Examination_System_DATA_Dictionary

Data Dictionary

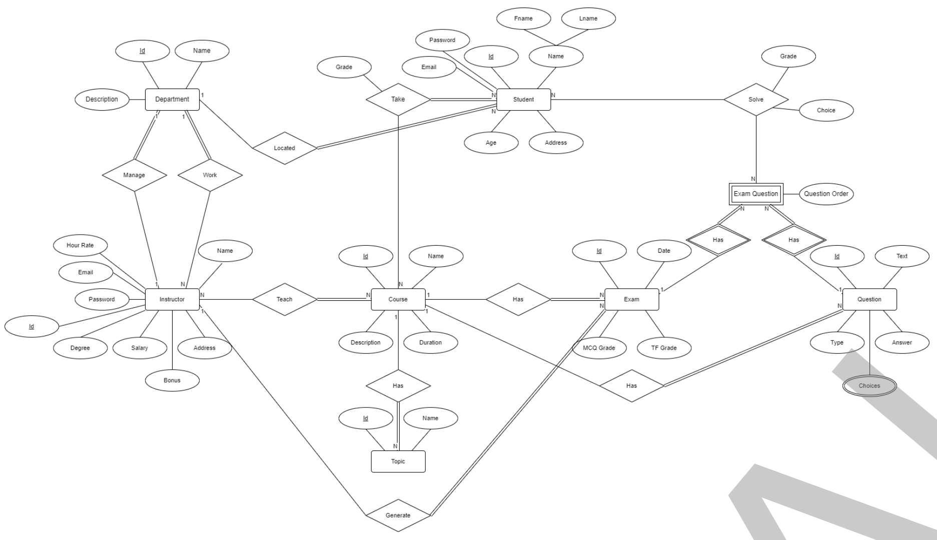
2025-02-01

Table of contents

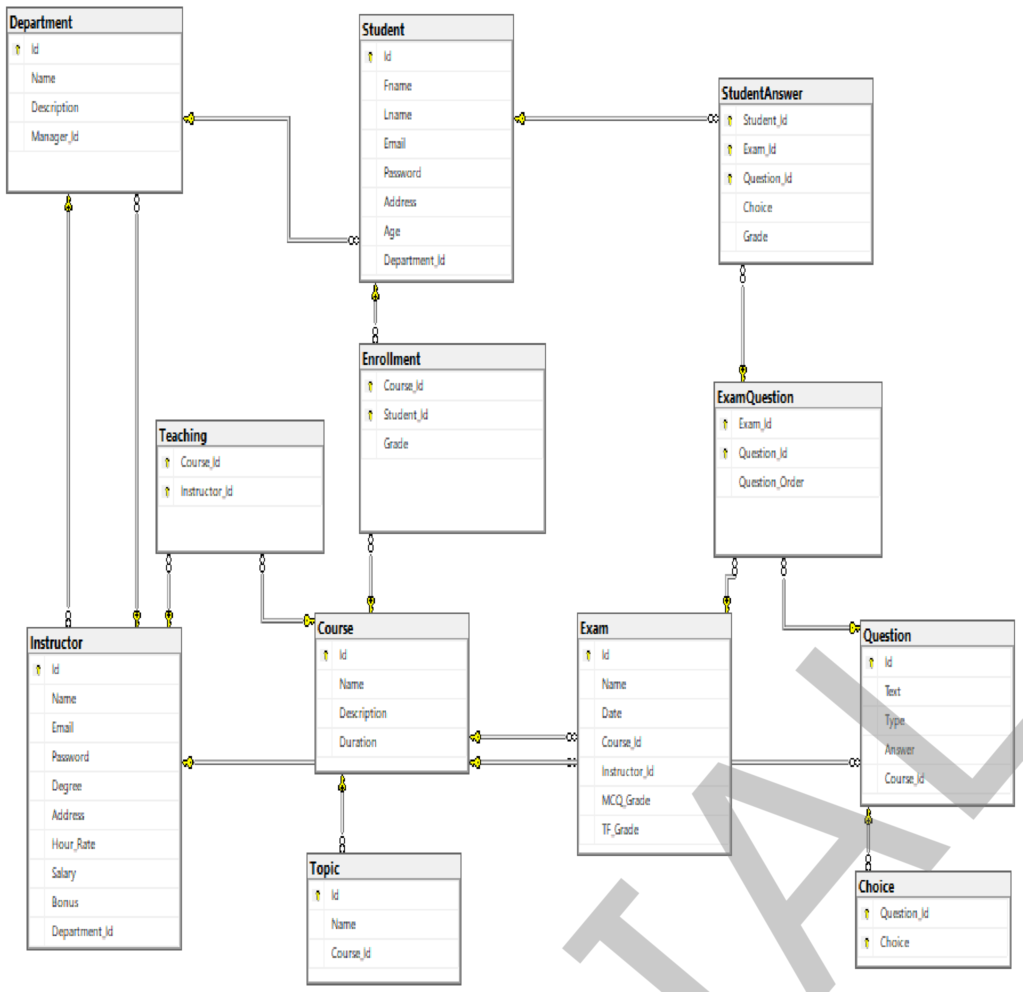
Legend

- Primary key
- Primary key disabled
- User-defined primary key
- Unique key
- Unique key disabled
- User-defined unique key
- Active trigger
- Disabled trigger
- Many to one relationship
- User-defined many to one relationship
- One to many relationship
- User-defined one to many relationship
- Many to many relationship
- User-defined many to many relationship
- One to one relationship
- User-defined one to one relationship
- Input
- Output
- Input/Output
- Uses dependency
- User-defined uses dependency
- Used by dependency
- User-defined used by dependency

ERD:



THE Mapping Schema


Department
Id
Name
Description
Manager_Id
Student
Id
Fname
Lname
Email
Password
Address
Age
Department_Id
StudentAnswer
Student_Id
Exam_Id
Question_Id
Choice
Grade
Enrollment
Course_Id
Student_Id
Grade
ExamQuestion
Exam_Id
Question_Id
Question_Order
Teaching
Course_Id
Instructor_Id
Instructor
Id
Name
Email
Password
Degree
Address
Hour_Rate
Salary
Bonus
Department_Id
Course
Id
Name
Description
Duration
Exam
Id
Name
Date
Course_Id
Instructor_Id
MCQ_Grade
TF_Grade
Question
Id
Text
Type
Answer
Course_Id
Topic
Id
Name
Course_Id
Choice
Question_Id
Choice

1. Tables

1.1. Table: Choice

Columns

		Name	Data type	Description / Attributes
		Question_Id	int	**References**: Question
		Choice	varchar(100)	

Links to

	Table	Join	Title / Name / Description
	Question	**Choice**Question_Id = QuestionId	FK_Choice_Question

Unique keys

	Columns	Name / Description
	Question_Id, Choice	PK_Choice

Uses

Name
Choice
Question

1.2. Table: Course

Columns

		Name	Data type	Description / Attributes
		Id	int	Identity / Auto increment
		Name	varchar(100)	
		Description	varchar(100)	Nullable
		Duration	int	Nullable

Linked from

	Table	Join	Title / Name / Description
	Enrollment	CourseId = EnrollmentCourse_Id	FK_Enrollment_Course
	Exam	CourseId = ExamCourse_Id	FK_Exam_Course
	Question	CourseId = QuestionCourse_Id	FK_Question_Course
	Teaching	CourseId = TeachingCourse_Id	FK_Teaching_Course
	Topic	CourseId = TopicCourse_Id	FK_Topic_Course

Unique keys

	Columns	Name / Description
	Id	PK_Course

Used By

Name
Course
Enrollment
Exam
Question
Teaching
Topic

1.3. Table: Department

Columns

		Name	Data type	Description / Attributes
	🔑	Id	int	**Identity / Auto increment**
		Name	varchar(50)	
		Description	varchar(100)	**Nullable**
		Manager_Id	int	**References**: Instructor

Links to

	Table	Join	Title / Name / Description
	Instructor	**Department**Manager_Id = InstructorId	FK_Department_Instructor

Linked from

	Table	Join	Title / Name / Description
	Instructor	**Department**Id = InstructorDepartment_Id	FK_Instructor_Department
	Student	**Department**Id = StudentDepartment_Id	FK_Student_Department

Unique keys

	Columns	Name / Description
🔑	Id	PK_Department

Uses

Name
Department
Instructor

Used By

Name
Department
Instructor
Student

1.4. Table: Enrollment

Columns

		Name	Data type	Description / Attributes
		Course_Id	int	**References**: Course
		Student_Id	int	**References**: Student
		Grade	decimal(18, 2)	**Nullable**

Links to

	Table	Join	Title / Name / Description
	Course	**Enrollment**Course_Id = CourseId	FK_Enrollment_Course
	Student	**Enrollment**Student_Id = StudentId	FK_Enrollment_Student

Unique keys

	Columns	Name / Description
	Course_Id, Student_Id	PK_Enrollment

Uses

Name
Enrollment
Course
Student

1.5. Table: Exam

Columns

		Name	Data type	Description / Attributes
		Id	int	**Identity / Auto increment**
		Name	varchar(100)	**Nullable**
		Date	date	
		Course_Id	int	**References**: Course
		Instructor_Id	int	**References**: Instructor
		MCQ_Grade	decimal(5, 2)	
		TF_Grade	decimal(5, 2)	

Links to

	Table	Join	Title / Name / Description
	Course	**Exam**Course_Id = CourseId	FK_Exam_Course
	Instructor	**Exam**Instructor_Id = InstructorId	FK_Exam_Instructor

Linked from

	Table	Join	Title / Name / Description
	ExamQuestion	**Exam**Id = ExamQuestionExam_Id	FK_ExamQuestion_Exam

Unique keys

	Columns	Name / Description
	Id	PK_Exam

Uses

Name
Exam
Course
Instructor

Used By

Name
Exam
ExamQuestion

1.6. Table: ExamQuestion

Columns

		Name	Data type	Description / Attributes
		Exam_Id	int	**References**: Exam
		Question_Id	int	**References**: Question
		Question_Order	int	

Links to

	Table	Join	Title / Name / Description
⤚	Exam	**ExamQuestion**Exam_Id = ExamId	FK_ExamQuestion_Exam
⤚	Question	**ExamQuestion**Question_Id = QuestionId	FK_ExamQuestion_Question

Linked from

	Table	Join	Title / Name / Description
⤙	StudentAnswer	**ExamQuestion**Exam_Id = StudentAnswerExam_Id, **ExamQuestion**Question_Id = StudentAnswerQuestion_Id	FK_StudentAnswer_ExamQuestion

Unique keys

	Columns	Name / Description
	Exam_Id, Question_Id	PK_ExamQuestion

Uses

Name
ExamQuestion
⤚ Exam
⤚ Question

Used By

Name
ExamQuestion
⤙ StudentAnswer

1.7. Table: Instructor

Columns

Name			Data type	Description / Attributes
▤	🔑	Id	int	Identity / Auto increment
▤		Name	varchar(100)	
▤		Email	varchar(100)	
▤		Password	varbinary(255)	
▤		Degree	varchar(100)	Nullable
▤		Address	varchar(100)	Nullable
▤		Hour_Rate	decimal(18, 0)	Nullable
▤		Salary	decimal(18, 0)	Nullable
▤		Bonus	decimal(18, 0)	Nullable
▤		Department_Id	int	Nullable References: Department

Links to

Table		Join	Title / Name / Description
≻	Department	InstructorDepartment_Id = DepartmentId	FK_Instructor_Department

Linked from

Table		Join	Title / Name / Description
≺	Department	InstructorId = DepartmentManager_Id	FK_Department_Instructor
≺	Exam	InstructorId = ExamInstructor_Id	FK_Exam_Instructor
≺	Teaching	InstructorId = TeachingInstructor_Id	FK_Teaching_Instructor

Unique keys

Columns		Name / Description
🔑	Id	PK_Instructor

Uses

Name
▦ Instructor
≻ Department

Used By

Name
▦ Instructor
≺ Department
≺ Exam
≺ Teaching

1.8. Table: Question

Columns

		Name	Data type	Description / Attributes
▤	🔑	Id	int	Identity / Auto increment
▤		Text	varchar(100)	
▤		Type	varchar(50)	
▤		Answer	varchar(100)	
▤		Course_Id	int	References: Course

Links to

	Table	Join	Title / Name / Description
≻	Course	QuestionCourse_Id = CourseId	FK_Question_Course

Linked from

	Table	Join	Title / Name / Description
≺	Choice	QuestionId = ChoiceQuestion_Id	FK_Choice_Question
≺	ExamQuestion	QuestionId = ExamQuestionQuestion_Id	FK_ExamQuestion_Question

Unique keys

	Columns	Name / Description
🔑	Id	PK_Question

Uses

Name
Question
≻ Course

Used By

Name
Question
≺ Choice
≺ ExamQuestion

1.9. Table: Student

Columns

		Name	Data type	Description / Attributes
		Id	int	Identity / Auto increment
		Fname	varchar(50)	
		Lname	varchar(50)	
		Email	varchar(100)	
		Password	varbinary(255)	
		Address	varchar(100)	Nullable
		Age	int	Nullable
		Department_Id	int	References: Department

Links to

	Table	Join	Title / Name / Description
	Department	StudentDepartment_Id = DepartmentId	FK_Student_Department

Linked from

	Table	Join	Title / Name / Description
	Enrollment	StudentId = EnrollmentStudent_Id	FK_Enrollment_Student
	StudentAnswer	StudentId = StudentAnswerStudent_Id	FK_StudentAnswer_Student

Unique keys

	Columns	Name / Description
	Id	PK_Student

Uses

Name
Student
Department

Used By

Name
Student
Enrollment
StudentAnswer

1.10. Table: StudentAnswer

Columns

		Name	Data type	Description / Attributes
		Student_Id	int	**References**: Student
		Exam_Id	int	**References**: ExamQuestion
		Question_Id	int	**References**: ExamQuestion
		Choice	varchar(100)	
		Grade	decimal(5, 2)	**Nullable**

Links to

	Table	Join	Title / Name / Description
	ExamQuestion	**StudentAnswer**Exam_Id = ExamQuestionExam_Id, **StudentAnswer**Question_Id = ExamQuestionQuestion_Id	FK_StudentAnswer_ExamQuestion
	Student	**StudentAnswer**Student_Id = StudentId	FK_StudentAnswer_Student

Unique keys

	Columns	Name / Description
	Student_Id, Exam_Id, Question_Id	PK_StudentAnswer

Uses

Name
StudentAnswer
ExamQuestion
Student

1.11. Table: Teaching

Columns

		Name	Data type	Description / Attributes
		Course_Id	int	**References**: Course
		Instructor_Id	int	**References**: Instructor

Links to

	Table	Join	Title / Name / Description
	Course	**Teaching**Course_Id = CourseId	FK_Teaching_Course
	Instructor	**Teaching**Instructor_Id = InstructorId	FK_Teaching_Instructor

Unique keys

	Columns	Name / Description
	Course_Id, Instructor_Id	PK_Teaching

Uses

Name
Teaching
Course
Instructor

1.12. Table: Topic

Columns

		Name	Data type	Description / Attributes
		Id	int	**Identity / Auto increment**
		Name	varchar(100)	
		Course_Id	int	**References**: Course

Links to

	Table	Join	Title / Name / Description
	Course	**Topic**Course_Id = CourseId	FK_Topic_Course

Unique keys

	Columns	Name / Description
	Id	PK_Topic

Uses

Name
Topic
Course

2. Procedures

2.1. Procedure: Delete_Course

Input/Output

	Name	Data type	Description
→@	Id	int	

Script

```
CREATE   PROCEDURE [dbo].[Delete_Course]
    @Id INT
AS
BEGIN
    SET NOCOUNT ON;

    DELETE FROM [dbo].[Course] WHERE [Id] = @Id;

    PRINT 'Course deleted successfully.';
END;
```

2.2. Procedure: Delete_Department

Input/Output

	Name	Data type	Description
→@	DID	int	

Script

```
--- Delete
create   proc Delete_Department
           @DID int
as
           Begin try
                      Delete from dbo.Department
                      where Id=@DID;
           end try
           begin catch
                      SELECT ERROR_MESSAGE() AS ErrorMessage;
           end catch
```

2.3. Procedure: Delete_Student

Input/Output

	Name	Data type	Description
→@	SID	int	

Script

```
create   proc Delete_Student
           @SID int
as
           Begin try
                      Delete from dbo.Student
                      where Id=@SID;
           end try
           begin catch
                      SELECT ERROR_MESSAGE() AS ErrorMessage;
           end catch
```

2.4. Procedure: Delete_StudentAnswer

Input/Output

	Name	Data type	Description
→@	SID	int	

Script

```
--- Delete
create   proc Delete_StudentAnswer
           @SID int
as
           Begin try
                      Delete from dbo.StudentAnswer
                      where Student_Id=@SID;
           end try
           begin catch
                      SELECT ERROR_MESSAGE() AS ErrorMessage;
           end catch
```

2.5. Procedure: Delete_Topic

Input/Output

	Name	Data type	Description
→@	Id	int	

Script

```
CREATE   PROCEDURE [dbo].[Delete_Topic]
    @Id INT
AS
BEGIN
    SET NOCOUNT ON;

    DELETE FROM [dbo].[Topic] WHERE [Id] = @Id;

    PRINT 'Topic deleted successfully.';
END;
```

2.6. Procedure: DeleteEnrollment

Input/Output

	Name	Data type	Description
→@	Course_Id	int	
→@	Student_Id	int	

Script

```
--DeleteEnrollment

CREATE   PROCEDURE DeleteEnrollment
    @Course_Id INT,
    @Student_Id INT
AS
BEGIN
    BEGIN TRY
                      begin transaction
                                 DELETE FROM Enrollment WHERE Course_Id = @Course_Id AND Student_Id = @Student_Id;
                      commit

    END TRY
    BEGIN CATCH
                      select 'UpdateEnrollment Proc ERROR => ' ,ERROR_LINE(), ERROR_MESSAGE();

        rollback
    END CATCH
END;
```

2.7. Procedure: DeleteExam

Input/Output

	Name	Data type	Description
→@	Id	int	

Script

```
--DeleteExam
CREATE   PROCEDURE DeleteExam
    @Id INT
AS
BEGIN
    BEGIN TRY
        DELETE FROM Exam WHERE Id = @Id;
    END TRY
    BEGIN CATCH
                      select 'DeleteExam ERROR => ' ,ERROR_LINE(), ERROR_MESSAGE();
    END CATCH
END;
```

2.8. Procedure: DeleteInstructor

Input/Output

	Name	Data type	Description
→@	Id	int	

Script

```
-- Delete Instructor
CREATE   PROCEDURE DeleteInstructor
    @Id INT
AS
BEGIN
   BEGIN TRY
              BEGIN TRANSACTION;
              DELETE FROM Instructor WHERE Id = @Id;
              COMMIT;
   END TRY
   BEGIN CATCH
                      SELECT 'DeleteInstructor PROC  ERROR =>', ERROR_LINE(),ERROR_MESSAGE()
                      ROLLBACK;
   END CATCH

END;
```

2.9. Procedure: Exam_Generation

Input/Output

	Name	Data type	Description
→@	Ins_ID	int	
→@	C_ID	int	
→@	EName	varchar(100)	
→@	MCQ_num	int	
→@	TF_num	int	
→@	MCQ_grade	int	
→@	TF_grade	int	

Script

```
create   proc Exam_Generation
           @Ins_ID int,
           @C_ID int,
           @EName varchar(100),
           @MCQ_num int,
           @TF_num int,
           @MCQ_grade int,
           @TF_grade int
as
           begin try
                      begin transaction
                                 --Check if instructor teaches said Course
                                 Declare @E_ID int
                                 if (Exists(Select * from Teaching where Instructor_Id=@Ins_ID and Course_Id=@C_ID))
                                 begin
                                            insert into dbo.Exam(Instructor_Id,Course_Id,Name,MCQ_Grade,TF_Grade,Date)
                                            values (@Ins_ID,@C_ID,@EName,@MCQ_grade,@TF_grade,GETDATE());
                                            set @E_ID=SCOPE_IDENTITY();
                                 end
                                 else
                                 begin
                                            THROW 50001, 'Instructor does not teach the specified course.', 1;
                                 end
                                 --Retrive the Course questions
                                 Declare @MCQ_Final_Table table( Q_ID int);
                                 Declare @TF_Final_Table table( Q_ID int);
                                 Declare @MCQ_Table table( Q_ID int );
                                 Insert into @MCQ_Table(Q_ID) Select Id from Question where Type='MCQ' and Course_Id=@C_ID
                                 Declare @MCQ_COUNT int
                                 set @MCQ_COUNT=(Select COUNT(Q_ID) From @MCQ_Table)
                                 if (@MCQ_COUNT >= @MCQ_num)
                                 begin
                                            insert into @MCQ_Final_Table select top(@MCQ_num) Q_ID from @MCQ_Table order by
NEWID();
                                 end
                                 else
                                 begin
                                            THROW 50002, 'Not enough MCQ questions.', 1;
                                 end

                                 Declare @TF_Table table( Q_ID int );
                                 Insert into @TF_Table Select Id from Question where Type='TF' and Course_Id=@C_ID
                                 Declare @TF_COUNT int
                                 set @TF_COUNT=(Select COUNT(Q_ID) From @TF_Table)
                                 if (@TF_COUNT >= @TF_num)
                                            begin
                                                       insert into @TF_Final_Table select top(@TF_num) Q_ID from @TF_Table
order by NEWID();
                                            end
                                 else
                                 begin
                                            THROW 50003, 'Not enough True or false questions.', 1;
                                 end

                                 Declare @RowNumberedData table(
                                            Exam_ID int,
                                            Question_ID int,
                                            Q_Order int)
                                 insert into @RowNumberedData SELECT
                                                       @E_ID,  -- First column (constant value)
                                                       t1.Q_ID,  -- Second column (merging two columns)
                                                       ROW_NUMBER() OVER (ORDER BY NEWID())  -- Third column (sequential
count starting from 1)
                                            FROM (Select Q_ID from @MCQ_Final_Table union all Select Q_ID from
@TF_Final_Table) t1;


                                 --insert into ExamQuestion
                                 insert into dbo.ExamQuestion(Exam_Id,Question_Id,Question_Order)
                                 select Exam_ID,Question_ID,Q_Order from @RowNumberedData
                      commit transaction
           end try
           begin catch
                      SELECT ERROR_MESSAGE() AS ErrorMessage;
                      rollback transaction
           end catch;
```

2.10. Procedure: ExamCorrection

Input/Output

Name		Data type	Description
→@	Exam_Id	int	
→@	Student_Id	int	

Script

```
CREATE   PROCEDURE ExamCorrection
    @Exam_Id INT,
    @Student_Id INT
AS
BEGIN
    DECLARE @Course_Id INT;
    DECLARE @MCQ_Grade DECIMAL(5,2);
    DECLARE @TF_Grade DECIMAL(5,2);
    DECLARE @Total_MCQ INT;
    DECLARE @Total_TF INT;
    DECLARE @Correct_MCQ INT;
    DECLARE @Correct_TF INT;
    DECLARE @Total_Exam_Marks DECIMAL(18,2);
    DECLARE @Student_Score DECIMAL(18,2);
    DECLARE @Percentage DECIMAL(18,2);

    BEGIN TRY
        BEGIN TRANSACTION;

        -- Get Exam Details
        SELECT @Course_Id = Course_Id,
               @MCQ_Grade = MCQ_Grade,
               @TF_Grade = TF_Grade
        FROM Exam
        WHERE Id = @Exam_Id;

        -- Validate if Exam exists
        IF @Course_Id IS NULL
        BEGIN
            select 'Exam not found!';
            ROLLBACK;
            RETURN;
        END

                      IF not exists(select * from StudentAnswer where Student_Id =@Student_Id)
        BEGIN
            select 'the stundent did not slove exam yet !';
            ROLLBACK;
            RETURN;
        END

        -- Get total number of MCQ and TF questions in the exam

                      SELECT  @Total_TF=COUNT(*) FROM ExamQuestion EQ
                      inner join  Question Q
                      on EQ .Question_Id = Q.Id and Type ='TF' and EQ.Exam_Id = @Exam_Id

                      SELECT @Total_MCQ = COUNT(*) FROM ExamQuestion EQ
                      inner join  Question Q
                      on EQ .Question_Id = Q.Id and Type ='MCQ'and EQ.Exam_Id = @Exam_Id



                      update  SA
                                            set Grade = case when
                                                                                                   Q.type ='MCQ' and Q.Answer
=SA.Choice then @MCQ_Grade
                                                                                        when
                                                                                                   Q.type ='TF' and Q.Answer
=SA.Choice then @TF_Grade
                                                                                        ELSE 0
                                                                             END
                                            from StudentAnswer SA
                                                        inner join ExamQuestion EQ
                                                        on  SA.Question_Id= EQ.Question_Id
                                                        inner join Question Q
                                                        on Q.Id =EQ.Question_Id
                                                        WHERE SA.Student_Id =@Student_Id AND SA.Exam_Id =@Exam_Id


        -- Get number of correct answers by the student

        SELECT @Correct_MCQ = COUNT(*)
```

```
                      from StudentAnswer SA
                                                        inner join ExamQuestion EQ
                                                        on  SA.Question_Id= EQ.Question_Id
                                                        inner join Question Q
                                                        on Q.Id =EQ.Question_Id
                                                        WHERE SA.Student_Id =@Student_Id
                                                                             AND SA.Exam_Id =@Exam_Id
                                                                             and Type = 'MCQ'
                                                                             and SA.Grade >0


                      SELECT @Correct_TF = COUNT(*)
                      from StudentAnswer SA
                                                        inner join ExamQuestion EQ
                                                        on  SA.Question_Id= EQ.Question_Id
                                                        inner join Question Q
                                                        on Q.Id =EQ.Question_Id
                                                        WHERE SA.Student_Id =@Student_Id
                                                                             AND SA.Exam_Id =@Exam_Id
                                                                             and Type = 'TF'
                                                                             and SA.Grade >0

        -- Calculate Total Possible Score
        SET @Total_Exam_Marks = (@Total_MCQ * @MCQ_Grade) + (@Total_TF * @TF_Grade);

        -- Calculate Student's Score
        SET @Student_Score = (@Correct_MCQ * @MCQ_Grade) + (@Correct_TF * @TF_Grade);

        -- Avoid division by zero
        IF @Total_Exam_Marks = 0
        BEGIN
            select 'Error: Exam has no questions!';
            ROLLBACK;
            RETURN;
        END

        -- Calculate Percentage
        SET @Percentage = (@Student_Score / @Total_Exam_Marks) * 100;

        -- Update Enrollment Table with the student's percentage
        UPDATE Enrollment
        SET Grade = @Percentage
        WHERE Course_Id = @Course_Id AND Student_Id = @Student_Id;

        -- Check if Enrollment record exists before updating
        IF @@ROWCOUNT = 0
        BEGIN
            select 'Student is not enrolled in the course!';
            ROLLBACK;
            RETURN;
        END

        COMMIT;
    END TRY
    BEGIN CATCH
        ROLLBACK;
        select 'ExamCorrection ERROR => ' + ERROR_MESSAGE();
    END CATCH
END;
```

2.11. Procedure: GetInstructor

Input/Output

	Name	Data type	Description
→@	Id	int	

Script

```
CREATE   PROCEDURE GetInstructor @Id int=null
AS
BEGIN
    Begin try
                  Begin transaction
                             if @Id is not null
                             SELECT * FROM Instructor
                             where Id =@Id
                             else
                             SELECT * FROM Instructor

                  commit;
           End Try
           Begin Catch

        SELECT 'GET Instructor PROC  ERROR =>', ERROR_LINE(),ERROR_MESSAGE()
                  Rollback;
           End Catch
END;
--exec UpdateInstructor 121,'mo khaled','mo@testUpdate','mo khaled','MODEGREE','cairo',null,null,2,2.5
```

2.12. Procedure: Insert_Course

Input/Output

	Name	Data type	Description
→@	Name	nvarchar(100)	
→@	Description	nvarchar(100)	
→@	Duration	int	

Script

```
CREATE   PROCEDURE [dbo].[Insert_Course]
    @Name NVARCHAR(100),
    @Description NVARCHAR(100) = NULL,
    @Duration INT = NULL
AS
BEGIN
    SET NOCOUNT ON;

    INSERT INTO [dbo].[Course] ([Name], [Description], [Duration])
    VALUES (@Name, @Description, @Duration);

    PRINT 'Course inserted successfully.';
END;
```

2.13. Procedure: Insert_Department

Input/Output

	Name	Data type	Description
→@	name	varchar(50)	
→@	Des	varchar(100)	
→@	MID	int	

Script

```
-- Department table
--- Insert
create   proc Insert_Department
           @name varchar(50),
           @Des varchar(100),
           @MID int
as
           Begin try
                      insert into dbo.Department(Name,Description,Manager_Id)
                      values (@name,@Des,@MID)
           end try
           begin catch
                      SELECT ERROR_MESSAGE() AS ErrorMessage;
           end catch
```

2.14. Procedure: Insert_Student

Input/Output

	Name	Data type	Description
→@	fname	varchar(50)	
→@	lname	varchar(50)	
→@	mail	varchar(100)	
→@	pass	varbinary(255)	
→@	address	varchar(100)	
→@	age	int	
→@	deparment_ID	int	

Script

```
create   proc Insert_Student
           @fname varchar(50),
           @lname varchar(50),
           @mail varchar(100),
           @pass varbinary(255),
           @address varchar(100),
           @age int,
           @deparment_ID int
as
           Begin try
                      insert into dbo.Student(Fname,Lname,Email,Password,Address,Age,Department_Id)
                      values (@fname,@lname,@mail,@pass,@address,@age,@deparment_ID)
           end try
           begin catch
                      SELECT ERROR_MESSAGE() AS ErrorMessage;
           end catch
```

2.15. Procedure: Insert_StudentAnswer

Input/Output

	Name	Data type	Description
→@	SID	int	
→@	EID	int	
→@	QID	int	
→@	Choice	varchar(100)	
→@	grade	decimal(5, 2)	

Script

```
-- StudnetAnswer table
--- Insert
create   proc Insert_StudentAnswer
           @SID int,
           @EID int,
           @QID int,
           @Choice varchar(100),
           @grade decimal(5,2)
as
           Begin try
                      insert into dbo.StudentAnswer(Student_Id,Exam_Id,Question_Id,Choice,Grade)
                      values (@SID,@EID,@QID,@Choice,@grade)
           end try
           begin catch
                      SELECT ERROR_MESSAGE() AS ErrorMessage;
           end catch
```

2.16. Procedure: Insert_Topic

Input/Output

	Name	Data type	Description
→@	Name	nvarchar(100)	
→@	Course_Id	int	

Script

```
CREATE   PROCEDURE [dbo].[Insert_Topic]
    @Name NVARCHAR(100),
    @Course_Id INT
AS
BEGIN
    SET NOCOUNT ON;

    -- Validate that the course exists
    IF EXISTS (SELECT 1 FROM [dbo].[Course] WHERE [Id] = @Course_Id)
    BEGIN
        INSERT INTO [dbo].[Topic] ([Name], [Course_Id])
        VALUES (@Name, @Course_Id);

        PRINT 'Topic inserted successfully.';
    END
    ELSE
    BEGIN
        PRINT 'Invalid Course_Id. The referenced course does not exist.';
    END
END;
```

2.17. Procedure: InsertEnrollment

Input/Output

	Name	Data type	Description
→@	Course_Id	int	
→@	Student_Id	int	
→@	Grade	decimal(18, 2)	

Script

```
CREATE   PROCEDURE InsertEnrollment
    @Course_Id INT,
    @Student_Id INT,
    @Grade DECIMAL(18,2) = NULL -- Grade is nullable
AS
BEGIN
    BEGIN TRY
                      BEGIN transaction
        INSERT INTO Enrollment (Course_Id, Student_Id, Grade)
        VALUES (@Course_Id, @Student_Id, @Grade);
                      commit
    END TRY
    BEGIN CATCH
                                 select 'InsertEnrollment Proc ERROR => ' ,ERROR_LINE(), ERROR_MESSAGE();
                                 Rollback
    END CATCH
END;
```

2.18. Procedure: InsertExam

Input/Output

Name		Data type	Description
→@	Name	varchar(100)	
→@	Date	date	
→@	Course_Id	int	
→@	Instructor_Id	int	
→@	MCQ_Grade	decimal(5, 2)	
→@	TF_Grade	decimal(5, 2)	

Script

```
CREATE   PROCEDURE InsertExam
  @Name VARCHAR(100),
    @Date DATE,
    @Course_Id INT,
    @Instructor_Id INT,
    @MCQ_Grade DECIMAL(5,2),
    @TF_Grade DECIMAL(5,2)
As
BEGIN
BEGIN TRY
              BEGIN TRANSACTION;

                         INSERT INTO Exam (Name, Date, Course_Id, Instructor_Id, MCQ_Grade, TF_Grade)
        VALUES (@Name, @Date, @Course_Id, @Instructor_Id, @MCQ_Grade, @TF_Grade);
                      COMMIT
END TRY
BEGIN CATCH
                      SELECT 'InsertExam PROC  ERROR =>',ERROR_LINE(),ERROR_MESSAGE()
                      ROLLBACK;
END CATCH

END;
```

2.19. Procedure: InsertInstructor

Input/Output

	Name	Data type	Description
→@	Name	varchar(255)	
→@	Email	varchar(255)	
→@	Password	varchar(255)	
→@	Degree	varchar(255)	
→@	Address	varchar(255)	
→@	HourRate	decimal(10, 2)	
→@	Salary	decimal(10, 2)	
→@	Department_Id	int	

Script

```
CREATE   PROCEDURE InsertInstructor
    @Name VARCHAR(255),
    @Email VARCHAR(255),
    @Password VARCHAR(255),
    @Degree VARCHAR(255),
    @Address VARCHAR(255),
    @HourRate DECIMAL(10,2),
    @Salary DECIMAL(10,2),
           @Department_Id int
As
BEGIN
BEGIN TRY
              BEGIN TRANSACTION;


              INSERT INTO Instructor (Name, Email, Password, Degree, Address,Hour_Rate, Salary,Department_Id)
                      VALUES (@Name, @Email,CONVERT(varbinary(255),@Password) , @Degree, @Address, @HourRate,
@Salary,@Department_Id);
                      COMMIT
END TRY
BEGIN CATCH
                      SELECT 'InsertInstructor PROC  ERROR =>',ERROR_LINE(),ERROR_MESSAGE()
                      ROLLBACK;
END CATCH

END;
```

2.20. Procedure: InsertStudentAnswer

Input/Output

	Name	Data type	Description
→@	Exam_id	int	
→@	Student_id	int	
→@	answers	nvarchar(255)	

Script

```sql
CREATE PROCEDURE InsertStudentAnswer
    @Exam_id INT,
    @Student_id INT,
    @answers NVARCHAR(255)
AS
BEGIN
    -- Start transaction
    BEGIN TRANSACTION;

    Declare @Question_id INT;
    Declare @QOrder INT = 1;
    Declare @choice NVARCHAR(255);

    -- Declare a cursor for the function result
    DECLARE choice_cursor CURSOR FOR
    SELECT Element FROM dbo.SplitString(@answers, ';');
           -- Open the cursor
OPEN choice_cursor;
FETCH NEXT FROM choice_cursor INTO @choice;

-- Loop through each choice and insert into StudentAnswer table
WHILE @@FETCH_STATUS = 0
BEGIN
    -- Insert into StudentAnswer table
    SET @Question_Id = (SELECT Question_Id
                       FROM Examination_System.dbo.ExamQuestion
                       WHERE Exam_Id = @Exam_Id AND Question_Order = @QOrder);

INSERT INTO Examination_System.dbo.StudentAnswer
    (Student_Id, Exam_Id, Question_Id, Choice)
VALUES
    (@Student_Id, @Exam_Id, @Question_Id, @choice);

-- Increment the question order
SET @QOrder = @QOrder + 1;

-- Fetch the next value
FETCH NEXT FROM choice_cursor INTO @choice;

END;

-- Close and deallocate the cursor
CLOSE choice_cursor;
DEALLOCATE choice_cursor;

-- Commit the transaction
COMMIT TRANSACTION;
END;
```

2.21. Procedure: Select_Course

Input/Output

	Name	Data type	Description
→@	Id	int	

Script

```
CREATE   PROCEDURE [dbo].[Select_Course]
    @Id INT = NULL -- Optional, if NULL selects all courses
AS
BEGIN
    SET NOCOUNT ON;

    IF @Id IS NOT NULL
    BEGIN
        SELECT * FROM [dbo].[Course] WHERE [Id] = @Id;
    END
    ELSE
    BEGIN
        SELECT * FROM [dbo].[Course];
    END
END;
```

2.22. Procedure: Select_Department

Input/Output

	Name	Data type	Description
→@	DID	int	

Script

```
--- Select
create   proc Select_Department
           @DID int
as
           Begin try
                      Select * from dbo.Department
                      where Id=@DID;
           end try
           begin catch
                      SELECT ERROR_MESSAGE() AS ErrorMessage;
           end catch

SET ANSI_NULLS ON
```

2.23. Procedure: Select_Student

Input/Output

	Name	Data type	Description
→@	SID	int	

Script

```
--- Select
create   proc Select_Student
           @SID int
as
           Begin try
                      Select * from dbo.Student
                      where Id=@SID;
           end try
           begin catch
                      SELECT ERROR_MESSAGE() AS ErrorMessage;
           end catch
```

2.24. Procedure: Select_StudentAnswer

Input/Output

	Name	Data type	Description
→@	SID	int	

Script

```
--- Select
create   proc Select_StudentAnswer
           @SID int
as
           Begin try
                      Select * from dbo.StudentAnswer
                      where Student_Id=@SID;
           end try
           begin catch
                      SELECT ERROR_MESSAGE() AS ErrorMessage;
           end catch
```

2.25. Procedure: Select_Topic

Input/Output

	Name	Data type	Description
→@	Course_Id	int	

Script

```
CREATE   PROCEDURE [dbo].[Select_Topic]
    @Course_Id INT = NULL -- Optional parameter
AS
BEGIN
    SET NOCOUNT ON;

    IF @Course_Id IS NOT NULL
    BEGIN
        SELECT * FROM [dbo].[Topic] WHERE [Course_Id] = @Course_Id;
    END
    ELSE
    BEGIN
        SELECT * FROM [dbo].[Topic];
    END
END;
```

2.26. Procedure: SelectEnrollment

Input/Output

	Name	Data type	Description
→@	Course_Id	int	
→@	Student_Id	int	

Script

```
--SelectEnrollment

CREATE   PROCEDURE SelectEnrollment
    @Course_Id INT = NULL,
    @Student_Id INT = NULL
AS
BEGIN
    IF @Course_Id IS NULL AND @Student_Id IS NULL
        SELECT * FROM Enrollment; -- Get all enrollments
    ELSE IF @Course_Id IS NOT NULL AND @Student_Id IS NULL
        SELECT * FROM Enrollment WHERE Course_Id = @Course_Id; -- Get enrollments for a course
    ELSE IF @Course_Id IS NULL AND @Student_Id IS NOT NULL
        SELECT * FROM Enrollment WHERE Student_Id = @Student_Id; -- Get enrollments for a student
    ELSE
        SELECT * FROM Enrollment WHERE Course_Id = @Course_Id AND Student_Id = @Student_Id; -- Specific record
END;
```

2.27. Procedure: SelectExam

Input/Output

	Name	Data type	Description
→@	Id	int	

Script

```
--SelectExaM
CREATE   PROCEDURE SelectExam
    @Id INT = NULL -- Optional parameter
AS
BEGIN
    IF @Id IS NULL
        SELECT * FROM Exam; -- Get all exams
    ELSE
        SELECT * FROM Exam WHERE Id = @Id; -- Get specific exam
END;
```

2.28. Procedure: SP_cChoice

Input/Output

	Name	Data type	Description
→@	questionId	int	
→@	choiceText	varchar(100)	

Script

```
CREATE   PROCEDURE SP_cChoice
           @questionId int,
           @choiceText varchar(100)
AS
BEGIN TRY
           INSERT INTO Choice VALUES (@questionId, @choiceText);
END TRY
BEGIN CATCH
           SELECT ERROR_MESSAGE() AS ErrorMessage;
END CATCH
```

2.29. Procedure: SP_cExamQuestion

Input/Output

	Name	Data type	Description
→@	examId	int	
→@	questionId	int	
→@	questionOrder	int	

Script

```sql
-- ============================================
-- Author:		GalalMohammed
-- Create date: 28-1-2025
-- Description:	Create an examQuestion record
-- ============================================
CREATE	PROCEDURE SP_cExamQuestion
	@examId int,
	@questionId int,
	@questionOrder int
AS
BEGIN TRY
	INSERT INTO ExamQuestion VALUES (@examId, @questionId, @questionOrder);
END TRY
BEGIN CATCH
	SELECT ERROR_MESSAGE() AS ErrorMessage;
END CATCH
```

2.30. Procedure: SP_cQuestion

Input/Output

	Name	Data type	Description
→@	questionText	varchar(100)	
→@	questionType	varchar(50)	
→@	questionAnswer	varchar(100)	
→@	courseId	int	

Script

```
CREATE   PROCEDURE SP_cQuestion
           @questionText varchar(100),
           @questionType varchar(50),
           @questionAnswer varchar(100),
           @courseId int
AS
BEGIN TRY
           IF @questionType='MCQ' OR @questionType='TF'
                      INSERT INTO Question(Text, Type, Answer, Course_Id) VALUES (@questionText, @questionType,
@questionAnswer, @courseId);
END TRY
BEGIN CATCH
           SELECT ERROR_MESSAGE() AS ErrorMessage;
END CATCH
```

2.31. Procedure: SP_dChoice

Input/Output

	Name	Data type	Description
→@	questionId	int	
→@	choiceText	varchar(100)	

Script

```
CREATE   PROCEDURE SP_dChoice
           @questionId int,
           @choiceText varchar(100)
AS
BEGIN TRY
           DELETE FROM Choice WHERE Question_Id=@questionId AND Choice=@choiceText;
END TRY
BEGIN CATCH
           SELECT ERROR_MESSAGE() AS ErrorMessage;
END CATCH
```

2.32. Procedure: SP_dExamQuestion

Input/Output

	Name	Data type	Description
→@	examId	int	
→@	questionId	int	

Script

```
-- ============================================
CREATE   PROCEDURE SP_dExamQuestion
           @examId int,
           @questionId int
AS
BEGIN TRY
           DELETE FROM ExamQuestion WHERE Exam_Id=@examId AND Question_Id=@questionId;
END        TRY
BEGIN CATCH
           SELECT ERROR_MESSAGE() AS ErrorMessage;
END CATCH
```

2.33. Procedure: SP_dQuestion

Input/Output

	Name	Data type	Description
→@	questionId	int	

Script

```
CREATE   PROCEDURE SP_dQuestion
           @questionId int
AS
BEGIN TRY
           DELETE FROM Question WHERE Id=@questionId;
END TRY
BEGIN CATCH
           SELECT ERROR_MESSAGE() AS ErrorMessage;
END CATCH
```

2.34. Procedure: SP_rChoices

Input/Output

	Name	Data type	Description
→@	questionId	int	

Script

```
CREATE   PROCEDURE SP_rChoices
           @questionId int
AS
BEGIN TRY
           -- SET NOCOUNT ON added to prevent extra result sets from
           -- interfering with SELECT statements.
           SET NOCOUNT ON;
           SELECT Choice FROM Choice WHERE Question_Id=@questionId;
END TRY
BEGIN CATCH
           SELECT ERROR_MESSAGE() AS ErrorMessage;
END CATCH
```

2.35. Procedure: SP_reportDepartmentStudents

Input/Output

	Name	Data type	Description
→@	departmentId	int	

Script

```
CREATE   PROCEDURE SP_reportDepartmentStudents
           @departmentId int
AS
BEGIN TRY
           -- SET NOCOUNT ON added to prevent extra result sets from
           -- interfering with SELECT statements.
           SET NOCOUNT ON;
  SELECT Id, Fname, Lname, Email, Address, Age FROM Student WHERE Department_Id=@departmentId;
END TRY
BEGIN CATCH
  SELECT ERROR_MESSAGE() AS ErrorMessage;
END CATCH
```

2.36. Procedure: SP_ReportingGetCourseTopics

Input/Output

	Name	Data type	Description
→@	Course_Id	int	

Script

```
CREATE   PROCEDURE SP_ReportingGetCourseTopics
    @Course_Id INT
AS
BEGIN
    SELECT T.Name AS TopicName
    FROM Topic T
    WHERE T.Course_Id = @Course_Id;
END;
```

2.37. Procedure: SP_ReportingGetExamQuestions

Input/Output

	Name	Data type	Description
→@	Exam_Id	int	

Script

```
CREATE   PROCEDURE SP_ReportingGetExamQuestions
    @Exam_Id INT
AS
BEGIN
    -- Select existing choices
    SELECT  QuestionID,
                                 QuestionText,
                                 ChoiceText
                                 FROM
                      (
                      SELECT
        Q.Id AS QuestionID,
        Q.Text AS QuestionText,
        C.Choice AS ChoiceText,EQ.Question_Order
                      FROM ExamQuestion EQ
                      INNER JOIN Question Q ON EQ.Question_Id = Q.Id
                      LEFT JOIN Choice C ON Q.Id = C.Question_Id
                      WHERE EQ.Exam_Id = @Exam_Id AND C.Choice IS NOT NULL

    UNION ALL

    -- Add "True" where there are no choices
    SELECT
        Q.Id AS QuestionID,
        Q.Text AS QuestionText,
        'True' AS ChoiceText,EQ.Question_Order
    FROM ExamQuestion EQ
    INNER JOIN Question Q ON EQ.Question_Id = Q.Id
    LEFT JOIN Choice C ON Q.Id = C.Question_Id
    WHERE EQ.Exam_Id = @Exam_Id AND C.Choice IS NULL

    UNION ALL

    -- Add "False" where there are no choices
    SELECT
        Q.Id AS QuestionID,
        Q.Text AS QuestionText,
        'False' AS ChoiceText,EQ.Question_Order
    FROM ExamQuestion EQ
    INNER JOIN Question Q ON EQ.Question_Id = Q.Id
    LEFT JOIN Choice C ON Q.Id = C.Question_Id
    WHERE EQ.Exam_Id = @Exam_Id AND C.Choice IS NULL
           ) AS TEMP

           ORDER BY Question_Order
END;
```

2.38. Procedure: SP_ReportingGetInstructorCourses

Input/Output

	Name	Data type	Description
→@	Instructor_Id	int	

Script

```
CREATE   PROCEDURE SP_ReportingGetInstructorCourses
    @Instructor_Id INT
AS
BEGIN

    SELECT C.Name,COUNT(E.Student_Id) AS StudentCount
           FROM Course C
           inner JOIN Enrollment E ON C.Id = E.Course_Id
           inner JOIN Teaching T ON T.Course_Id = E.Course_Id
           where T.Instructor_Id=@Instructor_Id
           group by c.Name
END;
```

2.39. Procedure: SP_ReportingGetStudentExamAnswers

Input/Output

	Name	Data type	Description
→@	Exam_Id	int	
→@	Student_Id	int	

Script

```
CREATE   PROCEDURE SP_ReportingGetStudentExamAnswers
    @Exam_Id INT,
    @Student_Id INT
AS
BEGIN
    SELECT Q.Id AS QuestionID, Q.Text AS QuestionText,
           SA.Choice AS StudentAnswer ,Q.Answer AS TheCorrectAnswer ,case when  SA.Choice =Q.Answer then



    FROM StudentAnswer SA
    INNER JOIN Question Q ON SA.Question_Id = Q.Id
    WHERE SA.Exam_Id = @Exam_Id AND SA.Student_Id = @Student_Id;
END;
```

2.40. Procedure: SP_ReportingGetStudentGrades

Input/Output

	Name	Data type	Description
→@	Student_Id	int	

Script

```
CREATE   PROCEDURE SP_ReportingGetStudentGrades
    @Student_Id INT
AS
BEGIN
    SELECT C.Name AS CourseName, E.Grade,
              convert(decimal(5,2),(E.Grade * 100) / 100) AS Percentage
    FROM Enrollment E
    INNER JOIN Course C ON E.Course_Id = C.Id
    WHERE E.Student_Id = @Student_Id;
END;
```

2.41. Procedure: SP_rExamQuestions

Input/Output

	Name	Data type	Description
→@	examId	int	

Script

```
-- ============================================
-- Author:          GalalMohammed
-- Create date: 28-1-2025
-- Description:     Retrieve exam questions
-- ============================================
CREATE   PROCEDURE SP_rExamQuestions
           @examId int
AS
BEGIN TRY
           -- SET NOCOUNT ON added to prevent extra result sets from
           -- interfering with SELECT statements.
           SET NOCOUNT ON;
           SELECT Question_Id, Question_Order FROM ExamQuestion WHERE Exam_Id=@examId ORDER BY Question_Order;
END        TRY
BEGIN CATCH
           SELECT ERROR_MESSAGE() AS ErrorMessage;
END CATCH
```

2.42. Procedure: SP_rQuestions

Input/Output

	Name	Data type	Description
→@	courseId	int	
→@	questionType	varchar(50)	

Script

```
CREATE   PROCEDURE SP_rQuestions
           @courseId int,
           @questionType varchar(50)
AS
BEGIN TRY
           -- SET NOCOUNT ON added to prevent extra result sets from
           -- interfering with SELECT statements.
           SET NOCOUNT ON;
           SELECT * FROM Question WHERE Course_Id=@courseId AND Type=@questionType;
END        TRY
BEGIN CATCH
           SELECT ERROR_MESSAGE() AS ErrorMessage;
END CATCH
```

2.43. Procedure: SP_uChoice

Input/Output

	Name	Data type	Description
→@	questionId	int	
→@	oldChoiceText	varchar(100)	
→@	newChoiceText	varchar(100)	

Script

```
CREATE   PROCEDURE SP_uChoice
           @questionId int,
           @oldChoiceText varchar(100),
           @newChoiceText varchar(100)
AS
BEGIN TRY
           UPDATE Choice SET Choice=@newChoiceText WHERE Question_Id=@questionId AND Choice=@oldChoiceText;
END TRY
BEGIN CATCH
           SELECT ERROR_MESSAGE() AS ErrorMessage;
END CATCH
```

2.44. Procedure: SP_uExamQuestion

Input/Output

	Name	Data type	Description
→@	examID	int	
→@	questionId	int	
→@	questionOrder	int	

Script

```
-- =============================================
-- Author:            GalalMohammed
-- Create date: 28-1-2025
-- Description:       Update an exam question
-- =============================================
CREATE   PROCEDURE SP_uExamQuestion
           @examID int,
           @questionId int,
           @questionOrder int
AS
BEGIN TRY
           UPDATE ExamQuestion SET Question_Id=@questionId, Question_Order=@questionOrder WHERE Exam_Id=@examID AND
Question_Id=@questionId;
END        TRY
BEGIN CATCH
           SELECT ERROR_MESSAGE() AS ErrorMessage;
END CATCH
```

2.45. Procedure: SP_uQuestion

Input/Output

	Name	Data type	Description
→@	questionId	int	
→@	questionText	varchar(100)	
→@	questionType	varchar(50)	
→@	questionAnswer	varchar(100)	
→@	courseId	int	

Script

```
-- =============================================
-- Author:          GalalMohammed
-- Create date: 28-1-2025
-- Description:     Update a question record
-- =============================================
CREATE   PROCEDURE SP_uQuestion
           @questionId int,
           @questionText varchar(100) = '',
           @questionType varchar(50),
           @questionAnswer varchar(100),
           @courseId int
AS
BEGIN TRY
           IF @questionType='MCQ' OR @questionType='TF'
                      UPDATE Question SET Text=@questionText, Type=@questionType, Answer=@questionAnswer, Course_Id=@courseId
WHERE Id=@questionId;
END TRY
BEGIN CATCH
           SELECT ERROR_MESSAGE() AS ErrorMessage;
END CATCH
```

2.46. Procedure: Update_Course

Input/Output

	Name	Data type	Description
→@	Id	int	
→@	Name	nvarchar(100)	
→@	Description	nvarchar(100)	
→@	Duration	int	

Script

```
CREATE   PROCEDURE [dbo].[Update_Course]
    @Id INT,
    @Name NVARCHAR(100),
    @Description NVARCHAR(100) = NULL,
    @Duration INT = NULL
AS
BEGIN
    SET NOCOUNT ON;

    UPDATE [dbo].[Course]
    SET [Name] = @Name,
        [Description] = @Description,
        [Duration] = @Duration
    WHERE [Id] = @Id;

    PRINT 'Course updated successfully.';
END;
```

2.47. Procedure: Update_Department

Input/Output

	Name	Data type	Description
→@	DID	int	
→@	name	varchar(50)	
→@	Des	varchar(100)	
→@	MID	int	

Script

```
--- Update
create   proc Update_Department
           @DID int,
           @name varchar(50),
           @Des varchar(100),
           @MID int
as
           Begin try
                      update dbo.Department
                      set Name=@name,
                                 Description=@Des,
                                 Manager_Id=@MID
                      where Id=@DID;
           end try
           begin catch
                      SELECT ERROR_MESSAGE() AS ErrorMessage;
           end catch
```

2.48. Procedure: Update_Student

Input/Output

	Name	Data type	Description
→@	SID	int	
→@	fname	varchar(50)	
→@	lname	varchar(50)	
→@	mail	varchar(100)	
→@	pass	varbinary(255)	
→@	address	varchar(100)	
→@	age	int	
→@	deparment_ID	int	

Script

```
--- Update
create   proc Update_Student
           @SID int,
           @fname varchar(50),
           @lname varchar(50),
           @mail varchar(100),
           @pass varbinary(255),
           @address varchar(100),
           @age int,
           @deparment_ID int
as
           Begin try
                      update dbo.Student
                      set Fname=@fname,
                                 Lname=@lname,
                                 Email=@mail,
                                 Password=@pass,
                                 Address=@address,
                                 Age=@age,
                                 Department_Id=@deparment_ID
                      where Id=@SID;
           end try
           begin catch
                      SELECT ERROR_MESSAGE() AS ErrorMessage;
           end catch

--- Delete
```

2.49. Procedure: Update_StudentAnswer

Input/Output

	Name	Data type	Description
→@	SID	int	
→@	EID	int	
→@	QID	int	
→@	Choice	varchar(100)	
→@	grade	decimal(5, 2)	

Script

```
--- Update
create   proc Update_StudentAnswer
           @SID int,
           @EID int,
           @QID int,
           @Choice varchar(100),
           @grade decimal(5,2)
as
           Begin try
                      update dbo.StudentAnswer
                      set Exam_Id=@EID,
                                 Question_Id=@QID,
                                 Choice=@Choice,
                                 Grade=@grade
                      where Student_Id=@SID;
           end try
           begin catch
                      SELECT ERROR_MESSAGE() AS ErrorMessage;
           end catch
```

2.50. Procedure: Update_Topic

Input/Output

	Name	Data type	Description
→@	Id	int	
→@	Name	nvarchar(100)	
→@	Course_Id	int	

Script

```
CREATE    PROCEDURE [dbo].[Update_Topic]
    @Id INT,
    @Name NVARCHAR(100),
    @Course_Id INT
AS
BEGIN
    SET NOCOUNT ON;

    -- Validate that the course exists
    IF EXISTS (SELECT 1 FROM [dbo].[Course] WHERE [Id] = @Course_Id)
    BEGIN
        UPDATE [dbo].[Topic]
        SET [Name] = @Name, [Course_Id] = @Course_Id
        WHERE [Id] = @Id;

        PRINT 'Topic updated successfully.';
    END
    ELSE
    BEGIN
        PRINT 'Invalid Course_Id. The referenced course does not exist.';
    END
END;
```

2.51. Procedure: UpdateEnrollment

Input/Output

	Name	Data type	Description
→@	Course_Id	int	
→@	Student_Id	int	
→@	Grade	decimal(18, 2)	

Script

```
CREATE   PROCEDURE UpdateEnrollment
    @Course_Id INT,
    @Student_Id INT,
    @Grade DECIMAL(18,2) = NULL
AS
BEGIN
    BEGIN TRY
        BEGIN TRANSACTION;

        UPDATE Enrollment
        SET Grade = @Grade
        WHERE Course_Id = @Course_Id AND Student_Id = @Student_Id;

        COMMIT;
    END TRY
    BEGIN CATCH
                      select 'UpdateEnrollment Proc ERROR => ' ,ERROR_LINE(), ERROR_MESSAGE();
        ROLLBACK;
    END CATCH
END;
```

2.52. Procedure: UpdateExam

Input/Output

	Name	Data type	Description
→@	Id	int	
→@	Name	varchar(100)	
→@	Date	date	
→@	Course_Id	int	
→@	Instructor_Id	int	
→@	MCQ_Grade	decimal(5, 2)	
→@	TF_Grade	decimal(5, 2)	

Script

```
--UpdateExam

CREATE   PROCEDURE UpdateExam
    @Id INT,
    @Name VARCHAR(100),
    @Date DATE,
    @Course_Id INT,
    @Instructor_Id INT,
    @MCQ_Grade DECIMAL(5,2),
    @TF_Grade DECIMAL(5,2)
AS
BEGIN
    BEGIN TRY
        BEGIN TRANSACTION;

        UPDATE Exam
        SET Name = @Name,
            Date = @Date,
            Course_Id = @Course_Id,
            Instructor_Id = @Instructor_Id,
            MCQ_Grade = @MCQ_Grade,
            TF_Grade = @TF_Grade
        WHERE Id = @Id;

        COMMIT;
    END TRY
    BEGIN CATCH
        select 'UpdateExam ERROR => ' ,ERROR_LINE(), ERROR_MESSAGE();
        ROLLBACK;
    END CATCH
END;
```

2.53. Procedure: UpdateInstructor

Input/Output

	Name	Data type	Description
→@	Id	int	
→@	Name	varchar(255)	
→@	Email	varchar(255)	
→@	Password	varchar(255)	
→@	Degree	varchar(255)	
→@	Address	varchar(255)	
→@	HourRate	decimal(10, 2)	
→@	Salary	decimal(10, 2)	
→@	Department_Id	int	
→@	Bonus	decimal(10, 2)	

Script

```
-- Update Instructor
CREATE   PROCEDURE UpdateInstructor
    @Id INT,
    @Name VARCHAR(255),
    @Email VARCHAR(255),
    @Password Varchar(255),
    @Degree VARCHAR(255),
    @Address VARCHAR(255),
    @HourRate DECIMAL(10,2),
    @Salary DECIMAL(10,2),
           @Department_Id int,
           @Bonus DECIMAL(10,2)
AS
BEGIN

           BEGIN TRY
                       BEGIN TRANSACTION;

    UPDATE Instructor
                                 SET Name = @Name, Email = @Email, Password = CONVERT(varbinary(255),@Password),
                                            Degree = @Degree, Address = @Address,
                                            Hour_Rate = @HourRate, Salary = @Salary,
                                            Department_Id =@Department_Id,
                                            Bonus =@Bonus
                                 WHERE Id = @Id;
                       COMMIT;
           END TRY
           BEGIN CATCH
        SELECT 'UpdateInstructor PROC  ERROR =>', ERROR_LINE(),ERROR_MESSAGE()
                      ROLLBACK;
           END CATCH


END;
```

3. Functions

3.1. Function: SplitString

Input/Output

	Name	Data type	Description
	Returns	table type	
	InputString	nvarchar(MAX)	
	Delimiter	char(1)	

Script

```
CREATE FUNCTION dbo.SplitString(@InputString NVARCHAR(MAX), @Delimiter CHAR(1))
RETURNS @Result TABLE (Element NVARCHAR(255))
AS
BEGIN
    DECLARE @pos INT = 0;
    DECLARE @nextPos INT;
    DECLARE @element NVARCHAR(255);

    WHILE CHARINDEX(@Delimiter, @InputString, @pos + 1) > 0
    BEGIN
        -- Find the next delimiter position
        SET @nextPos = CHARINDEX(@Delimiter, @InputString, @pos + 1);

        -- Extract the value and trim spaces
        SET @element = LTRIM(RTRIM(SUBSTRING(@InputString, @pos + 1, @nextPos - @pos - 1)));

        -- Insert only if it's not empty
        IF @element <> ''
            INSERT INTO @Result (Element) VALUES (@element);

        -- Move to the next position
        SET @pos = @nextPos;
    END;

    -- Handle the last value after the last `;`
    SET @element = LTRIM(RTRIM(SUBSTRING(@InputString, @pos + 1, LEN(@InputString) - @pos)));

    -- Insert only if it's not empty
    IF @element <> ''
        INSERT INTO @Result (Element) VALUES (@element);

    RETURN;
END;
```