

UNIVERSITY PORTAL

(DBMS PROJECT)

**Members:**

Muhammad Arsalan Warsi

(SP19-BSSE-0030 | [sp19bsse0030@maju.edu.pk](mailto:sp19bsse0030@maju.edu.pk))

Aliyan Ahmed

(SP19-BSSE-0028 | [sp19bsse0028@maju.edu.pk](mailto:sp19bsse0028@maju.edu.pk))

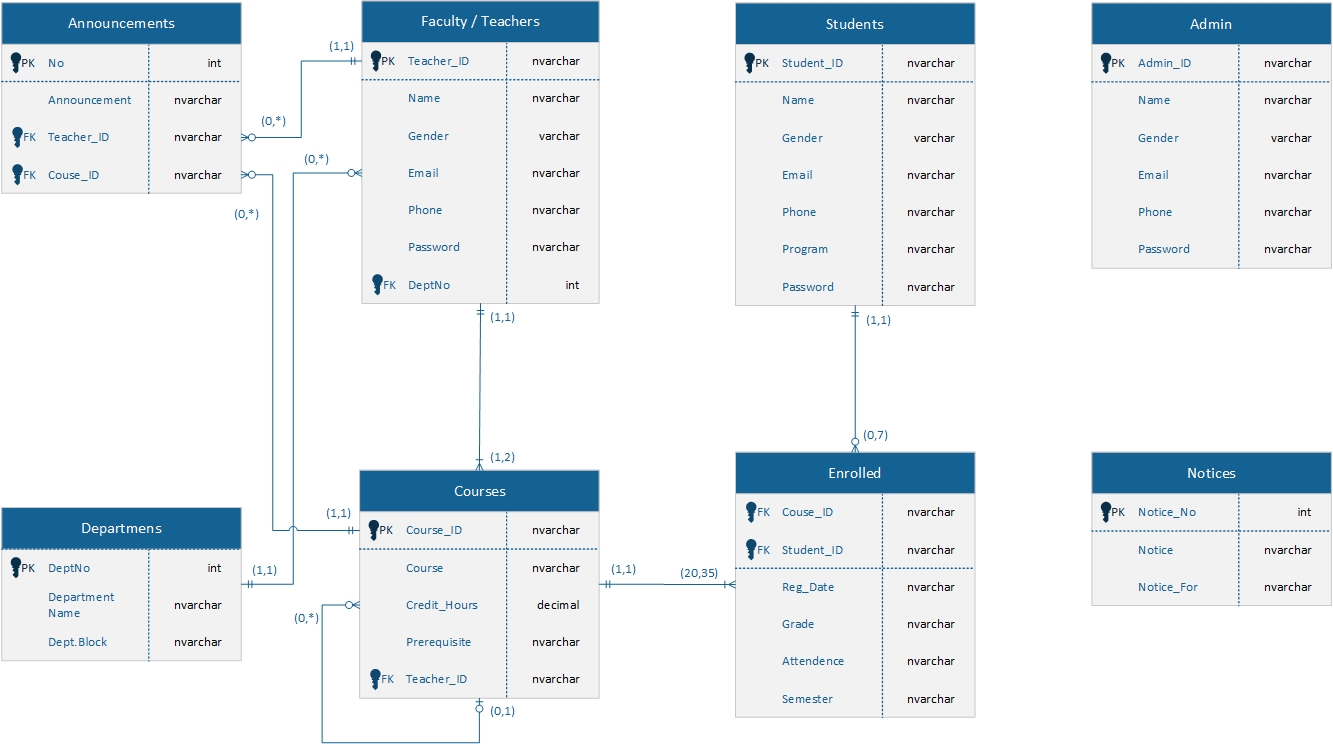
**Course:**

Database Management Systems Lab (BM)

INTRODUCTION

There are a number of peoples in a university having different roles. Admin, teachers and students are most important part of the university and it is not easy to manage and maintain all the records accurately. So, the university portal is the best way to maintain the record which allow the admin to view, update, add or delete the data and communicate with students and teachers. The university portal not only allows to do business easily but also saves time, manual paperwork, and manual work force which can help to reduce the cost and save the data accurately.

ENTITY RELATIONSHIP MODELING



NORMALIZATION

**1NF (1st Normal From):**

All the entities are in atomic column. All the tables in ERD have uniquely identifying keys (Primary Key).

**Reason:**

When we save the record in Database the new record will save in next row and each table have PK, so new similar record will save in table.

**2NF (2nd Normal From):**

In ERD the partially dependencies of tables have been solved. The attributes are functionally dependent on single attribute Primary key, except the enrolled tables because in us created to resolve many to many relationships.

**Reason:**

If we want to update, add or delete the record in table we only preform DML operation on single table and its impact on different tables.

**3NF (3rd Normal From):**

The Transitive dependency is resolve by breaking the tables due to non-primary key depend on non-primary key. Like Department table is created separately from teacher’s table.

**Reason:**

When we add new department or change the details of department it applies on each and every record in teacher’s table, so we did not need to update the details of every single record in teacher’s tables.

SQL DDL

**ADMIN Table:**

CREATE TABLE Admin(

Admin\_ID nvarchar(50) PRIMARY KEY,

Name nvarchar(50) NOT NULL,

Gender varchar(10),

Email nvarchar(70) NOT NULL,

PhoneNo nvarchar(20),

Password nvarchar(20) DEFAULT 'admin123' NOT NULL

)

**NOTICE Table:**

CREATE TABLE Notices(

Notice\_NO int PRIMARY KEY IDENTITY(1,1),

Notice nvarchar(400) NOT NULL,

Notice\_For varchar(20) NOT NULL

)

**DEPARTMENT Table:**

CREATE TABLE Departments(

Dept\_No int PRIMARY KEY,

Dept\_Name nvarchar(50) NOT NULL,

Dept\_Block varchar(20)

)

**TEACHERS Table:**

CREATE TABLE Teachers(

Teacher\_ID nvarchar(50) PRIMARY KEY,

Name nvarchar(50) NOT NULL,

Gender varchar(10),

Email nvarchar(70) NOT NULL,

PhoneNo nvarchar(20),

Password nvarchar(20) DEFAULT 'teacher123' NOT NULL,

Dept\_No int FOREIGN KEY REFERENCES Departments(Dept\_No)

ON DELETE NO ACTION

ON UPDATE CASCADE

NOT NULL

)

**STUDENTS Table:**

CREATE TABLE Students(

Student\_ID nvarchar(50) PRIMARY KEY,

Name nvarchar(50) NOT NULL,

Gender varchar(10),

Email nvarchar(70) NOT NULL,

PhoneNo nvarchar(20),

Program nvarchar(50) NOT NULL,

Password nvarchar(20) DEFAULT 'student123' NOT NULL

)

**COURSES Table:**

CREATE TABLE Courses(

Course\_ID nvarchar(10) PRIMARY KEY,

Course\_Name nvarchar(50) NOT NULL,

Credit\_Hours decimal(5,1) NOT NULL,

Prerequisite nvarchar(10) FOREIGN KEY REFERENCES Courses(Course\_ID) NULL,

Teacher\_ID nvarchar(50) FOREIGN KEY REFERENCES Teachers(Teacher\_ID)

ON DELETE NO ACTION

ON UPDATE CASCADE

NOT NULL

)

**ANNOUNCEMENTS Table:**

CREATE TABLE Announcements(

Ann\_No int PRIMARY KEY IDENTITY(1,1),

Announcement nvarchar(300) NOT NULL,

Teacher\_ID nvarchar(50) FOREIGN KEY REFERENCES Teachers(Teacher\_ID)

ON DELETE CASCADE

ON UPDATE CASCADE,

Course\_ID nvarchar(10) FOREIGN KEY REFERENCES Courses(Course\_ID)

ON DELETE CASCADE

)

**ENROLLED Table:**

CREATE TABLE Enrolled(

Course\_ID nvarchar(10) FOREIGN KEY REFERENCES Courses(Course\_ID)

ON DELETE SET NULL

ON UPDATE CASCADE,

Student\_ID nvarchar(50) FOREIGN KEY REFERENCES Students(Student\_ID)

ON DELETE SET NULL

ON UPDATE CASCADE,

Reg\_Date Date,

Attendence nvarchar(10),

Grade nvarchar(6),

Semester nvarchar(15)

)

USER ROLES

We have created the user login and assign a role or database owner so that user can manage the complete database.

The application users are:

Admin:

* Admin can add and delete students or faculty members.
* Admin can change or update details of students and faculty members.
* Admin can communicate with students and teachers through notices.

Faculty:

* Teacher can view student details.
* Teacher can communicate with students through announcements.

Student:

* Student can view their details and announcement from teachers and notices from admin.
* Student can check their enrolled course and check attendance.

FUNCTIONALITY DESCRIPTION

**Views:**

CREATE VIEW Admin\_Login\_Table

AS

SELECT Admin\_ID,Password

FROM Admin

CREATE VIEW Teacher\_Login\_Table

AS

SELECT Teacher\_ID,Password

FROM Teachers

CREATE VIEW Student\_Login\_Table

AS

SELECT Student\_ID,Password

FROM Students

Login views are created for admin, teachers and students that only contain the username and password which will used for matching the data.

CREATE VIEW Teacher\_Table

AS

SELECT t.Teacher\_ID,t.Name,t.Gender,t.Email,t.PhoneNo,t.Password,d.Dept\_Name FROM Teachers t

INNER JOIN Departments d

ON t.Dept\_No = d.Dept\_No

Teachers table view is created that will show the teacher details including teacher respective department name.

CREATE VIEW Course\_Table

AS

SELECT c.Course\_ID,c.Course\_Name,c.Credit\_Hours,d.Course\_Name AS 'Prerequisite',t.Name AS 'Teacher' FROM Courses c

LEFT JOIN Courses d

ON c.Prerequisite = d.Course\_ID

INNER JOIN Teachers t

ON c.Teacher\_ID = t.Teacher\_ID

The course table view is created to show the details of the course and the teacher’s name who is teaching that course.

CREATE VIEW Reg\_Course

AS

SELECT Course\_ID AS 'ID',Course\_Name AS 'Name',Credit\_Hours AS 'Cr.',Prerequisite AS 'Pre\_Req' FROM Courses

The Registration Course view will show the course detail and the prerequisite of that course.

CREATE VIEW Register\_Course

AS

SELECT e.Course\_ID AS 'ID',c.Course\_Name AS 'Name',e.Student\_ID,e.Semester FROM Enrolled e

INNER JOIN Courses c

on e.Course\_ID = c.Course\_ID

This view show student detail and student registered course details.

CREATE VIEW Course\_Registered

AS

SELECT e.Student\_ID as 'St\_ID',e.Course\_ID AS 'ID',c.Course\_Name AS 'Name',e.Attendence,e.Grade,t.Name AS 'Teacher',e.Semester FROM Enrolled e

INNER JOIN Courses c

ON e.Course\_ID = c.Course\_ID

INNER JOIN Teachers t

ON c.Teacher\_ID = t.Teacher\_ID

This view shows the course details in which the student is enrolled.

CREATE VIEW Notice\_Record

AS

SELECT Notice,Notice\_For FROM Notices

This view shows the notifications from the admin to teachers or students.

CREATE VIEW Student\_Announcement

AS

SELECT e.Student\_ID,e.Course\_ID AS 'Course', a.Announcement FROM Enrolled e

INNER JOIN Announcements a

ON e.Course\_ID = a.Course\_ID

This view shows the announcement from the teachers to students.

CREATE VIEW Teacher\_Announce

AS

SELECT Ann\_No AS 'ID',Announcement,Course\_ID AS 'Course',Teacher\_ID FROM Announcements

This view helps to delete the announcement from the teachers hands.

CREATE VIEW Teacher\_Course

AS

SELECT t.Teacher\_ID,c.Course\_ID AS 'ID',c.Course\_Name AS 'Course' FROM Teachers t

INNER JOIN Courses c

ON t.Teacher\_ID= c.Teacher\_ID

This view shows the details of the course that is taught by teacher.

CREATE VIEW Techer\_Assigned\_Course

AS

SELECT t.Teacher\_ID,c.Course\_ID AS 'ID',c.Course\_Name AS 'COURSE',c.Credit\_Hours AS 'Cr\_Hour' FROM Teachers t

INNER JOIN Courses c

ON t.Teacher\_ID = c.Teacher\_ID

This view helps the edit the details of the course that is taught by teacher.

CREATE VIEW Course\_Students

AS

SELECT c.Course\_ID,e.Student\_ID AS 'ID',s.Name,s.Email,e.Attendence,e.Grade,e.Semester FROM Courses c

INNER JOIN Enrolled e

on c.Course\_ID = e.Course\_ID

INNER JOIN Students s

ON e.Student\_ID = s.Student\_ID

This view shows the student that are registered in the course that is assigned to teacher.

**Procedures:**

CREATE PROCEDURE admin\_login

(

@ID nvarchar(50),

@Password nvarchar(20)

)

AS

BEGIN

SELECT 'true' FROM Admin\_Login\_Table WHERE Admin\_ID = @ID AND Password = @Password

END

CREATE PROCEDURE teacher\_login

(

@ID nvarchar(50),

@Password nvarchar(20)

)

AS

BEGIN

SELECT 'TRUE' FROM Teacher\_Login\_Table WHERE Teacher\_ID = @ID AND Password = @Password

END

CREATE PROCEDURE student\_login

(

@ID nvarchar(50),

@Password nvarchar(20)

)

AS

BEGIN

SELECT 'true' FROM Student\_Login\_Table WHERE Student\_ID = @ID AND Password = @Password

END

The procedure is created that verify the username and password of the user from the required tables.

CREATE PROCEDURE add\_student

(

@ID nvarchar(50),

@Name nvarchar(50),

@Gender varchar(10),

@Email nvarchar(70),

@PhoneNo nvarchar(20),

@Program nvarchar(50),

@Password nvarchar(20)

)

AS

BEGIN

INSERT INTO [dbo].[Students]

(Student\_ID,Name,Gender,Email,PhoneNo,Program,Password)

VALUES

(@ID,@Name,@Gender,@Email,@PhoneNo,@Program,@Password)

END

CREATE PROCEDURE add\_teacher

(

@ID nvarchar(50),

@Name nvarchar(50),

@Gender varchar(10),

@Email nvarchar(70),

@PhoneNo nvarchar(20),

@Password nvarchar(20),

@Dept\_No int

)

AS

BEGIN

INSERT INTO [dbo].[Teachers]

(Teacher\_ID,Name,Gender,Email,PhoneNo,Password,Dept\_No)

VALUES

(@ID,@Name,@Gender,@Email,@PhoneNo,@Password,@Dept\_No)

END

CREATE PROCEDURE add\_course

(

@Course\_ID nvarchar(10),

@Course\_Name nvarchar(50),

@Credit\_Hours decimal(5,1),

@Prerequisite nvarchar(10),

@Teacher\_ID nvarchar(50)

)

AS

BEGIN

INSERT INTO [dbo].[Courses]

(Course\_ID,Course\_Name,Credit\_Hours,Prerequisite,Teacher\_ID)

VALUES

(@Course\_ID,@Course\_Name,@Credit\_Hours,@Prerequisite,@Teacher\_ID)

END

CREATE PROCEDURE add\_admin

(

@Admin\_ID nvarchar(50),

@Name nvarchar(50),

@Gender varchar(10),

@Email nvarchar(70),

@PhoneNo nvarchar(20),

@Password nvarchar(20)

)

AS

BEGIN

INSERT INTO [dbo].[Admin]

(Admin\_ID,Name,Gender,Email,PhoneNo,Password)

VALUES

(@Admin\_ID,@Name,@Gender,@Email,@PhoneNo,@Password)

END

CREATE PROCEDURE add\_notice

(

@Notice nvarchar(400),

@Notice\_For varchar(20)

)

AS

BEGIN

INSERT INTO [dbo].[Notices]

(Notice,Notice\_For)

VALUES

(@Notice,@Notice\_For)

END

CREATE PROCEDURE add\_announcement

( @Announcement nvarchar(300),

@Teacher\_ID nvarchar(50),

@Course\_ID nvarchar(10)

)

AS

BEGIN

INSERT INTO [dbo].[Announcements]

(Announcement,Teacher\_ID,Course\_ID)

VALUES

(@Announcement,@Teacher\_ID,@Course\_ID)

END

The procedure is created to insert the new record of admin, teacher student, notice and announcement in the database.

CREATE PROCEDURE update\_student

(

@ID nvarchar(50),

@Name nvarchar(50),

@Email nvarchar(70),

@PhoneNo nvarchar(20),

@Password nvarchar(20)

)

AS

BEGIN

UPDATE Students

SET Name = @Name,

Email = @Email,

PhoneNo = @PhoneNo,

Password = @Password

WHERE Student\_ID = @ID

END

REATE PROCEDURE update\_teacher

(

@ID nvarchar(50),

@Name nvarchar(50),

@Email nvarchar(70),

@PhoneNo nvarchar(20),

@Password nvarchar(20),

@Dept\_No int

)

AS

BEGIN

UPDATE Teachers

SET Name = @Name,

Email = @Email,

PhoneNo = @PhoneNo,

Password = @Password,

Dept\_No = @Dept\_No

WHERE Teacher\_ID = @ID

END

CREATE PROCEDURE update\_course

(

@Course\_ID nvarchar(10),

@Course\_Name nvarchar(50),

@Credit\_Hours decimal(5,1),

@Prerequisite nvarchar(10),

@Teacher\_ID nvarchar(50)

)

AS

BEGIN

UPDATE Courses

SET Course\_Name = @Course\_Name,

Credit\_Hours = @Credit\_Hours,

Prerequisite = @Prerequisite,

Teacher\_ID = @Teacher\_ID

WHERE Course\_ID = @Course\_ID

END

CREATE PROCEDURE update\_admin

(

@Admin\_ID nvarchar(50),

@Name nvarchar(50),

@Email nvarchar(70),

@PhoneNo nvarchar(20),

@Password nvarchar(20)

)

AS

BEGIN

UPDATE Admin

SET Name = @Name,

Email = @Email,

PhoneNo = @PhoneNo,

Password = @Password

WHERE Admin\_ID = @Admin\_ID

END

The procedure is created to update the record of admin, teacher, student and course in the database.

CREATE PROCEDURE program\_students

(

@Program nvarchar(50)

)

AS

BEGIN

SELECT \* FROM Students WHERE Program LIKE '%'+@Program+'%';

END

CREATE PROCEDURE search\_student

(

@ID nvarchar(50)

)

AS

BEGIN

SELECT \* FROM Students WHERE Student\_ID LIKE '%'+@ID+'%'

END

CREATE PROCEDURE search\_teacher

(

@ID nvarchar(50)

)

AS

BEGIN

SELECT t.Teacher\_ID,t.Name,t.Gender,t.Email,t.PhoneNo,t.Password,d.Dept\_Name FROM Teachers t

INNER JOIN Departments d

ON t.Dept\_No = d.Dept\_No

WHERE Teacher\_ID LIKE '%'+@ID+'%'

END

CREATE PROCEDURE dept\_teacher

(

@Dept\_No nvarchar(50)

)

AS

BEGIN

SELECT t.Teacher\_ID,t.Name,t.Gender,t.Email,t.PhoneNo,t.Password,d.Dept\_Name FROM Teachers t

INNER JOIN Departments d

ON t.Dept\_No = d.Dept\_No

WHERE d.Dept\_No LIKE '%'+@Dept\_No+'%';

END

CREATE PROCEDURE search\_course

(

@ID nvarchar(50)

)

AS

BEGIN

SELECT c.Course\_ID,c.Course\_Name,c.Credit\_Hours,d.Course\_Name AS 'Prerequisite',t.Name AS 'Teacher' FROM Courses c

LEFT JOIN Courses d

ON c.Prerequisite = d.Course\_ID

INNER JOIN Teachers t

ON c.Teacher\_ID = t.Teacher\_ID

WHERE c.Course\_ID like '%'+@ID+'%'

END

The procedure is created to search the record of admin, teacher, student and course from the database.

CREATE PROCEDURE enroll\_student

(

@Course\_ID nvarchar(10),

@Student\_ID nvarchar(50),

@Reg\_Date date,

@Semester nvarchar(15)

)

AS

BEGIN

INSERT INTO [dbo].[Enrolled]

(Course\_ID,Student\_ID,Reg\_Date,Attendence,Grade,Semester)

VALUES

(@Course\_ID,@Student\_ID,@Reg\_Date,'99','I',@Semester)

END

The procedure is created to enroll the student in the course by admin.

**Triggers:**

CREATE TRIGGER count\_reg

ON Enrolled

AFTER INSERT

AS

BEGIN

IF(SELECT count(Student\_ID) FROM Enrolled WHERE Student\_ID = (SELECT Student\_ID FROM inserted) AND Semester = (SELECT Semester FROM inserted)) >5

BEGIN

ROLLBACK

END

IF(SELECT count(Course\_ID) FROM Enrolled WHERE Course\_ID = (select Course\_ID from inserted) AND Student\_ID = (SELECT Student\_ID FROM inserted) AND Semester = (SELECT Semester FROM inserted)) > 1

BEGIN

ROLLBACK

END

END

This trigger is created to apply the business rule in the database to ensure that the student is not enrolled in more than 5 courses.

CREATE TRIGGER count\_course

ON Courses

AFTER INSERT,UPDATE

AS

BEGIN

IF (SELECT count(Course\_ID) FROM Courses WHERE Teacher\_ID = (SELECT Teacher\_ID FROM inserted)) >2

BEGIN

ROLLBACK

END

END

This trigger is created to apply the business rule in the database to ensure that a teacher is only assigned to 1 or 2 courses.

CONCLUSION

The project was really informative and we learn many things while developing the project. We learn new tools like SQL Server, Visual studio and our login build was improved when executing this project. Some parts of the coding were difficult and some time the database work troubles us a lot. The most difficult part of the project is to handle the announcement from the teacher to students as there are many students enrolled in the courses that are assign to teacher. So, match the student with the course and teacher take a lot of time and effort. On the other hand, the easiest part was to insert the data into the tables. We have not imagined that we are able to implement these features in our project because at the start we are just working with basic things only like add, update and delete the record. View the records by joining the tables only. If we had to do this type of project again, we will be done exactly the project in the same manner but we remember the current mistakes we made in layout designing and, in a database, and resolve our mistakes on early-stage, not in the integration stage. Working on the project teaches us things like teamwork, meeting the deadline.