Information
Technology
Institute

Examination System Database

Prepared by: Maryam Salah Hassan, Mohamed Atef Ghazal, Ahmed Khaled Ahmed, Mohammad Anwar Ahmed, Sherif Khaled Khalifa
Supervised By: Eng.Sarah Salah

Date: August 2025
Project: **Examination System Database**

1. Abstract

This project develops a database system to manage an examination process. The system provides a question pool, exam creation, student answers, result calculation, and reporting. It supports multiple user roles (admin, training manager, instructors, and students) with different permissions and ensures data integrity, performance, and security.

2. Introduction

Problem Statement: Need for a system that automates exam creation, execution, and evaluation.
Objectives:

- Store and manage students, instructors, and course information.
- Provide a pool of questions (MCQ, True/False, Text).
- Allow instructors to generate exams automatically or manually.
- Secure user access with role-based permissions.

3. System Requirements

Functional Requirements

- Store course, instructor, student, and exam details.
- Instructors can create exams using random/manual question selection.
- Exams have metadata (type, course, branch, intake, time, allowance).
- Students submit answers; the system auto-checks objective questions.
- Store results per student per exam.

Non-Functional Requirements

- Daily automatic backup.
- Optimized queries using indexes.
- Secure login with SQL Server users and permissions.

4. ERD (Entity-Relationship Diagram) & Mapping *[All Members]*

Entities may include:

- **Student:** (NID$_{PK}$, Fullname, Gender, Grad_year, GPA, College, BD, Email, Phone, City)
- **Instructor:** (NID$_{PK}$, Gender, Name, BD, Phone, Email, City, Salary)
- **Course:** (ID$_{pk}$, Name, Description, Max_Degree, Min_Degree)
- **Exam:** (ID$_{PK}$, No_Questions, Type, Date, Start_Time, End_Time, Total_Time, Allowance_Options, Total_Degree)
- **Question:** (ID$_{PK}$, Head, Options, Type, Correct_Answer, Degree)
- **Branch:** (ID$_{PK}$, Name)
- **Intake:** (ID$_{PK}$, Year, Start_Date, End_Date)
- **Track:** (ID$_{PK}$, Name)
- **Department:** (ID$_{PK}$, Name)

Tables may include:

<table>
<tr><th colspan="13">Student</th></tr>
<tr><td>NID</td><td>Full_Name</td><td>Gender</td><td>Birth_Date</td><td>Email</td><td>Phone</td><td>City</td><td>College</td><td>GPA</td><td>Grad_Year</td><td>Track_ID$_{FK}$</td><td>Intake_ID$_{FK}$</td><td>Branch_ID$_{FK}$</td></tr>
</table>

“Track_ID” references “ID” in table “Track”

“Intake_ID” references “ID” in table “Intake”

“Branch_ID” references “ID” in table “Branch”

<table>
<tr><th colspan="9">Instructor</th></tr>
<tr><td>NID</td><td>Full_Name</td><td>Gender</td><td>Birth_Date</td><td>Phone</td><td>Email</td><td>City</td><td>Salary</td><td>Department_ID$_{FK}$</td></tr>
</table>

“Department_ID” references “ID” in table “Department”

Exam													
ID	No_Question	Type	Date	Start_Time	End_Time	Total_Time	Total_Degree	Ins_NID$_{FK}$	Course_ID$_{FK}$	Intake_ID$_{FK}$	Branch_ID$_{FK}$	Track_ID$_{FK}$

“Ins_NID” references “ID” in table “Instructor”

“Course_ID” references “ID” in table “Course”

“Intake_ID” references “ID” in table “Intake”

“Branch_ID” references “ID” in table “Branch”

“Track_ID” references “ID” in table “Track”

Questions						
ID	Head	Options	Type	Correct_Answer	Degree	Course_ID$_{FK}$

“Course_ID” references “ID” in table “Course”

Exam_Question	
Exam_ID	Question_ID

“Exam_ID” references “ID” in table “Exam”

“Question_ID” references “ID” in table “Question”

Track		
ID	Name	Departmen_ID$_{FK}$

“Department_ID” references “ID” in table “Department”

Track_Courses	
Track_ID$_{FK}$	Course_ID$_{FK}$

“Track_ID” references “ID” in table “Track”

“Course_ID” references “ID” in table “Course”

Intake_Branch_Track		
Intake_ID$_{FK}$	Branch_ID$_{FK}$	Track_ID$_{FK}$

“Intake_ID” references “ID” in table “Intake”

“Branch_ID” references “ID” in table “Branch”

“Track_ID” references “ID” in table “Track”

Exam_Student_Questions_Answers					
Exam_ID$_{FK}$	Student_ID$_{FK}$	Question_ID$_{FK}$	Student_Degree	Student_Answer	IS_Correct

“Exam_ID” references “ID” in table “Exam”

“Student_ID” references “ID” in table “Student”

“Question_ID” references “ID” in table “Questions”

<table>
<tr><th colspan="2">Instructor_Courses</th></tr>
<tr><td>Instructor_ID$_{FK}$</td><td>Course_ID$_{FK}$</td></tr>
</table>

“Instructor_ID” references “ID” in table “Instructor”

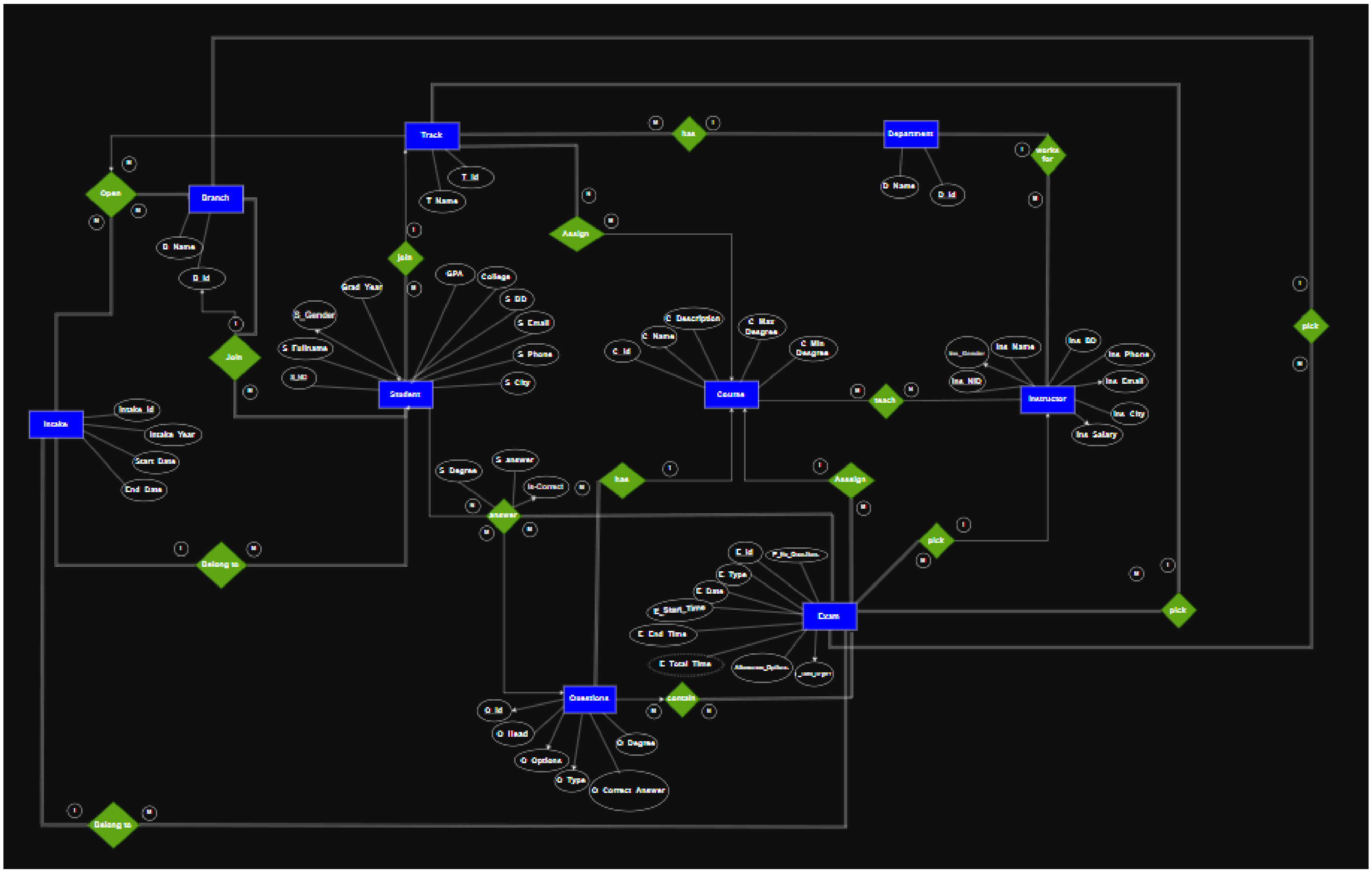
“Course_ID” references “ID” in table “Course”

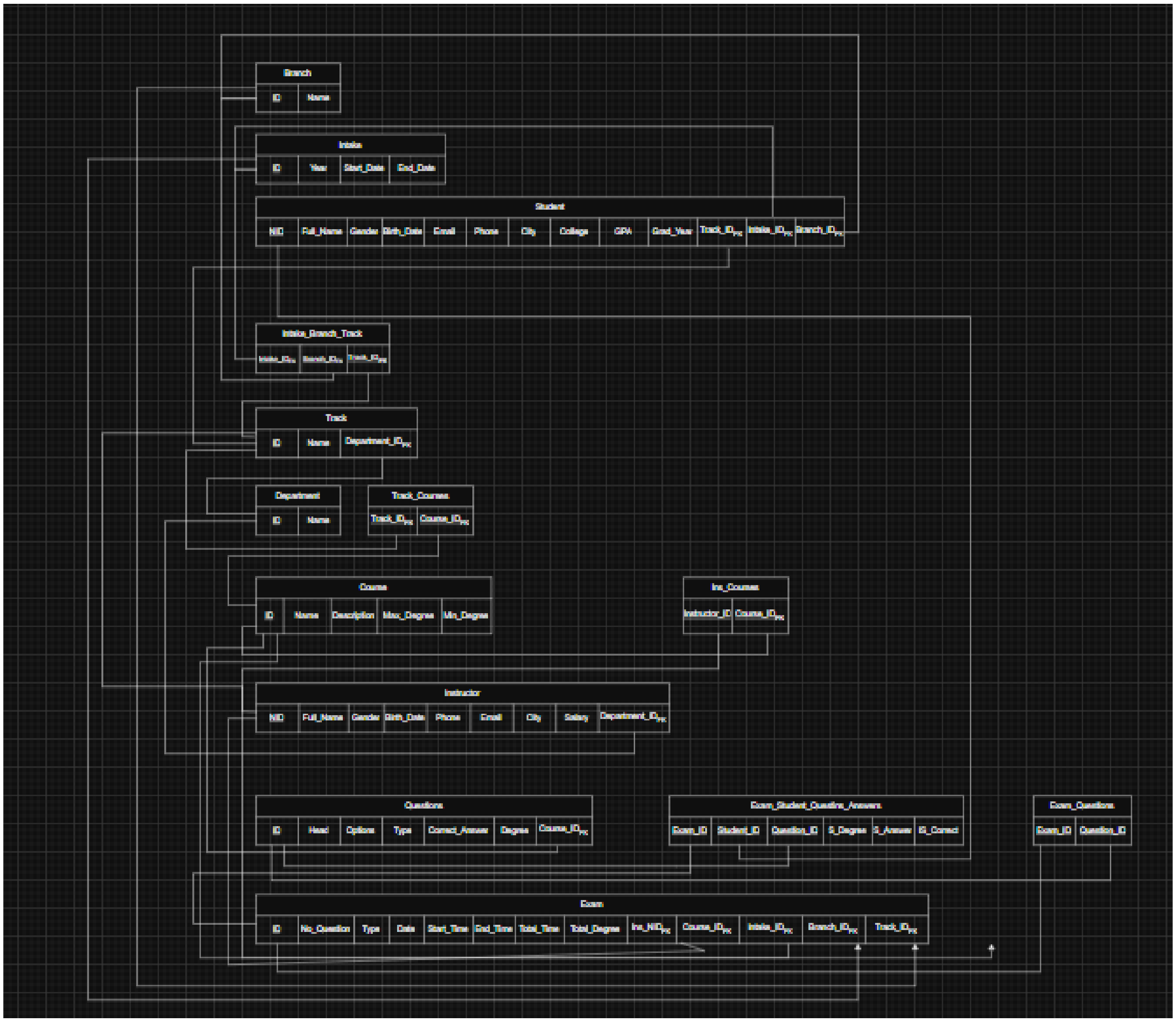
<table>
<tr><th colspan="4">Intake</th></tr>
<tr><td>ID</td><td>Year</td><td>Start_Date</td><td>End_Date</td></tr>
</table>

<table>
<tr><th colspan="5">Course</th></tr>
<tr><td>ID</td><td>Name</td><td>Description</td><td>Max_Degree</td><td>Min_Degree</td></tr>
</table>

<table>
<tr><th colspan="2">Branch</th></tr>
<tr><td>ID</td><td>Name</td></tr>
</table>

<table>
<tr><th colspan="2">Department</th></tr>
<tr><td>ID</td><td>Name</td></tr>
</table>


Track
has
Department
works for
T_Id
T Name
D Name
D_Id
Open
Branch
B Name
B_Id
Assign
Join
Join
Grad Year
GPA
College
S_Gender
S BD
S Email
S Fullname
S Phone
S_NID
S City
Student
C Description
C Name
C Max Degrees
C Min Degrees
C_Id
Course
teach
Ins_Gender
Ins Name
Ins BD
Ins Phone
Ins NID
Ins Email
Instructor
Ins City
Ins Salary
pick
Intake
Intake_Id
Intake Year
Start Date
End Date
S Degree
S answer
Is-Correct
has
Assign
answer
Belong to
pick
E_Id
E Type
E Date
E_Start_Time
E End Time
E Total Time
Exam
pick
Questions
contain
Q_Id
Q Head
Q Options
Q Type
Q Degree
Q Correct Answer
Belong to


Branch
ID
Name
Intake
ID
Year
Start_Date
End_Date
Student
NID
Full_Name
Gender
Birth_Date
Email
Phone
City
College
GPA
Grad_Year
Track_ID$_{FK}$
Intake_ID$_{FK}$
Branch_ID$_{FK}$
Intake_Branch_Track
Intake_ID$_{FK}$
Branch_ID$_{FK}$
Track_ID$_{FK}$
Track
ID
Name
Department_ID$_{FK}$
Department
ID
Name
Track_Courses
Track_ID$_{FK}$
Course_ID$_{FK}$
Course
ID
Name
Description
Max_Degree
Min_Degree
Ins_Courses
Instructor_ID
Course_ID$_{FK}$
Instructor
NID
Full_Name
Gender
Birth_Date
Phone
Email
City
Salary
Department_ID$_{FK}$
Questions
ID
Head
Options
Type
Correct_Answer
Degree
Course_ID$_{FK}$
Exam_Student_Questins_Answers
Exam_ID
Student_ID
Question_ID
S_Degree
S_Answer
IS_Correct
Exam_Questions
Exam_ID
Question_ID
Exam
ID
No_Question
Type
Date
Start_Time
End_Time
Total_Time
Total_Degree
Ins_NID$_{FK}$
Course_ID$_{FK}$
Intake_ID$_{FK}$
Branch_ID$_{FK}$
Track_ID$_{FK}$

5. Database Creation *[All Members]*

We designed the **(Examination System)** Database in SQL Server, including all required tables, relationships, and constraints.

File Group

A dedicated filegroup **(ExamDataFG.ndf)** was created for the **Exam Table, Questions Table, Exam_Question Table, and Exam_Student_Questions_Answers Table** to enhance database performance, storage management, and scalability.

—---

6. Constraints and Rules

[Mohamed Ghazal & Maryam Salah & Ahmed Khaled]:

1. Check Constraints

- **Constraint G** on Student.GPA:
  Ensures that the GPA of any student must be between **2.0 and 4.0**.
- **Constraint GY** on Student.Grad_Year:
  Ensures that the graduation year must fall within the last **five years up to the current year**.
- **Constraint ET** on Exam.Type:
  Restricts the exam type to only two valid values: **'exam'** or **'corrective'**.
- **Constraint QT** on Questions.Type:
  Restricts the type of a question to be one of the following: **'Multiple choice'**, **'True/False'**, or **'Text'**.

—---

2. Unique Constraints

- **Constraint YU** on Intake.Year:
  Guarantees that each intake year is unique in the system.
- **Constraint DNU** on Department.Name:
  Ensures that each department has a unique name.
- **Constraint TNU** on Track.Name:
  Ensures that each track has a unique name.
- **Constraint BNU** on Branch.Name:
  Ensures that each branch has a unique name.

- **Constraint CNU** on Course.Name:
  Ensures that each course has a unique name.
- **Constraint EIN** on Instructor.Email:
  Ensures that every instructor's email is unique.
- **Constraint ES** on Student.Email:
  Ensures that every student's email is unique.

—--

3. Rules

- **GEN_Rule** applied to Instructor.Gender:
  Restricts the gender field to only two values: **'M'** or **'F'**.
- **GEN_Rule** applied to Student.Gender:
  Restricts the gender field to only two values: **'M'** or **'F'**.

And there were also the Primary Key and Foreign Key constraints in the data creation.

—--

7. Data Creation and Insertion *[Mohammad Anwar & Sherif Khaled]*:

We created data sets for each table for testing, and inserted the data through bulk insertion

- Created 10 rows for table Branch
- Created 25 rows for table Course
- Created 6 rows for table Department
- Created 40 rows for table Exam
- Created 200 rows for table Exam_Question
- Created 2000 rows for table Exam_Student_Questions_Answers
- Created 60 rows for table Instructor_Courses
- Created 20 rows for table Instructor
- Created 2 rows for table Intake
- Created 40 rows for table Intake_Branch_Track
- Created 750 rows for table Questions
- Created 200 rows for table Student
- Created 10 rows for table Track
- Created 50 rows for table Track_Courses

8. Test Logic

After completing the ERD and Mapping, applying the necessary constraints and rules, as well as inserting sample data, we performed logic testing to ensure that the ERD and Mapping matched the required system logic.

—---

9. Stored Procedures

[Mohammad Anwar]:

Procedure: Add_Branch

- **Purpose:**
  This stored procedure inserts a new branch into the system.
- **Parameters:**
  (@ID - @Name)
- **Logic:**
  Inserts a new branch with its ID and name.
  If the ID already exists, an error is raised.
- **Output:**
  Confirmation message or error if duplicate ID

Procedure: Add_Department

- **Purpose:**
  This stored procedure inserts a new department into the system.
- **Parameters:**
  (@ID - @Name)
- **Logic:**
  Inserts a new department with its ID and name.
  If the ID already exists, an error is raised.
- **Output:**
  Confirmation message or error if duplicate ID.

Procedure: Add_Track

- **Purpose:**
  This stored procedure inserts a new track and associates it with a department.

- **Parameters:**
  (@ID - @Name - @Dept_ID)
- **Logic:**
  Inserts a new track with its ID, name, and related department ID.
  If the ID already exists, an error is raised.
- **Output**:
  Confirmation message or error if duplicate ID or invalid department.

Procedure: Add_Intake

- **Purpose:**
  This stored procedure inserts a new intake into the system.
- **Parameters:**
  (@ID - @Year - @Start_Date - @End_Date)
- **Logic:**
  Inserts a new intake record with year, start date, and end date.
  If the ID already exists, an error is raised.
- **Output:**
  Confirmation message or error if duplicate ID.

Procedure: Register_Student

- **Purpose:**
  Inserts a new student into the database with their personal and academic details
- **Parameters:** (@NID - @Full_Name - @Gender - @Birth_Date - @Email - @Phone - @City - @College - @GPA - @Grad_Year - @Track_ID - @Intake_ID - @Branch_ID)
- **Logic:**
  Checks if the student already exists.
  Inserts a new record into the student table if not found.
- **Output:**
  Confirmation message or error if the instructor exists

Procedure: Register_Instructor

- **Purpose:**
  This stored procedure inserts a new instructor into the system.
- **Parameters:**
  (@NID - @Full_Name - @Gender - @Birth_Date - @Phone - @Email - @City - @Salary - @Dept_ID)
- **Logic:**
  Checks if the instructor already exists.
  Inserts a new instructor record if not found.

- **Output:**
  Confirmation message or error if the instructor exists.

Procedure: RegisterCourse

- **Purpose:**
  This stored procedure inserts a new course into the system.
- **Parameters:**
  (@ID - @Name - @Description - @Min_Degree - @Max_Degree)
- **Logic:**
  Validates that Min_Degree < Max_Degree.
  Inserts a new course record.
- **Output:**
  Confirmation message or error if invalid degree range.

Procedure: Edit_Branch

- **Purpose:**
  This stored procedure updates the name of an existing branch.
- **Parameters:**
  (@ID - @Name)
- **Logic:**
  Updates the branch name where the given branch ID matches.
  If no rows are affected, it raises an error indicating the branch was not found.
- **Output:**
  Confirmation message or error if the branch is not found.

Procedure: Edit_Department

- **Purpose:**
  This stored procedure updates the name of an existing department.
- **Parameters:**
  (@ID - @Name)
- **Logic:**
  Updates the department name where the given department ID matches.
  If no rows are affected, it raises an error indicating the department was not found.
- **Output:**
  Confirmation message or error if the department is not found.

[Ahmed Khaled]

Procedure: Edit_Track

- **Purpose:**
  This stored procedure updates the details of an existing track, including its name and associated department.
- **Parameters:**
  (@ID - @Name - @Dept_ID)
- **Logic:**
  Updates the track name and department ID where the given track ID matches.
  If no rows are affected, it raises an error indicating the track was not found.
  If the department ID provided does not exist in the Department table, a foreign key violation error will be raised.
- **Output:**
  Confirmation message or error if track not found or invalid department ID.

Procedure: Edit_Intake

- **Purpose:**
  This stored procedure updates the details of an existing intake.
- **Parameters:**
  (@ID - @Year - @Start_Date - @End_Date)
- **Logic:**
  Updates the year, start date, and end date of the intake where the given intake ID matches.
  If no rows are affected, it raises an error indicating the intake was not found.
- **Output:**
  Confirmation message or error if intake not found.

Procedure: Edit_Student

- **Purpose:**
  This stored procedure updates student information. Only provided parameters will be updated, others remain unchanged.
- **Parameters:**
  (@NID - @Full_Name - @Gender - @Birth_Date - @Email - @Phone - @City - @College - @GPA - @Grad_Year - @Track_ID - @Intake_ID - @Branch_ID)

- **Logic:**
  Finds the student by NID.
  Updates provided fields using ISNULL.
- **Output:**
  Confirmation message or error if the student is not found.

Procedure: Edit_Instructor

- **Purpose:**
  This stored procedure updates instructor details.
- **Parameters:**
  (@NID - @Full_Name - @Gender - @Birth_Date - @Phone - @Email - @City - @Salary - @Dept_ID)
- **Logic:**
  Finds an instructor by NID.
  Updates provided fields using ISNULL.
- **Output:**
  Confirmation message or error if the instructor is not found.

Procedure: Edit_Course

- **Purpose:**
  This stored procedure updates course information.
- **Parameters:**
  (@ID - @Name - @Description - @Min_Degree - @Max_Degree)
- **Logic:**
  Validates that Min_Degree < Max_Degree and both are not null.
  Updates provided fields using ISNULL.
- **Output:**
  Confirmation message or error if the course is not found or invalid degree range.

Procedure: Remove_Branch

- **Purpose:**
  This stored procedure deletes an existing branch from the system.
- **Parameters:**
  (@ID)
- **Logic:**
  Deletes the branch where the given branch ID matches.
  If no rows are affected, it raises an error indicating the branch was not found.
- **Output:**
  Confirmation message or error if the branch is not found.

Procedure: Remove_Department

- **Purpose:**
  This stored procedure deletes an existing department from the system.
- **Parameters**:
  (@ID)
- **Logic:**
  Deletes the department where the given department ID matches.
  If no rows are affected, it raises an error indicating the department was not found.
- **Output:**
  Confirmation message or error if the department is not found.

Procedure: Remove_Track

- **Purpose:**
  This stored procedure deletes an existing track from the system.
- **Parameters:**
  (@ID)
- **Logic:**
  Deletes the track where the given track ID matches.
  If no rows are affected, it raises an error indicating the track was not found.
  If the track is referenced by other tables (e.g., Student, Exam, Track_Courses), a foreign key violation error will be raised.
- **Output:**
  Confirmation message or error if the track is not found or linked to other records.

Procedure: Remove_Intake

- **Purpose:**
  This stored procedure deletes an existing intake from the system.
- **Parameters:**
  (@ID)
- **Logic:**
  Deletes the intake where the given intake ID matches.
  If no rows are affected, it raises an error indicating the intake was not found.
  If the intake is referenced by other tables (e.g., Student, Exam, Intake_Branch_Track), a foreign key violation error will be raised.
- **Output:**
  Confirmation message or error if intake is not found or linked to other records.

Procedure: Remove_Student

- **Purpose:**
  This stored procedure deletes a student from the database.
- **Parameters:**
  (@NID)
- **Logic:**
  Finds students by NID.
  Deletes the student record.
- **Output:**
  Confirmation message or error if dependencies exist or student does not exist.

Procedure: Remove_Instructor

- **Purpose:**
  This stored procedure deletes an instructor from the database.
- **Parameters:**
  (@NID)
- **Logic:**
  Finds the instructor by NID.
  Deletes the instructor record.
- **Output:**
  Confirmation message or error if dependencies exist or instructor does not exist.

Procedure: Remove_Course

- **Purpose:**
  This stored procedure deletes a course from the system.
- **Parameters:**
  (@ID)
- **Logic:**
  Finds the course by ID.
  Deletes the course record.
- **Output:**
  Confirmation message or error if dependencies exist or course does not exist.

[Maryam Salah]

Procedure: Assign_Course_To_Instructor

- **Purpose:**
  Assigns a specific instructor to teach a specific course.
- **Parameters:**
  @InstructorID (Instructor ID), @CourseID (Course ID).
- **Logic:**
  Checks if the instructor is already assigned to the course.
  If the assignment already exists, it displays a message and does not insert a duplicate record.
  If not, inserts a new record into Instructor_Courses to establish the assignment.
- **Output:**
  Prints a message indicating whether the course was successfully assigned or if it was already assigned before.

Procedure: Create_Exam

- **Purpose:**
  Creates a new exam with full validation on time, degree, and instructor-course assignment.
- **Parameters:**
  @ID, @No_Question, @Type, @Date, @Start_Time, @End_Time, @Total_Degree, @InstructorID, @CourseID, @IntakeID, @BranchID, @TrackID.
- **Logic:**
  Validates that the exam end time is greater than the start time.
  Checks that the total degree does not exceed the maximum allowed degree for the course.
  Ensures the instructor is officially assigned to teach the given course.
  If all conditions are satisfied, inserts the new exam into the Exam table.
- **Output:**
  Prints a confirmation message: “EXAM CREATED SUCCESSFULLY.”

Procedure: Add_Questions_Randomly

- **Purpose:**
  Automatically assigns a random set of questions from a course’s question pool to an exam.

- **Parameters:**
  @ExamID (Exam ID), @NumberOfQuestions (Number of questions to be added).
- **Logic:**
  Retrieves the course ID associated with the given exam.
  Validates that the exam exists.
  Randomly selects the specified number of questions from the course's question pool, excluding those already assigned to the exam.
  Inserts the selected questions into Exam_Question.
- **Output:**
  Prints a confirmation message: "Random questions added successfully."

Procedure: Add_Questions_Manually

- **Purpose:**
  Allows manual assignment of a specific question to an exam, with course validation.
- **Parameters:**
  @ExamID (Exam ID), @QuestionID (Question ID).
- **Logic:**
  Retrieves the course ID of the exam and the course ID of the question.
  Validates that both the exam and question exist.
  Checks that the question belongs to the same course as the exam.
  If validation passes, inserts the question into Exam_Question.
- **Output:**
  Prints a confirmation message: "Question successfully added to exam."

—---

[Mohamed Atef Ghazal]

Procedure: View_Exam

- **Purpose:**
  This stored procedure allows a student to view the questions of a specific exam, but only if they are eligible and the exam is currently available.
- **Parameters:**
  @E_ID (Exam ID), @Student_ID (Student ID).
- **Logic:**
  Validates that the exam is assigned to the student's track and course.
  Checks that the exam date and time match the current system date and time (ensuring the exam is currently open).
  If the exam type is corrective, it verifies that the student has already failed the

corresponding normal exam before granting access.
If all checks pass, it retrieves and displays the exam questions with their options.

- **Output:**
  Returns the list of questions (Question ID, Question Head, and Options) for the specified exam if the student is allowed to access it.

Procedure: Submit_Answer

- **Purpose:**
  This stored procedure records or updates a student's answer to a specific exam question, automatically evaluating correctness and assigning a score.
- **Parameters:**
  @Exam_ID (Exam ID), @Student_ID (Student ID), @Question_ID (Question ID), @Answer (Submitted Answer).
- **Logic:**
  Checks if the student has already submitted an answer for the given exam and question:
  If yes → updates the existing record with the new answer, correctness, and score.
  If no → inserts a new record into EXAM_STUDENT_QUESTIONS_ANSWERS.
  For Multiple choice / True-False questions: compares the submitted answer with the correct answer.
  For Text questions: uses the dbo.Levenshtein_Similarity function to compare with the correct answer (must be ≥ 0.8 similarity to count as correct).
  Assigns a degree of 5 for correct answers, and 0 for incorrect ones.
- **Output:**
  Updates or inserts the student's answer into the database, along with correctness status and the assigned degree.

—---

[Sherif Khaled]

Procedure: Calculate_Result

- **Purpose:**
  This stored procedure calculates the final score for a specific student in a specific exam by comparing their submitted answers with the correct answers.
- **Parameters:**
  (@Student_ID - @Exam_ID)
- **Logic:**
  Joins the EXAM_STUDENT_QUESTIONS_ANSWERS and QUESTIONS tables.
  Compare each student's submitted answer with the correct answer.
  Sums the degrees for correctly answered questions only.
  Returns the total score for the specified student in the given exam.

- **Output:**
  (Student ID - Exam ID - Score)

Procedure: CourseResult

- **Purpose:**
  This stored procedure retrieves the total score of each student for a specific course taught by a given instructor.
  It is designed to provide instructors with an overview of how their students performed in exams related to their course.
- **Parameters:**
  (@instructor_id - @course_id)
- **Logic:**
  Joins the following tables: Student, Exam_Student_Questions_Answers, Exam, Course, and Instructor_Courses.
  Filters data by the instructor and course.
  Groups the data by student and exam to calculate the total degree scored by each student.
  Returns the exam type, student name, and total marks per exam.
- **Output:**
  (Student ID - Student Name - Exam Type - Exam ID - Course Name - Total Degree)

—---

11. Functions

Levenshtein Function (Text Question Evaluation)

- **Purpose:**
  To evaluate text-based answers by measuring the similarity between the student's answer and the model answer.

- **Parameters:**
  (@Student_Answer, @Correct_Answer)

- **Logic:**
  We used the function **dbo.Levenshtein_Similarity** which calculates a similarity score (0 → 1).
  If similarity ≥ 0.8 (80%), the answer is considered Correct.
  Otherwise, it is marked Incorrect.

12. Creating Views
[Ahmed Khaled & Mohammad Anwer & Sherif Khaled]

1. **Student_Info**
   This view shows basic profile details of each student, including Full student name, the year of the intake they belong to, the branch name, the track name

2. **Track_Info**
   This view provides details about tracks offered within intakes, branches, and departments. It shows which track belongs to which branch and department, and in which intake year it is available.

3. **Instructor_Info**
   provides details about instructors and the courses they teach, including course description, Max_Degree, and Min_Degree

4. **Student_Results**
   Lists exam participation records for students, including exam type, course, date, and total degree. It allows students and instructors to check exam allocations and the results overview.

5. **Exam_Statistics**
   provides exam-level statistics, including number of students who took the exam, highest score, lowest score, and average score. This is mainly for instructors and admins to evaluate exam difficulty and student performance**.**

13. Triggers

[Maryam Salah & Mohamed Ghazal]

1. trg_UpdateExamTotalDegree

- **Purpose**
  Automatically updates the total degree of an exam whenever questions are added to or removed from it.
- **Event**
  Executes after insert or delete on the Exam_Question table.

- **Logic**
  - Identifies the affected exam(s) using the inserted and deleted pseudo-tables.
  - Recalculates the total degree as the sum of the degrees of all associated questions.
  - Updates the Exam.Total_Degree column with the recalculated value.
- **Effect**
  Ensures that the total degree of any exam is always accurate and consistent with its assigned questions.

2. trg_UpdateExamQuestionCount

- **Purpose**
  Maintains the correct number of questions for each exam whenever questions are added or removed.
- **Event**
  Executes after insert or delete on the Exam_Question table.

- **Logic**
  - Identifies the affected exam(s) using the inserted and deleted pseudo-tables.
  - Recalculates the number of questions by counting all entries in Exam_Question for each exam.
  - Updates the Exam.No_Question column with the new count.
- **Effect**
  Keeps the number of exam questions up to date, avoiding manual updates and possible mismatches.

3. TRG_SUBMITANSWER

- **Purpose**
  Enforces the exam time window by preventing students from submitting or modifying answers outside the allowed exam period.
- **Event**
  Executes after insert or update on the Exam_Student_Questions_Answers table.
- **Logic**
  - Compares the current system date and time (GETDATE()) with the exam's scheduled date and time.
  - If a submission or update occurs outside the exam window, the transaction is rolled back.
  - Displays an error message:
    "YOU CANNOT SUBMIT OR UPDATE ANSWERS OUTSIDE THE EXAM TIME WINDOW."
- **Effect**
  Ensures strict compliance with exam scheduling and prevents late or early submissions.

—---

12. Login and Users *[All Members]*

We created SQL Server logins and corresponding database users for the main system roles:

- **Training Manager** User_Name: **(T_Manager) ,** Password **(222)**
- **Instructor** User_Name: **(Instructor) ,** Password (**333)**
- **Student** User_Name: **(Student) ,** Password **(444)**
- **Admin** User_Name: **(Admin) ,** Password **(111)**

This setup ensures secure access control and role-based management within the Examination System Database.

13. Permissions *[All Members]*

We defined permissions for each user on stored procedures and views according to their role in the Examination System Database.

Training Manager

- **Stored Procedures:**
  Add_Branch, Add_Department, Add_Track, Add_Intake, Register_Student, Register_Instructor, Register_Course,
  Edit_Branch, Edit_Department, Edit_Track, Edit_Intake, Edit_Student, Edit_Instructor, Edit_Course,
  Remove_Branch, Remove_Department, Remove_Track, Remove_Intake, Remove_Student, Remove_Instructor, Remove_Course,
  Assign_Course_To_Instructor

- **Views:**
  Track_Info, Student_Info, Instructor_Info, Student_Results, Exam_Statistics

Instructor

- **Stored Procedures:**
  Create_Exam, Add_Questions_Randomly, Add_Questions_Manually, Calculate_Result, Course_Result

Student

- **Stored Procedures:**
  View_Exam, Submit_Answer, Calculate_Result

—---

14. Daily Backup *[All Members]*

A scheduled **daily backup** is performed every day at **10:00 PM** to ensure data safety and recovery for the Examination System Database.