Information Technology Institute

Full stack.net Track

Database project: Examination system

Supervisor: Eng/Eman Gomaa

Presented by:

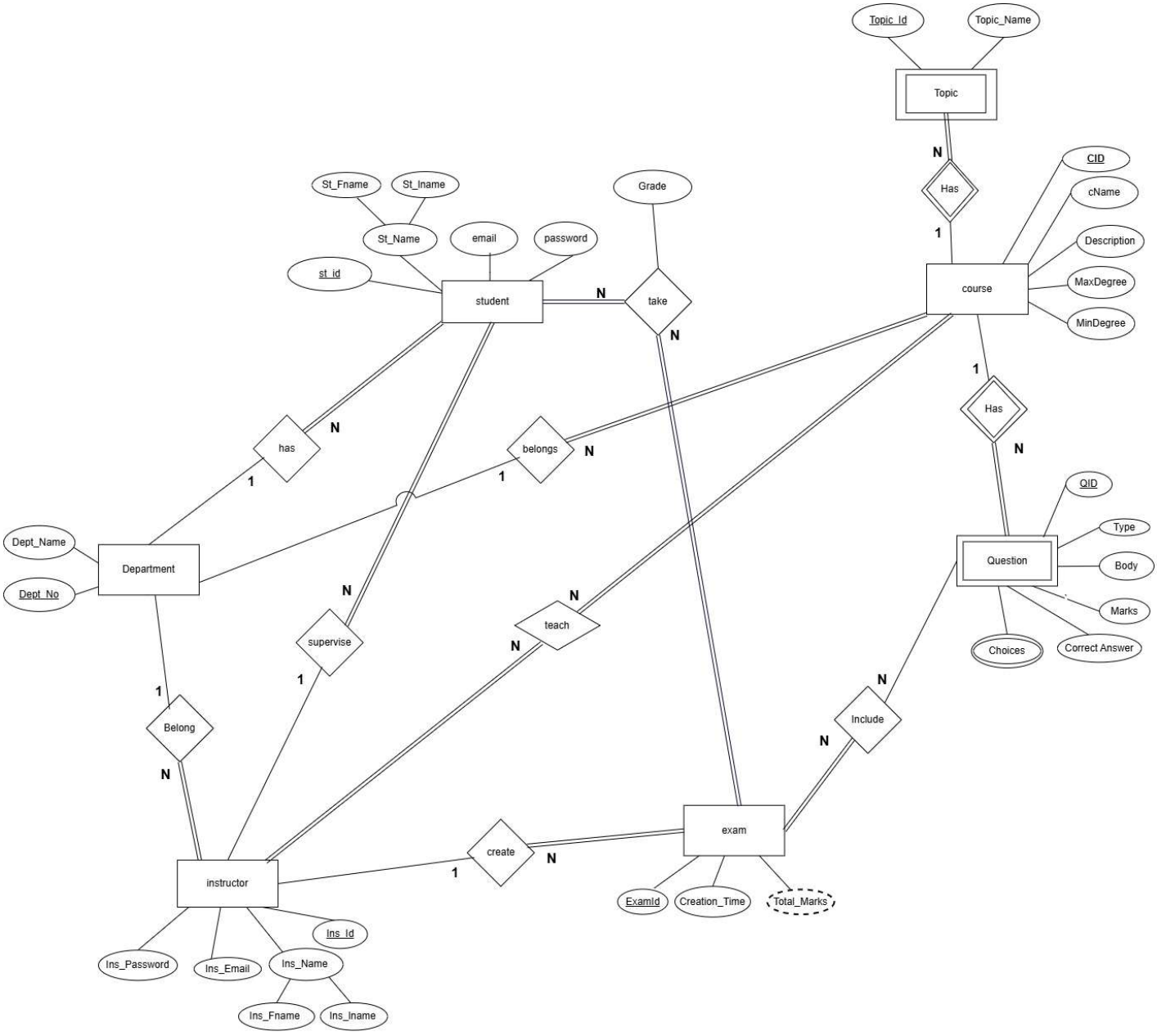
1. **Eman Ramadan Mostafa**
2. **Ahmed Ibrhim Alqaaspy**
3. **Mohamed Ahmed Ali**
4. **Nour Elhoda Abdallah**
5. **Rania Mahmoud Khalifa**

Contents

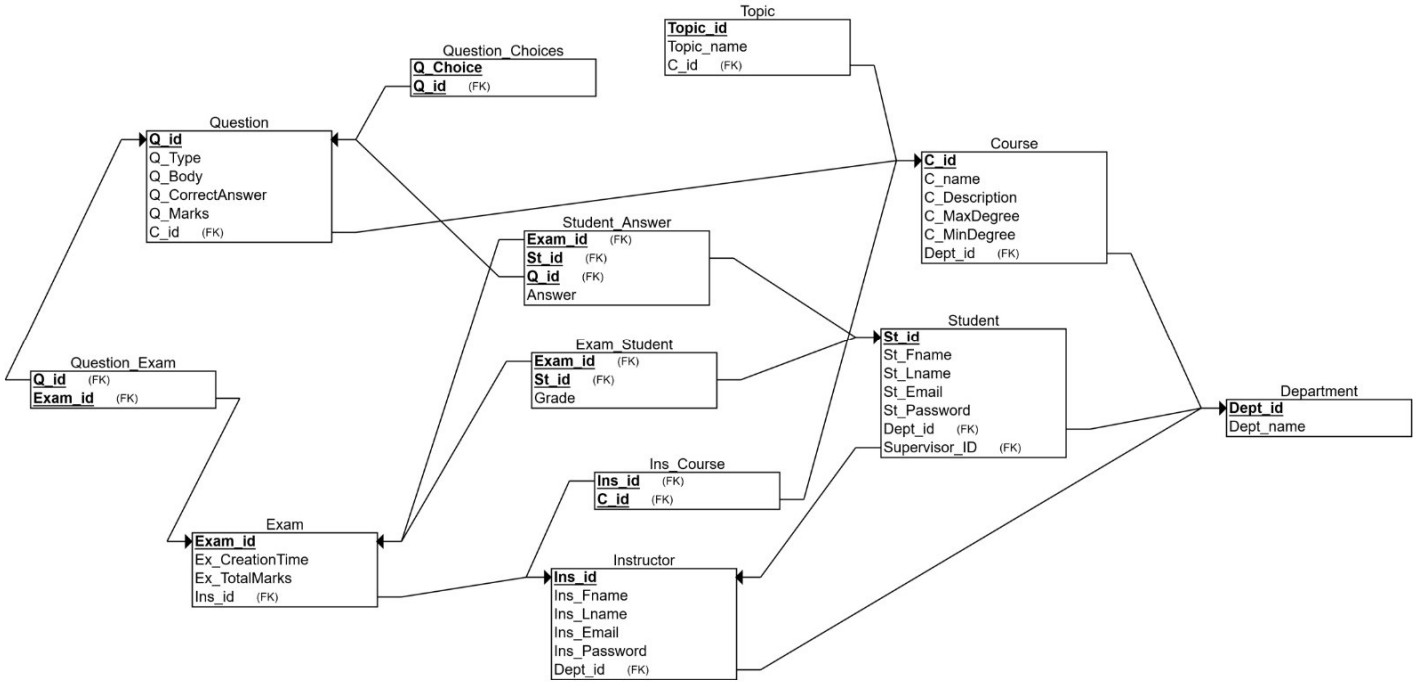
System requirements:

- The system should provide a question pool for each course.
- Instructors should be able to pick questions from the pool to create exams.
- Questions can be of the following types: Multiple Choice (MCQ) or True/False.
- For **Multiple Choice** and **True/False** questions: The system should store the correct answer, it should automatically check the student's answer and store the result.
- System should store courses information (Course name, description, Max degree, Min Degree), instructors' information, and students' information, each instructor can teach one or more course,and each course may be teacher by one instructor in each class
- Admin Manages departments, courses, and instructors. also monitors system logs and audits.
- Admin Prevent unauthorized inserts, updates, or deletes using triggers. Log all changes to critical tables for auditing.
- Instructor creates and manages exams, Grades student exams, and views course and student reports.
- Student takes exams., views grades and exam history, and submits exam answers.
- Admin can Add/update/delete departments, courses, and instructors. View system logs and audit trails.
- Instructor Generate exams with True/False and MCQ questions. Correct exams and assign grades.
- Student Submit exam answers. View grades...

Database design ERD



Database Design Mapping



Introduction

This document provides a detailed overview of the database schema designed for managing Examination system. The schema includes tables for departments, students, instructors, courses, exams, and questions, along with their relationships and constraints. It ensures data integrity through primary keys, foreign keys, and various constraints.

1-Tabels

1.1 Department Table

```
Create table Department (
    Dept_no int primary key,
    Dept_name varchar(100) not null
);
```

- **Primary Key:** Dept_no
- **Relationships:**
  - **Student:** A student belongs to a department (Dept_no in Student references Dept_no in Department).
  - **Instructor:** An instructor belongs to a department (Dept_id in Instructor references Dept_no in Department).
  - **Course:** A course belongs to a department (Dep_ID in Course references Dept_no in Department).

1.2 Student Table

```
Create table Student (
    stud_ID int primary key identity(1,1),
    fname varchar(50) not null,
    lname varchar(50) not null,
    email varchar(100) not null unique,
    password varchar(100) not null,
    Dept_no int,
    Supervisor_ID int,
    constraint c20 foreign key (Dept_no) references   Department(Dept_no) on update
    cascade,
    constraint c21 CHECK (email LIKE '%_@__%.__%'),
    constraint c22 foreign key (Supervisor_ID) references     Instructor(Ins_id)
       );
```

- **Primary Key:** stud_ID
- **Relationships:**
  - **Department:** A student belongs to a department (Dept_no in Student references Dept_no in Department).
  - **Instructor:** A student may have a supervisor (Supervisor_ID in Student references Ins_id in Instructor).
  - **ExamStudent:** A student can take multiple exams (stud_ID in ExamStudent references stud_ID in Student).
  - **StudentAnswer:** A student can answer multiple questions in exams (stud_ID in StudentAnswer references stud_ID in Student).

1.3 Course Table

```
create table Course(
	CID int primary key,
	Cname varchar(100) not null unique,
	CDescription varchar(500),
	CminDegree int default 50 not null ,
	CmaxDegree int default 100 not null ,
	Dep_ID int,
	constraint c30 foreign key(Dep_ID) references department(Dept_no)
on update cascade,
	constraint c31 check ( CminDegree < CmaxDegree ) ,
	constraint c32 check ( CminDegree >= 0 and CmaxDegree >= 0)
);
```

- **Primary Key:** CID
- **Relationships:**
  - **Department:** A course belongs to a department (Dep_ID in Course references Dept_no in Department).
  - **Topic:** A course can have multiple topics (CID in Topic references CID in Course).
  - **Ins_Course:** A course can be taught by multiple instructors (C_id in Ins_Course references CID in Course).
  - **QuestionCourse:** A course can have multiple questions (CID in QuestionCourse references CID in Course).

1.4 Topic Table

```
create table Topic(
	CID int,
	TID int identity(1,1),
	Tname varchar(200) not null,
	primary key (TID,Cid),
	constraint c40 foreign key(Cid) references Course(CID) on delete cascade on
update cascade
);
```

- **Primary Key:** TID (composite key with CID)
- **Relationships:**
  - **Course:** A topic belongs to a course (CID in Topic references CID in Course).

1.5 Instructor Table

```
Create Table Instructor(
	Ins_id int primary key identity(1,1),
	fname varchar(50) not null,
	lname varchar(50) not null,
	email varchar(200) unique not null,
	password varchar(100) not null,
	Dept_id int,
	constraint c50 foreign key(Dept_id) references Department(Dept_no) on update cascade,
	constraint c51 CHECK (email LIKE '%_@__%.__%'));
```

- **Primary Key:** Ins_id
- **Relationships:**
  - **Department:** An instructor belongs to a department (Dept_id in Instructor references Dept_no in Department).
  - **Student:** An instructor can supervise multiple students (Ins_id in Instructor is referenced by Supervisor_ID in Student).
  - **Ins_Course:** An instructor can teach multiple courses (Ins_id in Ins_Course references Ins_id in Instructor).
  - **Exam:** An instructor can create multiple exams (Ins_ID in Exam references Ins_id in Instructor).

1.6 Ins_Course Table

```
CREATE TABLE Ins_Course(
        Ins_id int,
        C_id int,
        PRIMARY KEY (Ins_id, C_id),
        constraint c60 foreign key (Ins_Id) references Instructor(Ins_id) on update no action on
        delete no action,
        constraint c61 foreign key (C_id) references Course(CID) on update no action on delete no
        action
        );
```

- **Primary Key:** Ins_id, C_id
- **Relationships:**
    - **Instructor:** An instructor can teach multiple courses (Ins_id in Ins_Course references Ins_id in Instructor).
    - **Course:** A course can be taught by multiple instructors (C_id in Ins_Course references CID in Course).

1.7 Question Table

```
Create table Question(
        QID int primary key identity(1,1),
        Body varchar(500) not null unique,
        Type int not null,
        CorrectAnswer varchar(500) not null,
        Marks int not null default 0,
        constraint c70 CHECK (Marks >= 0),
        constraint c71 check (Type IN (1, 2))
        );
```

- **Primary Key:** QID
- **Relationships:**
    - **QuestionChoices:** A question can have multiple choices (QID in QuestionChoices references QID in Question).
    - **QuestionCourse:** A question can belong to multiple courses (QID in QuestionCourse references QID in Question).
    - **Question_Exam:** A question can be part of multiple exams (Q_id in Question_Exam references QID in Question).
    - **StudentAnswer:** A question can be answered by multiple students (Q_ID in StudentAnswer references QID in Question).

1.8 QuestionChoices Table

```
Create table QuestionChoices(
        QID int,
        Choice varchar(500),
        primary key (QID,Choice),
        constraint c80 foreign key (QID) references Question(QID) on delete cascade on update
        cascade,);
```

- **Primary Key:** QID, Choice
- **Relationships:**
    - **Question:** A question can have multiple choices (QID in QuestionChoices references QID in Question).

1.9 QuestionCourse Table

```
create table QuestionCourse(
	CID int,
	QID int,
	primary key (Cid,Qid),
	constraint c90 Foreign key(CID) references Course(CID) on delete cascade on update
	cascade,
	constraint c91 Foreign key(QID) references Question(QID) on delete cascade on
	update cascade
	);
```

- **Primary Key:** CID, QID
- **Relationships:**
  - **Course:** A course can have multiple questions (CID in QuestionCourse references CID in Course).
  - **Question:** A question can belong to multiple courses (QID in QuestionCourse references QID in Question).

1.10 Exam Table

```
Create table Exam(
	ID int primary key identity(1,1),
	CreationTime datetime not null,
	TotalMarks int,
	Ins_ID int,
	constraint c100 foreign key (Ins_ID) references Instructor(Ins_ID),
	constraint c101 check (TotalMarks >= 0),
	);
```

- **Primary Key:** ID
- **Relationships:**
  - **Instructor:** An exam is created by an instructor (Ins_ID in Exam references Ins_id in Instructor).
  - **Question_Exam:** An exam can have multiple questions (Exam_id in Question_Exam references ID in Exam).
  - **ExamStudent:** An exam can be taken by multiple students (ExamID in ExamStudent references ID in Exam).

1.11 Question_Exam table

```
create table Question_Exam(
	Q_id int,
	Exam_id int,
	primary key (Q_id,Exam_id),
	constraint c110 foreign key(Q_id) references Question(QID) on delete cascade on
	update cascade,
	constraint c111 foreign key(Exam_id) references Exam(ID) on delete cascade on
	update cascade;
```

- **Primary Key:** Q_id, Exam_id
- **Relationships:**
  - **Question:** A question can be part of multiple exams (Q_id in Question_Exam references QID in Question).
  - **Exam:** An exam can have multiple questions (Exam_id in Question_Exam references ID in Exam).

1.12 ExamStudent tale

```
CREATE TABLE ExamStudent(
    grade int default 0 check (grade >= 0),
    stud_ID int,
    ExamID int,
    primary key (stud_ID, CID, ExamID),
    constraint c120 foreign key (stud_ID) references Student(stud_ID),
    constraint c122 foreign key (ExamID) references Exam(ID)
       );
```

- **Primary Key:** stud_ID, CID, ExamID
- **Relationships:**
  - **Student:** A student can take multiple exams (stud_ID in ExamStudent references stud_ID in Student).
  - **Exam:** An exam can be taken by multiple students (ExamID in ExamStudent references ID in Exam).

1.13 StudentAnswer table

```
CREATE TABLE StudentAnswer (
    Stud_ID int,
    Exam_ID int,
    Q_ID int,
    Answer varchar(500),
    primary key (Stud_ID, Exam_ID, Q_ID),
       constraint sa_foreign_stud foreign key (Stud_ID) references  Student(stud_ID) on delete
cascade on update cascade,
    constraint sa_foreign_exam foreign key (Exam_ID) references Exam(ID) on delete cascade on
update cascade,
    constraint sa_foreign_question foreign key(Q_ID) references Question(QID) on delete cascade
on update cascade,
       );
```

- **Primary Key:** Stud_ID, Exam_ID, Q_ID
- **Relationships:**
  - **Student:** A student can answer multiple questions (Stud_ID in StudentAnswer references stud_ID in Student).
  - **Exam:** An exam can have multiple answers (Exam_ID in StudentAnswer references ID in Exam).
  - **Question:** A question can be answered by multiple students (Q_ID in StudentAnswer references QID in Question).

Summary of Relationships:

- **Department** is the central table that links to **Student**, **Instructor**, and **Course**.
- **Student** is linked to **Department**, **Instructor** (as a supervisor), and **Exam** (through **ExamStudent** and **StudentAnswer**).
- **Instructor** is linked to **Department**, **Student** (as a supervisor), **Course** (through **Ins_Course**), and **Exam**.
- **Course** is linked to **Department**, **Topic**, **Instructor** (through **Ins_Course**), and **Question** (through **QuestionCourse**).
- **Question** is linked to **Course** (through **QuestionCourse**), **Exam** (through **Question_Exam**), and **Student** (through **StudentAnswer**).
- **Exam** is linked to **Instructor**, **Question** (through **Question_Exam**), and **Student** (through **ExamStudent** and **StudentAnswer**).

2- Main Implementation

2.1 Get_Questions_By_Type

```
create procedure Get_Questions_By_Type
    @courseName varchar(100),
    @QuesNum int,
    @type int
AS
begin
    select TOP (@QuesNum) QID, Body, Type
    from Question Q
    join Course C
       on Q.CID = C.CID
    where C.Cname = @courseName AND Q.Type = @type
    order by NEWID();
       end;
```

- **Purpose:** Retrieves a specified number of questions of a given type for a specific course.
- **Parameters:**
    - @courseName: Name of the course.
    - @QuesNum: Number of questions to retrieve.
    - @type: Type of questions to retrieve (1 for True/False, 2 for MCQ).

2.2 Cal_TotalMarks

```
alter procedure Cal_TotalMarks
    @examid int,
       @totalMarks int output
AS
begin
    begin Transaction;
    begin try
        select @totalMarks = SUM(Q.Marks)
        from dbo.Q_QE_join Q
        where Q.Exam_ID = @examid;

        update Exam
        set TotalMarks = @totalMarks
        where ID = @examid;

        commit Transaction;
    end try
    begin catch
             select 'Something error during calculate Total Marks of Current Exam';
        rollback Transaction;
    end catch;
end;
```

- **Purpose:** Calculates the total marks for an exam and updates the Exam table.
- **Parameters:**
    - @examid: ID of the exam.
    - @totalMarks: Output parameter to return the total marks.

2.3 ExamGeneration

```
alter procedure ExamGeneration
    @courseName varchar(100),
    @InsId int,
    @QnumT int,
    @QnumM int,
       @ExamId int OUTPUT

with encryption
AS
begin
    begin Transaction;
    begin try
        insert into Exam (Ins_ID, CreationTime)
              values (@InsId, GETDATE());

        set @ExamId = SCOPE_IDENTITY();

        create table #TempQuestions (
              QID int,
              QBody varchar(500),
              Type int);

        insert into #TempQuestions
        exec Get_Questions_By_Type @courseName, @QnumT, 1;

        insert into #TempQuestions
        exec Get_Questions_By_Type @courseName, @QnumM, 2;

        insert into Question_Exam (Q_id, Exam_ID)
        select QID, @ExamId
              from #TempQuestions;

        create table #TF_Choices (c varchar(10));

        insert into #TF_Choices (c)
              values ('True'), ('False');

        select TQ.QID, TQ.QBody AS QuestionBody,
            CASE
                WHEN TQ.Type = 1 THEN TC.c
                ELSE QC.Choice
            end AS Choice
        from #TempQuestions TQ
        LEFT join QuestionChoices QC
              on TQ.QID = QC.QID
        LEFT join #TF_Choices TC
              on TQ.Type = 1
        order by TQ.QID;

              declare @ExamTotalMarks int;
              exec Cal_TotalMarks @ExamId, @ExamTotalMarks OUTPUT;
              select 'Total Marks for Exam: ' + CAST(@ExamTotalMarks AS varchar);

        drop table #TempQuestions;
        drop table #TF_Choices;

        commit Transaction;
    end try
    begin catch
              select 'Something error at During Generating an Exam';
        rollback Transaction;
    end catch;
end;
```

- **Purpose:** The ExamGeneration stored procedure is designed to generate an exam by selecting questions from a database based on a given course and instructor. It inserts the generated exam into the `Exam` table and retrieves the questions, storing them temporarily before inserting them into the Question_Exam table. Additionally, it calculates the total marks for the exam and presents the final question paper structure.
- **Parameters:**
    - @courseName: Name of the course.
    - @InsId: ID of the instructor creating the exam.
    - @QnumT: Number of True/False questions.
    - @QnumM: Number of MCQ questions.
    - @ExamId: Output parameter to return the generated exam ID.

2.4 ExamAnswers

```
alter procedure ExamAnswers
    @stuId int,
    @examid int,
    @Answer varchar(MAX)

with encryption
AS
begin
    begin Transaction;
    begin try

        declare @totalQ int;
        select @totalQ = COUNT(QID)
        from Q_QE_join
        where Exam_ID = @examid;

        declare @Questiontable table (
              QNum int IDENTITY(1, 1),
              QID int);

        insert into @Questiontable (QID)
        select QID
              from Q_QE_join
              where Exam_ID = @examid;

        declare @ind int = 1, @curQ int, @curAns varchar(500);

        while @ind <= @totalQ
        begin

            select @curQ = QID
                    from @Questiontable
                    where QNum = @ind;

            select @curAns = newtable.ans
            from (
                select Value AS ans, ROW_NUMBER() OVER (order by (select NULL)) AS Num
                from STRING_SPLIT(@Answer, ',')
            ) AS newtable
            where Num = @ind;

            if EXISTS (
                select 1
                from StudentAnswer
                where Stud_ID = @stuId AND Exam_ID = @examid AND Q_ID = @curQ
            )
            begin
                PRint 'Answer for question ' + CAST(@curQ AS varchar) + ' already exists.';
            end
            ELSE
            begin
                insert into StudentAnswer (Stud_ID, Exam_ID, Q_ID, Answer)
                values (@stuId, @examid, @curQ, @curAns);
            end;
            set @ind = @ind + 1;
        end;
        commit Transaction;
    end try
    begin catch
        select 'Something error at Storing Student Answers';
        rollback Transaction;
    end catch;
end;
```

- **Purpose:** Stores the answers provided by a student for an exam.
- **Parameters:**
    - @stuId: ID of the student.
    - @examid: ID of the exam.
    - @Answer: Comma-separated list of answers.

2.5 ExamCorrection

```
alter procedure ExamCorrection
    @examid int,
    @stuId int,
       @finalGrade Money OUTPUT

with encryption
AS
begin
    begin Transaction;
    begin try

        create table #Temp_Stud_Ans(
              QID int,
              Ans varchar(500)
              );

        insert into #Temp_Stud_Ans
        select Q_ID, Answer
        from StudentAnswer
        where Exam_ID = @examid AND Stud_ID = @stuId;

        declare @curQ int, @curAns varchar(500), @mark int, @stud_grade int = 0;
        declare @studAns varchar(500);
        declare c1 CURSOR
              for
                     select QID, CorrectAnswer, Marks from Question
              for read only;
        open c1;
        fetch c1 into @curQ, @curAns, @mark;
        while @@fetch_STATUS = 0
        begin
            select @studAns = Ans
                     from #Temp_Stud_Ans
                     where QID = @curQ;

            if @studAns = @curAns
                set @stud_grade = @stud_grade + @mark;

            fetch c1 into @curQ, @curAns, @mark;
        end;
        close c1;
        deallocate c1;

        declare @totalM int;
              exec Cal_TotalMarks @examid, @totalM OUTPUT;
        set @finalGrade = (@stud_grade * 1.0 / @totalM) * 100;

           if not exists(
                     select 1 from ExamStudent
                     where Stud_ID = @stuId AND ExamID = @examid
              )
                     begin
                            insert into ExamStudent (Grade, Stud_ID, ExamID)
                            values (@finalGrade, @stuId, @examid);
                     end
              ELSE
                     begin
                            select 'Student Already has a grade for this exam!';
                     end
        commit Transaction;
    end try
    begin catch
              select 'Something error during Correcting Exam';
        rollback Transaction;
    end catch;
end;
```

- **Purpose:** Corrects an exam and calculates the final grade for a student.
- **Parameters:**
    - @examid: ID of the exam.
    - @stuId: ID of the student.
    - @finalGrade: Output parameter to return the final grade.

2.5 Q_QE_Join view

```
create view Q_QE_Join
as
	select *
	FROM Question Q
	JOIN Question_Exam QE
	ON Q.QID = QE.Q_id
```

- **Purpose:** Joins the Question and Question_Exam tables to provide a combined view of questions and their associated exams.
- **Columns:**
    - All columns from Question and Question_Exam.

3-Validation

Validation section, including the triggers, audit tables, and non-clustered indexes. This section ensures data integrity, logs changes, and prevents unauthorized modifications to critical tables.

3.1Global Tables

3.1.1 Error_Log

- **Purpose:** Logs errors that occur during the execution of triggers or procedures.
- **Columns:**
  - ErrorMessage: Description of the error.
  - ErrorTime: Timestamp of the error.
  - UserName: User who encountered the error.

3.2 Exam Validation

3.2.1 Audit Table

1. Exam_Audit
   - **Purpose:** Logs actions performed on the Exam table.
   - **Columns:**
     - AuditID: Unique identifier for the audit record.
     - Action: Type of action (Insert Blocked, Update, Delete).
     - _user: User who performed the action.
     - ActionTime: Timestamp of the action.
     - OldValue: Old value of the ID column (for updates and deletes).
     - NewValue: New value of the ID column (for updates).

3.2.2 Triggers

1. **Exam_Insert**
   - **Purpose:** Prevents direct inserts into the Exam table and logs the attempt.
   - **Behavior:**
     - Blocks inserts and logs the action in Exam_Audit.
     - Logs errors in Error_Log if any occur.
   - **Usage:** `INSERT INTO Exam (Ins_ID, CreationTime) VALUES (1, GETDATE());`
2. **Exam_Update**

- **Purpose:** Logs updates to the Exam table.
- **Behavior:**
  - Logs the old and new values of the ID column in Exam_Audit.
  - Logs errors in Error_Log if any occur.

3. **Exam_Delete**

- Purpose: Logs deletions from the Exam table.
- Behavior:
  - Logs the deleted ID in Exam_Audit.
  - Logs errors in Error_Log if any occur.

3.2.3 Non-Clustered Index

1. **Ins_exam_index**
    - **Purpose:** Improves query performance for searches on the Ins_ID column in the Exam table.
    - **Usage:**
```
create nonclustered index Ins_exam_index
on Exam (ins_ID)
```

3.3 Student Exam Validation

3.3.1 Audit Table

1. **Student_Exam_Audit**
    - **Purpose:** Logs actions performed on the ExamStudent table.
    - **Columns:**
        - AuditID: Unique identifier for the audit record.
        - Action: Type of action (Insert, Update, Delete).
        - _user: User who performed the action.
        - ActionTime: Timestamp of the action.
        - OldValue: Old value of the column being updated or deleted.
        - NewValue: New value of the column being updated.

3.3.2 Triggers

1. **Student_Exam_Insert**
    - **Purpose:** Prevents direct inserts into the ExamStudent table and logs the attempt.
    - **Behavior:**
        - Blocks inserts and logs the action in Student_Exam_Audit.
        - Logs errors in Error_Log if any occur.
2. **Student_Exam_Update**
    - **Purpose:** Logs updates to the ExamStudent table.
    - **Behavior:**
        - Logs the old and new values of the Stud_ID, ExamID, and Grade columns in Student_Exam_Audit.
        - Logs errors in Error_Log if any occur.
3. **Student_Exam_Delete**
    - **Purpose:** Logs deletions from the ExamStudent table.
    - **Behavior:**
        - Logs the deleted ExamID in Student_Exam_Audit.
        - Logs errors in Error_Log if any occur.

3.3.3 Non-Clustered Index

1. **Exam_Student_grade**
    - **Purpose:** Improves query performance for searches on the Grade column in the ExamStudent table.
    - **Usage:**
```
create nonclustered index Exam_Student_grade
on ExamStudent(grade)
```

3.4 Question Exam Validation

3.4.1 Audit Tables

1. **Question_Exam_Audit_insert**
   - **Purpose:** Logs insert attempts on the Question_Exam table.
   - **Columns:**
     - _user: User who attempted the insert.
     - Q_id: Question ID.
     - Exam_id: Exam ID.
     - _date: Timestamp of the attempt.
2. **Question_Exam_Audit_update**
   - **Purpose:** Logs updates to the Question_Exam table.
   - **Columns:**
     - _user: User who performed the update.
     - _date: Timestamp of the update.
     - OldValue: Old value of the Exam_id column.
     - NewValue: New value of the Exam_id column.

3.4.2 Triggers

1. **Question_Exam_Insert**
   - **Purpose:** Prevents direct inserts into the Question_Exam table and logs the attempt.
   - **Behavior:**
     - Blocks inserts and logs the action in Question_Exam_Audit_insert.
     - Logs errors in Error_Log if any occur.
   - **Usage:** `insert into Question_Exam values(2,2)`
2. **Question_Exam_Update**
   - **Purpose:** Logs updates to the Question_Exam table.
   - **Behavior:**
     - Logs the old and new values of the Exam_id column in Question_Exam_Audit_update.
     - Logs errors in Error_Log if any occur.
   - **Usage:**
     ```
     UPDATE Question_Exam SET Exam_id = 2 WHERE Q_id = 12 AND Exam_id = 21;
     ```
3. **Question_Exam_Delete**
   - **Purpose:** Logs deletions from the Question_Exam table.
   - **Behavior:**
     - Logs the deleted Exam_id in Question_Exam_Audit_update.
     - Logs errors in Error_Log if any occur.
   - **Usage:** `delete from Question_Exam where Q_id=12`

3.5 Student Answer Validation

3.5.1 Audit Table

1. **Student_Answer_Audit**
    - **Purpose:** Logs actions performed on the StudentAnswer table.
    - **Columns:**
        - AuditID: Unique identifier for the audit record.
        - Action: Type of action (Insert, Update, Delete).
        - _user: User who performed the action.
        - ActionTime: Timestamp of the action.
        - OldValue: Old value of the Exam_id column.
        - NewValue: New value of the Exam_id column.

3.5.2 Triggers

1. **Student_Answer_Insert**
    - **Purpose:** Prevents direct inserts into the StudentAnswer table and logs the attempt.
    - **Behavior:**
        - Blocks inserts and logs the action in Student_Answer_Audit.
        - Logs errors in Error_Log if any occur.
2. **Student_Answer_Update**
    - **Purpose:** Logs updates to the StudentAnswer table.
    - **Behavior:**
        - Logs the old and new values of the Exam_id column in Student_Answer_Audit.
        - Logs errors in Error_Log if any occur.
3. **Student_Answer_Delete**
    - **Purpose:** Logs deletions from the StudentAnswer table.
    - **Behavior:**
        - Logs the deleted Exam_id in Student_Answer_Audit.
        - Logs errors in Error_Log if any occur.

3.6 Summary of Validation

- **Audit Tables:**
    - Track changes to critical tables (Exam, ExamStudent, Question_Exam, StudentAnswer).
- **Triggers:**
    - Prevent unauthorized inserts, updates, or deletes.
    - Log actions for auditing and debugging.
- **Non-Clustered Indexes:**
    - Improve query performance for frequently searched columns.

4-Schemas

4.1 Admin Schema

Contains administrative procedures and views for managing the database.

Tables

1. **Login_Audit**
    - **Purpose:** Logs login attempts by instructors.
    - **Columns:**
        - Email: Email address used for login.
        - password: Password used for login.
        - Status: Status of the login attempt (Success or Failed).
        - _user: User who attempted to log in.
        - Time: Timestamp of the login attempt.

-Procedure to control select , insert , update and delete a student

1. sp_Student_Select procedure

- **Purpose:** Retrieves student details.
- **Parameters:**
    - @stud_id (optional): Student ID. If not provided, retrieves all students.

2. sp_Student_Insert procedure

- **Purpose:** Inserts a new student record.
- **Parameters:**
    - @fname: First name.
    - @lname: Last name.
    - @email: Email address.
    - @pass: Password.
    - @dno: Department number.
    - @superid: Supervisor ID.

3. sp_Student_Update procedure

- **Purpose:** Updates an existing student record.
- **Parameters:**
    - @stud_id: Student ID.
    - @fname: First name.
    - @lname: Last name.
    - @email: Email address.
    - @pass: Password.
    - @dno: Department number.
    - @superid: Supervisor ID.

4. sp_Student_Delete procedure

- **Purpose:** Deletes a student record.
- **Parameters:**
    - @stud_id: Student ID.

-Procedure to control select , insert , update and delete a department

1. sp_Department_Select procedure

- **Purpose:** Retrieves department details.
- **Parameters:**
  - @dno (optional): Department number. If not provided, retrieves all departments.

2. sp_Department_Insert procedure

- **Purpose:** Inserts a new department record.
- **Parameters:**
  - @dno: Department number.
  - @dname: Department name.

3. sp_Department_Update procedure

- **Purpose:** Updates an existing department record.
- **Parameters:**
  - @dno: Department number.
  - @dname: Department name.

4. sp_Department_Delete procedure

- **Purpose:** Deletes a department record.
- **Parameters:**
  - @dno: Department number.

-Procedure to control select , insert , update and delete a topic

1. sp_CourseTopic_Select procedure

- **Purpose:** Retrieves topics for a course.
- **Parameters:**
  - @cid (optional): Course ID. If not provided, retrieves all topics.

2. sp_CourseTopic_Insert procedure

- **Purpose:** Inserts a new topic for a course.
- **Parameters:**
  - @cid: Course ID.
  - @tname: Topic name.

3. sp_CourseTopic_Update procedure

- **Purpose:** Updates an existing topic.
- **Parameters:**
  - @tid: Topic ID.
  - @tname: Topic name.

4. sp_CourseTopic_Delete procedure

- **Purpose:** Deletes a topic.
- **Parameters:**
  - @tid: Topic ID.

-Procedure to control select , insert , update and delete a question

1. sp_Question_Select procedure

- **Purpose:** Retrieves question details.
- **Parameters:**
  - @QID (optional): Question ID. If not provided, retrieves all questions.

2. sp_Question_Insert procedure

- **Purpose:** Inserts a new question.
- **Parameters:**
  - @Body: Question text.
  - @Type: Question type (1 for True/False, 2 for MCQ).
  - @CorrectAnswer: Correct answer.
  - @Marks: Marks allocated.

3. sp_Question_Update procedure

- **Purpose:** Updates an existing question.
- **Parameters:**
  - @QID: Question ID.
  - @Body: Question text.
  - @Type: Question type.
  - @CorrectAnswer: Correct answer.
  - @Marks: Marks allocated.

4. sp_Question_Delete procedure

- **Purpose:** Deletes a question.
- **Parameters:**
  - @QID: Question ID.

-Procedure to control select , insert , update and delete an Exam

1. sp_Exam_Select procedure

- **Purpose:** Retrieves Exam details.
- **Parameters**: @ID: Exam ID.

2. **2. sp_Exam_Insert procedure**
   - **Purpose:** Inserts a new exam.
   - **Parameters:**
     - @creationtime : creation time.
     - @totalmarks: Total exam marks.
     - @ins_id: Exam instructor id

3. sp_Exam_Update procedure

- **Purpose:** Updates an existing exam.
- **Parameters:**
  - @id: Exam Id.
  - @creationtime: creation time.
  - @totalmarks: Total exam marks.
  - @ins_id: Exam instructor id

4. sp_Exam_Delete procedure

- **Purpose:** Deletes an existing exam.
- **Parameters:**
  - @ID: Exam ID.

-**Procedure to control select , insert , update and delete a course**

1. sp_course_select procedure

- **Purpose:** Retrieves course details.
- **Parameters:**
  - @cid (optional): Course ID. If not provided, retrieves all courses.

2. sp_course_insert procedure

- **Purpose:** Inserts a new course.
- **Parameters:**
  - @cname: Course name.
  - @cdescription: Course description.
  - @cmindegree: Minimum degree.
  - @cmaxdegree: Maximum degree.
  - @depid: Department ID.

3. sp_course_update procedure

- **Purpose:** Updates an existing course.
- **Parameters:**
  - @cid: Course ID.
  - @cname: Course name.
  - @cdescription: Course description.
  - @cmindegree: Minimum degree.
  - @cmaxdegree: Maximum degree.
  - @depid: Department ID.

4. sp_course_delete procedure

- **Purpose:** Deletes a course.
- **Parameters:**
  - @cid: Course ID.

-**Procedure to control select , insert , update and delete an instructor**

1. sp_Instructor_Select procedure

- **Purpose:** Retrieves instructor details.
- **Parameters:**
  - @Ins_ID (optional): Instructor ID. If not provided, retrieves all instructors.

2. sp_Instructor_Insert procedure

- **Purpose:** Inserts a new instructor.
- **Parameters:**
  - @FName: First name.
  - @LName: Last name.
  - @Email: Email address.
  - @Pass: Password.
  - @Dept_ID: Department ID.

3. sp_Instructor_Update

- **Purpose:** Updates an existing instructor.
- **Parameters:**
  - @Ins_ID: Instructor ID.
  - @FName: First name.
  - @LName: Last name.
  - @Email: Email address.
  - @Pass: Password.
  - @Dept_ID: Department ID.

4. sp_Instructor_Delete

- **Purpose:** Deletes an instructor.
- **Parameters:**
  - @Ins_ID: Instructor ID.

Stored Procedure of Reports

1. **Report_StudentInfo**

```
alter proc Report_StudentInfo
	@Dept_no int
as
begin
		select *
		from Student
		where Dept_no=@Dept_no
		order by Dept_no
end;
```

   - **Purpose:** Retrieves student information for a specific department.
   - **Parameters:**
     - @Dept_no: Department number.

2. **StudentExam**

```
alter proc StudentExam
	@stud_id int
as
begin
	select Distinct Cname,grade
	from ExamStudent ES
	join Exam E
	on ES.ExamID=E.ID
	join Question_Exam QE
	on QE.Exam_id=E.ID
	join Question Q
	on Q.QID=QE.Q_id
	join Course C
	on C.CID=Q.CID
	where ES.stud_ID = @stud_id;

end
```

   - **Purpose:** Retrieves exam details for a specific student.
   - **Parameters:**
     - @stud_id: Student ID.

3. **GetInstructorCourses_StudentsCountCourse**

```
CREATE PROCEDURE GetInstructorCourses_StudentsCountCourse
    @InstructorID INT
AS
BEGIN
    IF NOT EXISTS (SELECT 1 FROM Instructor WHERE Ins_id = @InstructorID)
    BEGIN
        PRINT 'Instructor ID does not exist.';
        RETURN;
    END;
    SELECT
        C.Cname AS 'Course Name',
        COUNT(S.stud_ID) AS 'Students Count'
    FROM
        Instructor I
    JOIN Ins_Course IC
       ON I.Ins_id = IC.Ins_id
    JOIN Course C
       ON IC.C_id = C.CID
    JOIN Department D
       ON C.Dep_ID = D.Dept_no
    JOIN Student S
       ON S.Dept_no = D.Dept_no
    WHERE
        I.Ins_id = @InstructorID
    GROUP BY
        C.Cname
    ORDER BY
        C.Cname;
END;
```

   - **Purpose:** Retrieves the number of students enrolled in courses taught by a specific instructor.
   - **Parameters:**
     - @InstructorID: Instructor ID.

4. **ExamQuestionReport**

```
create proc ExamQuestionReport(@examid int)
as
begin
       select q.Body
       from Question q
       join Question_Exam qe
       on q.QID=qe.Q_id where qe.Exam_id=@examid
end;
```

   - **Purpose:** Retrieves questions for a specific exam.
   - **Parameters:**
     - @examid: Exam ID.

5. **StudentAnswers**

```
create proc StudentAnswers
       @stud_id int,
       @exam_id int
as
begin
       select Body,Answer
       from StudentAnswer SA
       join Question Q
       on SA.Q_ID=Q.QID and sa.Stud_ID=@stud_id and sa.Exam_ID=@exam_id
end;
```

   - **Purpose:** Retrieves answers provided by a student for a specific exam.
   - **Parameters:**
     - @stud_id: Student ID.
     - @exam_id: Exam ID.

6. **CourseTopics**

```
create proc CourseTopics
       @CID int
as
   select Tname
   from Topic
   where CID=@CID
end;
```

   - **Purpose:** Retrieves topics for a specific course.
   - **Parameters:**
     - @CID: Course ID.

Error Handling

- **Error Log Table:**
  - **Purpose:** Logs errors that occur during the execution of stored procedures.
  - **Columns:**
    - ErrorMessage: Description of the error.
    - ErrorTime: Timestamp of the error.
    - UserName: User who encountered the error.

4.2 Instructor Schema

The **Instructor Schema** contains procedures and tables related to instructor operations, such as login,, exam generation, and exam correction.

Stored Procedures

1. **Instructors.Login**
   - **Purpose:** Authenticates an instructor and logs the login attempt.
   - **Parameters:**
     - @Email: Instructor's email address.
     - @Pass: Instructor's password.
     - @ID (output): Instructor ID if login is successful.
   - **Behavior:**
     - If login is successful, the instructor ID is returned, and a success message is displayed.
     - If login fails, a failure message is displayed.
     - All login attempts are logged in the Login_Audit table.
   - **Usage:**
```
declare @id int
exec Instructors.Login 'ahmed@gmail.com','ahmed1234',@id output
```
2. **Instructors.AvailbleCourses**
   - **Purpose:** Retrieves the list of courses available for a specific instructor.
   - **Parameters:**
     - @ins_id: Instructor ID.
   - **Behavior:**
     - Returns the names of courses assigned to the instructor.
   - **Usage:**
```
declare @id int
exec Instructors.AvailbleCourses @id
```
3. **Instructors.GenerateExamByIns**
   - **Purpose:** Generates an exam for a specific course with a specified number of True/False and MCQ questions.
   - **Parameters:**
     - @courseName: Name of the course.
     - @QnumT: Number of True/False questions.
     - @QnumM: Number of MCQ questions.
     - @ins_id: Instructor ID.
   - **Behavior:**
     - Validates the instructor's login status and the course name.
     - Calls the ExamGeneration procedure to generate the exam.
     - Logs errors in the Error_Log table if any occur.
   - Usage:
```
declare @id int
exec Instructors.GenerateExamByIns 'Advanced C#',2,5,@id
```
4. **Instructors.ExamCorrection**
   - **Purpose:** Corrects an exam and calculates the final grade for a student.
   - **Parameters:**
     - @examid: Exam ID.
     - @stuId: Student ID.
     - @finalGrade (output): Final grade of the student.
   - **Behavior:**
     - Calculates the student's grade based on correct answers.
     - Updates the ExamStudent table with the final grade.
   - **Usage:**
```
DECLARE @Grade MONEY;
EXEC Instructors.ExamCorrection 19, 30, @Grade OUTPUT;
select 'Final Grade: ' + CAST(@Grade AS VARCHAR);
```

4.3 Student Schema

The **Student Schema** contains procedures related to student operations, such as login, viewing available exams, submitting exam answers, and checking grades.

Stored Procedures

1. **Students.Login**
   - **Purpose:** Authenticates a student and returns their student ID if successful.
   - **Parameters:**
     - @email: Student's email address.
     - @password: Student's password.
     - @stud_id (output): Student ID if login is successful.
   - **Behavior:**
     - If login is successful, the student ID is returned, and a success message is displayed.
     - If login fails, a failure message is displayed.
   - **Usage:**
```
declare @studId int;
exec Students.Login 'moali@example.com','89%452',@studId output
```

2. **Students.AvailbleExams**
   - **Purpose:** Retrieves the list of available exams for a specific student.
   - **Parameters:**
     - @stud_id: Student ID.
   - **Behavior:**
     - Validates the student's login status.
     - Returns the exam IDs and course names for exams the student has attempted.
   - **Usage:**
```
declare @studId int;
exec Students.AvailbleExams @studId;
```

3. **Students.ExamAnswers**
   - **Purpose:** Stores the answers provided by a student for a specific exam.
   - **Parameters:**
     - @stud_id: Student ID.
     - @exam_id: Exam ID.
     - @Answer: Comma-separated list of answers.
   - **Behavior:**
     - Inserts the student's answers into the StudentAnswer table.
   - **Usage:**
```
Students.ExamAnswers1,2,'True,True,False,string,default,const,for';
```

4. **Students.Show_Grade_SpecificExam**
   - **Purpose:** Retrieves the grade for a specific exam taken by a student.
   - **Parameters:**
     - @exam_id: Exam ID.
     - @stud_id: Student ID.
   - **Behavior:**
     - Validates the student's login status.
     - Returns the grade for the specified exam.
   - **Usage:**
```
declare @studId int;
exec Students.Show_Grade_SpecificExam 3,@studId
```

5. **Students.Show_AllGrades**
    - **Purpose:** Retrieves all grades for a specific student across all exams.
    - **Parameters:**
        - @stud_id: Student ID.
    - **Behavior:**
        - Validates the student's login status.
        - Returns the exam IDs and grades for all exams taken by the student.
    - **Usage:**

```
declare @studId int;
exec Students.Show_AllGrades @studId
```