**ERD 🡪** I created an Entity relationship model for Examination system.

**Mapping 🡪** I created mapping to schema from an Entity relationship model for Examination system.

* **Tables**

1. **Department Table** 🡪Storesinformationabout department such as department identification, department name and department location.
2. **Training Manager Table 🡪** Storesinformationabout training manager such as training manager identification, training manager first name and last name, Email and his password.
3. **Branch Table 🡪** Storesinformationabout branches such as branch identification, branch name and department identification.
4. **Update\_adds\_inbranch Table 🡪** Storesoperationsthat training manager do in branches such as adding new branches in departments or update in existence branches. Information that stores in this table are training manager identification, branch identification and department identification.
5. **Intake Table 🡪** Storesinformation about intakes such as intake identification, intake name and branch identification.
6. **Intake\_addedby\_manager Table 🡪** stores information about new intakes that added by specific training manager. Information that stores training manageridentification and intake identification.
7. **Track Table 🡪** Stores information about track such as track identification, track name and department identification.
8. **Update\_adds\_inTrack🡪** Stores information about new tracks that added by training manager or update in existence tracks.
9. **Course Table 🡪** Stores information about courses such as course identification, course name, course description, course’s minimum degree and course’s maximum degree.

**10-Student Table 🡪** Storesinformation about student such as student identification, student first name and last name, student email and password, department identification, supervisor identification, intake identification, branch identification, track identification, training manager identification.

**11-Instructor Table 🡪** Storesinformation about instructor such as instructor identification, instructor first name and last name, instructor degree, instructor age, instructor email and password, instructor salary and department identification.

**12-Instructor\_courses 🡪** Storesinformation about courses that teached by specific instructor. Information such as instructor identification and course identification.

* **Stored procedures**

1. **Insert Department**

**Usage:** used to insert data into department table.

**Parameters:** @dept\_id, @dept\_name, @dept\_location.

**SQL query:**

create PROCEDURE DatabaseAdmin.InsertDepartment

@dept\_id int,

@dept\_name nvarchar (50),

@dept\_location nvarchar(50)

with encryption

AS

BEGIN

INSERT INTO DatabaseAdmin.Department (Dept\_id,dept\_name, dept\_location)

VALUES (@dept\_id,@dept\_name,@dept\_location);

END;

1. **Insert TrainngManger**

**Usage:** used to insert data into table Training manager.

**Parameters:** @Manager\_Fname, @Manager\_Lname, @Email, @password.

**SQL query:**

create PROCEDURE DatabaseAdmin.InsertTraninngManger

@Manager\_Fname nvarchar(50),

@Manager\_Lname nvarchar(50),

@Email nvarchar(50),

@password nvarchar(50)

with encryption

As

begin

insert into TrainingManagers.Training\_Manager(Manager\_Fname,Manager\_Lname,Email,password)

values (@Manager\_Fname,@Manager\_Lname,@Email,@password);

end;

1. **AddNewBranch**

**Usage:** used to insert new branch into table branch and also insert new branch id and training manager id that add new branch into table Update\_adds\_inbranch

**Parameters:** @Manager\_id, @Branch\_name, @Dept\_id.

**SQL query:**

CREATE PROCEDURE TrainingManagers.AddNewBranch

@Manager\_id INT,

@Branch\_name NVARCHAR(MAX),

@Dept\_id INT

AS

BEGIN

-- Declare a variable to hold the new Branch\_id

DECLARE @Branch\_id INT;

-- Insert the new branch into the Branch table

INSERT INTO TrainingManagers.Branch (Barnch\_name, dept\_id)

VALUES (@Branch\_name, @Dept\_id);

-- Get the newly generated Branch\_id

SET @Branch\_id = SCOPE\_IDENTITY();

-- Insert the Manager\_id, Branch\_id, and Dept\_id into Update\_adds\_inbranch table

INSERT INTO TrainingManagers.Update\_adds\_inbranch (Manager\_id, Branch\_id, Dept\_it)

VALUES (@Manager\_id, @Branch\_id, @Dept\_id);

-- Return success message

SELECT 'New branch added successfully and recorded by manager.' AS Message;

END;

1. **UpdateBranch**

**Usage:** use to make any update in table branch and store training manager id that make update on branch and branch id that updated into table Update\_adds\_inbranch

**Parameters:** @Manager\_id, @Branch\_id, @New\_Branch\_name, @New\_Dept\_id.

**SQL query:**

CREATE PROCEDURE TrainingManagers.UpdateBranch

@Manager\_id INT,

@Branch\_id INT,

@New\_Branch\_name NVARCHAR(MAX),

@New\_Dept\_id INT

AS

BEGIN

-- Check if the branch exists

IF NOT EXISTS (SELECT 1 FROM Branch WHERE Baranch\_id = @Branch\_id)

BEGIN

SELECT 'Branch does not exist.' AS Message;

RETURN;

END

-- Update the branch in the Branch table

UPDATE Branch

SET Barnch\_name = @New\_Branch\_name,

dept\_id = @New\_Dept\_id

WHERE Baranch\_id = @Branch\_id;

-- Insert the Manager\_id, Branch\_id, and Dept\_id into Update\_adds\_inbranch table to log the update

INSERT INTO TrainingManagers.Update\_adds\_inbranch (Manager\_id, Branch\_id, Dept\_it)

VALUES (@Manager\_id, @Branch\_id, @New\_Dept\_id);

-- Return success message

SELECT 'Branch updated successfully and update recorded by manager.' AS Message;

END;

1. **AddNewIntake**

**Usage:** insert new intake into table Intake by Training manager and also store manager id and intake id that added by training manager into table Intake\_addedby\_manager.

**Parameters:** @Manager\_id, @Intake\_name, @Branch\_id.

**SQL query:**

CREATE or alter PROCEDURE TrainingManagers.AddNewIntake

@Manager\_id INT,

@Intake\_name NVARCHAR(MAX),

@Branch\_id INT

with encryption

AS

BEGIN

-- Declare a variable to hold the new Intake\_id

DECLARE @Intake\_id INT;

-- Insert the new intake into the Intake table

INSERT INTO TrainingManagers.Intake (Intake\_name, Branch\_id)

VALUES (@Intake\_name, @Branch\_id);

-- Get the newly generated Intake\_id

SET @Intake\_id = SCOPE\_IDENTITY();

-- Insert the Manager\_id and Intake\_id into Intake\_addedby\_manager table

INSERT INTO TrainingManagers.Intake\_addedby\_manager (Manager\_id, Intack\_id)

VALUES (@Manager\_id, @Intake\_id);

-- Return success message

SELECT 'New intake added successfully and recorded by manager.' AS Message;

END;

1. **AddNewTrack**

**Usage:** used to add new track into table Track by Training manager and also store the manager id that add new track and track id that manager added into table Update\_adds\_inTrack.

**Parameters:** @Manager\_id, @Track\_name, @Dept\_id.

**SQL query:**

CREATE or alter PROCEDURE TrainingManagers.AddNewTrack

@Manager\_id INT,

@Track\_name NVARCHAR(MAX),

@Dept\_id INT

with encryption

AS

BEGIN

-- Declare a variable to hold the new Track\_id

DECLARE @Track\_id INT;

-- Insert the new track into the Track table

INSERT INTO TrainingManagers.Track (Track\_name, dept\_id)

VALUES (@Track\_name, @Dept\_id);

-- Get the newly generated Track\_id

SET @Track\_id = SCOPE\_IDENTITY();

-- Insert the Manager\_id, Track\_id, and Dept\_id into Update\_adds\_inTrack table

INSERT INTO TrainingManagers.Update\_adds\_inTrack (Manager\_id, Track\_id, dept\_id)

VALUES (@Manager\_id, @Track\_id, @Dept\_id);

-- Return success message

SELECT 'New track added successfully and recorded by manager.' AS Message;

END;

1. **UpdateTrack**

**Usage:** use to make update into table track by training manager and store update information like training manager id and track id which updated by training manager into table Update\_adds\_inTrack.

**Parameters:** @Manager\_id, @Track\_id, @New\_Track\_name, @New\_Dept\_id.

**SQL query:**

CREATE or alter PROCEDURE TrainingManagers.UpdateTrack

@Manager\_id INT,

@Track\_id INT,

@New\_Track\_name NVARCHAR(MAX),

@New\_Dept\_id INT

with encryption

AS

BEGIN

-- Check if the track exists

IF NOT EXISTS (SELECT 1 FROM Track WHERE Track\_id = @Track\_id)

BEGIN

SELECT 'Track does not exist.' AS Message;

RETURN;

END

-- Update the track in the Track table

UPDATE TrainingManagers.Track

SET Track\_name = @New\_Track\_name,

dept\_id = @New\_Dept\_id

WHERE Track\_id = @Track\_id;

-- Insert the Manager\_id, Track\_id, and Dept\_id into Update\_adds\_inTrack table to log the update

INSERT INTO TrainingManagers.Update\_adds\_inTrack (Manager\_id, Track\_id, dept\_id)

VALUES (@Manager\_id, @Track\_id, @New\_Dept\_id);

-- Return success message

SELECT 'Track updated successfully and update recorded by manager.' AS Message;

END;

1. **InsertCourse**

**Usage:** used to insert courses into table course.

**Parameters:** @Crs\_ID, @Crs\_name, @Description, @Min\_Degree, @Max\_Degree.

**SQL query:**

CREATE or alter PROCEDURE Instructors.InsertCourse

@Crs\_ID INT,

@Crs\_name NVARCHAR(100),

@Description NVARCHAR(MAX),

@Min\_Degree INT,

@Max\_Degree INT

AS

BEGIN

-- Check if a course with the same Crs\_ID already exists

IF EXISTS (SELECT 1 FROM Course WHERE Crs\_ID = @Crs\_ID)

BEGIN

SELECT 'Course with this ID already exists.' AS Message;

RETURN;

END

-- Insert the new course into the Course table

INSERT INTO Instructors.Course (Crs\_ID, Crs\_name, Description, Min\_Degree, Max\_Degree)

VALUES (@Crs\_ID, @Crs\_name, @Description, @Min\_Degree, @Max\_Degree);

-- Return success message

SELECT 'New course added successfully.' AS Message;

END;

1. **AddNewStudent**

**Usage:** use to insert new student into student table.

**Parameters:** @St\_ID, @St\_FName, @St\_LName, @Email, @Password, @Dept\_ID, @Supervisor\_ID, @Intake\_ID, @Track\_ID, @Manager\_ID, @Branch\_ID.

**SQL query:**

CREATE or alter PROCEDURE Students.AddNewStudent

@St\_ID INT,

@St\_FName NVARCHAR(50),

@St\_LName NVARCHAR(50),

@Email NVARCHAR(100),

@Password NVARCHAR(100),

@Dept\_ID INT,

@Supervisor\_ID INT,

@Intake\_ID INT,

@Track\_ID INT,

@Manager\_ID INT,

@Branch\_ID INT

with encryption

AS

BEGIN

-- Check if a student with the same St\_ID already exists

IF EXISTS (SELECT 1 FROM Student WHERE St\_ID = @St\_ID)

BEGIN

SELECT 'Student with this ID already exists.' AS Message;

RETURN;

END

-- Check if a student with the same Email already exists

IF EXISTS (SELECT 1 FROM Student WHERE Email = @Email)

BEGIN

SELECT 'Student with this Email already exists.' AS Message;

RETURN;

END

-- Insert the new student into the Student table

INSERT INTO Students.Student (St\_ID, St\_FName, St\_LName, Email, Password, Dept\_ID, Supervisor\_ID, Intake\_ID, Track\_ID, Manager\_ID, Branch\_ID)

VALUES (@St\_ID, @St\_FName, @St\_LName, @Email, @Password, @Dept\_ID, @Supervisor\_ID, @Intake\_ID, @Track\_ID, @Manager\_ID, @Branch\_ID);

-- Return success message

SELECT 'New student added successfully.' AS Message;

END;

1. **InsertInstructor**

**Usage:** used to insert instructor’s data into instructor table.

**Parameters:** @Ins\_ID, @Ins\_FName, @Ins\_LName, @Ins\_degree, @Age, @Email, @Password, @Salary, @Dept\_ID.

**SQL query:**

CREATE or alter PROCEDURE Instructors.InsertInstructor

@Ins\_ID INT,

@Ins\_FName NVARCHAR(50),

@Ins\_LName NVARCHAR(50),

@Ins\_degree NVARCHAR(50),

@Age INT,

@Email NVARCHAR(100),

@Password NVARCHAR(100),

@Salary DECIMAL(18, 2),

@Dept\_ID INT

with Encryption

AS

BEGIN

-- Check if an instructor with the same Ins\_ID already exists

IF EXISTS (SELECT 1 FROM Instructor WHERE Ins\_ID = @Ins\_ID)

BEGIN

SELECT 'Instructor with this ID already exists.' AS Message;

RETURN;

END

-- Check if an instructor with the same Email already exists

IF EXISTS (SELECT 1 FROM Instructor WHERE Email = @Email)

BEGIN

SELECT 'Instructor with this Email already exists.' AS Message;

RETURN;

END

-- Insert the new instructor into the Instructor table

INSERT INTO Instructors.Instructor (Ins\_ID, Ins\_FName, Ins\_LName, Ins\_degree, Age, Email, Password, Salary, Dept\_ID)

VALUES (@Ins\_ID, @Ins\_FName, @Ins\_LName, @Ins\_degree, @Age, @Email, @Password, @Salary, @Dept\_ID);

-- Return success message

SELECT 'New instructor added successfully.' AS Message;

END;

1. **AssignInstructorToCourse**

**Usage:** used to insert courses that teached by specific instructor into table Instructor\_Courses.

**Parameters:** @Course\_ID, @Instructor\_ID.

**SQL query:**

CREATE or alter PROCEDURE Instructors.AssignInstructorToCourse

@Course\_ID INT,

@Instructor\_ID INT

with encryption

AS

BEGIN

BEGIN TRY

-- Check if the association already exists

IF EXISTS (SELECT 1 FROM Instructor\_Courses WHERE Course\_ID = @Course\_ID AND Instructor\_ID = @Instructor\_ID)

BEGIN

SELECT 'Instructor is already assigned to teach this course.' AS Message;

RETURN;

END

-- Insert the association into Instructor\_Courses table

INSERT INTO Instructors.Instructor\_Courses (Course\_ID, Instructor\_ID)

VALUES (@Course\_ID, @Instructor\_ID);

-- Return success message

SELECT 'Instructor assigned to teach course successfully.' AS Message;

END TRY

BEGIN CATCH

-- Return error message

SELECT ERROR\_MESSAGE() AS ErrorMessage;

END CATCH;

END;

* **Views**

1. **Courses\_Teached\_ByInstructor**

**Usage:** This view displays for each instructor which courses he teaches.

**SQL query:**

create view Instructors.Courses\_Teached\_ByInstructor

with encryption

as

select i.Ins\_ID , i.Ins\_FName + ' ' + i.Ins\_LName as'Instructor Full Name',

i.Ins\_degree,c.Crs\_ID,c.Crs\_name,c.Description

from Instructor i inner join Instructor\_Courses ic

on ic.Instructor\_ID = i.Ins\_ID inner join Course c

on ic.Course\_ID = c.Crs\_ID

group by i.Ins\_ID, i.Ins\_FName + ' ' + i.Ins\_LName ,i.Ins\_degree,c.Crs\_ID,c.Crs\_name,c.Description

1. **Updates\_and\_AddsInBranch\_ByTriningManager**

**Usage:** This view shows branches that added and updated in each department by each Training Manager.

**SQL query:**

create view TrainingManagers.Updates\_and\_AddsInBranch\_ByTriningManager

with encryption

as

select t.Manager\_id,t.Manager\_Fname + ' ' + t.Manager\_Lname as'Manager Full Name',

b.Baranch\_id,b.Barnch\_name,d.Dept\_id,d.Dept\_name

from TrainingManagers.Training\_Manager t inner join TrainingManagers.Update\_adds\_inbranch un

on un.Manager\_id = t.Manager\_id inner join TrainingManagers.Branch b

on un.Branch\_id = b.Baranch\_id inner join DatabaseAdmin.Department d

on un.Dept\_it = d.Dept\_id

group by d.Dept\_id,d.Dept\_name,t.Manager\_id,t.Manager\_Fname + ' ' + t.Manager\_Lname,b.Baranch\_id,b.Barnch\_name

1. **Updates\_and\_AddsInTrack\_ByTriningManager**

**Usage:** This view shows Tracks that added and updated in each department by each Training Manager.

**SQL query:**

create view TrainingManagers.Updates\_and\_AddsInTrack\_ByTriningManager

with encryption

as

select t.Manager\_id,t.Manager\_Fname + ' ' + t.Manager\_Lname as'Manager Full Name',

tr.Track\_id,tr.Track\_name,d.Dept\_id,d.Dept\_name

from TrainingManagers.Training\_Manager t inner join TrainingManagers.Update\_adds\_inTrack ut

on ut.Manager\_id = t.Manager\_id inner join TrainingManagers.Track tr

on ut.Track\_id = tr.Track\_id inner join DatabaseAdmin.Department d

on ut.dept\_id = d.Dept\_id

group by t.Manager\_id,t.Manager\_Fname + ' ' + t.Manager\_Lname,tr.Track\_id,tr.Track\_name,d.Dept\_id,d.Dept\_name

* Functions

1. **GetIntakesByManager**

**Usage:** This inline table function displays intakes that added by each training manager.

**SQL query:**

CREATE FUNCTION TrainingManagers.GetIntakesByManager()

RETURNS TABLE

AS

RETURN

(

SELECT t.Manager\_id,t.Manager\_Fname + ' ' + t.Manager\_Lname as'Manager Name',

i.Intake\_id,i.Intake\_name,i.Branch\_Id

from TrainingManagers.Training\_Manager t inner join TrainingManagers.Intake\_addedby\_manager im

on im.Manager\_id = t.Manager\_id inner join Intake i

on im.Intack\_id = i.Intake\_id

);

1. **GetStudentsByManager**

**Usage:** This inline functions display students that added by training manager in a system.

**SQL query:**

CREATE FUNCTION TrainingManagers.GetStudentsByManager()

RETURNS TABLE

AS

RETURN

(

SELECT

tm.Manager\_id,

tm.Manager\_Fname,

tm.Manager\_Lname,

s.St\_ID,

s.St\_FName,

s.St\_LName,

s.Email,

s.Dept\_ID,

s.Intake\_ID,

s.Track\_ID,

s.Branch\_ID

FROM

TrainingManagers.Training\_Manager tm

JOIN

Students.Student s ON tm.Manager\_id = s.Manager\_ID

);

1. **GetCourseDetails**

**Usage:** This function to get Course Name and Description from Course ID.

**Parameters:** @Crs\_ID

**SQL query:**

CREATE FUNCTION Instructors.GetCourseDetails (@Crs\_ID INT)

RETURNS TABLE

AS

RETURN

(

SELECT

Crs\_name,

Description

FROM

Instructors.Course

WHERE

Crs\_ID = @Crs\_ID

);

1. **GetStudentNameBYID**

**Usage:** Function to get student full name by Enter ID.

**Parameters:** @St\_ID

**SQL query:**

CREATE FUNCTION TrainingManagers.GetStudentNameBYID (@St\_ID INT)

RETURNS NVARCHAR(101)

AS

BEGIN

DECLARE @FullName NVARCHAR(101);

SELECT @FullName = St\_FName + ' ' + St\_LName

FROM Students.Student

WHERE St\_ID = @St\_ID;

RETURN @FullName;

END;

1. **GetInstructorNameByID**

**Usage:** Function to get the instructor full name.

**Parameters:** @Ins\_ID

**SQL query:**

CREATE FUNCTION DatabaseAdmin.GetInstructorNameByID (@Ins\_ID INT)

RETURNS NVARCHAR(101)

AS

BEGIN

DECLARE @FullName NVARCHAR(Max);

SELECT @FullName = Ins\_FName + ' ' + Ins\_LName

FROM Instructors.Instructor

WHERE Ins\_ID = @Ins\_ID;

RETURN @FullName;

END;

1. **GetManagerNameByID**

**Usage:** This function return Training Manager Name by his id.

**Parameters:** @Manager\_ID

**SQL query:**

CREATE FUNCTION DatabaseAdmin.GetManagerNameByID (@Manager\_ID INT)

RETURNS NVARCHAR(max)

as

begin

DECLARE @FullName NVARCHAR(Max);

select @FullName = Manager\_Fname + ' ' + Manager\_Lname

from TrainingManagers.Training\_Manager

where Manager\_id = @Manager\_ID

RETURN @FullName;

END;

* **Triggers**

1. **trg\_CheckCourseDegree**

**usage:** This trigger will ensure that any new course inserted has valid Min\_Degree and Max\_Degree values.

**SQL query:**

CREATE TRIGGER trg\_CheckCourseDegree

ON Instructors.Course

for INSERT

AS

BEGIN

IF EXISTS (

SELECT 1

FROM inserted

WHERE Min\_Degree <50 OR Max\_Degree < Min\_Degree

)

BEGIN

RAISERROR ('Min\_Degree should be greater than or equal to 50 and less than or equal to Max\_Degree.', 16, 1);

ROLLBACK TRANSACTION;

END

END;

1. **trg\_CheckInstructorEmail**

**usage:** This trigger will ensure that the email for a new instructor or in update existence instructor is unique.

**SQL query:**

CREATE TRIGGER trg\_CheckInstructorEmail

ON Instructors.Instructor

FOR INSERT, UPDATE

AS

BEGIN

IF EXISTS (

SELECT 1

FROM inserted i

JOIN Instructors.Instructor ins ON i.Email = ins.Email AND i.Ins\_ID != ins.Ins\_ID

)

BEGIN

RAISERROR ('Email must be unique.', 16, 1);

ROLLBACK TRANSACTION;

END

END;

1. **trg\_CheckStudentEmailAndSupervisor**

**usage:** This trigger will ensure that the email for a new student or updated student is unique and the supervisor is a valid student ID.

**SQL query:**

CREATE TRIGGER trg\_CheckStudentEmailAndSupervisor

ON Students.Student

FOR INSERT, UPDATE

AS

BEGIN

IF EXISTS (

SELECT 1

FROM inserted i

JOIN Students.Student s ON i.Email = s.Email AND i.St\_ID != s.St\_ID

)

BEGIN

RAISERROR ('Email must be unique.', 16, 1);

ROLLBACK TRANSACTION;

END

IF EXISTS (

SELECT 1

FROM inserted i

WHERE i.Supervisor\_ID IS NOT NULL AND NOT EXISTS (SELECT 1 FROM Student s WHERE s.St\_ID = i.Supervisor\_ID)

)

BEGIN

RAISERROR ('Supervisor\_ID must be a valid Student ID.', 16, 1);

ROLLBACK TRANSACTION;

END

END;

1. **trg\_AssignInstructorToCourse**

**usage:** This trigger ensures that each instructor can teach one or more courses, and each course may be taught by one instructor per class.

**SQL query:**

CREATE TRIGGER trg\_AssignInstructorToCourse

ON Instructors.Instructor\_Courses

FOR INSERT

AS

BEGIN

IF EXISTS (

SELECT 1

FROM inserted i

JOIN Instructors.Instructor ins ON i.Instructor\_ID = ins.Ins\_ID

JOIN Instructors.Course c ON i.Course\_ID = c.Crs\_ID

WHERE NOT EXISTS (SELECT 1 FROM Instructor WHERE Ins\_ID = i.Instructor\_ID)

OR NOT EXISTS (SELECT 1 FROM Course WHERE Crs\_ID = i.Course\_ID)

)

BEGIN

RAISERROR ('Instructor or Course does not exist.', 16, 1);

ROLLBACK TRANSACTION;

END

END;

* **Permissions**

1. **Admin Permissions**

* Admins have access to Database Admin schema.

**SQL query**

GRANT CONTROL ON SCHEMA::DatabaseAdmin TO AdminRole;

* Admins have access to TrainingManagers schema

**SQL query**

GRANT CONTROL ON SCHEMA::TrainingManagers TO AdminRole;

* Admins have access to Instructors schema

**SQL query**

GRANT CONTROL ON SCHEMA::Instructors TO AdminRole;

* Admins have access to Students schema

**SQL query**

GRANT CONTROL ON SCHEMA::Students TO AdminRole;

* Admins have access to stored procedure that called InsertInstructor

**SQL query**

GRANT EXECUTE ON OBJECT::Instructors.InsertInstructor TO AdminRole;

* Admins have access to use stored procedure that called InsertDepartment

**SQL query**

GRANT EXECUTE ON OBJECT::DatabaseAdmin.InsertDepartment TO AdminRole;

* Admins have access to use stored procedure InsertTraninngManger

**SQL query**

GRANT EXECUTE ON OBJECT::DatabaseAdmin.InsertTraninngManger TO AdminRole;

* Admins have access to use stored procedure InsertTraninngManger

**SQL query**

GRANT EXECUTE ON OBJECT::Instructors.InsertInstructor TO AdminRole;

1. **Training Manager permissions**

* Training Managers have access to make select or insert or updates on tables that belongs to Training managers schema

**SQL query**

GRANT SELECT, INSERT, UPDATE, DELETE ON SCHEMA::TrainingManagers TO TrainingManagerRole;

* Training Managers have access to make select or insert or updates on table Student

**SQL query**

GRANT SELECT, INSERT, UPDATE, DELETE ON Students.Student TO TrainingManagerRole

* Training Managers have access to use stored procedure that called AddNewStudent

**SQL query**

GRANT EXECUTE ON OBJECT::Students.AddNewStudent TO TrainingManagerRole

* Training Managers have access to use stored procedure AddNewBranch

**SQL query**

GRANT EXECUTE ON OBJECT::TrainingManagers.AddNewBranch TO TrainingManagerRole

* Training Managers have access to use stored procedure UpdateBranch

**SQL query**

GRANT EXECUTE ON OBJECT::TrainingManagers.UpdateBranch TO TrainingManagerRole

* Training Managers have access to use stored procedure AddNewIntake

**SQL query**

GRANT EXECUTE ON OBJECT::TrainingManagers.AddNewIntake TO TrainingManagerRole

* Training Managers have access to use stored procedure AddNewTrack

**SQL query**

GRANT EXECUTE ON OBJECT::TrainingManagers.AddNewTrack TO TrainingManagerRole

* Training Managers have access to use stored procedure UpdateTrack

**SQL query**

GRANT EXECUTE ON OBJECT::TrainingManagers.UpdateTrack TO TrainingManagerRole

1. **Instructor Permissions**

* Instructors have permissions to select and update on tables in Instructors schema

**SQL query**

GRANT SELECT,Update ON SCHEMA::Instructors TO InstructorRole

* Instructors have permissions to select from table students

**SQL query**

GRANT SELECT ON Students.Student TO InstructorRole

* Instructors have permissions to SELECT, INSERT, UPDATE, DELETE into table Exam

**SQL query**

GRANT SELECT, INSERT, UPDATE, DELETE ON Instructors.Exam TO InstructorRole

* Instructors have permissions to SELECT, INSERT, UPDATE, DELETE into Exam\_Question

**SQL query**

GRANT SELECT, INSERT, UPDATE, DELETE ON Instructors.Exam\_Question TO InstructorRole

* Instructors have permissions to use stored procedure CreateExamWithQuestions

**SQL query**

GRANT EXECUTE ON OBJECT::Instructors.CreateExamWithQuestions TO InstructorRole

* Instructors have permissions to use stored procedure AssignInstructorToCourse

**SQL query**

GRANT EXECUTE ON OBJECT::Instructors.AssignInstructorToCourse TO InstructorRole

* Instructors have permissions to use stored procedure InsertQuestionWithAnswer

**SQL query**

GRANT EXECUTE ON OBJECT::Instructors.InsertQuestionWithAnswer TO InstructorRole

* Instructors have permissions to use stored procedure InsertCourse

**SQL query**

GRANT EXECUTE ON OBJECT::Instructors.InsertCourse TO InstructorRole

* Instructors have permissions to use stored procedure StudentExams

**SQL query**

GRANT SELECT, INSERT, UPDATE, DELETE ON Students.StudentExams TO InstructorRole;

1. **Student Permissions**

* Students have access to select only from table student

**SQL query**

GRANT SELECT ON SCHEMA::Students TO StudentRole;