UNIVERSITY PORTAL
(DBMS PROJECT)

Members:

Muhammad Arsalan Warsi

(SP19-BSSE-0030 | sp19bsse0030@maju.edu.pk)

Aliyan Ahmed

(SP19-BSSE-0028 | sp19bsse0028@maju.edu.pk)

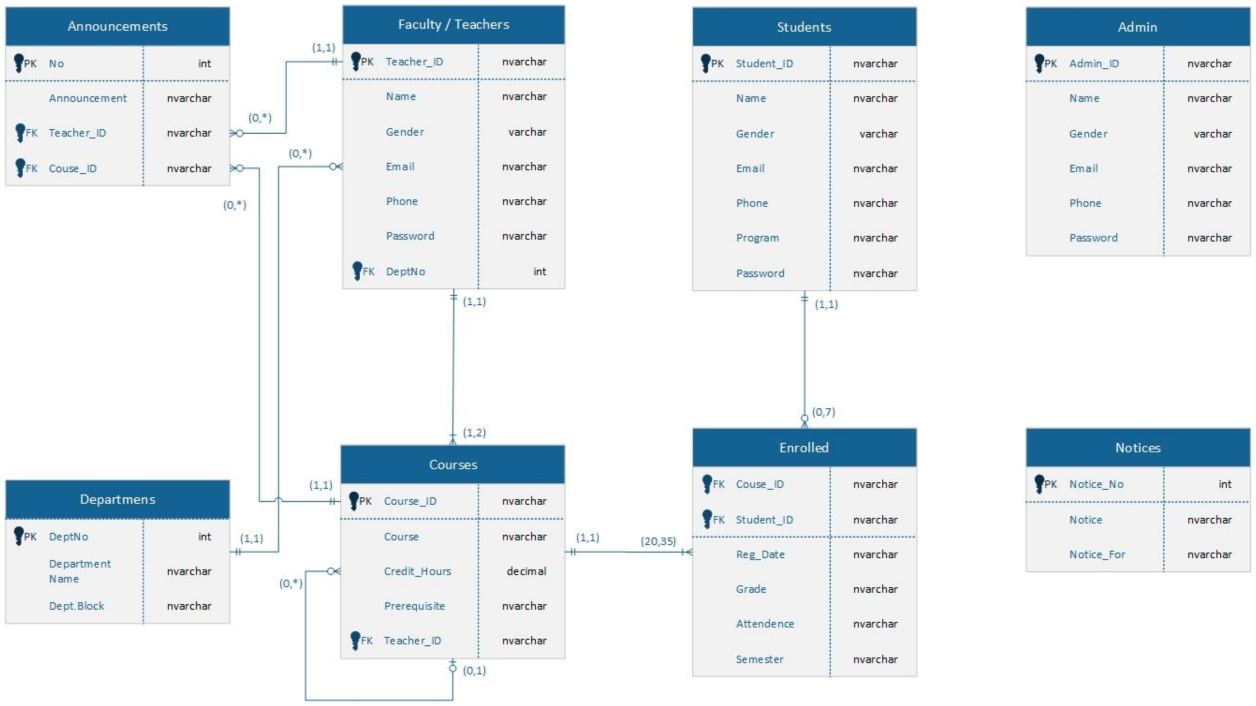
Course:

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.