;
END;
```

----- Course Table =>

----- Insert

```
CREATE PROCEDURE InsertCourse
    @UserID INT,
    @Title NVARCHAR(255),
    @Description NVARCHAR(MAX)
AS
BEGIN
    INSERT INTO Course (user_id, title, description)
    VALUES (@UserID, @Title, @Description);
END;
```

----- Delete

```
CREATE PROCEDURE DeleteCourse
    @CourseID INT
AS
BEGIN
    DELETE FROM Course
    WHERE course_id = @CourseID;
END;
```

----- Insert

```
CREATE PROCEDURE InsertModule
    @CourseID INT,
    @Title NVARCHAR(255),
    @Description NVARCHAR(MAX)
AS
BEGIN
    INSERT INTO Module (course_id, title, description)
    VALUES (@CourseID, @Title, @Description);
END;
```

----- Update

```
CREATE PROCEDURE DeleteModule
    @ModuleID INT
AS
BEGIN
    DELETE FROM Module
    WHERE module_id = @ModuleID;
END;
```

Stored Procedure

```
----- Lesson Table =>

-----insert

CREATE PROCEDURE InsertLesson
@ModuleID INT,
@Title NVARCHAR(255),
@Content NVARCHAR(MAX)
AS
BEGIN
    INSERT INTO Lesson (module_id, title, content)
    VALUES (@ModuleID, @Title, @Content);
END;

----- Delete

CREATE PROCEDURE DeleteLesson
@LessonID INT
AS
BEGIN
    DELETE FROM Lesson
    WHERE lesson_id = @LessonID;
END;

----- Content Table =>

----- Insert

CREATE PROCEDURE InsertContent
@LessonID INT,
@Title NVARCHAR(255),
@ContentType NVARCHAR(50),
@FilePath NVARCHAR(255)
AS
BEGIN
    INSERT INTO Content (lesson_id, title, content_type, file_path)
    VALUES (@LessonID, @Title, @ContentType, @FilePath);
END;

----- Delete

CREATE PROCEDURE DeleteContent
@ContentID INT
AS
BEGIN
    DELETE FROM Content
    WHERE content_id = @ContentID;
END;

----- Enrollment Table=>

----- insert

CREATE PROCEDURE InsertEnrollment
@studentID INT,
@courseID INT,
@enrollmentDate DATE
AS
BEGIN
    INSERT INTO Enrollment (student_id, course_id, enrollment_date)
    VALUES (@StudentID, @CourseID, @EnrollmentDate);
END;

----- Delete

CREATE PROCEDURE DeleteEnrollment
@enrollmentID INT
AS
BEGIN
    DELETE FROM Enrollment
    WHERE enrollment_id = @enrollmentID;
END;
```

Stored Procedure

```
----- Grade Table =>

---- insert

CREATE PROCEDURE InsertGrade
    @CourseID INT,
    @StudentID INT,
    @Grade NVARCHAR(2)
AS
BEGIN
    INSERT INTO Grade (course_id, student_id, grade)
    VALUES (@CourseID, @StudentID, @Grade);
END;

---- delete

CREATE PROCEDURE DeleteGrade
    @GradeID INT
AS
BEGIN
    DELETE FROM Grade
    WHERE grade_id = @GradeID;
END;
```

```
----- Message Table =>

---- insert

CREATE PROCEDURE InsertMessage
    @SenderID INT,
    @ReceiverID INT,
    @Subject NVARCHAR(255),
    @Content NVARCHAR(MAX)
AS
BEGIN
    INSERT INTO Message (sender_id, receiver_id, subject, content, Timestamp )
    VALUES (@SenderID, @ReceiverID, @Subject, @Content, getdate());
END;

---- delete

CREATE PROCEDURE DeleteMessage
    @MessageID INT
AS
BEGIN
    DELETE FROM Message
    WHERE message_id = @MessageID;
END;
```

Business Requirements:

DB Views

==== 1. View to Show Students Only

```
CREATE VIEW StudentsOnly AS  
SELECT user_id, username, email, role  
FROM [User]  
WHERE role = 'student';
```

====2. View to Show Instructors Only

```
CREATE VIEW InstructorsOnly AS  
SELECT user_id, username, email, role  
FROM [User]  
WHERE role = 'instructor';
```

==== 3. View to Show Enrolled Courses Only

```
alter VIEW courses.EnrolledCoursesOnly AS  
select * from (  
SELECT c.course_id, c.title, c.description,  
e.student_id, e.enrollment_date ,  
row_number() over (partition by c.course_id order by c.title) as RN  
FROM Course c INNER JOIN Enrollment e  
ON c.course_id = e.course_id) as tab  
where RN = 1  
--select * from courses.EnrolledCoursesOnly
```

==== 4. View to Show Non-Enrolled Courses Only

```
alter VIEW courses.NonEnrolledCoursesOnly AS  
SELECT c.course_id, c.title, c.description  
FROM Course c  
WHERE c.course_id NOT IN (SELECT e.course_id FROM Enrollment e);
```

DB Views

--== 5. View to Show Instructors Without Courses

```
alter VIEW ins.InstructorsWithoutCourses AS  
SELECT u.user_id, u.username, u.email  
FROM [User] u  
WHERE u.role = 'instructor'  
AND u.user_id NOT IN (SELECT c.user_id FROM Course c);  
select * from ins.InstructorsWithoutCourses
```

--== 6. View to Show Students Not Enrolled in Any Courses

```
CREATE VIEW StudentsNotEnrolledInCourses AS  
SELECT u.user_id, u.username, u.email, u.role  
FROM [User] u LEFT JOIN Enrollment e ON u.user_id = e.student_id  
WHERE u.role = 'student' AND e.course_id IS NULL;
```

Business Requirements:

DB Functions

```
- - - - - Functions  
- - - - - get the top n courses with the highest number of enrolled  
CREATE FUNCTION dbo.GetTopNCoursesWithHighestSold (@TopN INT)  
RETURNS TABLE  
AS  
RETURN  
(  
    SELECT TOP (@TopN) c.course_id, c.title, COUNT(e.course_id) AS enrollment_count  
    FROM Course c JOIN Enrollment e  
    ON c.course_id = e.course_id  
    GROUP BY c.course_id, c.title  
    ORDER BY enrollment_count DESC  
);
```

DB Functions

```
--4= the top n courses with the highest number of student enrolled in  
--=> Function GetTopNCoursesWithHieghestSold()  
  
CREATE FUNCTION dbo.GetTopNCoursesWithHighestSold (@TopN INT)  
RETURNS TABLE  
AS  
RETURN  
(  
    SELECT TOP (@TopN) c.course_id, c.title, COUNT(e.course_id) AS enrollment_count  
    FROM Course c JOIN Enrollment e  
    ON c.course_id = e.course_id  
    GROUP BY c.course_id, c.title  
    ORDER BY enrollment_count DESC  
);
```

DB Functions

```
----- student Questions & answers -----  
  
--7= show a specific course with it's grade  
---> FUNCTION show_course_with_grade(student_id INT, course_id INT)  
  
CREATE FUNCTION show_course_with_grade(@student_id INT, @course_id INT)  
RETURNS TABLE  
AS  
RETURN  
    SELECT c.title AS course_title, g.grade  
    FROM Course c  
    JOIN Grade g ON c.course_id = g.course_id  
    WHERE g.student_id = @student_id AND c.course_id = @course_id;
```

```
--8= show his all courses and his grades  
---> FUNCTION show_student_courses_and_grades(student_id INT)  
  
CREATE FUNCTION show_student_courses_and_grades(@student_id INT)  
RETURNS TABLE  
AS  
RETURN  
    SELECT c.title AS course_title, g.grade  
    FROM Course c  
    JOIN Grade g ON c.course_id = g.course_id  
    WHERE g.student_id = @student_id;  
  
----- instructor Questions & answers -----
```

```
--9= show all his courses and the count of student enrolled in  
---> FUNCTION show_instructor_courses_and_enrollment_count(instructor_id INT)  
  
CREATE FUNCTION show_instructor_courses_and_enrollment_count(@instructor_id INT)  
RETURNS TABLE  
AS  
RETURN  
    SELECT c.title AS course_title, COUNT(e.student_id) AS student_count  
    FROM Course c  
    LEFT JOIN Enrollment e ON c.course_id = e.course_id  
    WHERE c.user_id = @instructor_id  
    GROUP BY c.course_id, c.title;
```

DB Functions

```
--10= show all his courses with the highest enrolled
--=> FUNCTION show_instructor_courses_with_highest_enrollment(instructor_id INT)

CREATE FUNCTION show_instructor_courses_with_highest_enrollment(@instructor_id INT)
RETURNS TABLE
AS
RETURN
    WITH CourseEnrollments AS (
        SELECT c.course_id, c.title, COUNT(e.student_id) AS student_count
        FROM Course c
        LEFT JOIN Enrollment e ON c.course_id = e.course_id
        WHERE c.user_id = @instructor_id
        GROUP BY c.course_id, c.title
    )
    SELECT ce.title, ce.student_count
    FROM CourseEnrollments ce
    WHERE ce.student_count = (
        SELECT MAX(student_count)
        FROM CourseEnrollments
    );
}
```

DB Functions

```
--11= show all his courses with no enrollments  
--=> FUNCTION show_instructor_courses_with_no_enrollments(instructor_id INT)  
  
CREATE FUNCTION show_instructor_courses_with_no_enrollments(@instructor_id INT)  
RETURNS TABLE  
AS  
RETURN  
    SELECT c.title AS course_title  
    FROM Course c  
    LEFT JOIN Enrollment e ON c.course_id = e.course_id  
    WHERE c.user_id = @instructor_id  
    GROUP BY c.course_id, c.title  
    HAVING COUNT(e.student_id) = 0;
```

DB Functions

```
--===== mix
--12= show all the instrucors name with he searches for
---> FUNCTION search_instructors_by_name(search_term VARCHAR)

CREATE FUNCTION search_instructors_by_name(@search_term VARCHAR)
RETURNS TABLE
AS
RETURN
    SELECT u.username AS instructor_name
    FROM [User] u
    WHERE u.role = 'instructor' AND u.username LIKE '%' + @search_term + '%';

--13= show all the courses to a spesific instructor
---> FUNCTION show_instructor_courses_and_student_count(instructor_id INT)

CREATE FUNCTION show_instructor_courses_and_student_count(@instructor_id INT)
RETURNS TABLE
AS
RETURN
    SELECT c.title AS course_title, COUNT(e.student_id) AS student_count
    FROM Course c
    LEFT JOIN Enrollment e ON c.course_id = e.course_id
    WHERE c.user_id = @instructor_id
    GROUP BY c.course_id, c.title;
```

DB Functions

```
--14= Show all courses name with he searches for
---> FUNCTION search_courses_by_name(search_term VARCHAR)

CREATE FUNCTION search_courses_by_name(@search_term VARCHAR)
RETURNS TABLE
AS
RETURN
    SELECT c.title AS course_title
    FROM Course c
    WHERE c.title LIKE '%' + @search_term + '%';

--15= show The Course data , course's modules , course's lessons and course's contents to a specific course
---> get_course_details(@course_id INT)
CREATE FUNCTION get_course_details(@course_id INT)
RETURNS TABLE
AS
RETURN
    WITH CourseDetails AS (
        SELECT c.course_id, c.title AS course_title, c.description AS course_description, m.module_id,
               m.title AS module_title, m.description AS module_description, l.lesson_id,
               l.title AS lesson_title, co.content_id, co.title AS content_title,
               co.content_type, co.file_path
        FROM Course c
        LEFT JOIN Module m ON c.course_id = m.course_id
        LEFT JOIN Lesson l ON m.module_id = l.module_id
        LEFT JOIN Content co ON l.lesson_id = co.lesson_id
        WHERE c.course_id = @course_id
    )
    SELECT *
    FROM CourseDetails;
```

Testing DB Functions

```
|- ===== Functions =====  
  
---- Note that all the Functions are already create in the Bussiness Q&A Query file!  
---- we will not write them here  
---- but we can test them here ^_^  
  
select * from student.show_course_with_grade(1,1)  
----  
select * from student.show_student_courses_and_grades(1)  
----  
select * from ins.show_instructor_courses_and_enrollment_count(61)  
--select * from ins.InstructorsOnly  
--select * from courses.EnrolledCoursesOnly  
----  
select * from ins.show_instructor_courses_with_highest_enrollment(61)  
----  
select * from ins.show_instructor_courses_with_no_enrollments(87)  
--select * from courses.NonEnrolledCoursesOnly  
----  
select * from courses.GetTopNCoursesWithHighestSold(5)  
----  
select * from courses.get_course_details(12)  
----  
select * from sys_admin.search_courses_by_name('law')  
----  
select * from sys_admin.search_instructors_by_name('ins')
```

DB Index

==== Index on username in the User table

```
CREATE INDEX IDX_User_Username ON [User] (username);
```

==== Index on title in the Course table

```
CREATE INDEX IDX_Course_Title ON Course (title);
```

DB Triggers

```
- - - - - Triggers  
- - - - - stop all the delete process from enrollment table  
- - - - - create TRIGGER stop deleting  
on Enrollment  
INSTEAD OF delete  
as  
SELECT 'can't delete any record in that table'
```

DB Triggers

```
--==== Store all the log info for the insertion on enrollment table
-- create a log table (Audit)
create table Audit_log(
    st_id int,
    cu_id int,
    name varchar(50),
    cnew_Id int,
    oldhours date,
    date2 date
)
---- create Trigger to insert the log data to the Audit Table
create trigger Log_data
on Enrollment
after update
as
declare @st_ID int,@cu_ID int,@oldhours date ,@date2 date,@cnew_Id int
select @st_id=student_id from deleted
select @cu_id=course_id from deleted
select @oldhours= enrollment_date from deleted
select @cnew_Id=course_id from inserted
select @date2= enrollment_date from inserted
INSERT INTO Audit_log (st_id,cu_id,oldhours,cnew_Id,date2,name)
VALUES(@st_id,@cu_id , @oldhours , @cnew_Id , getdate(),SUSER_NAME())
```

Schema

```
--> Schema  
==== Create Schema for all the student data & Process  
create schema Student  
alter schema student transfer dbo.StudentsOnly  
alter schema student transfer dbo.StudentsNotEnrolledInCourses  
alter schema student transfer dbo.show_course_with_grade  
alter schema student transfer dbo.show_student_courses_and_grades  
  
==== Create Schema for all the Instructors data & process  
create schema ins  
alter schema ins transfer dbo.InstructorsWithoutCourses  
alter schema ins transfer dbo.InstructorsOnly  
alter schema ins transfer dbo.show_instructor_courses_and_enrollment_count  
alter schema ins transfer dbo.show_instructor_courses_with_highest_enrollment  
alter schema ins transfer dbo.show_instructor_courses_with_no_enrollments
```

Schema

```
--== Create Schema for all the Courses data
```

```
create schema courses
alter schema courses transfer dbo.EnrolledCoursesOnly
alter schema courses transfer dbo.NonEnrolledCoursesOnly
alter schema courses transfer dbo.GetTopNCoursesWithHighestSold
alter schema courses transfer dbo.get_course_details
```

```
--== create schema sys_admin for all admin process
```

```
create schema sys_admin
alter schema sys_admin transfer dbo.search_courses_by_name
alter schema sys_admin transfer dbo.search_instructors_by_name
```

DB Cursor

```
--===== Cursor
---- Using the Cursor, we will return all the students names and grades
---- that Enrolled in this course
CREATE FUNCTION GetUsersAndGradesInCourse (@user_id INT, @course_name NVARCHAR(100))
RETURNS @result TABLE (
    username NVARCHAR(100),
    grade CHAR(1)
)
AS
BEGIN
    DECLARE @course_id INT
    DECLARE @student_id INT
    DECLARE @username NVARCHAR(100)
    DECLARE @grade CHAR(1)
    -- Find the course_id from the course_name
    SELECT @course_id = course_id
    FROM Course
    WHERE title = @course_name;
    -- Cursor to iterate through students enrolled in the course
    DECLARE student_cursor CURSOR FOR
        SELECT e.student_id, u.username, g.grade
        FROM Enrollment e
        JOIN [User] u ON e.student_id = u.user_id
        JOIN Grade g ON e.course_id = g.course_id AND e.student_id = g.student_id
        WHERE e.course_id = @course_id;
    OPEN student_cursor;
```

DB Cursor

```
        FETCH NEXT FROM student_cursor INTO @student_id, @username, @grade;
        WHILE @@FETCH_STATUS = 0
        BEGIN
            -- Insert each student's name and grade into the result table
            INSERT INTO @result (username, grade)
            VALUES (@username, @grade);

            FETCH NEXT FROM student_cursor INTO @student_id, @username, @grade;
        END;

        CLOSE student_cursor;
        DEALLOCATE student_cursor;

        RETURN;
    END;
-- Example usage
SELECT * FROM GetUsersAndGradesInCourse(54, 'Introduction to Biology');
```

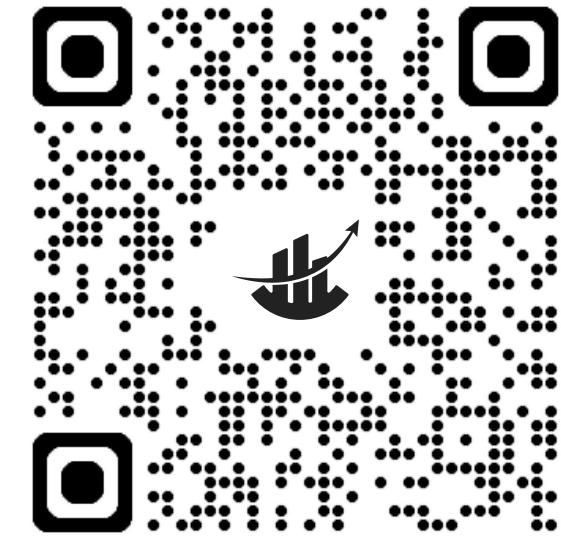
CONSTRAINTS & RULES

```
-->===== CONSTRAINTS & Rules =====  
  
----- putting a constrain to make sure that we will have just a Students and instructor  
----- in the User Table  
ALTER TABLE [user] ADD CONSTRAINT check_role CHECK (role IN ('student', 'instructor'));  
  
----- Buting a Rule to the Grade to make sure  
----- that what we will insert will be a real grade  
CREATE RULE Grade_Values AS @vales IN ('A', 'B', 'C', 'D', 'F')  
sp_bindrule 'Grade_Values', 'dbo.[grade].[grade]'  
-----
```



Thank You

Project Link on GitHub



Gerente General