Examination System Database Documentation

Examinatin_System

Server	KARIM
Author	Karim Essam
Created	11/01/2024
File Path	E:\frontend Projectes\SQL\SQL Project\Documentation\Examination System Database Docu-2024-01-11T11-56-50.pdf

The Examination System Database is a comprehensive solution designed to streamline and enhance the management of examinations within an educational institution. The project aims to automate and optimize the entire examination process, from question creation and student registration to result generation. Key features include efficient data management, secure user authentication, and insightful reporting capabilities.

Table of Contents

KARIM

Databases (1)

- Examinatin_System

User databases

Databases (1)

- Examinatin_System

Examinatin_System Database

Tables

Objects

Name
dbo.Branch
dbo.Branch_Track_Intake
dbo.Course
dbo.Department
dbo.Exam
dbo.Instructor
dbo.Intake
dbo.Intake_Instructor
dbo.Questions
dbo.student
dbo.Student_Course
dbo.Student_Exam_Questions
dbo.Track
dbo.Track_Course

[dbo].[Branch]

Columns

Key	Name	Data Type	Max Length (Bytes)	Nullability
PK C	Branch_Id	int	4	NOT NULL
	Branch_Name	varchar(50)	50	NOT NULL
	Branch_Address	nvarchar(50)	100	NULL allowed
	Branch_Phone	char(11)	11	NOT NULL
FK	Dept_ID	int	4	NOT NULL

Indexes

Key	Name	Key Columns	Unique
PK C	BranchPK	Branch_Id	True
	UQ__Branch__B90BCAA8813CFF69	Branch_Phone	True

Check Constraints

Name	On Column	Constraint
BranchPhoneCheck	Branch_Phone	(len([Branch_Phone])=(11))

Foreign Keys

Name	Columns
BranchDepartmentFK	Dept_ID->[dbo].[Department].[Dept_ID]

Permissions

Type	Action	Owning Principal
Grant	INSERT	RoleManger
Grant	SELECT	RoleManger
Grant	UPDATE	RoleManger

SQL Script

```
CREATE TABLE [dbo].[Branch]
(
```

```
[Branch_Id] [int] NOT NULL,
[Branch_Name] [varchar] (50) COLLATE SQL_Latin1_General_CP1_CI_AS NOT NULL,
[Branch_Address] [nvarchar] (50) COLLATE SQL_Latin1_General_CP1_CI_AS NULL,
[Branch_Phone] [char] (11) COLLATE SQL_Latin1_General_CP1_CI_AS NOT NULL,
[Dept_ID] [int] NOT NULL
) ON [PRIMARY]
GO
ALTER TABLE [dbo].[Branch] ADD CONSTRAINT [BranchPhoneCheck] CHECK ((len([Branch_Phone])=(11)))
GO
ALTER TABLE [dbo].[Branch] ADD CONSTRAINT [BranchPK] PRIMARY KEY CLUSTERED ([Branch_Id]) ON
[PRIMARY]
GO
ALTER TABLE [dbo].[Branch] ADD CONSTRAINT [UQ__Branch__B90BCAA8813CFF69] UNIQUE NONCLUSTERED
([Branch_Phone]) ON [PRIMARY]
GO
ALTER TABLE [dbo].[Branch] ADD CONSTRAINT [BranchDepartmentFK] FOREIGN KEY ([Dept_ID]) REFERENCES
[dbo].[Department] ([Dept_ID])
GO
GRANT INSERT ON  [dbo].[Branch] TO [RoleManger]
GO
GRANT SELECT ON  [dbo].[Branch] TO [RoleManger]
GO
GRANT UPDATE ON  [dbo].[Branch] TO [RoleManger]
GO
```

Uses

[dbo].[Department]

Used By

[dbo].[Branch_Track_Intake]

[dbo].[Branch_Track_Intake]

Columns

Key	Name	Data Type	Max Length (Bytes)	Nullability
PK FK C	Branch_Id	int	4	NOT NULL
PK FK C	Track_Id	int	4	NOT NULL
PK FK C	Intake_Id	int	4	NOT NULL

Indexes

Key	Name	Key Columns	Unique
PK C	BranchTrackIntakePK	Branch_Id, Track_Id, Intake_Id	True

Foreign Keys

Name	Columns
Branch_BranchTrackIntakeFK	Branch_Id->[dbo].[Branch].[Branch_Id]
Intake_BranchTrackIntakeFK	Intake_Id->[dbo].[Intake].[Intake_Id]
Track_BranchTrackIntakeFK	Track_Id->[dbo].[Track].[Track_Id]

SQL Script

```
CREATE TABLE [dbo].[Branch_Track_Intake]
(
[Branch_Id] [int] NOT NULL,
[Track_Id] [int] NOT NULL,
[Intake_Id] [int] NOT NULL
) ON [PRIMARY]
GO
ALTER TABLE [dbo].[Branch_Track_Intake] ADD CONSTRAINT [BranchTrackIntakePK] PRIMARY KEY
CLUSTERED ([Branch_Id], [Track_Id], [Intake_Id]) ON [PRIMARY]
GO
ALTER TABLE [dbo].[Branch_Track_Intake] ADD CONSTRAINT [Branch_BranchTrackIntakeFK] FOREIGN KEY
([Branch_Id]) REFERENCES [dbo].[Branch] ([Branch_Id])
GO
ALTER TABLE [dbo].[Branch_Track_Intake] ADD CONSTRAINT [Intake_BranchTrackIntakeFK] FOREIGN KEY
([Intake_Id]) REFERENCES [dbo].[Intake] ([Intake_Id])
GO
ALTER TABLE [dbo].[Branch_Track_Intake] ADD CONSTRAINT [Track_BranchTrackIntakeFK] FOREIGN KEY
([Track_Id]) REFERENCES [dbo].[Track] ([Track_Id])
GO
```

Uses

[dbo].[Branch]

[dbo].[Intake]

[dbo].[Track]

[dbo].[Course]

Columns

Key	Name	Data Type	Max Length (Bytes)	Nullability
PK C	Crs_Id	int	4	NOT NULL
	Crs_Name	nvarchar(50)	100	NOT NULL
	Crs_Description	nvarchar(max)	max	NULL allowed
	MinDegree	int	4	NULL allowed
	MaxDegree	int	4	NULL allowed
FK	Ins_Id	int	4	NULL allowed

Indexes

Key	Name	Key Columns	Unique
PK C	CoursePK	Crs_Id	True

Foreign Keys

Name	Columns
CourseInstructorFK	Ins_Id->[dbo].[Instructor].[Ins_Id]

Permissions

Type	Action	Owning Principal
Grant	DELETE	RoleManger
Grant	INSERT	RoleManger
Grant	SELECT	RoleManger
Grant	UPDATE	RoleManger

SQL Script

```
CREATE TABLE [dbo].[Course]
(
[Crs_Id] [int] NOT NULL,
[Crs_Name] [nvarchar] (50) COLLATE SQL_Latin1_General_CP1_CI_AS NOT NULL,
[Crs_Description] [nvarchar] (max) COLLATE SQL_Latin1_General_CP1_CI_AS NULL,
[MinDegree] [int] NULL,
[MaxDegree] [int] NULL,
[Ins_Id] [int] NULL
```

```
) ON [PRIMARY]
GO
ALTER TABLE [dbo].[Course] ADD CONSTRAINT [CoursePK] PRIMARY KEY CLUSTERED ([Crs_Id]) ON
[PRIMARY]
GO
ALTER TABLE [dbo].[Course] ADD CONSTRAINT [CourseInstructorFK] FOREIGN KEY ([Ins_Id]) REFERENCES
[dbo].[Instructor] ([Ins_Id])
GO
GRANT DELETE ON  [dbo].[Course] TO [RoleManger]
GO
GRANT INSERT ON  [dbo].[Course] TO [RoleManger]
GO
GRANT SELECT ON  [dbo].[Course] TO [RoleManger]
GO
GRANT UPDATE ON  [dbo].[Course] TO [RoleManger]
GO
```

Uses

[dbo].[Instructor]

Used By

[dbo].[Exam]
[dbo].[Questions]
[dbo].[Student_Course]
[dbo].[Track_Course]
[dbo].[InsertExam]
[dbo].[InsertExamRandom]
[dbo].[SeeAllCoursesWithInstructorsNames]
[dbo].[SeeExamesOfSpecificCourse]
[dbo].[SeeExamesScheduleOfAllCourses]
[dbo].[SeeResultOfAllCourses]
[dbo].[SeeResultOfSpecificCourse]
[dbo].[setStudentStatus]
[dbo].[ShowDegreeStudentsInCourse]
[dbo].[ShowInstructorCourseinAllIntake]
[dbo].[ShowInstructorCourseinThisIntake]
[dbo].[ShowQuestionsInCourse]
[dbo].[ShowStudentsCourse]

[dbo].[Department]

Columns

Key	Name	Data Type	Max Length (Bytes)	Nullability
PK C	Dept_ID	int	4	NOT NULL
	Dept_Name	varchar(50)	50	NOT NULL
	Dept_Location	nvarchar(50)	100	NULL allowed

Indexes

Key	Name	Key Columns	Unique
PK C	DepartmentPK	Dept_ID	True

SQL Script

```
CREATE TABLE [dbo].[Department]
(
[Dept_ID] [int] NOT NULL,
[Dept_Name] [varchar] (50) COLLATE SQL_Latin1_General_CP1_CI_AS NOT NULL,
[Dept_Location] [nvarchar] (50) COLLATE SQL_Latin1_General_CP1_CI_AS NULL
) ON [PRIMARY]
GO
ALTER TABLE [dbo].[Department] ADD CONSTRAINT [DepartmentPK] PRIMARY KEY CLUSTERED ([Dept_ID]) ON
[PRIMARY]
GO
```

Used By

[dbo].[Branch]

[dbo].[Exam]

Columns

Key	Name	Data Type	Max Length (Bytes)	Nullability
PK C	Exam_Id	int	4	NOT NULL
	E_Type	varchar(10)	10	NOT NULL
	Start_Time	datetime	8	NOT NULL
	End_Time	datetime	8	NOT NULL
	Total_time	varchar(30)	30	NOT NULL
FK	Intake_Id	int	4	NOT NULL
FK	Crs_Id	int	4	NOT NULL
FK	Ins_Id	int	4	NOT NULL

Indexes

Key	Name	Key Columns	Unique
PK C	ExamPK	Exam_Id	True

Check Constraints

Name	On Column	Constraint
ExamType	E_Type	([E_Type]='Corrective' OR [E_Type]='Regular')

Foreign Keys

Name	Columns
ExamCourseFK	Crs_Id->[dbo].[Course].[Crs_Id]
ExamInstructorFK	Ins_Id->[dbo].[Instructor].[Ins_Id]
ExamIntakeFK	Intake_Id->[dbo].[Intake].[Intake_Id]

SQL Script

```
CREATE TABLE [dbo].[Exam]
(
[Exam_Id] [int] NOT NULL,
[E_Type] [varchar] (10) COLLATE SQL_Latin1_General_CP1_CI_AS NOT NULL,
[Start_Time] [datetime] NOT NULL,
[End_Time] [datetime] NOT NULL,
```

```
[Total_time] [varchar] (30) COLLATE SQL_Latin1_General_CP1_CI_AS NOT NULL,
[Intake_Id] [int] NOT NULL,
[Crs_Id] [int] NOT NULL,
[Ins_Id] [int] NOT NULL
) ON [PRIMARY]
GO
ALTER TABLE [dbo].[Exam] ADD CONSTRAINT [ExamType] CHECK (([E_Type]='Corrective' OR [E_-
Type]='Regular'))
GO
ALTER TABLE [dbo].[Exam] ADD CONSTRAINT [ExamPK] PRIMARY KEY CLUSTERED ([Exam_Id]) ON [PRIMARY]
GO
ALTER TABLE [dbo].[Exam] ADD CONSTRAINT [ExamCourseFK] FOREIGN KEY ([Crs_Id]) REFERENCES
[dbo].[Course] ([Crs_Id])
GO
ALTER TABLE [dbo].[Exam] ADD CONSTRAINT [ExamInstructorFK] FOREIGN KEY ([Ins_Id]) REFERENCES
[dbo].[Instructor] ([Ins_Id])
GO
ALTER TABLE [dbo].[Exam] ADD CONSTRAINT [ExamIntakeFK] FOREIGN KEY ([Intake_Id]) REFERENCES
[dbo].[Intake] ([Intake_Id])
GO
```

Uses

[dbo].[Course]
[dbo].[Instructor]
[dbo].[Intake]

Used By

[dbo].[Student_Exam_Questions]
[dbo].[InsertExam]
[dbo].[InsertExamRandom]
[dbo].[InsertStudentAnswers]
[dbo].[SeeExamesOfSpecificCourse]
[dbo].[SeeExamesScheduleOfAllCourses]
[dbo].[setStudentStatus]
[dbo].[ShowQuestionsExam]
[dbo].[VerifyStudentTimeOfEnteredExamAndGivePermission]

[dbo].[Instructor]

Columns

Key	Name	Data Type	Max Length (Bytes)	Nullability	Default
PK C	Ins_Id	int	4	NOT NULL	
	Ins_Name	nvarchar(50)	100	NOT NULL	
	Ins_Age	int	4	NULL allowed	
	Ins_Address	nvarchar(50)	100	NULL allowed	('Cairo')
	Ins_Phone	char(11)	11	NOT NULL	

Indexes

Key	Name	Key Columns	Unique
PK C	InstructorPK	Ins_Id	True

Check Constraints

Name	On Column	Constraint
InsPhoneCheck	Ins_Phone	(len([Ins_Phone])=(11))

Permissions

Type	Action	Owning Principal
Grant	DELETE	RoleManger
Grant	INSERT	RoleManger
Grant	SELECT	RoleManger
Grant	UPDATE	RoleManger

SQL Script

```
CREATE TABLE [dbo].[Instructor]
(
[Ins_Id] [int] NOT NULL,
[Ins_Name] [nvarchar] (50) COLLATE SQL_Latin1_General_CP1_CI_AS NOT NULL,
[Ins_Age] [int] NULL,
[Ins_Address] [nvarchar] (50) COLLATE SQL_Latin1_General_CP1_CI_AS NULL CONSTRAINT [DF__-
Instructo__Ins_A__52593CB8] DEFAULT ('Cairo'),
[Ins_Phone] [char] (11) COLLATE SQL_Latin1_General_CP1_CI_AS NOT NULL
) ON [PRIMARY]
```

```
GO
ALTER TABLE [dbo].[Instructor] ADD CONSTRAINT [InsPhoneCheck] CHECK ((len([Ins_Phone])=(11)))
GO
ALTER TABLE [dbo].[Instructor] ADD CONSTRAINT [InstructorPK] PRIMARY KEY CLUSTERED ([Ins_Id]) ON
[PRIMARY]
GO
GRANT DELETE ON  [dbo].[Instructor] TO [RoleManger]
GO
GRANT INSERT ON  [dbo].[Instructor] TO [RoleManger]
GO
GRANT SELECT ON  [dbo].[Instructor] TO [RoleManger]
GO
GRANT UPDATE ON  [dbo].[Instructor] TO [RoleManger]
GO
```

Used By

[dbo].[Course]
[dbo].[Exam]
[dbo].[Intake_Instructor]
[dbo].[SeeAllCoursesWithInstructorsNames]
[dbo].[ShowDegreeStudentsInCourse]
[dbo].[ShowInstructorCourseinAllIntake]
[dbo].[ShowInstructorCourseinThisIntake]
[dbo].[ShowStudentsCourse]

[dbo].[Intake]

Columns

Key	Name	Data Type	Max Length (Bytes)	Nullability
PK C	Intake_Id	int	4	NOT NULL
	Intake_Name	varchar(20)	20	NOT NULL

Indexes

Key	Name	Key Columns	Unique
PK C	IntakePK	Intake_Id	True

Permissions

Type	Action	Owning Principal
Grant	INSERT	RoleManger
Grant	SELECT	RoleManger

SQL Script

```
CREATE TABLE [dbo].[Intake]
(
[Intake_Id] [int] NOT NULL,
[Intake_Name] [varchar] (20) COLLATE SQL_Latin1_General_CP1_CI_AS NOT NULL
) ON [PRIMARY]
GO
ALTER TABLE [dbo].[Intake] ADD CONSTRAINT [IntakePK] PRIMARY KEY CLUSTERED ([Intake_Id]) ON
[PRIMARY]
GO
GRANT INSERT ON  [dbo].[Intake] TO [RoleManger]
GO
GRANT SELECT ON  [dbo].[Intake] TO [RoleManger]
GO
```

Used By

[dbo].[Branch_Track_Intake]
[dbo].[Exam]
[dbo].[Intake_Instructor]
[dbo].[student]
[dbo].[InsertExam]

[dbo].[InsertExamRandom]
[dbo].[SeeExamesOfSpecificCourse]
[dbo].[SeeExamesScheduleOfAllCourses]
[dbo].[ShowDegreeStudentsInCourse]
[dbo].[ShowInstructorCourseinAllIntake]
[dbo].[ShowInstructorCourseinThisIntake]
[dbo].[ShowStudentsCourse]

[dbo].[Intake_Instructor]

Columns

Key	Name	Data Type	Max Length (Bytes)	Nullability
PK FK C	Intake_Id	int	4	NOT NULL
PK FK C	Ins_Id	int	4	NOT NULL

Indexes

Key	Name	Key Columns	Unique
PK C	IntakeInstructorPK	Intake_Id, Ins_Id	True

Foreign Keys

Name	Columns
Ins_IntakeInstructorFK	Ins_Id->[dbo].[Instructor].[Ins_Id]
Intake_IntakeInstructorFK	Intake_Id->[dbo].[Intake].[Intake_Id]

SQL Script

```
CREATE TABLE [dbo].[Intake_Instructor]
(
[Intake_Id] [int] NOT NULL,
[Ins_Id] [int] NOT NULL
) ON [PRIMARY]
GO
ALTER TABLE [dbo].[Intake_Instructor] ADD CONSTRAINT [IntakeInstructorPK] PRIMARY KEY CLUSTERED
([Intake_Id], [Ins_Id]) ON [PRIMARY]
GO
ALTER TABLE [dbo].[Intake_Instructor] ADD CONSTRAINT [Ins_IntakeInstructorFK] FOREIGN KEY ([Ins_-
Id]) REFERENCES [dbo].[Instructor] ([Ins_Id])
GO
ALTER TABLE [dbo].[Intake_Instructor] ADD CONSTRAINT [Intake_IntakeInstructorFK] FOREIGN KEY
([Intake_Id]) REFERENCES [dbo].[Intake] ([Intake_Id])
GO
```

Uses

[dbo].[Instructor]
[dbo].[Intake]

Used By

[dbo].[InsertExam]

[dbo].[InsertExamRandom]

[dbo].[ShowInstructorCourseinAllIntake]

[dbo].[ShowInstructorCourseinThisIntake]

[dbo].[Questions]

Columns

Key	Name	Data Type	Max Length (Bytes)	Nullability
PK C	Question_Id	int	4	NOT NULL
	Q_Type	varchar(10)	10	NOT NULL
	Correct_Answer	nvarchar(max)	max	NULL allowed
	Question_Text	nvarchar(max)	max	NULL allowed
	Text_Keywords	nvarchar(max)	max	NULL allowed
FK	Crs_Id	int	4	NOT NULL

Indexes

Key	Name	Key Columns	Unique
PK C	QuestionsPK	Question_Id	True

Check Constraints

Name	On Column	Constraint
QuestionType	Q_Type	([Q_Type]='Text' OR [Q_Type]='MCQ' OR [Q_Type]='T/F')

Foreign Keys

Name	Columns
QuestionsCourseFK	Crs_Id->[dbo].[Course].[Crs_Id]

SQL Script

```
CREATE TABLE [dbo].[Questions]
(
[Question_Id] [int] NOT NULL,
[Q_Type] [varchar] (10) COLLATE SQL_Latin1_General_CP1_CI_AS NOT NULL,
[Correct_Answer] [nvarchar] (max) COLLATE SQL_Latin1_General_CP1_CI_AS NULL,
[Question_Text] [nvarchar] (max) COLLATE SQL_Latin1_General_CP1_CI_AS NULL,
[Text_Keywords] [nvarchar] (max) COLLATE SQL_Latin1_General_CP1_CI_AS NULL,
[Crs_Id] [int] NOT NULL
) ON [PRIMARY]
GO
ALTER TABLE [dbo].[Questions] ADD CONSTRAINT [QuestionType] CHECK (([Q_Type]='Text' OR [Q_-
Type]='MCQ' OR [Q_Type]='T/F'))
```

```
GO
ALTER TABLE [dbo].[Questions] ADD CONSTRAINT [QuestionsPK] PRIMARY KEY CLUSTERED ([Question_Id])
ON [PRIMARY]
GO
ALTER TABLE [dbo].[Questions] ADD CONSTRAINT [QuestionsCourseFK] FOREIGN KEY ([Crs_Id])
REFERENCES [dbo].[Course] ([Crs_Id])
GO
```

Uses

[dbo].[Course]

Used By

[dbo].[Student_Exam_Questions]
[dbo].[InsertExam]
[dbo].[InsertExamRandom]
[dbo].[ShowQuestionsExam]
[dbo].[ShowQuestionsInCourse]
[dbo].[VerifyStudentTimeOfEnteredExamAndGivePermission]

[dbo].[student]

Columns

Key	Name	Data Type	Max Length (Bytes)	Nullability	Default
PK C	Std_Id	int	4	NOT NULL	
	Std_Fname	nvarchar(20)	40	NOT NULL	
	Std_Lname	nvarchar(20)	40	NOT NULL	
	Std_Age	int	4	NOT NULL	
	Std_Address	nvarchar(50)	100	NULL allowed	('Cairo')
	Std_Phone	char(11)	11	NOT NULL	
FK	Intake_Id	int	4	NOT NULL	

Indexes

Key	Name	Key Columns	Unique
PK C	StudentPK	Std_Id	True

Check Constraints

Name	On Column	Constraint
StdPhoneCheck	Std_Phone	(len([Std_Phone])=(11))

Foreign Keys

Name	Columns
IntakeStudentFK	Intake_Id->[dbo].[Intake].[Intake_Id]

Permissions

Type	Action	Owning Principal
Grant	INSERT	RoleManger
Grant	SELECT	RoleManger

SQL Script

```
CREATE TABLE [dbo].[student]
(
```

```
[Std_Id] [int] NOT NULL,
[Std_Fname] [nvarchar] (20) COLLATE SQL_Latin1_General_CP1_CI_AS NOT NULL,
[Std_Lname] [nvarchar] (20) COLLATE SQL_Latin1_General_CP1_CI_AS NOT NULL,
[Std_Age] [int] NOT NULL,
[Std_Address] [nvarchar] (50) COLLATE SQL_Latin1_General_CP1_CI_AS NULL CONSTRAINT
[DF__student__Std_Add__534D60F1] DEFAULT ('Cairo'),
[Std_Phone] [char] (11) COLLATE SQL_Latin1_General_CP1_CI_AS NOT NULL,
[Intake_Id] [int] NOT NULL
) ON [PRIMARY]
GO
ALTER TABLE [dbo].[student] ADD CONSTRAINT [StdPhoneCheck] CHECK ((len([Std_Phone])=(11)))
GO
ALTER TABLE [dbo].[student] ADD CONSTRAINT [StudentPK] PRIMARY KEY CLUSTERED ([Std_Id]) ON
[PRIMARY]
GO
ALTER TABLE [dbo].[student] ADD CONSTRAINT [IntakeStudentFK] FOREIGN KEY ([Intake_Id]) REFERENCES
[dbo].[Intake] ([Intake_Id])
GO
GRANT INSERT ON  [dbo].[student] TO [RoleManger]
GO
GRANT SELECT ON  [dbo].[student] TO [RoleManger]
GO
```

Uses

[dbo].[Intake]

Used By

[dbo].[Student_Course]
[dbo].[Student_Exam_Questions]
[dbo].[InsertExam]
[dbo].[InsertExamRandom]
[dbo].[SeeAllCoursesWithInstructorsNames]
[dbo].[SeeExamesOfSpecificCourse]
[dbo].[SeeExamesScheduleOfAllCourses]
[dbo].[SeeResultOfAllCourses]
[dbo].[SeeResultOfSpecificCourse]
[dbo].[ShowDegreeStudentsInCourse]
[dbo].[ShowStudentsCourse]

[dbo].[Student_Course]

Columns

Key	Name	Data Type	Max Length (Bytes)	Nullability	Default
PK FK C	Course_Id	int	4	NOT NULL	
PK FK C	Student_Id	int	4	NOT NULL	
	Degree	int	4	NULL allowed	
	Status	varchar(10)	10	NULL allowed	('Pass')

Indexes

Key	Name	Key Columns	Unique
PK C	StudentCoursePK	Course_Id, Student_Id	True

Foreign Keys

Name	Columns
Crs_StudentCourseFK	Course_Id->[dbo].[Course].[Crs_Id]
Std_StudentCourseFK	Student_Id->[dbo].[student].[Std_Id]

SQL Script

```
CREATE TABLE [dbo].[Student_Course]
(
[Course_Id] [int] NOT NULL,
[Student_Id] [int] NOT NULL,
[Degree] [int] NULL,
[Status] [varchar] (10) COLLATE SQL_Latin1_General_CP1_CI_AS NULL CONSTRAINT [DF__Student_C__-
Statu__5441852A] DEFAULT ('Pass')
) ON [PRIMARY]
GO
ALTER TABLE [dbo].[Student_Course] ADD CONSTRAINT [StudentCoursePK] PRIMARY KEY CLUSTERED
([Course_Id], [Student_Id]) ON [PRIMARY]
GO
ALTER TABLE [dbo].[Student_Course] ADD CONSTRAINT [Crs_StudentCourseFK] FOREIGN KEY ([Course_Id])
REFERENCES [dbo].[Course] ([Crs_Id])
GO
ALTER TABLE [dbo].[Student_Course] ADD CONSTRAINT [Std_StudentCourseFK] FOREIGN KEY ([Student_-
Id]) REFERENCES [dbo].[student] ([Std_Id])
GO
```

Uses

[dbo].[Course]
[dbo].[student]

Used By

[dbo].[InsertExam]
[dbo].[InsertExamRandom]
[dbo].[SeeAllCoursesWithInstructorsNames]
[dbo].[SeeResultOfAllCourses]
[dbo].[SeeResultOfSpecificCourse]
[dbo].[setStudentStatus]
[dbo].[ShowDegreeStudentsInCourse]
[dbo].[ShowStudentsCourse]

[dbo].[Student_Exam_Questions]

Columns

Key	Name	Data Type	Max Length (Bytes)	Nullability
PK FK C	Exam_Id	int	4	NOT NULL
PK FK C	Question_Id	int	4	NOT NULL
PK FK C	Std_Id	int	4	NOT NULL
	Answers	nvarchar(max)	max	NULL allowed
	Result	int	4	NULL allowed

Indexes

Key	Name	Key Columns	Unique
PK C	StudentExamQuestionsPK	Exam_Id, Question_Id, Std_Id	True

Foreign Keys

Name	Columns
Exam_StudentExamQuestionsFK	Exam_Id->[dbo].[Exam].[Exam_Id]
Question_StudentExamQuestionsFK	Question_Id->[dbo].[Questions].[Question_Id]
Std_StudentExamQuestionsFK	Std_Id->[dbo].[student].[Std_Id]

SQL Script

```
CREATE TABLE [dbo].[Student_Exam_Questions]
(
[Exam_Id] [int] NOT NULL,
[Question_Id] [int] NOT NULL,
[Std_Id] [int] NOT NULL,
[Answers] [nvarchar] (max) COLLATE SQL_Latin1_General_CP1_CI_AS NULL,
[Result] [int] NULL
) ON [PRIMARY]
GO
ALTER TABLE [dbo].[Student_Exam_Questions] ADD CONSTRAINT [StudentExamQuestionsPK] PRIMARY KEY
CLUSTERED ([Exam_Id], [Question_Id], [Std_Id]) ON [PRIMARY]
GO
ALTER TABLE [dbo].[Student_Exam_Questions] ADD CONSTRAINT [Exam_StudentExamQuestionsFK] FOREIGN
KEY ([Exam_Id]) REFERENCES [dbo].[Exam] ([Exam_Id])
GO
ALTER TABLE [dbo].[Student_Exam_Questions] ADD CONSTRAINT [Question_StudentExamQuestionsFK]
FOREIGN KEY ([Question_Id]) REFERENCES [dbo].[Questions] ([Question_Id])
```

```
GO
ALTER TABLE [dbo].[Student_Exam_Questions] ADD CONSTRAINT [Std_StudentExamQuestionsFK] FOREIGN
KEY ([Std_Id]) REFERENCES [dbo].[student] ([Std_Id])
GO
```

Uses

[dbo].[Exam]
[dbo].[Questions]
[dbo].[student]

Used By

[dbo].[InsertExam]
[dbo].[InsertExamRandom]
[dbo].[InsertStudentAnswers]
[dbo].[setStudentStatus]
[dbo].[ShowQuestionsExam]
[dbo].[VerifyStudentTimeOfEnteredExamAndGivePermission]

[dbo].[Track]

Columns

Key	Name	Data Type	Max Length (Bytes)	Nullability
PK C	Track_Id	int	4	NOT NULL
	Track_Name	varchar(100)	100	NOT NULL

Indexes

Key	Name	Key Columns	Unique
PK C	TrackPK	Track_Id	True

Permissions

Type	Action	Owning Principal
Grant	INSERT	RoleManger
Grant	SELECT	RoleManger
Grant	UPDATE	RoleManger

SQL Script

```
CREATE TABLE [dbo].[Track]
(
[Track_Id] [int] NOT NULL,
[Track_Name] [varchar] (100) COLLATE SQL_Latin1_General_CP1_CI_AS NOT NULL
) ON [PRIMARY]
GO
ALTER TABLE [dbo].[Track] ADD CONSTRAINT [TrackPK] PRIMARY KEY CLUSTERED ([Track_Id]) ON
[PRIMARY]
GO
GRANT INSERT ON  [dbo].[Track] TO [RoleManger]
GO
GRANT SELECT ON  [dbo].[Track] TO [RoleManger]
GO
GRANT UPDATE ON  [dbo].[Track] TO [RoleManger]
GO
```

Used By

[dbo].[Branch_Track_Intake]
[dbo].[Track_Course]

[dbo].[Track_Course]

Columns

Key	Name	Data Type	Max Length (Bytes)	Nullability
PK FK C	Crs_Id	int	4	NOT NULL
PK FK C	Track_Id	int	4	NOT NULL

Indexes

Key	Name	Key Columns	Unique
PK C	Track_CoursePK	Crs_Id, Track_Id	True

Foreign Keys

Name	Columns
Course_Track_CourseFK	Crs_Id->[dbo].[Course].[Crs_Id]
Track_Track_CourseFK	Track_Id->[dbo].[Track].[Track_Id]

SQL Script

```
CREATE TABLE [dbo].[Track_Course]
(
[Crs_Id] [int] NOT NULL,
[Track_Id] [int] NOT NULL
) ON [PRIMARY]
GO
ALTER TABLE [dbo].[Track_Course] ADD CONSTRAINT [Track_CoursePK] PRIMARY KEY CLUSTERED ([Crs_Id],
[Track_Id]) ON [PRIMARY]
GO
ALTER TABLE [dbo].[Track_Course] ADD CONSTRAINT [Course_Track_CourseFK] FOREIGN KEY ([Crs_Id])
REFERENCES [dbo].[Course] ([Crs_Id])
GO
ALTER TABLE [dbo].[Track_Course] ADD CONSTRAINT [Track_Track_CourseFK] FOREIGN KEY ([Track_Id])
REFERENCES [dbo].[Track] ([Track_Id])
GO
```

Uses

[dbo].[Course]
[dbo].[Track]

Stored Procedures

Objects

Name
dbo.InsertExam
dbo.InsertExamRandom
dbo.InsertStudentAnswers
dbo.SeeAllCoursesWithInstructorsNames
dbo.SeeExamesOfSpecificCourse
dbo.SeeExamesScheduleOfAllCourses
dbo.SeeResultOfAllCourses
dbo.SeeResultOfSpecificCourse
dbo.setStudentStatus
dbo.ShowDegreeStudentsInCourse
dbo.ShowInstructorCourseinAllIntake
dbo.ShowInstructorCourseinThisIntake
dbo.ShowQuestionsExam
dbo.ShowQuestionsInCourse
dbo.ShowStudentsCourse
dbo.VerifyStudentTimeOfEnteredExamAndGivePermission

[dbo].[InsertExam]

Parameters

Name	Data Type	Max Length (Bytes)
@ins_id	int	4
@Crs_name	nvarchar(50)	100
@Exam_Type	varchar(10)	10
@Start_Time	datetime	8
@end_Time	datetime	8
@question_id	QuestionsTableType	max

Permissions

Type	Action	Owning Principal
Grant	EXECUTE	RoleInstructor

SQL Script

```
create   proc [dbo].[InsertExam] @ins_id int,@Crs_name nvarchar(50),@Exam_Type varchar(10),
                                 @Start_Time Datetime ,@end_Time Datetime, @question_id Questions-
TableType readonly
as
begin

begin try


    declare @Intake_Id int,@Crs_id int, @IdentityExam int ,@hour int,@minute int,@Total_Time
nvarchar(30)

    --set data Intake Id and Course id


    select @Intake_Id=max([Intake_Id]) from [dbo].[Intake]
    select @Crs_id=[Crs_Id] from [dbo].[Course] where [Crs_Name] = @Crs_name

    -- check Instructor id and course Name and Intake Year are EXIST or not

    IF EXISTS(SELECT * FROM [dbo].[Course] WHERE Course.Crs_Name = @Crs_name and [Ins_Id]=
@ins_id
                and[Ins_Id] in (SELECT [Ins_Id] FROM [dbo].[Intake_Instructor] WHERE [Intake_Id]
= @Intake_Id)
                and [Crs_Id] in (SELECT [Course_Id] FROM [dbo].[Student_Course] WHERE [Student_-
Id] in
                    (SELECT [Std_Id] FROM [dbo].[student] WHERE [Intake_Id]=@Intake_Id)
```

```
                )
            )

    BEGIN

        --Calculate Total Time
        set @hour = DATEDIFF(hour,@Start_Time,@end_Time)
        set @minute = DATEDIFF(minute,@Start_Time,@end_Time)-(@hour*60)
        set @Total_Time = CONCAT(@hour,' hour, ',@minute,' minute')

        --Calculate Identity for table Exam

        select @IdentityExam= max([Exam_Id]) from [dbo].[Exam]
        if(@IdentityExam IS NULL)
        BEGIN
            set @IdentityExam = 0
        end

        insert into [dbo].[Exam]
        values (@IdentityExam+1,@Exam_Type,@Start_Time,@end_Time,@Total_Time,@Intake_-
Id,@Crs_id,@ins_id)



        DECLARE @Id_Q  int , @Id_S int

        -- insert questions in exam

        DECLARE QuestionsInExam CURSOR

        -- select questions base on table for Questions Id

        FOR SELECT ids FROM @question_id
        OPEN QuestionsInExam
        FETCH NEXT FROM QuestionsInExam INTO @Id_Q
        WHILE @@FETCH_STATUS = 0
            BEGIN
            -- check number of questions in course and mix degree in course
            IF EXISTS(SELECT * FROM [dbo].[Questions] WHERE [Question_Id] = @Id_Q )
            BEGIN

                if(lower(@Exam_Type) = 'corrective')
                begin

                DECLARE StudentsInExam CURSOR
                FOR SELECT [Std_Id]
                FROM [dbo].[student]
                where [Intake_Id]=@Intake_Id
                and [Std_Id] in (
                    SELECT [Student_Id]
                    FROM [dbo].[Student_Course]
                    where [Course_Id]=@Crs_id
                    and [Status] = 'Corrective')
```

```
                end
                else
                begin

                DECLARE StudentsInExam CURSOR
                FOR SELECT [Std_Id]
                FROM [dbo].[student]
                where [Intake_Id]=@Intake_Id
                and [Std_Id] in (
                    SELECT [Student_Id]
                    FROM [dbo].[Student_Course]
                    where [Course_Id]=@Crs_id)


                end


                OPEN StudentsInExam
                FETCH NEXT FROM StudentsInExam INTO @Id_S
                WHILE @@FETCH_STATUS = 0
                    BEGIN
                    -- check Question Id and Exam Id and Student Id EXISTS or not
                    if not EXISTS(SELECT * FROM [dbo].[Student_Exam_Questions]
                    WHERE [Question_Id] = @Id_Q and [Exam_Id]=@IdentityExam+1 and [Std_Id] =
@Id_S)
                    BEGIN
                        -- insert Questions in exam
                        insert into [dbo].[Student_Exam_Questions]
                        values (@IdentityExam+1,@Id_Q,@Id_S,null,null)
                        FETCH NEXT FROM StudentsInExam INTO @Id_S
                    END
                    END;
                CLOSE StudentsInExam
                DEALLOCATE StudentsInExam
            end
            FETCH NEXT FROM QuestionsInExam INTO @Id_Q
            END;
        CLOSE QuestionsInExam
        DEALLOCATE QuestionsInExam

        select @IdentityExam + 1 [Exam ID]
    end

    else
    begin
        --massage error
        RAISERROR ('Data are not exist, Please enter correct data' ,10,1)
    end
    end try
    begin catch
        --massage error
        RAISERROR ('Data are not exist, Please enter correct data' ,10,1)
```



```
    end catch
end
GO
GRANT EXECUTE ON  [dbo].[InsertExam] TO [RoleInstructor]
GO
```

Uses

[dbo].[Course]
[dbo].[Exam]
[dbo].[Intake]
[dbo].[Intake_Instructor]
[dbo].[Questions]
[dbo].[student]
[dbo].[Student_Course]
[dbo].[Student_Exam_Questions]
[dbo].[QuestionsTableType]

[dbo].[InsertExamRandom]

Parameters

Name	Data Type	Max Length (Bytes)
@ins_id	int	4
@Crs_name	nvarchar(50)	100
@Exam_Type	varchar(10)	10
@Start_Time	datetime	8
@end_Time	datetime	8
@NumberOfQ	int	4

Permissions

Type	Action	Owning Principal
Grant	EXECUTE	RoleInstructor

SQL Script

```
--Insert Exam base on Instructor id and course Name and Number of Qusestions (Random)

create   proc [dbo].[InsertExamRandom] @ins_id int,@Crs_name nvarchar(50) ,
                                @Exam_Type varchar(10),@Start_Time Datetime ,@end_Time
Datetime,@NumberOfQ int
as
begin

begin try

    declare @Intake_Id int,@Crs_id int, @IdentityExam int ,@hour int,@minute int,@Total_Time
nvarchar(30)

    --set data Intake Id and Course id

    select @Intake_Id=max([Intake_Id]) from [dbo].[Intake]
    select @Crs_id=[Crs_Id] from [dbo].[Course] where [Crs_Name] = @Crs_name

    -- check Instructor id and course Name and Intake Year are EXIST or not

    IF EXISTS(SELECT * FROM [dbo].[Course] WHERE Course.Crs_Name = @Crs_name and [Ins_Id]=
@ins_id
                and[Ins_Id] in (SELECT [Ins_Id] FROM [dbo].[Intake_Instructor] WHERE [Intake_Id]
= @Intake_Id)
                and [Crs_Id] in (SELECT [Course_Id] FROM [dbo].[Student_Course] WHERE [Student_-
Id] in
                    (SELECT [Std_Id] FROM [dbo].[student] WHERE [Intake_Id]=@Intake_Id)
```

```
                )
            )
    BEGIN

        --Calculate Total Time

        set @hour = DATEDIFF(hour,@Start_Time,@end_Time)
        set @minute = DATEDIFF(minute,@Start_Time,@end_Time)-(@hour*60)
        set @Total_Time = CONCAT(@hour,' hour, ',@minute,' minute')

        --Calculate Identity for table Exam

        select @IdentityExam= max([Exam_Id]) from [dbo].[Exam]
        if(@IdentityExam IS NULL)
        BEGIN
            set @IdentityExam = 0
        end

        insert into [dbo].[Exam]
        values (@IdentityExam+1,@Exam_Type,@Start_Time,@end_Time,@Total_Time,@Intake_-
Id,@Crs_id,@ins_id)


        DECLARE @Id_Q  int , @Id_S int


        -- insert questions in exam

        DECLARE QuestionsInExam CURSOR

        -- select questions random base on Number Of questions

        FOR SELECT TOP (@NumberOfQ) [Question_Id] FROM [dbo].[Questions]
        where [Crs_Id] in (SELECT [Crs_Id] FROM [dbo].[Course] where [Crs_Id] = @Crs_id)
        ORDER BY NEWID();


        OPEN QuestionsInExam
        FETCH NEXT FROM QuestionsInExam INTO @Id_Q
        WHILE @@FETCH_STATUS = 0
        BEGIN

                if(lower(@Exam_Type) = 'corrective')
                begin

                DECLARE StudentsInExam CURSOR
                FOR SELECT [Std_Id]
                FROM [dbo].[student]
                where [Intake_Id]=@Intake_Id
                and [Std_Id] in (
                    SELECT [Student_Id]
                    FROM [dbo].[Student_Course]
                    where [Course_Id]=@Crs_id
```



```
                    and [Status] = 'Corrective')

                end
                else
                begin

                DECLARE StudentsInExam CURSOR
                FOR SELECT [Std_Id]
                FROM [dbo].[student]
                where [Intake_Id]=@Intake_Id
                and [Std_Id] in (
                    SELECT [Student_Id]
                    FROM [dbo].[Student_Course]
                    where [Course_Id]=@Crs_id)


                end


                OPEN StudentsInExam
                FETCH NEXT FROM StudentsInExam INTO @Id_S
                WHILE @@FETCH_STATUS = 0
                    BEGIN

                        insert into [dbo].[Student_Exam_Questions]
                        values (@IdentityExam+1,@Id_Q,@Id_S,null,null)
                        FETCH NEXT FROM StudentsInExam INTO @Id_S

                    END;
                CLOSE StudentsInExam
                DEALLOCATE StudentsInExam

            FETCH NEXT FROM QuestionsInExam INTO @Id_Q
        END;
        CLOSE QuestionsInExam
        DEALLOCATE QuestionsInExam

        select @IdentityExam + 1 [Exam ID]
    end

    else
    begin
        --massage error
        RAISERROR ('Data are not exist, Please enter correct data' ,10,1)
    end
    end try
    begin catch
        --massage error
        RAISERROR ('Data are not exist, Please enter correct data' ,10,1)
    end catch

end
GO
```



```
GRANT EXECUTE ON  [dbo].[InsertExamRandom] TO [RoleInstructor]
GO
```

Uses

[dbo].[Course]
[dbo].[Exam]
[dbo].[Intake]
[dbo].[Intake_Instructor]
[dbo].[Questions]
[dbo].[student]
[dbo].[Student_Course]
[dbo].[Student_Exam_Questions]

[dbo].[InsertStudentAnswers]

Parameters

Name	Data Type	Max Length (Bytes)
@e_id	int	4
@s_id	int	4
@Q_id	int	4
@answer	nvarchar(max)	max

Permissions

Type	Action	Owning Principal
Grant	EXECUTE	RoleStudent

SQL Script

```
CREATE   PROC [dbo].[InsertStudentAnswers]
    @e_id INT,
    @s_id INT,
    @Q_id INT,
    @answer NVARCHAR(MAX)
AS
BEGIN
    -- Check if the student has access to the specified question in the given exam
    IF EXISTS (
        SELECT *
        FROM [dbo].[Student_Exam_Questions]
        WHERE [Std_Id] = @s_id
        AND [Exam_Id] = @e_id
        AND [Question_Id] = @Q_id
    )
    BEGIN
        -- Declare variables for current time, exam start time, and exam end time
        DECLARE @C_T DATETIME = GETDATE()
        DECLARE @S_T DATETIME
        DECLARE @E_T DATETIME

        -- Fetch exam start and end times
        SELECT @S_T = [Start_Time], @E_T = [End_Time]
        FROM [dbo].[Exam]
        WHERE Exam_Id = @e_id

        -- Check if the current time is within the exam period
        IF @C_T BETWEEN @S_T AND @E_T
```

```
        BEGIN
            -- Update student's answer for the specified question in the exam
            UPDATE Student_Exam_Questions
            SET Answers = @answer
            WHERE [Std_Id] = @s_id
            AND [Exam_Id] = @e_id
            AND [Question_Id] = @Q_id
        END
        ELSE IF @C_T < @S_T
        BEGIN
            -- Exam has not started yet
            RAISERROR('Exam has not started yet', 10, 1)
        END
        ELSE IF @C_T > @E_T
        BEGIN
            -- Exam time is over
            RAISERROR('Exam time is over', 10, 1)
        END
    END
    ELSE
    BEGIN
        -- The specified question in the exam does not exist
        RAISERROR('Exam or question does not exist, please enter correct data', 10, 1)
    END
END
GO
GRANT EXECUTE ON  [dbo].[InsertStudentAnswers] TO [RoleStudent]
GO
```

Uses

[dbo].[Exam]
[dbo].[Student_Exam_Questions]

[dbo].[SeeAllCoursesWithInstructorsNames]

Parameters

Name	Data Type	Max Length (Bytes)
@St_id	int	4

Permissions

Type	Action	Owning Principal
Grant	EXECUTE	RoleStudent

SQL Script

```
CREATE   PROC [dbo].[SeeAllCoursesWithInstructorsNames]
    @St_id INT
AS
BEGIN
    -- Check if the provided Student Id exists
    IF EXISTS (SELECT * FROM [dbo].[student] WHERE [Std_Id] = @St_id)
    BEGIN
        -- Retrieve student information along with course and instructor name
        BEGIN
            SELECT
                CONCAT(S.Std_Fname , ' ' , S.Std_Lname) AS [Student Name],
                [Crs_Name] AS [Course Name],
                [Ins_Name] AS [Instructor Name]
            FROM
                [dbo].[student] S
                JOIN [dbo].[Student_Course] SC ON S.Std_Id = SC.Student_Id
                JOIN [dbo].[Course] C ON SC.Course_Id = C.Crs_Id
                JOIN [dbo].[Instructor] I ON C.Ins_Id = I.Ins_Id
            WHERE
                S.Std_Id = @St_id;
        END
    END
    ELSE
    BEGIN
        -- Student Id does not exist
        RAISERROR ('Student Id does not exist', 10, 1)
    END
END
GO
GRANT EXECUTE ON  [dbo].[SeeAllCoursesWithInstructorsNames] TO [RoleStudent]
GO
```

Uses

[dbo].[Course]

[dbo].[Instructor]

[dbo].[student]

[dbo].[Student_Course]

[dbo].[SeeExamesOfSpecificCourse]

Parameters

Name	Data Type	Max Length (Bytes)
@St_Id	int	4
@Crs_name	nvarchar(50)	100

Permissions

Type	Action	Owning Principal
Grant	EXECUTE	RoleStudent

SQL Script

```
CREATE   PROC [dbo].[SeeExamesOfSpecificCourse]
    @St_Id INT,
    @Crs_name NVARCHAR(50)
AS
BEGIN
    -- Check if student and course exist
    IF EXISTS(SELECT * FROM [dbo].[student] WHERE Std_Id = @St_Id)
       AND EXISTS(SELECT * FROM [dbo].[Course] WHERE Crs_Name = @Crs_name)
    BEGIN
        -- Declare and initialize variables
        DECLARE @Intake_Id INT
        SELECT @Intake_Id = [Intake_Id] FROM [dbo].[student] WHERE Std_Id = @St_Id

        -- Check if Intake_Id is 3
        IF @Intake_Id = 3
        BEGIN
            -- Fetch exam details
            SELECT
                CONCAT(S.Std_Fname, ' ', S.Std_Lname) AS [Student Name],
                I.Intake_Name AS [Intake Year],
                C.Crs_Name AS [Course Name],
                E.Exam_Id AS [Exam ID],
                E.E_Type AS [Exam Type],
                E.Start_Time AS [Start Time],
                E.End_Time AS [End Time],
                E.Total_time AS [Total Time]
            FROM
                [dbo].[student] S
                JOIN [dbo].[Intake] I ON I.Intake_Id = S.[Intake_Id]
```

```
                JOIN [dbo].[Exam] E ON E.[Intake_Id] = I.[Intake_Id]
                JOIN [dbo].[Course] C ON C.Crs_Id = E.Crs_Id
            WHERE
                S.Std_Id = @St_Id AND C.Crs_Name = @Crs_name
        END

        ELSE
        BEGIN
            -- Access denied if Intake_Id is not 3
            RAISERROR('Access denied, you do not have permission', 10, 1)
        END
    END

    ELSE IF NOT EXISTS(SELECT * FROM [dbo].[student] WHERE Std_Id = @St_Id)
        AND NOT EXISTS(SELECT * FROM [dbo].[Course] WHERE Crs_Name = @Crs_name)
    BEGIN
        -- Both student and course do not exist
        RAISERROR('Student id and Course id do not exist. Please enter correct data', 10, 1)
    END

    ELSE IF NOT EXISTS(SELECT * FROM [dbo].[student] WHERE Std_Id = @St_Id)
    BEGIN
        -- Student does not exist
        RAISERROR('Student Id does not exist', 10, 1)
    END

    ELSE IF NOT EXISTS(SELECT * FROM [dbo].[Course] WHERE Crs_Name = @Crs_name)
    BEGIN
        -- Course does not exist
        RAISERROR('Course name does not exist', 10, 1)
    END
END
GO
GRANT EXECUTE ON  [dbo].[SeeExamesOfSpecificCourse] TO [RoleStudent]
GO
```

Uses

[dbo].[Course]
[dbo].[Exam]
[dbo].[Intake]
[dbo].[student]

[dbo].[SeeExamesScheduleOfAllCourses]

Parameters

Name	Data Type	Max Length (Bytes)
@St_Id	int	4

Permissions

Type	Action	Owning Principal
Grant	EXECUTE	RoleStudent

SQL Script

```
CREATE   PROCEDURE [dbo].[SeeExamesScheduleOfAllCourses]
    @St_Id INT
AS
BEGIN
    -- Check if the student exists
    IF EXISTS (SELECT * FROM [dbo].[student] WHERE Std_Id = @St_Id)
    BEGIN
        -- Declare and initialize variables
        DECLARE @Intake_Id INT
        SELECT @Intake_Id = [Intake_Id] FROM [dbo].[student] WHERE Std_Id = @St_Id

        -- Check if the student is in Intake 3
        IF @Intake_Id = 3
        BEGIN
            -- Fetch exam details for all courses
            SELECT
                CONCAT(S.Std_Fname, ' ', S.Std_Lname) AS [Student Name],
                I.Intake_Name AS [Intake Year],
                C.Crs_Name AS [Course Name],
                E.Exam_Id AS [Exam ID],
                E.E_Type AS [Exam Type],
                E.Start_Time AS [Start Time],
                E.End_Time AS [End Time],
                E.Total_time AS [Total Time]
            FROM
                [dbo].[student] S
                JOIN [dbo].[Intake] I ON I.Intake_Id = S.[Intake_Id]
                JOIN [dbo].[Exam] E ON E.[Intake_Id] = I.[Intake_Id]
                JOIN [dbo].[Course] C ON C.Crs_Id = E.Crs_Id
            WHERE
```

```
                S.Std_Id = @St_Id;
        END
        ELSE
        BEGIN
            -- Access denied if the student is not in Intake 3
            RAISERROR('Access denied, you do not have permission', 10, 1)
        END
    END
    ELSE
    BEGIN
        -- Error if the student does not exist
        RAISERROR('Student Id does not exist', 10, 1)
    END
END
GO
GRANT EXECUTE ON  [dbo].[SeeExamesScheduleOfAllCourses] TO [RoleStudent]
GO
```

Uses

[dbo].[Course]
[dbo].[Exam]
[dbo].[Intake]
[dbo].[student]

[dbo].[SeeResultOfAllCourses]

Parameters

Name	Data Type	Max Length (Bytes)
@St_id	int	4

Permissions

Type	Action	Owning Principal
Grant	EXECUTE	RoleStudent

SQL Script

```
CREATE   PROC [dbo].[SeeResultOfAllCourses]
    @St_id INT
AS
BEGIN
    -- Check if the provided Student Id exists
    IF EXISTS (SELECT * FROM [dbo].[student] WHERE [Std_Id] = @St_id)
    BEGIN
        -- Retrieve student information along with course results
        BEGIN
            SELECT
                CONCAT(S.Std_Fname , ' ' , S.Std_Lname) AS [Student name],
                [Crs_Name],
                [Status] = CASE WHEN [Status] IS NULL THEN 'The result has not yet appeared' ELSE
[Status] END
            FROM
                [dbo].[student] S
                JOIN [dbo].[Student_Course] SC ON S.Std_Id = SC.Student_Id
                JOIN [dbo].[Course] C ON SC.Course_Id = C.Crs_Id
            WHERE
                S.Std_Id = @St_id;
        END
    END
    ELSE
    BEGIN
        -- Student Id does not exist
        RAISERROR ('Student Id does not exist', 10, 1)
    END
END
GO
GRANT EXECUTE ON  [dbo].[SeeResultOfAllCourses] TO [RoleStudent]
GO
```

Uses

[dbo].[Course]

[dbo].[student]

[dbo].[Student_Course]

[dbo].[SeeResultOfSpecificCourse]

Parameters

Name	Data Type	Max Length (Bytes)
@St_id	int	4
@Course_Name	nvarchar(50)	100

Permissions

Type	Action	Owning Principal
Grant	EXECUTE	RoleStudent

SQL Script

```
CREATE   PROC [dbo].[SeeResultOfSpecificCourse]
    @St_id INT,
    @Course_Name NVARCHAR(50)
AS
BEGIN
    -- Check if the student and course exist
    IF EXISTS (SELECT * FROM [dbo].[student] WHERE [Std_Id] = @St_id)
        AND EXISTS (SELECT * FROM [dbo].[Course] WHERE [Crs_Name] = @Course_Name)
    BEGIN
        -- Retrieve student details and course result
        BEGIN
            SELECT
                CONCAT(S.Std_Fname, ' ', S.Std_Lname) AS [Student name],
                [Crs_Name],
                [Status] = CASE WHEN [Status] IS NULL THEN 'The result has not yet appeared' ELSE
[Status] END
            FROM
                [dbo].[student] S
                JOIN [dbo].[Student_Course] SC ON S.Std_Id = SC.Student_Id
                JOIN [dbo].[Course] C ON SC.Course_Id = C.Crs_Id
            WHERE
                S.Std_Id = @St_id AND C.Crs_Name = @Course_Name;
        END
    END

    ELSE IF NOT EXISTS (SELECT * FROM [dbo].[student] WHERE [Std_Id] = @St_id)
        AND NOT EXISTS (SELECT * FROM [dbo].[Course] WHERE [Crs_Name] = @Course_Name)
    BEGIN
        -- Raise an error if the student or course does not exist
        RAISERROR('Student id and course name are not exist , enter correct data', 10, 1);
```

```
    END

    ELSE IF NOT EXISTS (SELECT * FROM [dbo].[student] WHERE [Std_Id] = @St_id)
    BEGIN
        -- Raise an error if the student or course does not exist
        RAISERROR('Student id does not exist , enter correct data', 10, 1);
    END

    ELSE IF NOT EXISTS (SELECT * FROM [dbo].[Course] WHERE [Crs_Name] = @Course_Name)
    BEGIN
        -- Raise an error if the student or course does not exist
        RAISERROR('Course name does not exist , enter correct data', 10, 1);
    END
END
GO
GRANT EXECUTE ON  [dbo].[SeeResultOfSpecificCourse] TO [RoleStudent]
GO
```

Uses

[dbo].[Course]
[dbo].[student]
[dbo].[Student_Course]

[dbo].[setStudentStatus]

Parameters

Name	Data Type	Max Length (Bytes)
@e_id	int	4
@Course_Name	nvarchar(50)	100

Permissions

Type	Action	Owning Principal
Grant	EXECUTE	RoleInstructor

SQL Script

```
create   proc [dbo].[setStudentStatus] @e_id int, @Course_Name nvarchar(50)
as
begin

    --check if Student Id and Exam Id and Course Name

    if exists(select * from [dbo].[Course]
                where [Crs_Name] = @Course_Name
                and [Crs_Id] in (select [Crs_Id] from [dbo].[Exam] where [Exam_Id] = @e_id))

    begin

        declare @countExamQuesions int

        select @countExamQuesions = count ([Question_Id])
        from [dbo].[Student_Exam_Questions]
        where [Exam_Id] = 30
        and [Std_Id] = (
            SELECT min([Std_Id]) FROM [dbo].[Student_Exam_Questions]
            WHERE [Exam_Id] = 30
        )

        -- update Status

        declare @C_T datetime , @E_T datetime
        set @C_T = GETDATE()
        set @E_T = (Select [End_Time] from [dbo].[Exam] where Exam_Id = @e_id)


        if (@C_T>@E_T)
        begin
```

```
            declare @Id_S int

            DECLARE StudentsInExam CURSOR
            FOR SELECT [Std_Id]
            FROM [dbo].[Student_Exam_Questions]
            where [Exam_Id] = @e_id
            and [Question_Id] = (
                SELECT min([Question_Id]) FROM [dbo].[Student_Exam_Questions]
                WHERE [Exam_Id] = @e_id
            )

            OPEN StudentsInExam
            FETCH NEXT FROM StudentsInExam INTO @Id_S
            WHILE @@FETCH_STATUS = 0
            BEGIN

                update [dbo].[Student_Course]
                set [Status] = case when [Degree]>=(@countExamQuesions*0.5) then 'Pass' else
'Corrective' end
                where [Student_Id] = @Id_S
                and [Course_Id] in (select [Crs_Id] from [dbo].[Course] where [Crs_Name] =
@Course_Name)

                update [dbo].[Student_Course]
                set [Degree] = 0
                where [Student_Id] = @Id_S
                and [Course_Id] in (select [Crs_Id] from [dbo].[Course] where [Crs_Name] =
@Course_Name)


                FETCH NEXT FROM StudentsInExam INTO @Id_S
            END;
            CLOSE StudentsInExam
            DEALLOCATE StudentsInExam


        end
        else
        begin
            RAISERROR ('The exam has not ended yet.' ,10,1)
        end

    end

    else
    begin
        RAISERROR ('Data is not exist, Please enter correct data' ,10,1)
    end

end
GO
GRANT EXECUTE ON  [dbo].[setStudentStatus] TO [RoleInstructor]
```



```
GO
```

Uses

[dbo].[Course]
[dbo].[Exam]
[dbo].[Student_Course]
[dbo].[Student_Exam_Questions]

[dbo].[ShowDegreeStudentsInCourse]

Parameters

Name	Data Type	Max Length (Bytes)
@ins_id	int	4
@Course_Name	nvarchar(50)	100

Permissions

Type	Action	Owning Principal
Grant	EXECUTE	RoleInstructor

SQL Script

```
create   proc [dbo].[ShowDegreeStudentsInCourse] @ins_id int,@Course_Name nvarchar(50)
as
begin
    -- check Instructor Id and Course Name EXIST or not

    IF EXISTS(SELECT * FROM [dbo].[Course]
       WHERE [Crs_Name] = @Course_Name
       and[Ins_Id]=@ins_id )
    BEGIN
        begin
        declare @Intake_Id int
        select @Intake_Id=max([Intake_Id]) from [dbo].[Intake]


            -- select Instructor and Course in last Intake

            select
                [Ins_Name] as [Instructor Name],
                C.Crs_Name as [Course Name],
                CONCAT(S.Std_Fname , ' ' , S.Std_Lname) as [Student name],
                [Degree],
                IT.[Intake_Name]
            from [dbo].[student] S
            join [dbo].[Student_Course] SC on SC.Student_Id = S.Std_Id
            join [dbo].[Course] C on C.Crs_Id = SC.Course_Id
            join [dbo].[Instructor] I on I.Ins_Id = C.Ins_Id
            join [dbo].[Intake] IT on IT.Intake_Id = S.Intake_Id

            where I.Ins_Id = @ins_id and S.Intake_Id = @Intake_Id  and C.Crs_Name = @Course_Name
```

```
        end
    end
    else
    begin

        --Error massage

        RAISERROR ('Instructor Id and Course Name or does not exist' ,10,1)
    end
end
GO
GRANT EXECUTE ON  [dbo].[ShowDegreeStudentsInCourse] TO [RoleInstructor]
GO
```

Uses

[dbo].[Course]
[dbo].[Instructor]
[dbo].[Intake]
[dbo].[student]
[dbo].[Student_Course]

[dbo].[ShowInstructorCourseinAllIntake]

Parameters

Name	Data Type	Max Length (Bytes)
@ins_id	int	4

Permissions

Type	Action	Owning Principal
Grant	EXECUTE	RoleInstructor

SQL Script

```
create   proc [dbo].[ShowInstructorCourseinAllIntake] @ins_id int
as
begin
    -- check Instructor Id EXIST or not

    IF EXISTS(SELECT * FROM [dbo].[Instructor] WHERE [Ins_Id] = @ins_id )
    BEGIN
        begin

            -- select Instructor and Course in All Intake

            select [Ins_Name] , [Crs_Name] , [Intake_Name]
            from [dbo].[Instructor] I
            join [dbo].[Course] C on I.Ins_Id =C.Ins_Id
            join [dbo].[Intake_Instructor] II on I.Ins_Id =II.Ins_Id
            join [dbo].[Intake] IT on IT.Intake_Id = II.Intake_Id
            where I.Ins_Id = @ins_id
        end
    end
    else
    begin

        --Error massage

        RAISERROR ('Instructor Id does not exist' ,10,1)
    end
end
GO
GRANT EXECUTE ON  [dbo].[ShowInstructorCourseinAllIntake] TO [RoleInstructor]
GO
```

Uses

[dbo].[Course]

[dbo].[Instructor]

[dbo].[Intake]

[dbo].[Intake_Instructor]

[dbo].[ShowInstructorCourseinThisIntake]

Parameters

Name	Data Type	Max Length (Bytes)
@ins_id	int	4
@Intake_Year	varchar(20)	20

Permissions

Type	Action	Owning Principal
Grant	EXECUTE	RoleInstructor

SQL Script

```
create   proc [dbo].[ShowInstructorCourseinThisIntake] @ins_id int,@Intake_Year varchar(20)
as
begin
    -- check Instructor Id EXIST or not

    IF EXISTS(SELECT * FROM [dbo].[Instructor] WHERE [Ins_Id] = @ins_id )
       and EXISTS(SELECT * FROM [dbo].[Intake] WHERE [Intake_Name] = @Intake_Year )
    BEGIN
        begin

            -- select Instructor and Course in All Intake

            select [Ins_Name] , [Crs_Name] , [Intake_Name]
            from [dbo].[Instructor] I
            join [dbo].[Course] C on I.Ins_Id =C.Ins_Id
            join [dbo].[Intake_Instructor] II on I.Ins_Id =II.Ins_Id
            join [dbo].[Intake] IT on IT.Intake_Id = II.Intake_Id
            where I.Ins_Id = @ins_id and IT.[Intake_Name] = @Intake_Year
        end
    end
    else
    begin

        --Error massage

        RAISERROR ('Instructor Id or Intake Year does not exist' ,10,1)
    end
end
GO
GRANT EXECUTE ON  [dbo].[ShowInstructorCourseinThisIntake] TO [RoleInstructor]
```

```
GO
```

Uses

[dbo].[Course]

[dbo].[Instructor]

[dbo].[Intake]

[dbo].[Intake_Instructor]

[dbo].[ShowQuestionsExam]

Parameters

Name	Data Type	Max Length (Bytes)
@Exam_Id	int	4

Permissions

Type	Action	Owning Principal
Grant	EXECUTE	RoleInstructor

SQL Script

```
create   proc [dbo].[ShowQuestionsExam] @Exam_Id int
as
begin
    -- check Exam Id EXIST or not

    IF EXISTS(SELECT * FROM [dbo].[Exam] WHERE [Exam_Id] = @Exam_Id)

    BEGIN
        begin

            -- select Data Questions base on Exam

            SELECT [Question_Id], [Q_Type], [Question_Text],[Correct_Answer]
                FROM [dbo].[Questions] Q
                WHERE Q.[Question_Id] IN (
                    SELECT SEQ.[Question_Id]
                    FROM [dbo].[Student_Exam_Questions] SEQ
                    where SEQ.Exam_Id in (
                        SELECT E.[Exam_Id]
                        FROM [dbo].[Exam] E
                        WHERE E.[Exam_Id] = @Exam_Id
                    )
                );
        end
    end
    else
    begin

        --Error massage

        RAISERROR ('Exam Id does not exist' ,10,1)
```

```
    end
end
GO
GRANT EXECUTE ON  [dbo].[ShowQuestionsExam] TO [RoleInstructor]
GO
```

Uses

[dbo].[Exam]
[dbo].[Questions]
[dbo].[Student_Exam_Questions]

[dbo].[ShowQuestionsInCourse]

Parameters

Name	Data Type	Max Length (Bytes)
@Crs_name	nvarchar(50)	100

Permissions

Type	Action	Owning Principal
Grant	EXECUTE	RoleInstructor

SQL Script

```
create   proc [dbo].[ShowQuestionsInCourse] @Crs_name nvarchar(50)
as
begin
    -- check course Name EXIST or not

    IF EXISTS(SELECT * FROM [dbo].[Course] WHERE Course.Crs_Name = @Crs_name )
    BEGIN
        begin

            -- select Data Questions base on Course

            select Q.[Question_Id] , [Q_Type] , [Question_Text],[Correct_Answer]
            from [dbo].[Questions] Q
            where Q.Crs_Id =(select [Crs_Id] from [dbo].[Course] C where [Crs_Name] = @Crs_name)
        end
    end
    else
    begin

        --Error massage

        RAISERROR ('Course name does not exist' ,10,1)
    end
end
GO
GRANT EXECUTE ON  [dbo].[ShowQuestionsInCourse] TO [RoleInstructor]
GO
```

Uses

[dbo].[Course]

[dbo].[Questions]

[dbo].[ShowStudentsCourse]

Parameters

Name	Data Type	Max Length (Bytes)
@ins_id	int	4
@Course_Name	nvarchar(50)	100

Permissions

Type	Action	Owning Principal
Grant	EXECUTE	RoleInstructor

SQL Script

```
create   proc [dbo].[ShowStudentsCourse] @ins_id int,@Course_Name nvarchar(50)
as
begin
    -- check Instructor Id and Intake Year EXIST or not

    IF EXISTS(SELECT * FROM [dbo].[Course]
       WHERE [Crs_Name] = @Course_Name
       and[Ins_Id]=@ins_id )
    BEGIN
        begin
        declare @Intake_Id int
        select @Intake_Id=max([Intake_Id]) from [dbo].[Intake]
            -- select Instructor and Course in this Intake

            select
                [Ins_Name] as [Instructor Name],
                C.Crs_Name as [Course Name],
                CONCAT(S.Std_Fname , ' ' , S.Std_Lname) as [Student name],
                [Status] = case when [Status] is null then 'Not taken yet.' else  [Status] end,
                IT.[Intake_Name]
            from [dbo].[student] S
            join [dbo].[Student_Course] SC on SC.Student_Id = S.Std_Id
            join [dbo].[Course] C on C.Crs_Id = SC.Course_Id
            join [dbo].[Instructor] I on I.Ins_Id = C.Ins_Id
            join [dbo].[Intake] IT on IT.Intake_Id = S.Intake_Id

            where I.Ins_Id = @ins_id and S.Intake_Id = @Intake_Id  and C.Crs_Name = @Course_Name

        end
    end
    else
```

```
    begin

        --Error massage

        RAISERROR ('Instructor Id and Course Name or does not exist' ,10,1)
    end
end
GO
GRANT EXECUTE ON  [dbo].[ShowStudentsCourse] TO [RoleInstructor]
GO
```

Uses

[dbo].[Course]
[dbo].[Instructor]
[dbo].[Intake]
[dbo].[student]
[dbo].[Student_Course]

[dbo].[VerifyStudentTimeOfEnteredExamAndGivePermission]

Parameters

Name	Data Type	Max Length (Bytes)
@Exam_Id	int	4

Permissions

Type	Action	Owning Principal
Grant	EXECUTE	RoleStudent

SQL Script

```
CREATE   PROC [dbo].[VerifyStudentTimeOfEnteredExamAndGivePermission]
    @Exam_Id INT
AS
BEGIN
    -- Check if the provided Exam_Id exists
    IF EXISTS (SELECT 1 FROM [dbo].[Exam] WHERE Exam_Id = @Exam_Id)
    BEGIN
        -- Declare variables
        DECLARE @C_T DATETIME = GETDATE()
        DECLARE @S_T DATETIME
        DECLARE @E_T DATETIME

        -- Fetch exam start and end times
        SELECT @S_T = [Start_Time], @E_T = [End_Time]
        FROM [dbo].[Exam]
        WHERE Exam_Id = @Exam_Id

        -- Check if the current time is within the exam period
        IF @C_T BETWEEN @S_T AND @E_T
        BEGIN
            -- Fetch questions for the exam
            SELECT Q.[Question_Id], [Q_Type], [Question_Text]
            FROM [dbo].[Questions] Q
            WHERE Q.[Question_Id] IN (
                SELECT SEQ.[Question_Id]
                FROM [dbo].[Student_Exam_Questions] SEQ
                WHERE SEQ.Exam_Id = @Exam_Id
            );
        END
        ELSE IF @C_T < @S_T
        BEGIN
```

```
            -- Exam has not started yet
            RAISERROR('Exam has not started yet', 10, 1)
        END
        ELSE IF @C_T > @E_T
        BEGIN
            -- Exam time is over
            RAISERROR('Exam time is over', 10, 1)
        END
    END
    ELSE
    BEGIN
        -- Exam does not exist
        RAISERROR('Exam does not exist', 10, 1);
    END
END
GO
GRANT EXECUTE ON  [dbo].[VerifyStudentTimeOfEnteredExamAndGivePermission] TO [RoleStudent]
GO
```

Uses

[dbo].[Exam]
[dbo].[Questions]
[dbo].[Student_Exam_Questions]

User-Defined Table Types

Objects

Name
dbo.QuestionsTableType

[dbo].[QuestionsTableType]

Columns

Name	Data Type	Max Length (Bytes)	Nullability
ids	int	4	NULL allowed

Permissions

Type	Action	Owning Principal
Grant	EXECUTE	RoleInstructor

SQL Script

```
CREATE TYPE [dbo].[QuestionsTableType] AS TABLE
(
[ids] [int] NULL
)
GO
GRANT EXECUTE ON TYPE:: [dbo].[QuestionsTableType] TO [RoleInstructor]
GO
```

Used By

[dbo].[InsertExam]

Users

Objects

Name
Admin
Instructor
Manger
Student

Admin

Database Level Permissions

Type	Action
CONNECT	Grant

SQL Script

```
CREATE USER [Admin] FOR LOGIN [Admin]
GO
```

Used By

RoleAdmin

Instructor

Database Level Permissions

Type	Action
CONNECT	Grant

SQL Script

```
CREATE USER [Instructor] FOR LOGIN [Instructor]
GO
```

Used By

RoleInstructor

Manger

Database Level Permissions

Type	Action
CONNECT	Grant

SQL Script

```
CREATE USER [Manger] FOR LOGIN [Manger]
GO
```

Used By

RoleManger

Student

Database Level Permissions

Type	Action
CONNECT	Grant

SQL Script

```
CREATE USER [Student] FOR LOGIN [Student]
GO
```

Used By

RoleStudent

Database Roles

Objects

Name
db_accessadmin
db_backupoperator
db_datareader
db_datawriter
db_ddladmin
db_denydatareader
db_denydatawriter
db_owner
db_securityadmin
public
RoleAdmin
RoleInstructor
RoleManger
RoleStudent

db_owner

Members

- Admin
- RoleAdmin

SQL Script

```
ALTER ROLE [db_owner] ADD MEMBER [Admin]
GO
ALTER ROLE [db_owner] ADD MEMBER [RoleAdmin]
GO
```

Uses

Admin
RoleAdmin

RoleAdmin

Members

- Admin

SQL Script

```
CREATE ROLE [RoleAdmin]
AUTHORIZATION [dbo]
GO
ALTER ROLE [RoleAdmin] ADD MEMBER [Admin]
GO
```

Uses

Admin

RoleInstructor

Members

- Instructor

SQL Script

```
CREATE ROLE [RoleInstructor]
AUTHORIZATION [dbo]
GO
ALTER ROLE [RoleInstructor] ADD MEMBER [Instructor]
GO
```

Uses

Instructor

RoleManger

Members

- Manger

SQL Script

```
CREATE ROLE [RoleManger]
AUTHORIZATION [dbo]
GO
ALTER ROLE [RoleManger] ADD MEMBER [Manger]
GO
```

Uses

Manger

RoleStudent

Members

- Student

SQL Script

```
CREATE ROLE [RoleStudent]
AUTHORIZATION [dbo]
GO
ALTER ROLE [RoleStudent] ADD MEMBER [Student]
GO
```

Uses

Student

