. Documentation

ExamSystem

Server	.
Author	Salah
Created	Tuesday, January 6, 2026 7:28:27 PM
File Path	C:\Users\Electronica Care\Desktop\documentation.pdf

Table of Contents

.

Databases (1)

- ExamSystem

Server Properties

Property	Value
Product	Microsoft SQL Server
Version	16.0.1000.6
Language	English (United States)
Platform	NT x64
Edition	Developer Edition (64-bit)
Engine Edition	3 (Enterprise)
Processors	8
OS Version	6.3 (26100)
Physical Memory	8060
Is Clustered	False
Root Directory	D:\MSSQL\MSSQL16.MSSQLSERVER\MSSQL
Collation	Arabic_CI_AS

Server Settings

Property	Value
Default data file path	D:\MSSQL\MSSQL16.MSSQLSERVER\MSSQL\DATA\
Default backup file path	D:\MSSQL\MSSQL16.MSSQLSERVER\MSSQL\Backup
Default log file path	D:\MSSQL\MSSQL16.MSSQLSERVER\MSSQL\DATA\
Recovery Interval (minutes)	0
Default index fill factor	0
Default backup media retention	0
Compress Backup	False

Advanced Server Settings

Property	Value
Full text upgrade option	0
Locks	0
Nested triggers enabled	True
Allow triggers to fire others	True

Default language	English
Network packet size	4096
Default fulltext language LCID	1033
Two-digit year cutoff	2049
Remote login timeout	10
Cursor threshold	-1
Max text replication size	65536
Parallelism cost threshold	5
Max degree of parallelism	8
Min server memory	16
Max server memory	2147483647
Scan for startup procs	False
Transform noise words	False
CLR enabled	False
Blocked process threshold	0
Filestream access level	False
Optimize for ad hoc workloads	False
CLR strict security	True

User databases

Databases (1)

- ExamSystem

ExamSystem Database

Database Properties

Property	Value
SQL Server Version	SqlServer2022
Compatibility Level	SqlServer2022
Last backup time	-
Last log backup time	-
Database size	16.00 MB
Unallocated space	3.25 MB

Files

Name	Type	Size	Maxsize	Autogrowth	File Name
ExamSystem2	Data	8.00 MB	unlimited	64.00 MB	D:\MSSQL\MSSQL16.MSSQLSERVER\MSSQL\DATA\ExamSystem2.mdf
ExamSystem2_log	Log	8.00 MB	2048.00 GB	64.00 MB	D:\MSSQL\MSSQL16.MSSQLSERVER\MSSQL\DATA\ExamSystem2_log.ldf

Tables

Objects

Name
dbo.branch
dbo.branch_department
dbo.course_questions_on_topic
dbo.courses
dbo.department
dbo.department_courses
dbo.exam
dbo.exam_questions
dbo.instructor
dbo.instructor_course
dbo.instructor_generate_course_exam
dbo.person
dbo.person_jong_department_branch
dbo.quesiton_choice
dbo.question
dbo.question_choise_bridge
dbo.student
dbo.student_answer_question
dbo.student_course
dbo.student_exam
dbo.topic

[dbo].[branch]

Properties

Property	Value
Collation	Arabic_CI_AS
Row Count (~)	0
Created	6:04:12 PM Tuesday, January 6, 2026
Last Modified	6:04:12 PM Tuesday, January 6, 2026

Columns

Key	Name	Data Type	Max Length (Bytes)	Nullability	Identity
PK C	branch_id	int	4	NOT NULL	1 - 1
	branch_name	varchar(255)	255	NULL allowed	
	branch_city	varchar(255)	255	NULL allowed	

Indexes

Key	Name	Key Columns	Unique
PK C	PK__branch__E55E37DE23EB48BE	branch_id	True

SQL Script

```
CREATE TABLE [dbo].[branch]
(
[branch_id] [int] NOT NULL IDENTITY(1, 1),
[branch_name] [varchar] (255) COLLATE Arabic_CI_AS NULL,
[branch_city] [varchar] (255) COLLATE Arabic_CI_AS NULL
) ON [PRIMARY]
GO
ALTER TABLE [dbo].[branch] ADD CONSTRAINT [PK__branch__E55E37DE23EB48BE] PRIMARY KEY CLUSTERED
([branch_id]) ON [PRIMARY]
GO
```

Used By

[dbo].[branch_department]
[dbo].[person_jong_department_branch]
[dbo].[AddBranch]
[dbo].[DeleteBranch]

[dbo].[GetBranchById]
[dbo].[UpdateBranch]

[dbo].[branch_department]

Properties

Property	Value
Row Count (~)	0
Created	6:04:12 PM Tuesday, January 6, 2026
Last Modified	6:04:12 PM Tuesday, January 6, 2026

Columns

Key	Name	Data Type	Max Length (Bytes)	Nullability
PK FK	branch_id	int	4	NOT NULL
PK FK	department_id	int	4	NOT NULL

Indexes

Key	Name	Key Columns	Unique
PK	PK__branch_d__D97C059CA147EF4E	branch_id, department_id	True

Foreign Keys

Name	Columns
FK__branch_de__branc__60A75C0F	branch_id->[dbo].[branch].[branch_id]
FK__branch_de__depar__619B8048	department_id->[dbo].[department].[department_id]

SQL Script

```
CREATE TABLE [dbo].[branch_department]
(
[branch_id] [int] NOT NULL,
[department_id] [int] NOT NULL
) ON [PRIMARY]
GO
ALTER TABLE [dbo].[branch_department] ADD CONSTRAINT [PK__branch_d__D97C059CA147EF4E] PRIMARY KEY
CLUSTERED ([branch_id], [department_id]) ON [PRIMARY]
GO
ALTER TABLE [dbo].[branch_department] ADD CONSTRAINT [FK__branch_de__branc__60A75C0F] FOREIGN KEY
([branch_id]) REFERENCES [dbo].[branch] ([branch_id])
GO
ALTER TABLE [dbo].[branch_department] ADD CONSTRAINT [FK__branch_de__depar__619B8048] FOREIGN KEY
```

```
([department_id]) REFERENCES [dbo].[department] ([department_id])
GO
```

Uses

[dbo].[branch]
[dbo].[department]

Used By

[dbo].[AddDepartmentToBranch]
[dbo].[DeleteBranch]
[dbo].[DeleteDepartment]

[dbo].[course_questions_on_topic]

Properties

Property	Value
Row Count (~)	0
Created	6:04:12 PM Tuesday, January 6, 2026
Last Modified	6:04:12 PM Tuesday, January 6, 2026

Columns

Key	Name	Data Type	Max Length (Bytes)	Nullability
	course_id	int	4	NOT NULL
	question_id	int	4	NOT NULL
	topic_id	int	4	NOT NULL

Indexes

Key	Name	Key Columns	Unique
	PK__course_q__E5270C5971C46708	course_id, question_id, topic_id	True

Foreign Keys

Name	Columns
FK__course_qu__cours__76969D2E	course_id->[dbo].[courses].[course_id]
FK__course_qu__quest__778AC167	question_id->[dbo].[question].[question_id]
FK__course_qu__topic__787EE5A0	topic_id->[dbo].[topic].[topic_id]

SQL Script

```
CREATE TABLE [dbo].[course_questions_on_topic]
(
[course_id] [int] NOT NULL,
[question_id] [int] NOT NULL,
[topic_id] [int] NOT NULL
) ON [PRIMARY]
GO
ALTER TABLE [dbo].[course_questions_on_topic] ADD CONSTRAINT [PK__course_q__E5270C5971C46708]
PRIMARY KEY CLUSTERED ([course_id], [question_id], [topic_id]) ON [PRIMARY]
GO
```

```
ALTER TABLE [dbo].[course_questions_on_topic] ADD CONSTRAINT [FK__course_qu__cours__76969D2E]
FOREIGN KEY ([course_id]) REFERENCES [dbo].[courses] ([course_id])
GO
ALTER TABLE [dbo].[course_questions_on_topic] ADD CONSTRAINT [FK__course_qu__quest__778AC167]
FOREIGN KEY ([question_id]) REFERENCES [dbo].[question] ([question_id])
GO
ALTER TABLE [dbo].[course_questions_on_topic] ADD CONSTRAINT [FK__course_qu__topic__787EE5A0]
FOREIGN KEY ([topic_id]) REFERENCES [dbo].[topic] ([topic_id])
GO
```

Uses

[dbo].[courses]
[dbo].[question]
[dbo].[topic]

Used By

[dbo].[DeleteCourse]
[dbo].[DeleteTopic]
[dbo].[GetCourseTopics]
[dbo].[GetQuestionsForCourseTopic]
[dbo].[LinkQuestionToCourseTopic]
[dbo].[UnlinkQuestionFromCourseTopic]

[dbo].[courses]

Properties

Property	Value
Collation	Arabic_CI_AS
Row Count (~)	0
Created	6:04:12 PM Tuesday, January 6, 2026
Last Modified	6:04:12 PM Tuesday, January 6, 2026

Columns

Key	Name	Data Type	Max Length (Bytes)	Nullability	Identity
PK C	course_id	int	4	NOT NULL	1 - 1
	course_code	int	4	NULL allowed	
	description	varchar(255)	255	NULL allowed	
	course_title	varchar(255)	255	NULL allowed	
	credits	varchar(255)	255	NULL allowed	

Indexes

Key	Name	Key Columns	Unique
PK C	PK__courses__8F1EF7AEE75C5EEA	course_id	True

SQL Script

```
CREATE TABLE [dbo].[courses]
(
[course_id] [int] NOT NULL IDENTITY(1, 1),
[course_code] [int] NULL,
[description] [varchar] (255) COLLATE Arabic_CI_AS NULL,
[course_title] [varchar] (255) COLLATE Arabic_CI_AS NULL,
[credits] [varchar] (255) COLLATE Arabic_CI_AS NULL
) ON [PRIMARY]
GO
ALTER TABLE [dbo].[courses] ADD CONSTRAINT [PK__courses__8F1EF7AEE75C5EEA] PRIMARY KEY CLUSTERED
([course_id]) ON [PRIMARY]
GO
```

Used By

[dbo].[course_questions_on_topic]
[dbo].[department_courses]
[dbo].[instructor_course]
[dbo].[instructor_generate_course_exam]
[dbo].[student_course]
[dbo].[AddCourse]
[dbo].[AddCourseToDepartment]
[dbo].[AssignInstructorToCourse]
[dbo].[DeleteCourse]
[dbo].[EnrollStudentInCourse]
[dbo].[GetAllExams]
[dbo].[GetCourseById]
[dbo].[GetCourseTopics]
[dbo].[GetDepartmentCourses]
[dbo].[GetExamById]
[dbo].[GetInstructorCourses]
[dbo].[GetQuestionsForCourseTopic]
[dbo].[GetStudentCourses]
[dbo].[LinkQuestionToCourseTopic]
[dbo].[UpdateCourse]

[dbo].[department]

Properties

Property	Value
Collation	Arabic_CI_AS
Row Count (~)	0
Created	6:04:12 PM Tuesday, January 6, 2026
Last Modified	6:04:12 PM Tuesday, January 6, 2026

Columns

Key	Name	Data Type	Max Length (Bytes)	Nullability	Identity
PK C	department_id	int	4	NOT NULL	1 - 1
	department_name	varchar(255)	255	NULL allowed	

Indexes

Key	Name	Key Columns	Unique
PK C	PK__departme__C2232422BF01BF99	department_id	True

SQL Script

```
CREATE TABLE [dbo].[department]
(
[department_id] [int] NOT NULL IDENTITY(1, 1),
[department_name] [varchar] (255) COLLATE Arabic_CI_AS NULL
) ON [PRIMARY]
GO
ALTER TABLE [dbo].[department] ADD CONSTRAINT [PK__departme__C2232422BF01BF99] PRIMARY KEY
CLUSTERED ([department_id]) ON [PRIMARY]
GO
```

Used By

[dbo].[branch_department]
[dbo].[department_courses]
[dbo].[person_jong_department_branch]
[dbo].[AddCourseToDepartment]
[dbo].[AddDepartment]
[dbo].[DeleteDepartment]

[dbo].[GetDepartmentById]
[dbo].[GetDepartmentCourses]
[dbo].[UpdateDepartment]

[dbo].[department_courses]

Properties

Property	Value
Row Count (~)	0
Created	6:04:12 PM Tuesday, January 6, 2026
Last Modified	6:04:12 PM Tuesday, January 6, 2026

Columns

Key	Name	Data Type	Max Length (Bytes)	Nullability
PK FK	course_id	int	4	NOT NULL
PK FK	department_id	int	4	NOT NULL

Indexes

Key	Name	Key Columns	Unique
PK	PK__departme__B33CC5ECF8A1B3BB	course_id, department_id	True

Foreign Keys

Name	Columns
FK__departmen__cours__628FA481	course_id->[dbo].[courses].[course_id]
FK__departmen__depar__6383C8BA	department_id->[dbo].[department].[department_id]

SQL Script

```
CREATE TABLE [dbo].[department_courses]
(
[course_id] [int] NOT NULL,
[department_id] [int] NOT NULL
) ON [PRIMARY]
GO
ALTER TABLE [dbo].[department_courses] ADD CONSTRAINT [PK__departme__B33CC5ECF8A1B3BB] PRIMARY
KEY CLUSTERED ([course_id], [department_id]) ON [PRIMARY]
GO
ALTER TABLE [dbo].[department_courses] ADD CONSTRAINT [FK__departmen__cours__628FA481] FOREIGN
KEY ([course_id]) REFERENCES [dbo].[courses] ([course_id])
GO
ALTER TABLE [dbo].[department_courses] ADD CONSTRAINT [FK__departmen__depar__6383C8BA] FOREIGN
```

```
KEY ([department_id]) REFERENCES [dbo].[department] ([department_id])
GO
```

Uses

[dbo].[courses]
[dbo].[department]

Used By

[dbo].[AddCourseToDepartment]
[dbo].[DeleteCourse]
[dbo].[DeleteDepartment]
[dbo].[GetDepartmentCourses]
[dbo].[RemoveCourseFromDepartment]

[dbo].[exam]

Properties

Property	Value
Collation	Arabic_CI_AS
Row Count (~)	0
Created	6:04:12 PM Tuesday, January 6, 2026
Last Modified	6:04:12 PM Tuesday, January 6, 2026

Columns

Key	Name	Data Type	Max Length (Bytes)	Nullability	Identity
PK C	exam_id	int	4	NOT NULL	1 - 1
	exam_title	varchar(255)	255	NULL allowed	
	total_grade	int	4	NULL allowed	
	exam_date	datetime	8	NULL allowed	
▤	exam_type	varchar(50)	50	NULL allowed	
	duration_mins	int	4	NULL allowed	

Indexes

Key	Name	Key Columns	Unique
PK C	PK__exam__9C8C7BE9812C2180	exam_id	True

Check Constraints

Name	On Column	Constraint
CK__exam__exam_type__3F466844	exam_type	([exam_type]='semifinal' OR [exam_type]='mid' OR [exam_type]='final')

SQL Script

```
CREATE TABLE [dbo].[exam]
(
[exam_id] [int] NOT NULL IDENTITY(1, 1),
[exam_title] [varchar] (255) COLLATE Arabic_CI_AS NULL,
[total_grade] [int] NULL,
[exam_date] [datetime] NULL,
[exam_type] [varchar] (50) COLLATE Arabic_CI_AS NULL,
[duration_mins] [int] NULL
) ON [PRIMARY]
GO
ALTER TABLE [dbo].[exam] ADD CONSTRAINT [CK__exam__exam_type__3F466844] CHECK
(([exam_type]='semifinal' OR [exam_type]='mid' OR [exam_type]='final'))
GO
ALTER TABLE [dbo].[exam] ADD CONSTRAINT [PK__exam__9C8C7BE9812C2180] PRIMARY KEY CLUSTERED
([exam_id]) ON [PRIMARY]
GO
```

Used By

[dbo].[exam_questions]
[dbo].[instructor_generate_course_exam]
[dbo].[student_answer_question]
[dbo].[student_exam]
[dbo].[CorrectExam]
[dbo].[DeleteExam]
[dbo].[GetAllExams]
[dbo].[GetExamById]
[dbo].[GetExamQuestions]
[dbo].[GetExamQuestionsWithChoicesPivoted]
[dbo].[sp_GetExamForStudent]
[dbo].[StudentSubmitAnswers]
[dbo].[UpdateExam]

[dbo].[exam_questions]

Properties

Property	Value
Row Count (~)	0
Created	6:04:12 PM Tuesday, January 6, 2026
Last Modified	6:04:12 PM Tuesday, January 6, 2026

Columns

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

Indexes

Key	Name	Key Columns	Unique
PK C	PK__exam_que__E8EC932AC9FDE305	exam_id, questoin_id	True

Foreign Keys

Name	Columns
FK__exam_ques__exam___656C112C	exam_id->[dbo].[exam].[exam_id]
FK__exam_ques__quest__66603565	questoin_id->[dbo].[question].[question_id]

SQL Script

```
CREATE TABLE [dbo].[exam_questions]
(
[exam_id] [int] NOT NULL,
[questoin_id] [int] NOT NULL
) ON [PRIMARY]
GO
ALTER TABLE [dbo].[exam_questions] ADD CONSTRAINT [PK__exam_que__E8EC932AC9FDE305] PRIMARY KEY
CLUSTERED ([exam_id], [questoin_id]) ON [PRIMARY]
GO
ALTER TABLE [dbo].[exam_questions] ADD CONSTRAINT [FK__exam_ques__exam___656C112C] FOREIGN KEY
([exam_id]) REFERENCES [dbo].[exam] ([exam_id])
GO
ALTER TABLE [dbo].[exam_questions] ADD CONSTRAINT [FK__exam_ques__quest__66603565] FOREIGN KEY
```

```
([questoin_id]) REFERENCES [dbo].[question] ([question_id])
GO
```

Uses

[dbo].[exam]
[dbo].[question]

Used By

[dbo].[DeleteExam]
[dbo].[GetExamQuestions]
[dbo].[GetExamQuestionsWithChoicesPivoted]
[dbo].[sp_GetExamForStudent]
[dbo].[StudentSubmitAnswers]

[dbo].[instructor]

Properties

Property	Value
Row Count (~)	0
Created	6:04:12 PM Tuesday, January 6, 2026
Last Modified	6:04:12 PM Tuesday, January 6, 2026

Columns

Key	Name	Data Type	Max Length (Bytes)	Nullability
PK FK	instructor_id	int	4	NOT NULL
	hire_date	date	3	NULL allowed

Indexes

Key	Name	Key Columns	Unique
PK	PK__instruct__A1EF56E8E930E1E4	instructor_id	True

Foreign Keys

Name	Columns
FK__instructo__instr__6A30C649	instructor_id->[dbo].[person].[person_id]

SQL Script

```
CREATE TABLE [dbo].[instructor]
(
[instructor_id] [int] NOT NULL,
[hire_date] [date] NULL
) ON [PRIMARY]
GO
ALTER TABLE [dbo].[instructor] ADD CONSTRAINT [PK__instruct__A1EF56E8E930E1E4] PRIMARY KEY
CLUSTERED ([instructor_id]) ON [PRIMARY]
GO
ALTER TABLE [dbo].[instructor] ADD CONSTRAINT [FK__instructo__instr__6A30C649] FOREIGN KEY
([instructor_id]) REFERENCES [dbo].[person] ([person_id])
GO
```

Uses

[dbo].[person]

Used By

[dbo].[instructor_course]
[dbo].[instructor_generate_course_exam]
[dbo].[AddInstructor]
[dbo].[AssignInstructorToCourse]
[dbo].[DeleteInstructor]
[dbo].[GetAllInstructors]
[dbo].[GetInstructorById]
[dbo].[GetInstructorCourses]
[dbo].[UpdateInstructor]

[dbo].[instructor_course]

Properties

Property	Value
Row Count (~)	0
Created	6:04:12 PM Tuesday, January 6, 2026
Last Modified	6:04:12 PM Tuesday, January 6, 2026

Columns

Key	Name	Data Type	Max Length (Bytes)	Nullability
PK FK	instructor_id	int	4	NOT NULL
PK FK	course_id	int	4	NOT NULL

Indexes

Key	Name	Key Columns	Unique
PK	PK__instruct__591EB99228A61141	instructor_id, course_id	True

Foreign Keys

Name	Columns
FK__instructo__cours__6FE99F9F	course_id->[dbo].[courses].[course_id]
FK__instructo__instr__6EF57B66	instructor_id->[dbo].[instructor].[instructor_id]

SQL Script

```
CREATE TABLE [dbo].[instructor_course]
(
[instructor_id] [int] NOT NULL,
[course_id] [int] NOT NULL
) ON [PRIMARY]
GO
ALTER TABLE [dbo].[instructor_course] ADD CONSTRAINT [PK__instruct__591EB99228A61141] PRIMARY KEY
CLUSTERED ([instructor_id], [course_id]) ON [PRIMARY]
GO
ALTER TABLE [dbo].[instructor_course] ADD CONSTRAINT [FK__instructo__cours__6FE99F9F] FOREIGN KEY
([course_id]) REFERENCES [dbo].[courses] ([course_id])
GO
ALTER TABLE [dbo].[instructor_course] ADD CONSTRAINT [FK__instructo__instr__6EF57B66] FOREIGN KEY
```

```
([instructor_id]) REFERENCES [dbo].[instructor] ([instructor_id])
GO
```

Uses

[dbo].[courses]
[dbo].[instructor]

Used By

[dbo].[AssignInstructorToCourse]
[dbo].[DeleteCourse]
[dbo].[DeleteInstructor]
[dbo].[GetInstructorCourses]
[dbo].[RemoveInstructorFromCourse]

[dbo].[instructor_generate_course_exam]

Properties

Property	Value
Row Count (~)	0
Created	6:04:12 PM Tuesday, January 6, 2026
Last Modified	6:04:12 PM Tuesday, January 6, 2026

Columns

Key	Name	Data Type	Max Length (Bytes)	Nullability
🔑🔑	instructor_id	int	4	NOT NULL
🔑	exam_id	int	4	NULL allowed
🔑🔑	course_id	int	4	NOT NULL
🔑	genrate_date	datetime	8	NOT NULL

Indexes

Key	Name	Key Columns	Unique
🔑	PK__instruct__0961CD79A6F47A7A	instructor_id, course_id, genrate_date	True

Foreign Keys

Name	Columns
FK__instructo__cours__75A278F5	course_id->[dbo].[courses].[course_id]
FK__instructo__exam___74AE54BC	exam_id->[dbo].[exam].[exam_id]
FK__instructo__instr__73BA3083	instructor_id->[dbo].[instructor].[instructor_id]

SQL Script

```
CREATE TABLE [dbo].[instructor_generate_course_exam]
(
[instructor_id] [int] NOT NULL,
[exam_id] [int] NULL,
[course_id] [int] NOT NULL,
[genrate_date] [datetime] NOT NULL
) ON [PRIMARY]
GO
```

```
ALTER TABLE [dbo].[instructor_generate_course_exam] ADD CONSTRAINT
[PK__instruct__0961CD79A6F47A7A] PRIMARY KEY CLUSTERED ([instructor_id], [course_id],
[genrate_date]) ON [PRIMARY]
GO
ALTER TABLE [dbo].[instructor_generate_course_exam] ADD CONSTRAINT
[FK__instructo__cours__75A278F5] FOREIGN KEY ([course_id]) REFERENCES [dbo].[courses]
([course_id])
GO
ALTER TABLE [dbo].[instructor_generate_course_exam] ADD CONSTRAINT
[FK__instructo__exam___74AE54BC] FOREIGN KEY ([exam_id]) REFERENCES [dbo].[exam] ([exam_id])
GO
ALTER TABLE [dbo].[instructor_generate_course_exam] ADD CONSTRAINT
[FK__instructo__instr__73BA3083] FOREIGN KEY ([instructor_id]) REFERENCES [dbo].[instructor]
([instructor_id])
GO
```

Uses

[dbo].[courses]
[dbo].[exam]
[dbo].[instructor]

Used By

[dbo].[DeleteCourse]
[dbo].[DeleteExam]
[dbo].[DeleteInstructor]
[dbo].[GetAllExams]
[dbo].[GetExamById]
[dbo].[RemoveInstructorFromCourse]
[dbo].[sp_GetExamForStudent]
[dbo].[StudentSubmitAnswers]
[dbo].[UnenrollStudentFromCourse]

[dbo].[person]

Properties

Property	Value
Collation	Arabic_CI_AS
Row Count (~)	0
Created	6:04:12 PM Tuesday, January 6, 2026
Last Modified	6:04:12 PM Tuesday, January 6, 2026

Columns

Key	Name	Data Type	Max Length (Bytes)	Nullability	Identity
	person_id	int	4	NOT NULL	1 - 1
	first_name	varchar(255)	255	NULL allowed	
	last_name	varchar(255)	255	NULL allowed	
	email	varchar(255)	255	NULL allowed	

Indexes

Key	Name	Key Columns	Unique
	PK__person__543848DFA727EAA4	person_id	True

SQL Script

```
CREATE TABLE [dbo].[person]
(
[person_id] [int] NOT NULL IDENTITY(1, 1),
[first_name] [varchar] (255) COLLATE Arabic_CI_AS NULL,
[last_name] [varchar] (255) COLLATE Arabic_CI_AS NULL,
[email] [varchar] (255) COLLATE Arabic_CI_AS NULL
) ON [PRIMARY]
GO
ALTER TABLE [dbo].[person] ADD CONSTRAINT [PK__person__543848DFA727EAA4] PRIMARY KEY CLUSTERED
([person_id]) ON [PRIMARY]
GO
```

Used By

[dbo].[instructor]

[dbo].[person_jong_department_branch]
[dbo].[student]
[dbo].[AddInstructor]
[dbo].[AddStudent]
[dbo].[DeleteInstructor]
[dbo].[DeleteStudent]
[dbo].[GetAllExams]
[dbo].[GetAllInstructors]
[dbo].[GetAllStudents]
[dbo].[GetExamById]
[dbo].[GetInstructorById]
[dbo].[GetStudentById]
[dbo].[UpdateInstructor]
[dbo].[UpdateStudent]

[dbo].[person_jong_department_branch]

Properties

Property	Value
Row Count (~)	0
Created	6:04:12 PM Tuesday, January 6, 2026
Last Modified	6:04:12 PM Tuesday, January 6, 2026

Columns

Key	Name	Data Type	Max Length (Bytes)	Nullability
PK FK	person_id	int	4	NOT NULL
FK	branch_id	int	4	NULL allowed
FK	department_id	int	4	NULL allowed
PK	join_date	datetime	8	NOT NULL

Indexes

Key	Name	Key Columns	Unique
PK	PK__person_j__F2DF8C677263D0D0	person_id, join_date	True

Foreign Keys

Name	Columns
FK__person_jo__branc__71D1E811	branch_id->[dbo].[branch].[branch_id]
FK__person_jo__depar__72C60C4A	department_id->[dbo].[department].[department_id]
FK__person_jo__perso__70DDC3D8	person_id->[dbo].[person].[person_id]

SQL Script

```
CREATE TABLE [dbo].[person_jong_department_branch]
(
[person_id] [int] NOT NULL,
[branch_id] [int] NULL,
[department_id] [int] NULL,
[join_date] [datetime] NOT NULL
) ON [PRIMARY]
GO
ALTER TABLE [dbo].[person_jong_department_branch] ADD CONSTRAINT [PK__person_j__F2DF8C677263D0D0]
```

```
PRIMARY KEY CLUSTERED ([person_id], [join_date]) ON [PRIMARY]
GO
ALTER TABLE [dbo].[person_jong_department_branch] ADD CONSTRAINT [FK__person_jo__branc__71D1E811]
FOREIGN KEY ([branch_id]) REFERENCES [dbo].[branch] ([branch_id])
GO
ALTER TABLE [dbo].[person_jong_department_branch] ADD CONSTRAINT [FK__person_jo__depar__72C60C4A]
FOREIGN KEY ([department_id]) REFERENCES [dbo].[department] ([department_id])
GO
ALTER TABLE [dbo].[person_jong_department_branch] ADD CONSTRAINT [FK__person_jo__perso__70DDC3D8]
FOREIGN KEY ([person_id]) REFERENCES [dbo].[person] ([person_id])
GO
```

Uses

[dbo].[branch]

[dbo].[department]

[dbo].[person]

Used By

[dbo].[DeleteDepartment]

[dbo].[quesiton_choice]

Properties

Property	Value
Collation	Arabic_CI_AS
Row Count (~)	0
Created	6:04:12 PM Tuesday, January 6, 2026
Last Modified	6:04:12 PM Tuesday, January 6, 2026

Columns

Key	Name	Data Type	Max Length (Bytes)	Nullability	Identity
PK C	choice_id	int	4	NOT NULL	1 - 1
	choice_text	varchar(255)	255	NULL allowed	

Indexes

Key	Name	Key Columns	Unique
PK C	PK__quesiton__33CAF83ABEEE4ADC	choice_id	True

SQL Script

```
CREATE TABLE [dbo].[quesiton_choice]
(
[choice_id] [int] NOT NULL IDENTITY(1, 1),
[choice_text] [varchar] (255) COLLATE Arabic_CI_AS NULL
) ON [PRIMARY]
GO
ALTER TABLE [dbo].[quesiton_choice] ADD CONSTRAINT [PK__quesiton__33CAF83ABEEE4ADC] PRIMARY KEY
CLUSTERED ([choice_id]) ON [PRIMARY]
GO
```

Used By

[dbo].[question]
[dbo].[question_choise_bridge]
[dbo].[DeleteChoice]
[dbo].[GetAllQuestionsWithChoicesPivoted]
[dbo].[GetChoicesByQuestionId]
[dbo].[GetExamQuestions]

[dbo].[GetExamQuestionsWithChoicesPivoted]
[dbo].[GetQuestionWithChoicesPivoted]
[dbo].[InsertMCQChoice]
[dbo].[InsertTrueFalseChoices]
[dbo].[sp_GetExamForStudent]
[dbo].[UpdateChoiceText]
[dbo].[UpdateQuestionCorrectAnswer]

[dbo].[question]

Properties

Property	Value
Collation	Arabic_CI_AS
Row Count (~)	0
Created	6:04:12 PM Tuesday, January 6, 2026
Last Modified	6:04:12 PM Tuesday, January 6, 2026

Columns

Key	Name	Data Type	Max Length (Bytes)	Nullability	Identity
FK C	question_id	int	4	NOT NULL	1 - 1
	question_text	varchar(255)	255	NULL allowed	
	question_type	varchar(50)	50	NULL allowed	
	question_difficulty	varchar(50)	50	NULL allowed	
FK	correct_ans_id	int	4	NULL allowed	

Indexes

Key	Name	Key Columns	Unique
FK C	PK__question__2EC215491D089675	question_id	True

Check Constraints

Name	On Column	Constraint
CK__question__questi__44FF419A	question_difficulty	([question_difficulty]='easy' OR [question_difficulty]='medium' OR [question_difficulty]='hard')
CK__question__questi__440B1D61	question_type	([question_type]='True_False' OR [question_type]='MCQ')

Foreign Keys

Name	Columns
FK__question__correc__6477ECF3	correct_ans_id->[dbo].[quesiton_choice].[choice_id]

SQL Script

```
CREATE TABLE [dbo].[question]
(
[question_id] [int] NOT NULL IDENTITY(1, 1),
[question_text] [varchar] (255) COLLATE Arabic_CI_AS NULL,
[question_type] [varchar] (50) COLLATE Arabic_CI_AS NULL,
[question_difficulty] [varchar] (50) COLLATE Arabic_CI_AS NULL,
[correct_ans_id] [int] NULL
) ON [PRIMARY]
GO
ALTER TABLE [dbo].[question] ADD CONSTRAINT [CK__question__questi__44FF419A] CHECK
(([question_difficulty]='easy' OR [question_difficulty]='medium' OR
[question_difficulty]='hard'))
GO
ALTER TABLE [dbo].[question] ADD CONSTRAINT [CK__question__questi__440B1D61] CHECK
(([question_type]='True_False' OR [question_type]='MCQ'))
GO
ALTER TABLE [dbo].[question] ADD CONSTRAINT [PK__question__2EC215491D089675] PRIMARY KEY
CLUSTERED ([question_id]) ON [PRIMARY]
GO
ALTER TABLE [dbo].[question] ADD CONSTRAINT [FK__question__correc__6477ECF3] FOREIGN KEY
([correct_ans_id]) REFERENCES [dbo].[quesiton_choice] ([choice_id])
GO
```

Uses

[dbo].[quesiton_choice]

Used By

[dbo].[course_questions_on_topic]
[dbo].[exam_questions]
[dbo].[question_choise_bridge]
[dbo].[student_answer_question]
[dbo].[CorrectExam]
[dbo].[DeleteChoice]
[dbo].[DeleteQuestion]
[dbo].[GetAllQuestionsWithChoicesPivoted]
[dbo].[GetChoicesByQuestionId]
[dbo].[GetExamQuestions]
[dbo].[GetExamQuestionsWithChoicesPivoted]
[dbo].[GetQuestionsForCourseTopic]
[dbo].[GetQuestionWithChoices]
[dbo].[GetQuestionWithChoicesPivoted]
[dbo].[InsertMCQChoice]
[dbo].[InsertQuestion]
[dbo].[InsertTrueFalseChoices]
[dbo].[LinkQuestionToCourseTopic]
[dbo].[sp_GetExamForStudent]
[dbo].[UpdateQuestion]
[dbo].[UpdateQuestionCorrectAnswer]

[dbo].[question_choise_bridge]

Properties

Property	Value
Row Count (~)	0
Created	6:04:12 PM Tuesday, January 6, 2026
Last Modified	6:04:12 PM Tuesday, January 6, 2026

Columns

Key	Name	Data Type	Max Length (Bytes)	Nullability
PK FK	question_id	int	4	NOT NULL
PK FK	choice_id	int	4	NOT NULL

Indexes

Key	Name	Key Columns	Unique
PK	PK__question__9DFEBACAC30EC42C	question_id, choice_id	True

Foreign Keys

Name	Columns
FK__question___choic__68487DD7	choice_id->[dbo].[quesiton_choice].[choice_id]
FK__question___quest__6754599E	question_id->[dbo].[question].[question_id]

SQL Script

```
CREATE TABLE [dbo].[question_choise_bridge]
(
[question_id] [int] NOT NULL,
[choice_id] [int] NOT NULL
) ON [PRIMARY]
GO
ALTER TABLE [dbo].[question_choise_bridge] ADD CONSTRAINT [PK__question__9DFEBACAC30EC42C]
PRIMARY KEY CLUSTERED ([question_id], [choice_id]) ON [PRIMARY]
GO
ALTER TABLE [dbo].[question_choise_bridge] ADD CONSTRAINT [FK__question___choic__68487DD7]
FOREIGN KEY ([choice_id]) REFERENCES [dbo].[quesiton_choice] ([choice_id])
GO
ALTER TABLE [dbo].[question_choise_bridge] ADD CONSTRAINT [FK__question___quest__6754599E]
```

```
FOREIGN KEY ([question_id]) REFERENCES [dbo].[question] ([question_id])
GO
```

Uses

[dbo].[quesiton_choice]
[dbo].[question]

Used By

[dbo].[DeleteChoice]
[dbo].[DeleteQuestion]
[dbo].[GetAllQuestionsWithChoicesPivoted]
[dbo].[GetChoicesByQuestionId]
[dbo].[GetExamQuestions]
[dbo].[GetExamQuestionsWithChoicesPivoted]
[dbo].[GetQuestionWithChoicesPivoted]
[dbo].[InsertMCQChoice]
[dbo].[InsertTrueFalseChoices]
[dbo].[UpdateQuestionCorrectAnswer]

[dbo].[student]

Properties

Property	Value
Row Count (~)	0
Created	6:04:12 PM Tuesday, January 6, 2026
Last Modified	6:04:12 PM Tuesday, January 6, 2026

Columns

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

Indexes

Key	Name	Key Columns	Unique
PK	PK__student__2A33069AE3FEFE95	student_id	True

Foreign Keys

Name	Columns
FK__student__student__693CA210	student_id->[dbo].[person].[person_id]

SQL Script

```
CREATE TABLE [dbo].[student]
(
[student_id] [int] NOT NULL
) ON [PRIMARY]
GO
ALTER TABLE [dbo].[student] ADD CONSTRAINT [PK__student__2A33069AE3FEFE95] PRIMARY KEY CLUSTERED
([student_id]) ON [PRIMARY]
GO
ALTER TABLE [dbo].[student] ADD CONSTRAINT [FK__student__student__693CA210] FOREIGN KEY
([student_id]) REFERENCES [dbo].[person] ([person_id])
GO
```

Uses

[dbo].[person]

Used By

[dbo].[student_answer_question]
[dbo].[student_course]
[dbo].[student_exam]
[dbo].[AddStudent]
[dbo].[DeleteDepartment]
[dbo].[DeleteStudent]
[dbo].[EnrollStudentInCourse]
[dbo].[GetAllStudents]
[dbo].[GetStudentById]
[dbo].[GetStudentCourses]
[dbo].[UpdateStudent]

[dbo].[student_answer_question]

Properties

Property	Value
Collation	Arabic_CI_AS
Row Count (~)	0
Created	6:04:12 PM Tuesday, January 6, 2026
Last Modified	6:04:12 PM Tuesday, January 6, 2026

Columns

Key	Name	Data Type	Max Length (Bytes)	Nullability
🔑🔑	student_id	int	4	NOT NULL
🔑	exam_id	int	4	NULL allowed
🔑🔑	quesiotn_id	int	4	NOT NULL
	student_answer	varchar(255)	255	NULL allowed
🔑	answer_date	datetime	8	NOT NULL

Indexes

Key	Name	Key Columns	Unique
🔑	PK__student___06872EC7BEAE4705	student_id, quesiotn_id, answer_date	True

Foreign Keys

Name	Columns
FK__student_a__exam___7A672E12	exam_id->[dbo].[exam].[exam_id]
FK__student_a__quesi__7B5B524B	quesiotn_id->[dbo].[question].[question_id]
FK__student_a__stude__797309D9	student_id->[dbo].[student].[student_id]

SQL Script

```
CREATE TABLE [dbo].[student_answer_question]
(
[student_id] [int] NOT NULL,
[exam_id] [int] NULL,
[quesiotn_id] [int] NOT NULL,
[student_answer] [varchar] (255) COLLATE Arabic_CI_AS NULL,
[answer_date] [datetime] NOT NULL
) ON [PRIMARY]
GO
ALTER TABLE [dbo].[student_answer_question] ADD CONSTRAINT [PK__student___06872EC7BEAE4705]
PRIMARY KEY CLUSTERED ([student_id], [quesiotn_id], [answer_date]) ON [PRIMARY]
GO
ALTER TABLE [dbo].[student_answer_question] ADD CONSTRAINT [FK__student_a__exam___7A672E12]
FOREIGN KEY ([exam_id]) REFERENCES [dbo].[exam] ([exam_id])
GO
ALTER TABLE [dbo].[student_answer_question] ADD CONSTRAINT [FK__student_a__quesi__7B5B524B]
FOREIGN KEY ([quesiotn_id]) REFERENCES [dbo].[question] ([question_id])
GO
ALTER TABLE [dbo].[student_answer_question] ADD CONSTRAINT [FK__student_a__stude__797309D9]
FOREIGN KEY ([student_id]) REFERENCES [dbo].[student] ([student_id])
GO
```

Uses

[dbo].[exam]
[dbo].[question]
[dbo].[student]

Used By

[dbo].[CorrectExam]
[dbo].[DeleteExam]
[dbo].[sp_GetExamForStudent]
[dbo].[StudentSubmitAnswers]

[dbo].[student_course]

Properties

Property	Value
Row Count (~)	0
Created	6:04:12 PM Tuesday, January 6, 2026
Last Modified	6:04:12 PM Tuesday, January 6, 2026

Columns

Key	Name	Data Type	Max Length (Bytes)	Nullability
PK FK	student_id	int	4	NOT NULL
PK FK	course_id	int	4	NOT NULL
	enrollment_date	datetime	8	NULL allowed

Indexes

Key	Name	Key Columns	Unique
PK	PK__student___D2C2E9E0AFECFFBE	student_id, course_id	True

Foreign Keys

Name	Columns
FK__student_c__cours__6E01572D	course_id->[dbo].[courses].[course_id]
FK__student_c__stude__6D0D32F4	student_id->[dbo].[student].[student_id]

SQL Script

```
CREATE TABLE [dbo].[student_course]
(
[student_id] [int] NOT NULL,
[course_id] [int] NOT NULL,
[enrollment_date] [datetime] NULL
) ON [PRIMARY]
GO
ALTER TABLE [dbo].[student_course] ADD CONSTRAINT [PK__student___D2C2E9E0AFECFFBE] PRIMARY KEY
CLUSTERED ([student_id], [course_id]) ON [PRIMARY]
GO
ALTER TABLE [dbo].[student_course] ADD CONSTRAINT [FK__student_c__cours__6E01572D] FOREIGN KEY
([course_id]) REFERENCES [dbo].[courses] ([course_id])
```

```
GO
ALTER TABLE [dbo].[student_course] ADD CONSTRAINT [FK__student_c__stude__6D0D32F4] FOREIGN KEY
([student_id]) REFERENCES [dbo].[student] ([student_id])
GO
```

Uses

[dbo].[courses]
[dbo].[student]

Used By

[dbo].[DeleteCourse]
[dbo].[EnrollStudentInCourse]
[dbo].[GetStudentCourses]
[dbo].[sp_GetExamForStudent]
[dbo].[StudentSubmitAnswers]
[dbo].[UnenrollStudentFromCourse]

[dbo].[student_exam]

Properties

Property	Value
Collation	Arabic_CI_AS
Row Count (~)	0
Created	6:04:12 PM Tuesday, January 6, 2026
Last Modified	6:04:12 PM Tuesday, January 6, 2026

Columns

Key	Name	Data Type	Max Length (Bytes)	Nullability
	student_id	int	4	NOT NULL
	exam_id	int	4	NOT NULL
	state	varchar(255)	255	NULL allowed
	exam_date	datetime	8	NULL allowed
	grade	int	4	NULL allowed

Indexes

Key	Name	Key Columns	Unique
	PK__student___A3FBC124783762FE	student_id, exam_id	True

Foreign Keys

Name	Columns
FK__student_e__exam___6C190EBB	exam_id->[dbo].[exam].[exam_id]
FK__student_e__stude__6B24EA82	student_id->[dbo].[student].[student_id]

SQL Script

```
CREATE TABLE [dbo].[student_exam]
(
[student_id] [int] NOT NULL,
[exam_id] [int] NOT NULL,
[state] [varchar] (255) COLLATE Arabic_CI_AS NULL,
[exam_date] [datetime] NULL,
[grade] [int] NULL
) ON [PRIMARY]
```

```
GO
ALTER TABLE [dbo].[student_exam] ADD CONSTRAINT [PK__student___A3FBC124783762FE] PRIMARY KEY
CLUSTERED ([student_id], [exam_id]) ON [PRIMARY]
GO
ALTER TABLE [dbo].[student_exam] ADD CONSTRAINT [FK__student_e__exam___6C190EBB] FOREIGN KEY
([exam_id]) REFERENCES [dbo].[exam] ([exam_id])
GO
ALTER TABLE [dbo].[student_exam] ADD CONSTRAINT [FK__student_e__stude__6B24EA82] FOREIGN KEY
([student_id]) REFERENCES [dbo].[student] ([student_id])
GO
```

Uses

[dbo].[exam]

[dbo].[student]

Used By

[dbo].[CorrectExam]

[dbo].[DeleteExam]

[dbo].[sp_GetExamForStudent]

[dbo].[StudentSubmitAnswers]

[dbo].[UnenrollStudentFromCourse]

[dbo].[topic]

Properties

Property	Value
Collation	Arabic_CI_AS
Row Count (~)	0
Created	6:04:12 PM Tuesday, January 6, 2026
Last Modified	6:04:12 PM Tuesday, January 6, 2026

Columns

Key	Name	Data Type	Max Length (Bytes)	Nullability	Identity
PK C	topic_id	int	4	NOT NULL	1 - 1
	topic_order	int	4	NULL allowed	
	topic_duration	varchar(255)	255	NULL allowed	
	topic_title	varchar(255)	255	NULL allowed	

Indexes

Key	Name	Key Columns	Unique
PK C	PK__topic__D5DAA3E9C54B619E	topic_id	True

SQL Script

```
CREATE TABLE [dbo].[topic]
(
[topic_id] [int] NOT NULL IDENTITY(1, 1),
[topic_order] [int] NULL,
[topic_duration] [varchar] (255) COLLATE Arabic_CI_AS NULL,
[topic_title] [varchar] (255) COLLATE Arabic_CI_AS NULL
) ON [PRIMARY]
GO
ALTER TABLE [dbo].[topic] ADD CONSTRAINT [PK__topic__D5DAA3E9C54B619E] PRIMARY KEY CLUSTERED
([topic_id]) ON [PRIMARY]
GO
```

Used By

[dbo].[course_questions_on_topic]
[dbo].[AddTopic]

[dbo].[DeleteTopic]
[dbo].[GetCourseTopics]
[dbo].[GetQuestionsForCourseTopic]
[dbo].[GetTopicById]
[dbo].[LinkQuestionToCourseTopic]
[dbo].[UpdateTopic]

Stored Procedures

Objects

Name
dbo.AddBranch
dbo.AddCourse
dbo.AddCourseToDepartment
dbo.AddDepartment
dbo.AddDepartmentToBranch
dbo.AddInstructor
dbo.AddStudent
dbo.AddTopic
dbo.AssignInstructorToCourse
dbo.CorrectExam
dbo.DeleteBranch
dbo.DeleteChoice
dbo.DeleteCourse
dbo.DeleteDepartment
dbo.DeleteExam
dbo.DeleteInstructor
dbo.DeleteQuestion
dbo.DeleteStudent
dbo.DeleteTopic
dbo.EnrollStudentInCourse
dbo.GetAllExams
dbo.GetAllInstructors
dbo.GetAllQuestionsWithChoicesPivoted
dbo.GetAllStudents
dbo.GetBranchById
dbo.GetChoicesByQuestionId
dbo.GetCourseById
dbo.GetCourseTopics
dbo.GetDepartmentById
dbo.GetDepartmentCourses
dbo.GetExamById
dbo.GetExamQuestions
dbo.GetExamQuestionsWithChoicesPivoted
dbo.GetInstructorById

dbo.GetInstructorCourses
dbo.GetQuestionsForCourseTopic
dbo.GetQuestionWithChoices
dbo.GetQuestionWithChoicesPivoted
dbo.GetStudentById
dbo.GetStudentCourses
dbo.GetTopicById
dbo.InsertMCQChoice
dbo.InsertQuestion
dbo.InsertTrueFalseChoices
dbo.LinkQuestionToCourseTopic
dbo.RemoveCourseFromDepartment
dbo.RemoveInstructorFromCourse
dbo.sp_GetExamForStudent
dbo.StudentSubmitAnswers
dbo.UnenrollStudentFromCourse
dbo.UnlinkQuestionFromCourseTopic
dbo.UpdateBranch
dbo.UpdateChoiceText
dbo.UpdateCourse
dbo.UpdateDepartment
dbo.UpdateExam
dbo.UpdateInstructor
dbo.UpdateQuestion
dbo.UpdateQuestionCorrectAnswer
dbo.UpdateStudent
dbo.UpdateTopic

[dbo].[AddBranch]

Properties

Property	Value
ANSI Nulls On	True
Quoted Identifier On	False

Parameters

Name	Data Type	Max Length (Bytes)
@name	varchar(255)	255
@city	varchar(255)	255

SQL Script

```
SET QUOTED_IDENTIFIER OFF
GO
CREATE PROCEDURE [dbo].[AddBranch](@name varchar(255), @city varchar(255))
AS
BEGIN
    BEGIN TRY
        BEGIN TRANSACTION
            IF NOT EXISTS (SELECT * FROM branch
            WHERE branch_name = @name COLLATE SQL_Latin1_General_CP1_CI_AS)
            BEGIN
                INSERT INTO branch(branch_name, branch_city)
                VALUES(@name, @city);
            END
            ELSE
            BEGIN
                DECLARE @error_message varchar(50) = 'Branch ' + @name + ' is already exists!';
                THROW 50010, @error_message, 1;
            END
        COMMIT TRANSACTION;
    END TRY
    BEGIN CATCH
        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION;
        THROW;
    END CATCH
END


EXEC AddBranch 'ITI Assiut', 'Assiut'
EXEC AddBranch 'ITI Alex', 'Alexandria'
EXEC AddBranch 'ITI Giza', 'Giza'
GO
```

Uses

[dbo].[branch]

[dbo].[AddCourse]

Properties

Property	Value
ANSI Nulls On	True
Quoted Identifier On	False

Parameters

Name	Data Type	Max Length (Bytes)	Direction
@course_code	int	4	
@description	varchar(255)	255	
@course_title	varchar(255)	255	
@credits	varchar(255)	255	
@course_id	int	4	Out

SQL Script

```
SET QUOTED_IDENTIFIER OFF
GO
CREATE   PROCEDURE [dbo].[AddCourse]
    @course_code INT,
    @description VARCHAR(255),
    @course_title VARCHAR(255),
    @credits VARCHAR(255),
    @course_id INT OUTPUT
AS
BEGIN
    BEGIN TRY
        BEGIN TRANSACTION
            IF NOT EXISTS (SELECT 1 FROM courses WHERE course_code = @course_code)
            BEGIN
                INSERT INTO courses (course_code, description, course_title, credits)
                VALUES (@course_code, @description, @course_title, @credits);

                SET @course_id = SCOPE_IDENTITY();
            END
            ELSE
            BEGIN
                DECLARE @error_message VARCHAR(100) = 'Course with code ' + CAST(@course_code AS
VARCHAR(20)) + ' already exists!';
                THROW 50001, @error_message, 1;
            END
        COMMIT TRANSACTION;
    END TRY
    BEGIN CATCH
        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION;
        THROW;
    END CATCH
END
GO
```

Uses

[dbo].[courses]

[dbo].[AddCourseToDepartment]

Properties

Property	Value
ANSI Nulls On	True
Quoted Identifier On	False

Parameters

Name	Data Type	Max Length (Bytes)
@CourseID	int	4
@DepartmentID	int	4

SQL Script

```
SET QUOTED_IDENTIFIER OFF
GO
-- =============================================
-- Procedure: AddCourseToDepartment
-- Description: Links a course to a department.
-- =============================================
CREATE PROCEDURE [dbo].[AddCourseToDepartment]
    @CourseID INT,
    @DepartmentID INT
AS
BEGIN
    SET NOCOUNT ON;

    -- Validate CourseID
    IF @CourseID IS NULL
    BEGIN
        RAISERROR('Course ID is required', 16, 1);
        RETURN -1;
    END

    -- Validate DepartmentID
    IF @DepartmentID IS NULL
    BEGIN
        RAISERROR('Department ID is required', 16, 1);
        RETURN -1;
    END

    -- Check if course exists
    IF NOT EXISTS (SELECT 1 FROM courses WHERE course_id = @CourseID)
    BEGIN
        RAISERROR('Course not found', 16, 1);
        RETURN -2;
    END

    -- Check if department exists
    IF NOT EXISTS (SELECT 1 FROM department WHERE department_id = @DepartmentID)
    BEGIN
        RAISERROR('Department not found', 16, 1);
        RETURN -2;
    END

    -- Check if already linked
    IF EXISTS (
        SELECT 1
        FROM department_courses
        WHERE course_id = @CourseID
          AND department_id = @DepartmentID
    )
    BEGIN
        RAISERROR('Course is already linked to this department', 16, 1);
        RETURN -3;
    END

    BEGIN TRY
        INSERT INTO department_courses (course_id, department_id)
        VALUES (@CourseID, @DepartmentID);

        SELECT 'Course added to department successfully' AS Message,
               @CourseID AS CourseID,
               @DepartmentID AS DepartmentID;
        RETURN 0;
    END TRY
    BEGIN CATCH
        DECLARE @ErrorMessage NVARCHAR(4000) = ERROR_MESSAGE();
        RAISERROR(@ErrorMessage, 16, 1);
        RETURN -4;
    END CATCH
END
GO
```

Uses

[dbo].[courses]
[dbo].[department]
[dbo].[department_courses]

[dbo].[AddDepartment]

Properties

Property	Value
ANSI Nulls On	True
Quoted Identifier On	False

Parameters

Name	Data Type	Max Length (Bytes)
@name	varchar(255)	255

SQL Script

```
SET QUOTED_IDENTIFIER OFF
GO
CREATE PROCEDURE [dbo].[AddDepartment](@name VARCHAR(255))
AS
BEGIN
    BEGIN TRY
        BEGIN TRANSACTION
            IF NOT EXISTS(SELECT * FROM department WHERE department_name = @name COLLATE SQL_-
Latin1_General_CP1_CI_AS)
            BEGIN
                INSERT INTO department(department_name)
                VALUES(@name);
            END
            ELSE
            BEGIN
                DECLARE @error_message varchar(50) = 'Department ' + @name + ' is already
exists!';
                THROW 50010, @error_message, 1;
            END
        COMMIT TRANSACTION;
    END TRY
    BEGIN CATCH
        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION;
        THROW;
    END CATCH
END

EXEC AddDepartment 'PWD'
GO
```

Uses

[dbo].[department]

[dbo].[AddDepartmentToBranch]

Properties

Property	Value
ANSI Nulls On	True
Quoted Identifier On	False

Parameters

Name	Data Type	Max Length (Bytes)
@branch_id	int	4
@department_id	int	4

SQL Script

```sql
SET QUOTED_IDENTIFIER OFF
GO
CREATE PROCEDURE [dbo].[AddDepartmentToBranch](@branch_id INT, @department_id INT)
AS
BEGIN
    BEGIN TRY
        BEGIN TRANSACTION
        DECLARE @error_message VARCHAR(50);
        IF NOT EXISTS(SELECT * FROM branch_department
        WHERE branch_id = @branch_id AND department_id = @department_id)
        BEGIN
            INSERT INTO branch_department(branch_id, department_id)
            VALUES(@branch_id, @department_id);
        END
        ELSE
        BEGIN
            SET @error_message = 'This department is already exists in this branch';
            THROW 50010, @error_message, 1;
        END
        COMMIT TRANSACTION;
    END TRY
    BEGIN CATCH
        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION;
        THROW;
    END CATCH
END

EXEC AddDepartmentToBranch 1, 1
```

```
GO
```

Uses

[dbo].[branch_department]

[dbo].[AddInstructor]

Properties

Property	Value
ANSI Nulls On	True
Quoted Identifier On	False

Parameters

Name	Data Type	Max Length (Bytes)
@FirstName	varchar(255)	255
@LastName	varchar(255)	255
@Email	varchar(255)	255

SQL Script

```
SET QUOTED_IDENTIFIER OFF
GO
CREATE PROCEDURE [dbo].[AddInstructor]
    @FirstName VARCHAR(255),
    @LastName VARCHAR(255),
    @Email VARCHAR(255)
AS
BEGIN
    SET NOCOUNT ON;
    BEGIN TRANSACTION;
    BEGIN TRY
        -- Validate input parameters
        IF @FirstName IS NULL OR LTRIM(RTRIM(@FirstName)) = ''
        BEGIN
            RAISERROR('First name cannot be empty', 16, 1);
            ROLLBACK TRANSACTION;
            RETURN;
        END

        IF @LastName IS NULL OR LTRIM(RTRIM(@LastName)) = ''
        BEGIN
            RAISERROR('Last name cannot be empty', 16, 1);
            ROLLBACK TRANSACTION;
            RETURN;
        END

        IF @Email IS NULL OR LTRIM(RTRIM(@Email)) = ''
        BEGIN
```

```
            RAISERROR('Email cannot be empty', 16, 1);
            ROLLBACK TRANSACTION;
            RETURN;
        END



        -- Check if email already exists
        IF EXISTS (SELECT 1 FROM [person] WHERE [email] = @Email)
        BEGIN
            RAISERROR('Email already exists in the system', 16, 1);
            ROLLBACK TRANSACTION;
            RETURN;
        END

        DECLARE @NewPersonId INT;

        -- Insert into person table
        INSERT INTO [person] ([first_name], [last_name], [email])
        VALUES (@FirstName, @LastName, @Email);

        SET @NewPersonId = SCOPE_IDENTITY();

        -- Insert into instructor table
        INSERT INTO [instructor] ([instructor_id], [hire_date])
        VALUES (@NewPersonId, GETDATE());

        COMMIT TRANSACTION;

        -- Return the new instructor ID
        SELECT @NewPersonId AS InstructorId, 'Instructor created successfully' AS Message;
    END TRY
    BEGIN CATCH
        ROLLBACK TRANSACTION;
        DECLARE @ErrorMessage NVARCHAR(4000) = ERROR_MESSAGE();
        DECLARE @ErrorSeverity INT = ERROR_SEVERITY();
        DECLARE @ErrorState INT = ERROR_STATE();
        RAISERROR(@ErrorMessage, @ErrorSeverity, @ErrorState);
    END CATCH
END
GO
```

Uses

[dbo].[instructor]
[dbo].[person]

[dbo].[AddStudent]

Properties

Property	Value
ANSI Nulls On	True
Quoted Identifier On	False

Parameters

Name	Data Type	Max Length (Bytes)
@FirstName	varchar(255)	255
@LastName	varchar(255)	255
@Email	varchar(255)	255

SQL Script

```
SET QUOTED_IDENTIFIER OFF
GO
CREATE PROCEDURE [dbo].[AddStudent]
    @FirstName VARCHAR(255),
    @LastName VARCHAR(255),
    @Email VARCHAR(255)
AS
BEGIN
    SET NOCOUNT ON;
    BEGIN TRANSACTION;
    BEGIN TRY
        -- Validate input parameters
        IF @FirstName IS NULL OR LTRIM(RTRIM(@FirstName)) = ''
        BEGIN
            RAISERROR('First name cannot be empty', 16, 1);
            ROLLBACK TRANSACTION;
            RETURN;
        END

        IF @LastName IS NULL OR LTRIM(RTRIM(@LastName)) = ''
        BEGIN
            RAISERROR('Last name cannot be empty', 16, 1);
            ROLLBACK TRANSACTION;
            RETURN;
        END

        IF @Email IS NULL OR LTRIM(RTRIM(@Email)) = ''
        BEGIN
```

```
            RAISERROR('Email cannot be empty', 16, 1);
            ROLLBACK TRANSACTION;
            RETURN;
        END

        -- Check if email already exists
        IF EXISTS (SELECT 1 FROM [person] WHERE [email] = @Email)
        BEGIN
            RAISERROR('Email already exists in the system', 16, 1);
            ROLLBACK TRANSACTION;
            RETURN;
        END

        DECLARE @NewPersonId INT;

        -- Insert into person table (correct column names with underscores)
        INSERT INTO [person] ([first_name], [last_name], [email])
        VALUES (@FirstName, @LastName, @Email);

        SET @NewPersonId = SCOPE_IDENTITY();

        -- Insert into student table (correct column name with underscore)
        INSERT INTO [student] ([student_id])
        VALUES (@NewPersonId);

        COMMIT TRANSACTION;

        -- Return the new student ID
        SELECT @NewPersonId AS StudentId, 'Student created successfully' AS Message;
    END TRY
    BEGIN CATCH
        ROLLBACK TRANSACTION;
        DECLARE @ErrorMessage NVARCHAR(4000) = ERROR_MESSAGE();
        DECLARE @ErrorSeverity INT = ERROR_SEVERITY();
        DECLARE @ErrorState INT = ERROR_STATE();
        RAISERROR(@ErrorMessage, @ErrorSeverity, @ErrorState);
    END CATCH
END
GO
```

Uses

[dbo].[person]
[dbo].[student]

[dbo].[AddTopic]

Properties

Property	Value
ANSI Nulls On	True
Quoted Identifier On	False

Parameters

Name	Data Type	Max Length (Bytes)	Direction
@topic_order	int	4	
@topic_duration	varchar(255)	255	
@topic_title	varchar(255)	255	
@topic_id	int	4	Out

SQL Script

```
SET QUOTED_IDENTIFIER OFF
GO
CREATE   PROCEDURE [dbo].[AddTopic]
    @topic_order INT,
    @topic_duration VARCHAR(255),
    @topic_title VARCHAR(255),
    @topic_id INT OUTPUT
AS
BEGIN
    BEGIN TRY
        BEGIN TRANSACTION
            INSERT INTO topic (topic_order, topic_duration, topic_title)
            VALUES (@topic_order, @topic_duration, @topic_title);

            SET @topic_id = SCOPE_IDENTITY();
        COMMIT TRANSACTION;
    END TRY
    BEGIN CATCH
        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION;
        THROW;
    END CATCH
END
GO
```

Uses

[dbo].[topic]

[dbo].[AssignInstructorToCourse]

Properties

Property	Value
ANSI Nulls On	True
Quoted Identifier On	False

Parameters

Name	Data Type	Max Length (Bytes)
@InstructorID	int	4
@CourseID	int	4

SQL Script

```
SET QUOTED_IDENTIFIER OFF
GO
-- =============================================
-- Procedure: AssignInstructorToCourse
-- Description: Assigns an instructor to a course.
--              Required for instructors to generate exams.
-- =============================================
CREATE PROCEDURE [dbo].[AssignInstructorToCourse]
    @InstructorID INT,
    @CourseID INT
AS
BEGIN
    SET NOCOUNT ON;

    -- Validate InstructorID
    IF @InstructorID IS NULL
    BEGIN
        RAISERROR('Instructor ID is required', 16, 1);
        RETURN -1;
    END

    -- Validate CourseID
    IF @CourseID IS NULL
    BEGIN
        RAISERROR('Course ID is required', 16, 1);
        RETURN -1;
    END

    -- Check if instructor exists
    IF NOT EXISTS (SELECT 1 FROM instructor WHERE instructor_id = @InstructorID)
    BEGIN
        RAISERROR('Instructor not found', 16, 1);
        RETURN -2;
    END

    -- Check if course exists
    IF NOT EXISTS (SELECT 1 FROM courses WHERE course_id = @CourseID)
    BEGIN
        RAISERROR('Course not found', 16, 1);
        RETURN -2;
    END

    -- Check if already assigned
    IF EXISTS (
        SELECT 1
        FROM instructor_course
        WHERE instructor_id = @InstructorID
          AND course_id = @CourseID
    )
    BEGIN
        RAISERROR('Instructor is already assigned to this course', 16, 1);
        RETURN -3;
    END

    BEGIN TRY
        INSERT INTO instructor_course (instructor_id, course_id)
        VALUES (@InstructorID, @CourseID);

        SELECT 'Instructor assigned to course successfully' AS Message,
               @InstructorID AS InstructorID,
               @CourseID AS CourseID;
        RETURN 0;
    END TRY
    BEGIN CATCH
        DECLARE @ErrorMessage NVARCHAR(4000) = ERROR_MESSAGE();
        RAISERROR(@ErrorMessage, 16, 1);
        RETURN -4;
    END CATCH
END
GO
```

Uses

[dbo].[courses]
[dbo].[instructor]
[dbo].[instructor_course]

[dbo].[CorrectExam]

Properties

Property	Value
ANSI Nulls On	True
Quoted Identifier On	False

Parameters

Name	Data Type	Max Length (Bytes)
@exam_id	int	4
@student_id	int	4

SQL Script

```
SET QUOTED_IDENTIFIER OFF
GO
CREATE PROCEDURE [dbo].[CorrectExam]
(
    @exam_id INT,
    @student_id INT
)
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRY
        BEGIN TRANSACTION;

        IF NOT EXISTS
        (
            SELECT 1
            FROM student_exam
            WHERE exam_id = @exam_id
              AND student_id = @student_id
        )
        BEGIN
            THROW 50001, 'Student has not submitted this exam.', 1;
        END;

         IF EXISTS
        (
            SELECT 1
            FROM student_exam
            WHERE exam_id = @exam_id
              AND student_id = @student_id
              AND grade IS NOT NULL
              AND state = 'Graded'
        )
        BEGIN
            THROW 50002, 'Exam already corrected for this student.', 1;
        END;

        DECLARE @total_grade INT;

        SELECT @total_grade = e.total_grade
        FROM Exam e
        WHERE exam_id = @exam_id;

        DECLARE @correct_answers INT;

        SELECT @correct_answers = COUNT(*)
        FROM student_answer_question sa
        INNER JOIN question q
            ON sa.quesiotn_id = q.question_id
        WHERE sa.exam_id = @exam_id
          AND sa.student_id = @student_id
          AND sa.student_answer = q.correct_ans_id;

        DECLARE @final_score DECIMAL(5,2);

        SET @final_score =
            CAST(@correct_answers AS DECIMAL(5,2)) / @total_grade * 100;

        UPDATE student_exam
        SET grade = @final_score
        , state = 'Graded'
        WHERE exam_id = @exam_id
        AND student_id = @student_id


        COMMIT TRANSACTION;

        SELECT
            @exam_id AS exam_id,
            @student_id AS student_id,
            @total_grade AS total_questions,
            @correct_answers AS correct_answers,
            @final_score AS final_score;

        RETURN @final_score;

    END TRY
    BEGIN CATCH
        ROLLBACK TRANSACTION;
        THROW;
    END CATCH
```

```
END;
GO
```

Uses

[dbo].[exam]
[dbo].[question]
[dbo].[student_answer_question]
[dbo].[student_exam]

Used By

[dbo].[StudentSubmitAnswers]

[dbo].[DeleteBranch]

Properties

Property	Value
ANSI Nulls On	True
Quoted Identifier On	False

Parameters

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

SQL Script

```
SET QUOTED_IDENTIFIER OFF
GO
CREATE PROCEDURE [dbo].[DeleteBranch](@id INT)
AS
BEGIN
    BEGIN TRY
        BEGIN TRANSACTION
            DELETE
            FROM branch_department
            WHERE branch_id = @id;

            DELETE
            FROM branch
            WHERE branch_id = @id;
            IF @@ROWCOUNT = 0
            BEGIN
                DECLARE @error_message VARCHAR(50) = CONCAT('Sorry branch with id ', @id, ' does
not exist!');
                THROW 50010, @error_message, 1;
            END
        COMMIT TRANSACTION;
    END TRY
    BEGIN CATCH
        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION;
        THROW;
    END CATCH
END

EXEC DeleteBranch 3
GO
```

Uses

[dbo].[branch]
[dbo].[branch_department]

[dbo].[DeleteChoice]

Properties

Property	Value
ANSI Nulls On	True
Quoted Identifier On	False

Parameters

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

SQL Script

```
SET QUOTED_IDENTIFIER OFF
GO
CREATE PROCEDURE [dbo].[DeleteChoice]
    @ChoiceID INT
AS
BEGIN
    SET NOCOUNT ON;

    IF @ChoiceID IS NULL
        RETURN -1;

    DECLARE @TrueChoiceID INT;
    DECLARE @FalseChoiceID INT;

    SELECT @TrueChoiceID = choice_id FROM quesiton_choice WHERE LOWER(choice_text) = 'true';
    SELECT @FalseChoiceID = choice_id FROM quesiton_choice WHERE LOWER(choice_text) = 'false';

    IF (@TrueChoiceID IS NOT NULL AND @ChoiceID = @TrueChoiceID) OR
       (@FalseChoiceID IS NOT NULL AND @ChoiceID = @FalseChoiceID)
        RETURN -2;

    IF NOT EXISTS (SELECT 1 FROM quesiton_choice WHERE choice_id = @ChoiceID)
        RETURN -3;


    IF EXISTS (SELECT 1 FROM question WHERE correct_ans_id = @ChoiceID)
        RETURN -5;

    BEGIN TRY
        BEGIN TRANSACTION;
        -- Step 1: Delete from bridge table
        DELETE FROM question_choise_bridge
        WHERE choice_id = @ChoiceID;

        -- Step 2: Delete from choice table
        DELETE FROM quesiton_choice
        WHERE choice_id = @ChoiceID;

        COMMIT TRANSACTION;

        RETURN 0;
    END TRY
    BEGIN CATCH
        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION;

        RETURN -4;
    END CATCH
END
GO
```

Uses

[dbo].[quesiton_choice]
[dbo].[question]
[dbo].[question_choise_bridge]

[dbo].[DeleteCourse]

Properties

Property	Value
ANSI Nulls On	True
Quoted Identifier On	False

Parameters

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

SQL Script

```
SET QUOTED_IDENTIFIER OFF
GO
CREATE   PROCEDURE [dbo].[DeleteCourse]
    @course_id INT
AS
BEGIN
    BEGIN TRY
        BEGIN TRANSACTION
            IF EXISTS (SELECT 1 FROM courses WHERE course_id = @course_id)
            BEGIN
                -- Delete dependencies first
                DELETE FROM department_courses WHERE course_id = @course_id;
                DELETE FROM student_course WHERE course_id = @course_id;
                DELETE FROM instructor_course WHERE course_id = @course_id;
                DELETE FROM instructor_generate_course_exam WHERE course_id = @course_id;
                DELETE FROM course_questions_on_topic WHERE course_id = @course_id;

                -- Delete the course
                DELETE FROM courses WHERE course_id = @course_id;
            END
            ELSE
            BEGIN
                DECLARE @error_message VARCHAR(100) = 'Course with ID ' + CAST(@course_id AS
VARCHAR(20)) + ' does not exist!';
                THROW 50004, @error_message, 1;
            END
        COMMIT TRANSACTION;
    END TRY
    BEGIN CATCH
        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION;
        THROW;
    END CATCH
END
GO
```

Uses

[dbo].[course_questions_on_topic]
[dbo].[courses]
[dbo].[department_courses]
[dbo].[instructor_course]
[dbo].[instructor_generate_course_exam]
[dbo].[student_course]

[dbo].[DeleteDepartment]

Properties

Property	Value
ANSI Nulls On	True
Quoted Identifier On	False

Parameters

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

SQL Script

```
SET QUOTED_IDENTIFIER OFF
GO
CREATE PROCEDURE [dbo].[DeleteDepartment](@id INT)
AS
BEGIN
    BEGIN TRY
        BEGIN TRANSACTION
            DELETE FROM department_courses
            WHERE department_id = @id;

            DELETE FROM branch_department
            WHERE department_id = @id;

            DELETE FROM
            person_jong_department_branch
            WHERE department_id = @id AND person_id IN
            (SELECT student_id FROM student);

            DELETE FROM department
            WHERE department_id = @id;
        COMMIT TRANSACTION;
    END TRY
    BEGIN CATCH
        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION;
        THROW;
    END CATCH
END
GO
```

Uses

[dbo].[branch_department]
[dbo].[department]
[dbo].[department_courses]
[dbo].[person_jong_department_branch]
[dbo].[student]

[dbo].[DeleteExam]

Properties

Property	Value
ANSI Nulls On	True
Quoted Identifier On	False

Parameters

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

SQL Script

```
SET QUOTED_IDENTIFIER OFF
GO
-- =============================================
-- Procedure: DeleteExam
-- Description: Deletes an exam and all related records.
--              Use with caution as this removes all student answers and grades.
-- =============================================
CREATE PROCEDURE [dbo].[DeleteExam]
    @ExamID INT
AS
BEGIN
    SET NOCOUNT ON;

    -- Validate ExamID
    IF @ExamID IS NULL OR @ExamID <= 0
    BEGIN
        RAISERROR('Valid Exam ID is required', 16, 1);
        RETURN -1;
    END

    -- Check if exam exists
    IF NOT EXISTS (SELECT 1 FROM exam WHERE exam_id = @ExamID)
    BEGIN
        RAISERROR('Exam not found', 16, 1);
        RETURN -2;
    END

    BEGIN TRY
        BEGIN TRANSACTION;

        -- Delete student answers for this exam
        DELETE FROM student_answer_question
        WHERE exam_id = @ExamID;

        -- Delete student exam records
        DELETE FROM student_exam
        WHERE exam_id = @ExamID;

        -- Delete exam questions associations
        DELETE FROM exam_questions
        WHERE exam_id = @ExamID;

        -- Delete instructor-exam-course association
        DELETE FROM instructor_generate_course_exam
        WHERE exam_id = @ExamID;

        -- Delete the exam itself
        DELETE FROM exam
        WHERE exam_id = @ExamID;

        COMMIT TRANSACTION;

        SELECT 'Exam deleted successfully' AS Message,
               @ExamID AS DeletedExamID;
        RETURN 0;
    END TRY
    BEGIN CATCH
        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION;

        DECLARE @ErrorMessage NVARCHAR(4000) = ERROR_MESSAGE();
        RAISERROR(@ErrorMessage, 16, 1);
        RETURN -3;
    END CATCH
END
GO
```

Uses

[dbo].[exam]
[dbo].[exam_questions]
[dbo].[instructor_generate_course_exam]
[dbo].[student_answer_question]
[dbo].[student_exam]

[dbo].[DeleteInstructor]

Properties

Property	Value
ANSI Nulls On	True
Quoted Identifier On	False

Parameters

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

SQL Script

```
SET QUOTED_IDENTIFIER OFF
GO
CREATE PROCEDURE [dbo].[DeleteInstructor]
    @InstructorId INT
AS
BEGIN
    SET NOCOUNT ON;
    BEGIN TRANSACTION;
    BEGIN TRY
        -- Validate InstructorId
        IF @InstructorId IS NULL OR @InstructorId <= 0
        BEGIN
            RAISERROR('Invalid Instructor ID', 16, 1);
            ROLLBACK TRANSACTION;
            RETURN;
        END

        -- Check if instructor exists
        IF NOT EXISTS (SELECT 1 FROM [instructor] WHERE [instructor_id] = @InstructorId)
        BEGIN
            RAISERROR('Instructor not found', 16, 1);
            ROLLBACK TRANSACTION;
            RETURN;
        END

        -- Check for related records in instructor_course
        IF EXISTS (SELECT 1 FROM [instructor_course] WHERE [instructor_id] = @InstructorId)
        BEGIN
            RAISERROR('Cannot delete instructor: Instructor is assigned to courses', 16, 1);
            ROLLBACK TRANSACTION;
            RETURN;
        END

        -- Check for related records in instructor_generate_course_exam
        IF EXISTS (SELECT 1 FROM [instructor_generate_course_exam] WHERE [instructor_id] =
@InstructorId)
        BEGIN
            RAISERROR('Cannot delete instructor: Instructor has generated exams', 16, 1);
            ROLLBACK TRANSACTION;
            RETURN;
        END

        -- Delete from instructor table first (FK constraint)
        DELETE FROM [instructor] WHERE [instructor_id] = @InstructorId;

        -- Delete from person table
        DELETE FROM [person] WHERE [person_id] = @InstructorId;

        COMMIT TRANSACTION;

        SELECT 'Instructor deleted successfully' AS Message;
    END TRY
    BEGIN CATCH
        ROLLBACK TRANSACTION;
        DECLARE @ErrorMessage NVARCHAR(4000) = ERROR_MESSAGE();
        DECLARE @ErrorSeverity INT = ERROR_SEVERITY();
        DECLARE @ErrorState INT = ERROR_STATE();
        RAISERROR(@ErrorMessage, @ErrorSeverity, @ErrorState);
    END CATCH
END
GO
```

Uses

[dbo].[instructor]
[dbo].[instructor_course]
[dbo].[instructor_generate_course_exam]
[dbo].[person]

[dbo].[DeleteQuestion]

Properties

Property	Value
ANSI Nulls On	True
Quoted Identifier On	False

Parameters

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

SQL Script

```
SET QUOTED_IDENTIFIER OFF
GO

CREATE PROCEDURE [dbo].[DeleteQuestion]
    @QuestionID INT
AS
BEGIN
    SET NOCOUNT ON;

    IF @QuestionID IS NULL
        RETURN -1;

    IF NOT EXISTS (SELECT 1 FROM question WHERE question_id = @QuestionID)
        RETURN -2;

    BEGIN TRY
        BEGIN TRANSACTION;

        DELETE FROM question_choise_bridge
        WHERE question_id = @QuestionID;

        DELETE FROM question
        WHERE question_id = @QuestionID;

        COMMIT TRANSACTION;

        RETURN 0;
    END TRY
    BEGIN CATCH
        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION;
        RETURN -3;
    END CATCH
END
GO
```

Uses

[dbo].[question]
[dbo].[question_choise_bridge]

[dbo].[DeleteStudent]

Properties

Property	Value
ANSI Nulls On	True
Quoted Identifier On	False

Parameters

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

SQL Script

```
SET QUOTED_IDENTIFIER OFF
GO
CREATE PROCEDURE [dbo].[DeleteStudent]
    @StudentId INT
AS
BEGIN
    SET NOCOUNT ON;
    BEGIN TRANSACTION;
    BEGIN TRY
        -- Validate StudentId
        IF @StudentId IS NULL OR @StudentId <= 0
        BEGIN
            RAISERROR('Invalid Student ID', 16, 1);
            ROLLBACK TRANSACTION;
            RETURN;
        END

        -- Check if student exists
        IF NOT EXISTS (SELECT 1 FROM [student] WHERE [student_id] = @StudentId)
        BEGIN
            RAISERROR('Student not found', 16, 1);
            ROLLBACK TRANSACTION;
            RETURN;
        END

        -- Check for related records ??


        -- Delete from student table first (FK constraint)
        DELETE FROM [student] WHERE [student_id] = @StudentId;
        -- Delete from person table
        DELETE FROM [person] WHERE [person_id] = @StudentId;

        COMMIT TRANSACTION;

        SELECT 'Student deleted successfully' AS Message;
    END TRY
    BEGIN CATCH
        ROLLBACK TRANSACTION;
        DECLARE @ErrorMessage NVARCHAR(4000) = ERROR_MESSAGE();
        DECLARE @ErrorSeverity INT = ERROR_SEVERITY();
        DECLARE @ErrorState INT = ERROR_STATE();
        RAISERROR(@ErrorMessage, @ErrorSeverity, @ErrorState);
    END CATCH
END
GO
```

Uses

[dbo].[person]
[dbo].[student]

[dbo].[DeleteTopic]

Properties

Property	Value
ANSI Nulls On	True
Quoted Identifier On	False

Parameters

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

SQL Script

```
SET QUOTED_IDENTIFIER OFF
GO
CREATE   PROCEDURE [dbo].[DeleteTopic]
    @topic_id INT
AS
BEGIN
    BEGIN TRY
        BEGIN TRANSACTION
            IF EXISTS (SELECT 1 FROM topic WHERE topic_id = @topic_id)
            BEGIN
                -- Delete dependencies
                DELETE FROM course_questions_on_topic WHERE topic_id = @topic_id;

                -- Delete topic
                DELETE FROM topic WHERE topic_id = @topic_id;
            END
            ELSE
            BEGIN
                DECLARE @error_message VARCHAR(100) = 'Topic with ID ' + CAST(@topic_id AS
VARCHAR(20)) + ' does not exist!';
                THROW 50006, @error_message, 1;
            END
        COMMIT TRANSACTION;
    END TRY
    BEGIN CATCH
        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION;
        THROW;
    END CATCH
END
GO
```

Uses

[dbo].[course_questions_on_topic]
[dbo].[topic]

[dbo].[EnrollStudentInCourse]

Properties

Property	Value
ANSI Nulls On	True
Quoted Identifier On	False

Parameters

Name	Data Type	Max Length (Bytes)
@StudentID	int	4
@CourseID	int	4
@EnrollmentDate	datetime	8

SQL Script

```
SET QUOTED_IDENTIFIER OFF
GO
-- =============================================
-- Procedure: EnrollStudentInCourse
-- Description: Enrolls a student in a course.
--              Required for students to submit exam answers.
-- =============================================
CREATE PROCEDURE [dbo].[EnrollStudentInCourse]
    @StudentID INT,
    @CourseID INT,
    @EnrollmentDate DATETIME = NULL
AS
BEGIN
    SET NOCOUNT ON;

    -- Set default enrollment date to now if not provided
    IF @EnrollmentDate IS NULL
        SET @EnrollmentDate = GETDATE();

    -- Validate StudentID
    IF @StudentID IS NULL
    BEGIN
        RAISERROR('Student ID is required', 16, 1);
        RETURN -1;
    END

    -- Validate CourseID
    IF @CourseID IS NULL
    BEGIN
        RAISERROR('Course ID is required', 16, 1);
        RETURN -1;
    END

    -- Check if student exists
    IF NOT EXISTS (SELECT 1 FROM student WHERE student_id = @StudentID)
    BEGIN
        RAISERROR('Student not found', 16, 1);
        RETURN -2;
    END

    -- Check if course exists
    IF NOT EXISTS (SELECT 1 FROM courses WHERE course_id = @CourseID)
    BEGIN
        RAISERROR('Course not found', 16, 1);
        RETURN -2;
    END

    -- Check if already enrolled
    IF EXISTS (
        SELECT 1
        FROM student_course
        WHERE student_id = @StudentID
          AND course_id = @CourseID
    )
    BEGIN
        RAISERROR('Student is already enrolled in this course', 16, 1);
        RETURN -3;
    END

    BEGIN TRY
        INSERT INTO student_course (student_id, course_id, enrollment_date)
        VALUES (@StudentID, @CourseID, @EnrollmentDate);

        SELECT 'Student enrolled in course successfully' AS Message,
               @StudentID AS StudentID,
               @CourseID AS CourseID,
               @EnrollmentDate AS EnrollmentDate;
        RETURN 0;
    END TRY
    BEGIN CATCH
        DECLARE @ErrorMessage NVARCHAR(4000) = ERROR_MESSAGE();
        RAISERROR(@ErrorMessage, 16, 1);
        RETURN -4;
    END CATCH
END
GO
```

Uses

[dbo].[courses]
[dbo].[student]
[dbo].[student_course]

[dbo].[GetAllExams]

Properties

Property	Value
ANSI Nulls On	True
Quoted Identifier On	False

Parameters

Name	Data Type	Max Length (Bytes)
@CourseID	int	4
@ExamType	varchar(50)	50
@InstructorID	int	4

SQL Script

```
SET QUOTED_IDENTIFIER OFF
GO
-- =============================================
-- Procedure: GetAllExams
-- Description: Retrieves all exams with optional filtering.
-- =============================================
CREATE PROCEDURE [dbo].[GetAllExams]
    @CourseID INT = NULL,
    @ExamType VARCHAR(50) = NULL,
    @InstructorID INT = NULL
AS
BEGIN
    SET NOCOUNT ON;

    SELECT
        e.exam_id,
        e.exam_title,
        e.total_grade,
        e.exam_date,
        e.exam_type,
        e.duration_mins,
        igce.course_id,
        c.course_title,
        igce.instructor_id,
        p.first_name + ' ' + p.last_name AS instructor_name
    FROM exam e
    LEFT JOIN instructor_generate_course_exam igce ON e.exam_id = igce.exam_id
    LEFT JOIN courses c ON igce.course_id = c.course_id
    LEFT JOIN person p ON igce.instructor_id = p.person_id
    WHERE (@CourseID IS NULL OR igce.course_id = @CourseID)
      AND (@ExamType IS NULL OR e.exam_type = @ExamType)
      AND (@InstructorID IS NULL OR igce.instructor_id = @InstructorID)
    ORDER BY e.exam_date DESC;

    RETURN 0;
END
GO
```

Uses

[dbo].[courses]
[dbo].[exam]
[dbo].[instructor_generate_course_exam]
[dbo].[person]

[dbo].[GetAllInstructors]

Properties

Property	Value
ANSI Nulls On	True
Quoted Identifier On	False

SQL Script

```
SET QUOTED_IDENTIFIER OFF
GO
CREATE PROCEDURE [dbo].[GetAllInstructors]
AS
BEGIN
    SET NOCOUNT ON;

    SELECT
        i.[instructor_id],
        p.[first_name],
        p.[last_name],
        p.[email],
        i.[hire_date]
    FROM [instructor] i
    INNER JOIN [person] p ON i.[instructor_id] = p.[person_id];
END
GO
```

Uses

[dbo].[instructor]
[dbo].[person]

[dbo].[GetAllQuestionsWithChoicesPivoted]

Properties

Property	Value
ANSI Nulls On	True
Quoted Identifier On	False
Encrypted	True

SQL Script

```
SET QUOTED_IDENTIFIER OFF
GO

-- =============================================
-- Procedure: GetAllQuestionsWithChoicesPivoted
-- Description: Returns ALL questions with their choices in a single row each
--              Perfect for bulk report generation
-- Returns:
--   0  : Success (returns result set of all questions)
-- =============================================
CREATE PROCEDURE [dbo].[GetAllQuestionsWithChoicesPivoted]
WITH ENCRYPTION
AS
BEGIN
    SET NOCOUNT ON;

    SELECT
        q.question_id AS QuestionID,
        q.question_text AS QuestionText,
        q.question_type AS QuestionType,
        q.question_difficulty AS Difficulty,
        q.correct_ans_id AS CorrectAnswerID,
        MAX(CASE WHEN rn = 1 THEN c.choice_id END) AS Choice1_ID,
        MAX(CASE WHEN rn = 1 THEN c.choice_text END) AS Choice1_Text,
        MAX(CASE WHEN rn = 2 THEN c.choice_id END) AS Choice2_ID,
        MAX(CASE WHEN rn = 2 THEN c.choice_text END) AS Choice2_Text,
        MAX(CASE WHEN rn = 3 THEN c.choice_id END) AS Choice3_ID,
        MAX(CASE WHEN rn = 3 THEN c.choice_text END) AS Choice3_Text,
        MAX(CASE WHEN rn = 4 THEN c.choice_id END) AS Choice4_ID,
        MAX(CASE WHEN rn = 4 THEN c.choice_text END) AS Choice4_Text
    FROM question q
    LEFT JOIN (
        SELECT
            qcb.question_id,
            qc.choice_id,
            qc.choice_text,
            ROW_NUMBER() OVER (PARTITION BY qcb.question_id ORDER BY qc.choice_id) AS rn
        FROM question_choise_bridge qcb
        JOIN quesiton_choice qc ON qc.choice_id = qcb.choice_id
    ) c ON c.question_id = q.question_id
    GROUP BY
        q.question_id,
        q.question_text,
        q.question_type,
        q.question_difficulty,
        q.correct_ans_id
    ORDER BY q.question_id;

    RETURN 0;
END
GO
```

Uses

[dbo].[quesiton_choice]
[dbo].[question]
[dbo].[question_choise_bridge]

[dbo].[GetAllStudents]

Properties

Property	Value
ANSI Nulls On	True
Quoted Identifier On	False

SQL Script

```
SET QUOTED_IDENTIFIER OFF
GO
CREATE PROCEDURE [dbo].[GetAllStudents]
AS
BEGIN
    SET NOCOUNT ON;

    SELECT
        s.[student_id],
        p.[first_name],
        p.[last_name],
        p.[email]
    FROM [student] s
    INNER JOIN [person] p ON s.[student_id] = p.[person_id];
END
GO
```

Uses

[dbo].[person]
[dbo].[student]

[dbo].[GetBranchById]

Properties

Property	Value
ANSI Nulls On	True
Quoted Identifier On	False

Parameters

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

SQL Script

```sql
SET QUOTED_IDENTIFIER OFF
GO
CREATE PROCEDURE [dbo].[GetBranchById](@id INT)
AS
BEGIN
    SELECT branch_name AS [Branch Name], branch_city AS [Branch City]
    FROM branch
    WHERE branch_id = @id;
    IF @@ROWCOUNT = 0
    BEGIN
        DECLARE @error_message VARCHAR(50) = CONCAT('Branch with id ', @id, ' does not exist!');
        THROW 50010, @error_message, 1;
    END
END

EXEC GetBranchById 1
GO
```

Uses

[dbo].[branch]

[dbo].[GetChoicesByQuestionId]

Properties

Property	Value
ANSI Nulls On	True
Quoted Identifier On	False
Encrypted	True

Parameters

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

SQL Script

```
SET QUOTED_IDENTIFIER OFF
GO

CREATE PROCEDURE [dbo].[GetChoicesByQuestionId]
    @QuestionID INT
WITH ENCRYPTION
AS
BEGIN
    SET NOCOUNT ON;

    IF @QuestionID IS NULL
        RETURN -1;

    IF NOT EXISTS (SELECT 1 FROM question WHERE question_id = @QuestionID)
        RETURN -2;

    SELECT
        qc.choice_id,
        qc.choice_text
    FROM quesiton_choice qc
    JOIN question_choise_bridge qcb
        ON qc.choice_id = qcb.choice_id
    WHERE qcb.question_id = @QuestionID;

    RETURN 0;
END

GO
```

Uses

[dbo].[quesiton_choice]
[dbo].[question]
[dbo].[question_choise_bridge]

Used By

[dbo].[GetQuestionWithChoices]

[dbo].[GetCourseById]

Properties

Property	Value
ANSI Nulls On	True
Quoted Identifier On	False

Parameters

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

SQL Script

```
SET QUOTED_IDENTIFIER OFF
GO
CREATE   PROCEDURE [dbo].[GetCourseById]
    @course_id INT = NULL
AS
BEGIN
    SELECT course_id, course_code, description, course_title, credits
    FROM courses
    WHERE @course_id IS NULL OR course_id = @course_id;
END
GO
```

Uses

[dbo].[courses]

[dbo].[GetCourseTopics]

Properties

Property	Value
ANSI Nulls On	True
Quoted Identifier On	False

Parameters

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

SQL Script

```sql
SET QUOTED_IDENTIFIER OFF
GO
-- =============================================
-- Procedure: GetCourseTopics
-- Description: Returns all topics studied in a specific course
--              Useful for course outline reports and curriculum display
-- Parameters:
--   @CourseID INT - The course to retrieve topics for
-- Returns:
--   0  : Success (returns result set of topics)
--  -1  : CourseID is NULL
--  -2  : Course not found
--  -99 : Database error
-- =============================================
CREATE PROCEDURE [dbo].[GetCourseTopics]
    @CourseID INT
AS
BEGIN
    SET NOCOUNT ON;

    -- Validate input
    IF @CourseID IS NULL
    BEGIN
        RAISERROR('Course ID cannot be NULL', 16, 1);
        RETURN -1;
    END;

    -- Check if course exists
    IF NOT EXISTS (SELECT 1 FROM courses WHERE course_id = @CourseID)
    BEGIN
        RAISERROR('Course not found', 16, 1);
        RETURN -2;
    END;

    BEGIN TRY
        -- Return course topics with enriched data
        SELECT DISTINCT
            c.course_id AS CourseID,
            c.course_title AS CourseTitle,
            c.course_code AS CourseCode,
            c.description AS CourseDescription,
            c.credits AS Credits,
            t.topic_id AS TopicID,
            t.topic_title AS TopicTitle,
            t.topic_order AS TopicOrder,
            t.topic_duration AS TopicDuration,
            -- Calculated fields for report display
            (SELECT COUNT(DISTINCT topic_id)
             FROM course_questions_on_topic
             WHERE course_id = @CourseID) AS TotalTopics,
            CONCAT('Topic ', t.topic_order, ' of ',
                   (SELECT COUNT(DISTINCT topic_id)
                    FROM course_questions_on_topic
                    WHERE course_id = @CourseID)) AS TopicProgress
        FROM courses c
        INNER JOIN course_questions_on_topic cqt ON cqt.course_id = c.course_id
        INNER JOIN topic t ON t.topic_id = cqt.topic_id
        WHERE c.course_id = @CourseID
        ORDER BY t.topic_order;

        RETURN 0;

    END TRY
    BEGIN CATCH
        DECLARE @ErrorMessage NVARCHAR(4000) = ERROR_MESSAGE();
        DECLARE @ErrorSeverity INT = ERROR_SEVERITY();
        DECLARE @ErrorState INT = ERROR_STATE();

        RAISERROR(@ErrorMessage, @ErrorSeverity, @ErrorState);
        RETURN -99;
    END CATCH
END;
GO
```

Uses

[dbo].[course_questions_on_topic]
[dbo].[courses]
[dbo].[topic]

[dbo].[GetDepartmentById]

Properties

Property	Value
ANSI Nulls On	True
Quoted Identifier On	False

Parameters

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

SQL Script

```
SET QUOTED_IDENTIFIER OFF
GO
CREATE PROCEDURE [dbo].[GetDepartmentById](@id INT)
AS
BEGIN
    SELECT department_name AS [Department Name]
    FROM department
    WHERE department_id = @id;
    IF @@ROWCOUNT = 0
    BEGIN
        DECLARE @error_message VARCHAR(50) = CONCAT('Department with id ', @id, ' does not
exist!');
        THROW 50010, @error_message, 1;
    END
END

EXEC GetDepartmentById 100
GO
```

Uses

[dbo].[department]

[dbo].[GetDepartmentCourses]

Properties

Property	Value
ANSI Nulls On	True
Quoted Identifier On	False

Parameters

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

SQL Script

```
SET QUOTED_IDENTIFIER OFF
GO
-- =============================================
-- Procedure: GetDepartmentCourses
-- Description: Retrieves all courses linked to a department.
-- =============================================
CREATE PROCEDURE [dbo].[GetDepartmentCourses]
    @DepartmentID INT
AS
BEGIN
    SET NOCOUNT ON;

    -- Validate DepartmentID
    IF @DepartmentID IS NULL OR @DepartmentID <= 0
    BEGIN
        RAISERROR('Valid Department ID is required', 16, 1);
        RETURN -1;
    END

    -- Check if department exists
    IF NOT EXISTS (SELECT 1 FROM department WHERE department_id = @DepartmentID)
    BEGIN
        RAISERROR('Department not found', 16, 1);
        RETURN -2;
    END

    SELECT
        c.course_id,
        c.course_code,
        c.course_title,
        c.description,
        c.credits
    FROM department_courses dc
    INNER JOIN courses c ON dc.course_id = c.course_id
    WHERE dc.department_id = @DepartmentID
    ORDER BY c.course_title;

    RETURN 0;
END
GO
```

Uses

[dbo].[courses]
[dbo].[department]
[dbo].[department_courses]

[dbo].[GetExamById]

Properties

Property	Value
ANSI Nulls On	True
Quoted Identifier On	False

Parameters

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

SQL Script

```
SET QUOTED_IDENTIFIER OFF
GO
-- =============================================
-- Procedure: GetExamById
-- Description: Retrieves exam details by ID.
-- =============================================
CREATE PROCEDURE [dbo].[GetExamById]
    @ExamID INT
AS
BEGIN
    SET NOCOUNT ON;

    -- Validate ExamID
    IF @ExamID IS NULL OR @ExamID <= 0
    BEGIN
        RAISERROR('Valid Exam ID is required', 16, 1);
        RETURN -1;
    END

    -- Check if exam exists
    IF NOT EXISTS (SELECT 1 FROM exam WHERE exam_id = @ExamID)
    BEGIN
        RAISERROR('Exam not found', 16, 1);
        RETURN -2;
    END

    SELECT
        e.exam_id,
        e.exam_title,
        e.total_grade,
        e.exam_date,
        e.exam_type,
        e.duration_mins,
        igce.course_id,
        c.course_title,
        igce.instructor_id,
        p.first_name + ' ' + p.last_name AS instructor_name
    FROM exam e
    LEFT JOIN instructor_generate_course_exam igce ON e.exam_id = igce.exam_id
    LEFT JOIN courses c ON igce.course_id = c.course_id
    LEFT JOIN person p ON igce.instructor_id = p.person_id
    WHERE e.exam_id = @ExamID;

    RETURN 0;
END
GO
```

Uses

[dbo].[courses]
[dbo].[exam]
[dbo].[instructor_generate_course_exam]
[dbo].[person]

[dbo].[GetExamQuestions]

Properties

Property	Value
ANSI Nulls On	True
Quoted Identifier On	False

Parameters

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

SQL Script

```
SET QUOTED_IDENTIFIER OFF
GO
-- =============================================
-- Procedure: GetExamQuestions
-- Description: Retrieves all questions for an exam.
-- =============================================
CREATE PROCEDURE [dbo].[GetExamQuestions]
    @ExamID INT
AS
BEGIN
    SET NOCOUNT ON;

    -- Validate ExamID
    IF @ExamID IS NULL OR @ExamID <= 0
    BEGIN
        RAISERROR('Valid Exam ID is required', 16, 1);
        RETURN -1;
    END

    -- Check if exam exists
    IF NOT EXISTS (SELECT 1 FROM exam WHERE exam_id = @ExamID)
    BEGIN
        RAISERROR('Exam not found', 16, 1);
        RETURN -2;
    END

    -- Return exam questions with their choices
    SELECT
        q.question_id,
        q.question_text,
        q.question_type,
        q.question_difficulty,
        qc.choice_id,
        qc.choice_text,
        CASE WHEN q.correct_ans_id = qc.choice_id THEN 1 ELSE 0 END AS is_correct
    FROM exam_questions eq
    INNER JOIN question q ON eq.questoin_id = q.question_id
    LEFT JOIN question_choise_bridge qcb ON q.question_id = qcb.question_id
    LEFT JOIN quesiton_choice qc ON qcb.choice_id = qc.choice_id
    WHERE eq.exam_id = @ExamID
    ORDER BY q.question_id, qc.choice_id;

    RETURN 0;
END
GO
```

Uses

[dbo].[exam]
[dbo].[exam_questions]
[dbo].[quesiton_choice]
[dbo].[question]
[dbo].[question_choise_bridge]

[dbo].[GetExamQuestionsWithChoicesPivoted]

Properties

Property	Value
ANSI Nulls On	True
Quoted Identifier On	False
Encrypted	True

Parameters

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

SQL Script

```
SET QUOTED_IDENTIFIER OFF
GO
-- =============================================
-- Procedure: GetExamQuestionsWithChoicesPivoted
-- Description: Returns all questions for a specific exam with their
--              choices pivoted into columns (single row per question)
--              Ideal for generating printable exam papers/reports
-- Parameters:
--   @ExamID INT - The exam to retrieve questions for
-- Returns:
--   0  : Success
--  -1  : ExamID is NULL
--  -2  : Exam not found
-- =============================================
CREATE PROCEDURE [dbo].[GetExamQuestionsWithChoicesPivoted]
    @ExamID INT
WITH ENCRYPTION
AS
BEGIN
    SET NOCOUNT ON;

    -- Validate input
    IF @ExamID IS NULL
        RETURN -1;

    IF NOT EXISTS (SELECT 1 FROM exam WHERE exam_id = @ExamID)
        RETURN -2;

    -- Return all questions for this exam with choices pivoted into columns
    SELECT
        e.exam_id AS ExamID,
        e.exam_title AS ExamTitle,
        q.question_id AS QuestionID,
        q.question_text AS QuestionText,
        q.question_type AS QuestionType,
        q.question_difficulty AS Difficulty,
        q.correct_ans_id AS CorrectAnswerID,
        MAX(CASE WHEN rn = 1 THEN c.choice_id END) AS Choice1_ID,
        MAX(CASE WHEN rn = 1 THEN c.choice_text END) AS Choice1_Text,
        MAX(CASE WHEN rn = 2 THEN c.choice_id END) AS Choice2_ID,
        MAX(CASE WHEN rn = 2 THEN c.choice_text END) AS Choice2_Text,
        MAX(CASE WHEN rn = 3 THEN c.choice_id END) AS Choice3_ID,
        MAX(CASE WHEN rn = 3 THEN c.choice_text END) AS Choice3_Text,
        MAX(CASE WHEN rn = 4 THEN c.choice_id END) AS Choice4_ID,
        MAX(CASE WHEN rn = 4 THEN c.choice_text END) AS Choice4_Text
    FROM exam e
    JOIN exam_questions eq ON eq.exam_id = e.exam_id
    JOIN question q ON q.question_id = eq.questoin_id
    LEFT JOIN (
        SELECT
            qcb.question_id,
            qc.choice_id,
            qc.choice_text,
            ROW_NUMBER() OVER (PARTITION BY qcb.question_id ORDER BY qc.choice_id) AS rn
        FROM question_choise_bridge qcb
        JOIN quesiton_choice qc ON qc.choice_id = qcb.choice_id
    ) c ON c.question_id = q.question_id
    WHERE e.exam_id = @ExamID
    GROUP BY
        e.exam_id,
        e.exam_title,
        q.question_id,
        q.question_text,
        q.question_type,
        q.question_difficulty,
        q.correct_ans_id
    ORDER BY q.question_id;

    RETURN 0;
END
GO
```

Uses

[dbo].[exam]
[dbo].[exam_questions]
[dbo].[quesiton_choice]
[dbo].[question]
[dbo].[question_choise_bridge]

[dbo].[GetInstructorById]

Properties

Property	Value
ANSI Nulls On	True
Quoted Identifier On	False

Parameters

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

SQL Script

```
SET QUOTED_IDENTIFIER OFF
GO
CREATE PROCEDURE [dbo].[GetInstructorById]
    @InstructorId INT
AS
BEGIN
    SET NOCOUNT ON;

    -- Validate input parameter
    IF @InstructorId IS NULL OR @InstructorId <= 0
    BEGIN
        RAISERROR('Invalid Instructor ID', 16, 1);
        RETURN;
    END

    SELECT
        i.[instructor_id],
        p.[first_name],
        p.[last_name],
        p.[email],
        i.[hire_date]
    FROM [instructor] i
    INNER JOIN [person] p ON i.[instructor_id] = p.[person_id]
    WHERE i.[instructor_id] = @InstructorId;

    -- Check if instructor exists
    IF @@ROWCOUNT = 0
    BEGIN
        RAISERROR('Instructor not found', 16, 1);
        RETURN;
    END
```

```
END
GO
```

Uses

[dbo].[instructor]
[dbo].[person]

[dbo].[GetInstructorCourses]

Properties

Property	Value
ANSI Nulls On	True
Quoted Identifier On	False

Parameters

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

SQL Script

```
SET QUOTED_IDENTIFIER OFF
GO
-- =============================================
-- Procedure: GetInstructorCourses
-- Description: Retrieves all courses assigned to an instructor.
-- =============================================
CREATE PROCEDURE [dbo].[GetInstructorCourses]
    @InstructorID INT
AS
BEGIN
    SET NOCOUNT ON;

    -- Validate InstructorID
    IF @InstructorID IS NULL OR @InstructorID <= 0
    BEGIN
        RAISERROR('Valid Instructor ID is required', 16, 1);
        RETURN -1;
    END

    -- Check if instructor exists
    IF NOT EXISTS (SELECT 1 FROM instructor WHERE instructor_id = @InstructorID)
    BEGIN
        RAISERROR('Instructor not found', 16, 1);
        RETURN -2;
    END

    SELECT
        c.course_id,
        c.course_code,
        c.course_title,
        c.description,
        c.credits
    FROM instructor_course ic
    INNER JOIN courses c ON ic.course_id = c.course_id
    WHERE ic.instructor_id = @InstructorID
    ORDER BY c.course_title;

    RETURN 0;
END
GO
```

Uses

[dbo].[courses]
[dbo].[instructor]
[dbo].[instructor_course]

[dbo].[GetQuestionsForCourseTopic]

Properties

Property	Value
ANSI Nulls On	True
Quoted Identifier On	False

Parameters

Name	Data Type	Max Length (Bytes)
@CourseID	int	4
@TopicID	int	4
@QuestionType	varchar(50)	50

SQL Script

```
SET QUOTED_IDENTIFIER OFF
GO
-- =============================================
-- Procedure: GetQuestionsForCourseTopic
-- Description: Retrieves all questions linked to a specific course and optionally a topic.
-- =============================================
CREATE PROCEDURE [dbo].[GetQuestionsForCourseTopic]
    @CourseID INT,
    @TopicID INT = NULL,
    @QuestionType VARCHAR(50) = NULL
AS
BEGIN
    SET NOCOUNT ON;

    -- Validate CourseID
    IF @CourseID IS NULL
    BEGIN
        RAISERROR('Course ID is required', 16, 1);
        RETURN -1;
    END

    -- Check if course exists
    IF NOT EXISTS (SELECT 1 FROM courses WHERE course_id = @CourseID)
    BEGIN
        RAISERROR('Course not found', 16, 1);
        RETURN -2;
    END
    -- If TopicID is provided, check if it exists
    IF @TopicID IS NOT NULL AND NOT EXISTS (SELECT 1 FROM topic WHERE topic_id = @TopicID)
    BEGIN
        RAISERROR('Topic not found', 16, 1);
        RETURN -2;
    END

    SELECT
        q.question_id,
        q.question_text,
        q.question_type,
        q.question_difficulty,
        q.correct_ans_id,
        cqt.topic_id,
        t.topic_title,
        cqt.course_id
    FROM question q
    INNER JOIN course_questions_on_topic cqt
        ON q.question_id = cqt.question_id
    INNER JOIN topic t
        ON cqt.topic_id = t.topic_id
    WHERE cqt.course_id = @CourseID
      AND (@TopicID IS NULL OR cqt.topic_id = @TopicID)
      AND (@QuestionType IS NULL OR q.question_type = @QuestionType)
    ORDER BY t.topic_order, q.question_id;

    RETURN 0;
END
GO
```

Uses

[dbo].[course_questions_on_topic]
[dbo].[courses]
[dbo].[question]
[dbo].[topic]

[dbo].[GetQuestionWithChoices]

Properties

Property	Value
ANSI Nulls On	True
Quoted Identifier On	False
Encrypted	True

Parameters

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

SQL Script

```
SET QUOTED_IDENTIFIER OFF
GO

CREATE PROCEDURE [dbo].[GetQuestionWithChoices]
    @QuestionID INT
WITH ENCRYPTION
AS
BEGIN
    SET NOCOUNT ON;

    IF @QuestionID IS NULL
        RETURN -1;

    IF NOT EXISTS (SELECT 1 FROM question WHERE question_id = @QuestionID)
        RETURN -2;

    SELECT
        q.question_id,
        q.question_text,
        q.question_type,
        q.question_difficulty,
        q.correct_ans_id
    FROM question q
    WHERE q.question_id = @QuestionID;

    -- make sure here to use the correct schema name
    EXEC dbo.GetChoicesByQuestionId @QuestionID = @QuestionID;

    RETURN 0;
```

```
END
GO
```

Uses

[dbo].[question]
[dbo].[GetChoicesByQuestionId]

[dbo].[GetQuestionWithChoicesPivoted]

Properties

Property	Value
ANSI Nulls On	True
Quoted Identifier On	False
Encrypted	True

Parameters

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

SQL Script

```
SET QUOTED_IDENTIFIER OFF
GO
-- =============================================
-- Procedure: GetQuestionWithChoicesPivoted
-- Description: Returns question details with all choices in a single row
--              Useful for report generation where each question needs
--              to display with its choices in columns (Ch1, Ch2, Ch3, Ch4)
-- Returns:
--    0  : Success
--   -1  : QuestionID is NULL
--   -2  : Question not found
-- =============================================
CREATE PROCEDURE [dbo].[GetQuestionWithChoicesPivoted]
    @QuestionID INT
WITH ENCRYPTION
AS
BEGIN
    SET NOCOUNT ON;

    -- Validate input
    IF @QuestionID IS NULL
        RETURN -1;

    IF NOT EXISTS (SELECT 1 FROM question WHERE question_id = @QuestionID)
        RETURN -2;

    -- Return question with choices pivoted into columns
    -- Uses ROW_NUMBER to assign choice positions (Ch1, Ch2, Ch3, Ch4)
    SELECT
```

```
        q.question_id AS QuestionID,
        q.question_text AS QuestionText,
        q.question_type AS QuestionType,
        q.question_difficulty AS Difficulty,
        q.correct_ans_id AS CorrectAnswerID,
        MAX(CASE WHEN rn = 1 THEN c.choice_id END) AS Choice1_ID,
        MAX(CASE WHEN rn = 1 THEN c.choice_text END) AS Choice1_Text,
        MAX(CASE WHEN rn = 2 THEN c.choice_id END) AS Choice2_ID,
        MAX(CASE WHEN rn = 2 THEN c.choice_text END) AS Choice2_Text,
        MAX(CASE WHEN rn = 3 THEN c.choice_id END) AS Choice3_ID,
        MAX(CASE WHEN rn = 3 THEN c.choice_text END) AS Choice3_Text,
        MAX(CASE WHEN rn = 4 THEN c.choice_id END) AS Choice4_ID,
        MAX(CASE WHEN rn = 4 THEN c.choice_text END) AS Choice4_Text
    FROM question q
    LEFT JOIN (
        SELECT
            qcb.question_id,
            qc.choice_id,
            qc.choice_text,
            ROW_NUMBER() OVER (PARTITION BY qcb.question_id ORDER BY qc.choice_id) AS rn
        FROM question_choise_bridge qcb
        JOIN quesiton_choice qc ON qc.choice_id = qcb.choice_id
    ) c ON c.question_id = q.question_id
    WHERE q.question_id = @QuestionID
    GROUP BY
        q.question_id,
        q.question_text,
        q.question_type,
        q.question_difficulty,
        q.correct_ans_id;

    RETURN 0;
END
GO
```

Uses

[dbo].[quesiton_choice]
[dbo].[question]
[dbo].[question_choise_bridge]

[dbo].[GetStudentById]

Properties

Property	Value
ANSI Nulls On	True
Quoted Identifier On	False

Parameters

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

SQL Script

```
SET QUOTED_IDENTIFIER OFF
GO
CREATE PROCEDURE [dbo].[GetStudentById]
    @StudentId INT
AS
BEGIN
    SET NOCOUNT ON;

    -- Validate input parameter
    IF @StudentId IS NULL OR @StudentId <= 0
    BEGIN
        RAISERROR('Invalid Student ID', 16, 1);
        RETURN;
    END

    SELECT
        s.[student_id],
        p.[first_name],
        p.[last_name],
        p.[email]
    FROM [student] s
    INNER JOIN [person] p ON s.[student_id] = p.[person_id]
    WHERE s.[student_id] = @StudentId;

    -- Check if student exists
    IF @@ROWCOUNT = 0
    BEGIN
        RAISERROR('Student not found', 16, 1);
        RETURN;
    END
END
```

```
GO
```

Uses

[dbo].[person]
[dbo].[student]

[dbo].[GetStudentCourses]

Properties

Property	Value
ANSI Nulls On	True
Quoted Identifier On	False

Parameters

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

SQL Script

```
SET QUOTED_IDENTIFIER OFF
GO
-- =============================================
-- Procedure: GetStudentCourses
-- Description: Retrieves all courses a student is enrolled in.
-- =============================================
CREATE PROCEDURE [dbo].[GetStudentCourses]
    @StudentID INT
AS
BEGIN
    SET NOCOUNT ON;

    -- Validate StudentID
    IF @StudentID IS NULL OR @StudentID <= 0
    BEGIN
        RAISERROR('Valid Student ID is required', 16, 1);
        RETURN -1;
    END

    -- Check if student exists
    IF NOT EXISTS (SELECT 1 FROM student WHERE student_id = @StudentID)
    BEGIN
        RAISERROR('Student not found', 16, 1);
        RETURN -2;
    END

    SELECT
        c.course_id,
        c.course_code,
        c.course_title,
        c.description,
        c.credits,
        sc.enrollment_date
    FROM student_course sc
    INNER JOIN courses c ON sc.course_id = c.course_id
    WHERE sc.student_id = @StudentID
    ORDER BY sc.enrollment_date DESC;

    RETURN 0;
END
GO
```

Uses

[dbo].[courses]
[dbo].[student]
[dbo].[student_course]

[dbo].[GetTopicById]

Properties

Property	Value
ANSI Nulls On	True
Quoted Identifier On	False

Parameters

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

SQL Script

```
SET QUOTED_IDENTIFIER OFF
GO
CREATE   PROCEDURE [dbo].[GetTopicById]
    @topic_id INT = NULL
AS
BEGIN
    SELECT topic_id, topic_order, topic_duration, topic_title
    FROM topic
    WHERE @topic_id IS NULL OR topic_id = @topic_id;
END
GO
```

Uses

[dbo].[topic]

[dbo].[InsertMCQChoice]

Properties

Property	Value
ANSI Nulls On	True
Quoted Identifier On	False
Encrypted	True

Parameters

Name	Data Type	Max Length (Bytes)	Direction
@QuestionID	int	4	
@ChoiceText	varchar(255)	255	
@IsCorrect	bit	1	
@NewChoiceID	int	4	Out

SQL Script

```
SET QUOTED_IDENTIFIER OFF
GO
CREATE PROCEDURE [dbo].[InsertMCQChoice]
    @QuestionID INT,
    @ChoiceText VARCHAR(255),
    @IsCorrect BIT,
    @NewChoiceID INT OUTPUT
WITH ENCRYPTION
AS
BEGIN
    SET NOCOUNT ON;
    SET @NewChoiceID = NULL;


    IF @QuestionID IS NULL
        RETURN -1;

    IF @ChoiceText IS NULL OR LTRIM(RTRIM(@ChoiceText)) = ''
        RETURN -1;

    IF @IsCorrect IS NULL
        RETURN -1;

    IF NOT EXISTS (SELECT 1 FROM question WHERE question_id = @QuestionID)
        RETURN -2;  -- Question not found
    IF NOT EXISTS (SELECT 1 FROM question WHERE question_id = @QuestionID AND question_type =
'MCQ')
        RETURN -3;  -- Wrong question type (not MCQ)


    BEGIN TRY
        BEGIN TRANSACTION;

        INSERT INTO quesiton_choice(choice_text)
        VALUES (LTRIM(RTRIM(@ChoiceText)));

        SET @NewChoiceID = SCOPE_IDENTITY();

        IF @IsCorrect = 1
        BEGIN
            UPDATE question
            SET correct_ans_id = @NewChoiceID
            WHERE question_id = @QuestionID;
        END

        INSERT INTO question_choise_bridge(question_id, choice_id)
        VALUES (@QuestionID, @NewChoiceID);

        COMMIT TRANSACTION;

        RETURN 0;
    END TRY
    BEGIN CATCH
        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION;
        RETURN -4;  -- Database error
    END CATCH
END
GO
```

Uses

[dbo].[quesiton_choice]
[dbo].[question]
[dbo].[question_choise_bridge]

[dbo].[InsertQuestion]

Properties

Property	Value
ANSI Nulls On	True
Quoted Identifier On	False
Encrypted	True

Parameters

Name	Data Type	Max Length (Bytes)	Direction
@QuestionText	varchar(255)	255	
@QuestionType	varchar(50)	50	
@QuestionDifficulty	varchar(50)	50	
@NewQuestionID	int	4	Out

SQL Script

```
SET QUOTED_IDENTIFIER OFF
GO
CREATE PROCEDURE [dbo].[InsertQuestion]
    @QuestionText VARCHAR(255),
    @QuestionType VARCHAR(50),
    @QuestionDifficulty VARCHAR(50),
    @NewQuestionID INT OUTPUT
WITH ENCRYPTION
AS
BEGIN
    SET NOCOUNT ON;
    SET @NewQuestionID = NULL;

    IF @QuestionText IS NULL OR LTRIM(RTRIM(@QuestionText)) = ''
    BEGIN
        SELECT NULL AS QuestionID, 0 AS Success, 'Question text is required' AS Message;
        RETURN -1;
    END

    IF @QuestionType IS NULL
    BEGIN
        SELECT NULL AS QuestionID, 0 AS Success, 'Question type is required' AS Message;
        RETURN -1;
    END
    IF @QuestionDifficulty IS NULL
    BEGIN
        SELECT NULL AS QuestionID, 0 AS Success, 'Question difficulty is required' AS Message;
        RETURN -1;
    END

    BEGIN TRY
        INSERT INTO question (question_text, question_type, question_difficulty, correct_ans_id)
        VALUES (LTRIM(RTRIM(@QuestionText)), @QuestionType, @QuestionDifficulty, NULL);

        SET @NewQuestionID = SCOPE_IDENTITY();

        SELECT @NewQuestionID AS QuestionID, 1 AS Success, 'Created successfully' AS Message;
        RETURN 0;
    END TRY
    BEGIN CATCH
        SELECT NULL AS QuestionID, 0 AS Success, ERROR_MESSAGE() AS Message;
        RETURN -1;
    END CATCH
END

-- example of usage from DB

DECLARE @QuestionId INT;

EXECUTE InsertQuestion
    @QuestionText = 'What is 1+1?',
    @QuestionType = 'MCQ',
    @QuestionDifficulty = 'easy',
    @NewQuestionID = @QuestionId OUTPUT;


SELECT @QuestionId AS 'Question ID';
GO
```

Uses

[dbo].[question]

[dbo].[InsertTrueFalseChoices]

Properties

Property	Value
ANSI Nulls On	True
Quoted Identifier On	False
Encrypted	True

Parameters

Name	Data Type	Max Length (Bytes)
@QuestionID	int	4
@CorrectAnswer	bit	1

SQL Script

```
SET QUOTED_IDENTIFIER OFF
GO
CREATE PROCEDURE [dbo].[InsertTrueFalseChoices]
    @QuestionID INT,
    @CorrectAnswer BIT  -- 1 = True is correct, 0 = False is correct
WITH ENCRYPTION
AS
BEGIN
    SET NOCOUNT ON;

    IF @QuestionID IS NULL
        RETURN -1;

    IF @CorrectAnswer IS NULL
        RETURN -1;

    IF NOT EXISTS (SELECT 1 FROM question WHERE question_id = @QuestionID)
        RETURN -2;  -- Question not found

    -- Check if question is True_False type
    IF NOT EXISTS (SELECT 1 FROM question WHERE question_id = @QuestionID AND question_type =
'True_False')
        RETURN -3;  -- Wrong question type

    BEGIN TRY
        -- Get True/False choice IDs dynamically from quesiton_choice
        DECLARE @TrueChoiceID INT;
        DECLARE @FalseChoiceID INT;
        DECLARE @CorrectChoiceID INT;

        SELECT @TrueChoiceID = choice_id FROM quesiton_choice WHERE LOWER(choice_text) = 'true';
        SELECT @FalseChoiceID = choice_id FROM quesiton_choice WHERE LOWER(choice_text) =
'false';

        IF @TrueChoiceID IS NULL OR @FalseChoiceID IS NULL
            RETURN -4; -- True/False choice entries missing

        SET @CorrectChoiceID = CASE WHEN @CorrectAnswer = 1 THEN @TrueChoiceID ELSE @FalseChoice-
ID END;

        UPDATE question
        SET correct_ans_id = @CorrectChoiceID
        WHERE question_id = @QuestionID;

        INSERT INTO question_choise_bridge (question_id, choice_id)
        VALUES (@QuestionID, @TrueChoiceID);

        INSERT INTO question_choise_bridge (question_id, choice_id)
        VALUES (@QuestionID, @FalseChoiceID);

        RETURN 0;  -- Success
    END TRY
    BEGIN CATCH
        RETURN -5;  -- Database error
    END CATCH
END
GO
```

Uses

[dbo].[quesiton_choice]
[dbo].[question]
[dbo].[question_choise_bridge]

[dbo].[LinkQuestionToCourseTopic]

Properties

Property	Value
ANSI Nulls On	True
Quoted Identifier On	False

Parameters

Name	Data Type	Max Length (Bytes)
@QuestionID	int	4
@CourseID	int	4
@TopicID	int	4

SQL Script

```
SET QUOTED_IDENTIFIER OFF
GO
-- =============================================
-- Procedure: LinkQuestionToCourseTopic
-- Description: Links a question to a specific course and topic.
--              This is required for the GenerateExam procedure to find questions.
-- =============================================
CREATE PROCEDURE [dbo].[LinkQuestionToCourseTopic]
    @QuestionID INT,
    @CourseID INT,
    @TopicID INT
AS
BEGIN
    SET NOCOUNT ON;

    -- Validate QuestionID
    IF @QuestionID IS NULL
    BEGIN
        RAISERROR('Question ID is required', 16, 1);
        RETURN -1;
    END

    -- Validate CourseID
    IF @CourseID IS NULL
    BEGIN
        RAISERROR('Course ID is required', 16, 1);
        RETURN -1;
    END
```

```
-- Validate TopicID
IF @TopicID IS NULL
BEGIN
    RAISERROR('Topic ID is required', 16, 1);
    RETURN -1;
END

-- Check if question exists
IF NOT EXISTS (SELECT 1 FROM question WHERE question_id = @QuestionID)
BEGIN
    RAISERROR('Question not found', 16, 1);
    RETURN -2;
END

-- Check if course exists
IF NOT EXISTS (SELECT 1 FROM courses WHERE course_id = @CourseID)
BEGIN
    RAISERROR('Course not found', 16, 1);
    RETURN -2;
END

-- Check if topic exists
IF NOT EXISTS (SELECT 1 FROM topic WHERE topic_id = @TopicID)
BEGIN
    RAISERROR('Topic not found', 16, 1);
    RETURN -2;
END

-- Check if the link already exists
IF EXISTS (
    SELECT 1
    FROM course_questions_on_topic
    WHERE course_id = @CourseID
      AND question_id = @QuestionID
      AND topic_id = @TopicID
)
BEGIN
    RAISERROR('This question is already linked to the specified course and topic', 16, 1);
    RETURN -3;
END

BEGIN TRY
    INSERT INTO course_questions_on_topic (course_id, question_id, topic_id)
    VALUES (@CourseID, @QuestionID, @TopicID);

    SELECT 'Question linked to course and topic successfully' AS Message;
    RETURN 0;
END TRY
BEGIN CATCH
    DECLARE @ErrorMessage NVARCHAR(4000) = ERROR_MESSAGE();
    RAISERROR(@ErrorMessage, 16, 1);
    RETURN -4;
    END CATCH
END
GO
```

Uses

[dbo].[course_questions_on_topic]
[dbo].[courses]
[dbo].[question]
[dbo].[topic]

[dbo].[RemoveCourseFromDepartment]

Properties

Property	Value
ANSI Nulls On	True
Quoted Identifier On	False

Parameters

Name	Data Type	Max Length (Bytes)
@CourseID	int	4
@DepartmentID	int	4

SQL Script

```
SET QUOTED_IDENTIFIER OFF
GO
-- =============================================
-- Procedure: RemoveCourseFromDepartment
-- Description: Removes the link between a course and a department.
-- =============================================
CREATE PROCEDURE [dbo].[RemoveCourseFromDepartment]
    @CourseID INT,
    @DepartmentID INT
AS
BEGIN
    SET NOCOUNT ON;

    -- Validate CourseID
    IF @CourseID IS NULL
    BEGIN
        RAISERROR('Course ID is required', 16, 1);
        RETURN -1;
    END

    -- Validate DepartmentID
    IF @DepartmentID IS NULL
    BEGIN
        RAISERROR('Department ID is required', 16, 1);
        RETURN -1;
    END

    -- Check if link exists
    IF NOT EXISTS (
        SELECT 1
        FROM department_courses
        WHERE course_id = @CourseID
          AND department_id = @DepartmentID
    )
    BEGIN
        RAISERROR('Link not found between specified course and department', 16, 1);
        RETURN -2;
    END

    BEGIN TRY
        DELETE FROM department_courses
        WHERE course_id = @CourseID
          AND department_id = @DepartmentID;

        SELECT 'Course removed from department successfully' AS Message;
        RETURN 0;
    END TRY
    BEGIN CATCH
        DECLARE @ErrorMessage NVARCHAR(4000) = ERROR_MESSAGE();
        RAISERROR(@ErrorMessage, 16, 1);
        RETURN -3;
    END CATCH
END
GO
```

Uses

[dbo].[department_courses]

[dbo].[RemoveInstructorFromCourse]

Properties

Property	Value
ANSI Nulls On	True
Quoted Identifier On	False

Parameters

Name	Data Type	Max Length (Bytes)
@InstructorID	int	4
@CourseID	int	4

SQL Script

```
SET QUOTED_IDENTIFIER OFF
GO
-- =============================================
-- Procedure: RemoveInstructorFromCourse
-- Description: Removes an instructor's assignment from a course.
-- =============================================
CREATE PROCEDURE [dbo].[RemoveInstructorFromCourse]
    @InstructorID INT,
    @CourseID INT
AS
BEGIN
    SET NOCOUNT ON;

    -- Validate InstructorID
    IF @InstructorID IS NULL
    BEGIN
        RAISERROR('Instructor ID is required', 16, 1);
        RETURN -1;
    END

    -- Validate CourseID
    IF @CourseID IS NULL
    BEGIN
        RAISERROR('Course ID is required', 16, 1);
        RETURN -1;
    END

    -- Check if assignment exists
    IF NOT EXISTS (
        SELECT 1
        FROM instructor_course
        WHERE instructor_id = @InstructorID
          AND course_id = @CourseID
    )
    BEGIN
        RAISERROR('Assignment not found', 16, 1);
        RETURN -2;
    END

    -- Check if instructor has generated exams for this course
    IF EXISTS (
        SELECT 1
        FROM instructor_generate_course_exam
        WHERE instructor_id = @InstructorID
          AND course_id = @CourseID
    )
    BEGIN
        RAISERROR('Cannot remove: Instructor has generated exams for this course', 16, 1);
        RETURN -3;
    END

    BEGIN TRY
        DELETE FROM instructor_course
        WHERE instructor_id = @InstructorID
          AND course_id = @CourseID;

        SELECT 'Instructor removed from course successfully' AS Message;
        RETURN 0;
    END TRY
    BEGIN CATCH
        DECLARE @ErrorMessage NVARCHAR(4000) = ERROR_MESSAGE();
        RAISERROR(@ErrorMessage, 16, 1);
        RETURN -4;
    END CATCH
END
GO
```

Uses

[dbo].[instructor_course]
[dbo].[instructor_generate_course_exam]

[dbo].[sp_GetExamForStudent]

Properties

Property	Value
ANSI Nulls On	True
Quoted Identifier On	False

Parameters

Name	Data Type	Max Length (Bytes)
@student_id	int	4
@exam_id	int	4

SQL Script

```
SET QUOTED_IDENTIFIER OFF
GO
-- =============================================
--   Return codes:
--    0  : Success
--   -1  : Invalid parameters (NULL values)
--   -2  : Exam not found
--   -3  : Student not enrolled in course
--   -4  : Student has not submitted this exam yet
-- =============================================

CREATE PROCEDURE [dbo].[sp_GetExamForStudent]
(
    @student_id INT,
    @exam_id INT
)
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRY
        -- Validate input parameters
        IF @student_id IS NULL OR @exam_id IS NULL
        BEGIN
            RAISERROR('Student ID and Exam ID cannot be NULL.', 16, 1);
            RETURN -1;
        END;

        -- Check if exam exists
        IF NOT EXISTS (SELECT 1 FROM exam WHERE exam_id = @exam_id)
        BEGIN
            RAISERROR('Exam not found.', 16, 1);
            RETURN -2;
        END;

        -- Get the course_id for this exam
        DECLARE @course_id INT;

        SELECT @course_id = course_id
        FROM instructor_generate_course_exam
        WHERE exam_id = @exam_id;

        IF @course_id IS NULL
        BEGIN
            RAISERROR('Exam is not associated with any course.', 16, 1);
            RETURN -2;
        END;

        -- Check if student is enrolled in the course
        IF NOT EXISTS (
            SELECT 1
            FROM student_course
            WHERE student_id = @student_id
              AND course_id = @course_id
        )
        BEGIN
            RAISERROR('Student is not enrolled in the course for this exam.', 16, 1);
            RETURN -3;
        END;

        -- Check if student has submitted this exam
        IF NOT EXISTS (
            SELECT 1
            FROM student_exam
            WHERE student_id = @student_id
              AND exam_id = @exam_id
              AND state = 'Submitted'
        )
        BEGIN
            RAISERROR('Student has not submitted this exam yet.', 16, 1);
            RETURN -4;
        END;

        -- All validations passed, return exam questions with choices, student answers, and
correct answers
        SELECT
            e.exam_id AS ExamID,
            e.exam_title AS ExamTitle,
            e.total_grade AS TotalGrade,
            e.exam_date AS ExamDate,
            e.duration_mins AS DurationMinutes,
            se.grade AS StudentGrade,
            se.state AS ExamState,
            CONCAT(CAST(se.grade AS VARCHAR), ' / ', CAST(e.total_grade AS VARCHAR)) AS Score-
Display,
            q.question_id AS QuestionID,
            q.question_text AS QuestionText,
            q.question_type AS QuestionType,
            q.question_difficulty AS Difficulty,
            student_choice.choice_text AS StudentChoice,
            correct_choice.choice_text AS CorrectChoice,
            CASE
                WHEN CAST(saq.student_answer AS INT) = q.correct_ans_id THEN 'Correct'
                WHEN saq.student_answer IS NOT NULL THEN 'Incorrect'
                ELSE 'Not Answered'
            END AS QuestionResult
        FROM exam e
        INNER JOIN exam_questions eq ON eq.exam_id = e.exam_id
        INNER JOIN question q ON q.question_id = eq.questoin_id
        INNER JOIN student_exam se ON se.exam_id = e.exam_id AND se.student_id = @student_id
        LEFT JOIN student_answer_question saq ON saq.exam_id = e.exam_id
            AND saq.student_id = @student_id
            AND saq.quesiotn_id = q.question_id
        LEFT JOIN quesiton_choice student_choice ON student_choice.choice_id =
CAST(saq.student_answer AS INT)
        LEFT JOIN quesiton_choice correct_choice ON correct_choice.choice_id = q.correct_ans_id
        WHERE e.exam_id = @exam_id
        ORDER BY q.question_id;

        RETURN 0;

    END TRY
    BEGIN CATCH
        DECLARE @ErrorMessage NVARCHAR(4000) = ERROR_MESSAGE();
        DECLARE @ErrorSeverity INT = ERROR_SEVERITY();
        DECLARE @ErrorState INT = ERROR_STATE();

        RAISERROR(@ErrorMessage, @ErrorSeverity, @ErrorState);
        RETURN -99;
    END CATCH
END;
GO
```

Uses

[dbo].[exam]
[dbo].[exam_questions]
[dbo].[instructor_generate_course_exam]
[dbo].[quesiton_choice]
[dbo].[question]
[dbo].[student_answer_question]
[dbo].[student_course]
[dbo].[student_exam]

[dbo].[StudentSubmitAnswers]

Properties

Property	Value
ANSI Nulls On	True
Quoted Identifier On	False

Parameters

Name	Data Type	Max Length (Bytes)
@student_id	int	4
@exam_id	int	4
@Answers	StudentAnswers	max

SQL Script

```
SET QUOTED_IDENTIFIER OFF
GO
CREATE PROCEDURE [dbo].[StudentSubmitAnswers]
(
    @student_id INT,
    @exam_id INT,
    @Answers StudentAnswers READONLY
)
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRY
        BEGIN TRANSACTION;


        DECLARE @course_id INT;

        SELECT @course_id = course_id
        FROM instructor_generate_course_exam
        WHERE exam_id = @exam_id;

        IF @course_id IS NULL
       BEGIN
           THROW 50001, 'Student is not allowed to answer this exam.', 1;
       END;

        IF NOT EXISTS
        (
            SELECT 1
            FROM student_course
            WHERE student_id = @student_id
              AND course_id = @course_id
        )
        BEGIN
            THROW 50001, 'Student is not allowed to answer this exam.', 1;
        END;


        IF NOT EXISTS
        (
            SELECT 1
            FROM Exam
            WHERE exam_id = @exam_id
              AND GETDATE() BETWEEN exam_date
              AND DATEADD(MINUTE, duration_mins, exam_date)
        )
        BEGIN
            THROW 50002, 'Exam is not currently active.', 1;
        END;


        IF NOT EXISTS
        (
            SELECT 1
            FROM student_exam
            WHERE student_id = @student_id
              AND exam_id = @exam_id
        )
        BEGIN
            INSERT INTO student_exam
            (
                student_id,
                exam_id,
                state,
                grade
            )
            VALUES
            (
                @student_id,
                @exam_id,
                'Submitted',
                NULL
            );
        END;


        IF EXISTS
        (
            SELECT 1
            FROM @Answers a
            LEFT JOIN Exam_Questions eq
                ON eq.questoin_id = a.question_id
               AND eq.exam_id = @exam_id
            WHERE eq.questoin_id IS NULL
        )
        BEGIN
            THROW 50003, 'One or more questions do not belong to this exam.', 1;
        END;


        INSERT INTO student_answer_question
        (
            student_id,
            exam_id,
            quesiotn_id,
            student_answer
        )
        SELECT
            @student_id,
            @exam_id,
            a.question_id,
            a.student_answer
        FROM @Answers a
        WHERE NOT EXISTS
        (
            SELECT 1
            FROM student_answer_question sa
            WHERE sa.student_id = @student_id
              AND sa.exam_id = @exam_id
              AND sa.quesiotn_id = a.question_id
        );

        EXEC CorrectExam @exam_id,@student_id;
        COMMIT TRANSACTION;

        SELECT 'Answers saved successfully' AS Result;

    END TRY
    BEGIN CATCH
        ROLLBACK TRANSACTION;
        THROW;
    END CATCH
END;
GO
```

Uses

[dbo].[exam]
[dbo].[exam_questions]
[dbo].[instructor_generate_course_exam]
[dbo].[student_answer_question]

[dbo].[student_course]
[dbo].[student_exam]
[dbo].[CorrectExam]
[dbo].[StudentAnswers]

[dbo].[UnenrollStudentFromCourse]

Properties

Property	Value
ANSI Nulls On	True
Quoted Identifier On	False

Parameters

Name	Data Type	Max Length (Bytes)
@StudentID	int	4
@CourseID	int	4

SQL Script

```
SET QUOTED_IDENTIFIER OFF
GO
-- =============================================
-- Procedure: UnenrollStudentFromCourse
-- Description: Removes a student's enrollment from a course.
-- =============================================
CREATE PROCEDURE [dbo].[UnenrollStudentFromCourse]
    @StudentID INT,
    @CourseID INT
AS
BEGIN
    SET NOCOUNT ON;

    -- Validate StudentID
    IF @StudentID IS NULL
    BEGIN
        RAISERROR('Student ID is required', 16, 1);
        RETURN -1;
    END

    -- Validate CourseID
    IF @CourseID IS NULL
    BEGIN
        RAISERROR('Course ID is required', 16, 1);
        RETURN -1;
    END

    -- Check if enrollment exists
    IF NOT EXISTS (
        SELECT 1
        FROM student_course
        WHERE student_id = @StudentID
          AND course_id = @CourseID
    )
    BEGIN
        RAISERROR('Enrollment not found', 16, 1);
        RETURN -2;
    END

    -- Check if student has exams in this course
    IF EXISTS (
        SELECT 1
        FROM student_exam se
        INNER JOIN instructor_generate_course_exam igce ON se.exam_id = igce.exam_id
        WHERE se.student_id = @StudentID
          AND igce.course_id = @CourseID
    )
    BEGIN
        RAISERROR('Cannot unenroll: Student has exam records in this course', 16, 1);
        RETURN -3;
    END

    BEGIN TRY
        DELETE FROM student_course
        WHERE student_id = @StudentID
          AND course_id = @CourseID;

        SELECT 'Student unenrolled from course successfully' AS Message;
        RETURN 0;
    END TRY
    BEGIN CATCH
        DECLARE @ErrorMessage NVARCHAR(4000) = ERROR_MESSAGE();
        RAISERROR(@ErrorMessage, 16, 1);
        RETURN -4;
    END CATCH
END
GO
```

Uses

[dbo].[instructor_generate_course_exam]
[dbo].[student_course]
[dbo].[student_exam]

[dbo].[UnlinkQuestionFromCourseTopic]

Properties

Property	Value
ANSI Nulls On	True
Quoted Identifier On	False

Parameters

Name	Data Type	Max Length (Bytes)
@QuestionID	int	4
@CourseID	int	4
@TopicID	int	4

SQL Script

```
SET QUOTED_IDENTIFIER OFF
GO
-- =============================================
-- Procedure: UnlinkQuestionFromCourseTopic
-- Description: Removes the link between a question and a course/topic.
-- =============================================
CREATE PROCEDURE [dbo].[UnlinkQuestionFromCourseTopic]
    @QuestionID INT,
    @CourseID INT,
    @TopicID INT
AS
BEGIN
    SET NOCOUNT ON;

    -- Validate QuestionID
    IF @QuestionID IS NULL
    BEGIN
        RAISERROR('Question ID is required', 16, 1);
        RETURN -1;
    END

    -- Validate CourseID
    IF @CourseID IS NULL
    BEGIN
        RAISERROR('Course ID is required', 16, 1);
        RETURN -1;
    END
    -- Validate TopicID
    IF @TopicID IS NULL
    BEGIN
        RAISERROR('Topic ID is required', 16, 1);
        RETURN -1;
    END

    -- Check if the link exists
    IF NOT EXISTS (
        SELECT 1
        FROM course_questions_on_topic
        WHERE course_id = @CourseID
          AND question_id = @QuestionID
          AND topic_id = @TopicID
    )
    BEGIN
        RAISERROR('Link not found between specifed question, course, and topic', 16, 1);
        RETURN -2;
    END

    BEGIN TRY
        DELETE FROM course_questions_on_topic
        WHERE course_id = @CourseID
          AND question_id = @QuestionID
          AND topic_id = @TopicID;

        SELECT 'Question unlinked from course and topic successfully' AS Message;
        RETURN 0;
    END TRY
    BEGIN CATCH
        DECLARE @ErrorMessage NVARCHAR(4000) = ERROR_MESSAGE();
        RAISERROR(@ErrorMessage, 16, 1);
        RETURN -3;
    END CATCH
END
GO
```

Uses

[dbo].[course_questions_on_topic]

[dbo].[UpdateBranch]

Properties

Property	Value
ANSI Nulls On	True
Quoted Identifier On	False

Parameters

Name	Data Type	Max Length (Bytes)
@id	int	4
@new_name	varchar(255)	255
@new_city	varchar(255)	255

SQL Script

```
SET QUOTED_IDENTIFIER OFF
GO
CREATE PROCEDURE [dbo].[UpdateBranch](@id int, @new_name varchar(255), @new_city varchar(255))
AS
BEGIN
    UPDATE branch
    SET branch_name = @new_name, branch_city = @new_city
    WHERE branch_id = @id;

    IF @@ROWCOUNT = 0
    BEGIN
        DECLARE @error_message VARCHAR(50) = CONCAT('Sorry branch with id ', @id, ' does not
exist');
        THROW 50010, @error_message, 1;
    END
END

EXEC UpdateBranch 1, 'ITI Assiut', 'Assiut'
GO
```

Uses

[dbo].[branch]

[dbo].[UpdateChoiceText]

Properties

Property	Value
ANSI Nulls On	True
Quoted Identifier On	False

Parameters

Name	Data Type	Max Length (Bytes)
@ChoiceID	int	4
@ChoiceText	varchar(255)	255

SQL Script

```
SET QUOTED_IDENTIFIER OFF
GO
CREATE PROCEDURE [dbo].[UpdateChoiceText]
    @ChoiceID INT,
    @ChoiceText VARCHAR(255)
AS
BEGIN
    SET NOCOUNT ON;

    IF @ChoiceID IS NULL
        RETURN -1;

    IF @ChoiceText IS NULL OR LTRIM(RTRIM(@ChoiceText)) = ''
        RETURN -2;

    IF NOT EXISTS (SELECT 1 FROM quesiton_choice WHERE choice_id = @ChoiceID)
        RETURN -3;

    -- Protect global True/False choice entries (lookup dynamically)
    DECLARE @TrueChoiceID INT;
    DECLARE @FalseChoiceID INT;

    SELECT @TrueChoiceID = choice_id FROM quesiton_choice WHERE LOWER(choice_text) = 'true';
    SELECT @FalseChoiceID = choice_id FROM quesiton_choice WHERE LOWER(choice_text) = 'false';

    IF (@TrueChoiceID IS NOT NULL AND @ChoiceID = @TrueChoiceID) OR
       (@FalseChoiceID IS NOT NULL AND @ChoiceID = @FalseChoiceID)
        RETURN -4;
    BEGIN TRY
        UPDATE quesiton_choice
        SET choice_text = LTRIM(RTRIM(@ChoiceText))
        WHERE choice_id = @ChoiceID;

        RETURN 0;
    END TRY
    BEGIN CATCH
        RETURN -5;
    END CATCH
END
GO
```

Uses

[dbo].[quesiton_choice]

[dbo].[UpdateCourse]

Properties

Property	Value
ANSI Nulls On	True
Quoted Identifier On	False

Parameters

Name	Data Type	Max Length (Bytes)
@course_id	int	4
@course_code	int	4
@description	varchar(255)	255
@course_title	varchar(255)	255
@credits	varchar(255)	255

SQL Script

```
SET QUOTED_IDENTIFIER OFF
GO
CREATE   PROCEDURE [dbo].[UpdateCourse]
    @course_id INT,
    @course_code INT,
    @description VARCHAR(255),
    @course_title VARCHAR(255),
    @credits VARCHAR(255)
AS
BEGIN
    BEGIN TRY
        BEGIN TRANSACTION
            IF EXISTS (SELECT 1 FROM courses WHERE course_id = @course_id)
            BEGIN
                -- Check for duplicate code on other courses
                IF NOT EXISTS (SELECT 1 FROM courses WHERE course_code = @course_code AND
course_id <> @course_id)
                BEGIN
                    UPDATE courses
                    SET course_code = @course_code,
                        description = @description,
                        course_title = @course_title,
                        credits = @credits
                    WHERE course_id = @course_id;
                END
                ELSE
                BEGIN
                     DECLARE @msg VARCHAR(100) = 'Course code ' + CAST(@course_code AS
VARCHAR(20)) + ' is already used by another course!';
                     THROW 50002, @msg, 1;
                END
            END
            ELSE
            BEGIN
                DECLARE @error_message VARCHAR(100) = 'Course with ID ' + CAST(@course_id AS
VARCHAR(20)) + ' does not exist!';
                THROW 50003, @error_message, 1;
            END
        COMMIT TRANSACTION;
    END TRY
    BEGIN CATCH
        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION;
        THROW;
    END CATCH
END
GO
```

Uses

[dbo].[courses]

[dbo].[UpdateDepartment]

Properties

Property	Value
ANSI Nulls On	True
Quoted Identifier On	False

Parameters

Name	Data Type	Max Length (Bytes)
@id	int	4
@new_name	varchar(255)	255

SQL Script

```
SET QUOTED_IDENTIFIER OFF
GO
CREATE PROCEDURE [dbo].[UpdateDepartment](@id INT, @new_name VARCHAR(255))
AS
BEGIN
    UPDATE department
    SET department_name = @new_name
    WHERE department_id = @id;

    IF @@ROWCOUNT = 0
    BEGIN
        DECLARE @error_message VARCHAR(50) = CONCAT('Department with id ', @id, ' does not
exist!');
        THROW 50010, @error_message, 1;
    END
END

EXEC UpdateDepartment 1, 'PWD'
GO
```

Uses

[dbo].[department]

[dbo].[UpdateExam]

Properties

Property	Value
ANSI Nulls On	True
Quoted Identifier On	False

Parameters

Name	Data Type	Max Length (Bytes)
@ExamID	int	4
@ExamTitle	varchar(255)	255
@ExamDate	datetime	8
@ExamType	varchar(50)	50
@DurationMins	int	4

SQL Script

```
SET QUOTED_IDENTIFIER OFF
GO
-- =============================================
-- Procedure: UpdateExam
-- Description: Updates exam properties.
--              Note: Cannot change associated course or add/remove questions.
--              For that, use GenerateExam to create a new exam.
-- =============================================
CREATE PROCEDURE [dbo].[UpdateExam]
    @ExamID INT,
    @ExamTitle VARCHAR(255) = NULL,
    @ExamDate DATETIME = NULL,
    @ExamType VARCHAR(50) = NULL,
    @DurationMins INT = NULL
AS
BEGIN
    SET NOCOUNT ON;

    -- Validate ExamID
    IF @ExamID IS NULL OR @ExamID <= 0
    BEGIN
        RAISERROR('Valid Exam ID is required', 16, 1);
        RETURN -1;
    END
    -- Check if at least one field to update
    IF @ExamTitle IS NULL AND @ExamDate IS NULL AND @ExamType IS NULL AND @DurationMins IS NULL
    BEGIN
        RAISERROR('At least one field must be provided for update', 16, 1);
        RETURN -2;
    END

    -- Check if exam exists
    IF NOT EXISTS (SELECT 1 FROM exam WHERE exam_id = @ExamID)
    BEGIN
        RAISERROR('Exam not found', 16, 1);
        RETURN -3;
    END

    -- Check if exam has already ended (prevent updates to completed exams)
    DECLARE @ExamEndTime DATETIME;
    SELECT @ExamEndTime = DATEADD(MINUTE, duration_mins, exam_date)
    FROM exam
    WHERE exam_id = @ExamID;

    IF GETDATE() > @ExamEndTime
    BEGIN
        RAISERROR('Cannot update exam that has already ended. Students may have completed it.',
16, 1);
        RETURN -7;
    END

    -- Validate exam type if provided
    IF @ExamType IS NOT NULL AND @ExamType NOT IN ('final', 'mid', 'semifinal')
    BEGIN
        RAISERROR('Invalid exam type. Must be: final, mid, or semifinal', 16, 1);
        RETURN -4;
    END

    -- Validate duration if provided
    IF @DurationMins IS NOT NULL AND @DurationMins <= 0
    BEGIN
        RAISERROR('Duration must be greater than 0', 16, 1);
        RETURN -5;
    END

    BEGIN TRY
        UPDATE exam
        SET
            exam_title = COALESCE(@ExamTitle, exam_title),
            exam_date = COALESCE(@ExamDate, exam_date),
            exam_type = COALESCE(@ExamType, exam_type),
            duration_mins = COALESCE(@DurationMins, duration_mins)
        WHERE exam_id = @ExamID;

        SELECT 'Exam updated successfully' AS Message,
               @ExamID AS ExamID;
        RETURN 0;
    END TRY
    BEGIN CATCH
        DECLARE @ErrorMessage NVARCHAR(4000) = ERROR_MESSAGE();
        RAISERROR(@ErrorMessage, 16, 1);
        RETURN -6;
    END CATCH
END
GO
```

Uses

[dbo].[exam]

[dbo].[UpdateInstructor]

Properties

Property	Value
ANSI Nulls On	True
Quoted Identifier On	False

Parameters

Name	Data Type	Max Length (Bytes)
@InstructorId	int	4
@FirstName	varchar(255)	255
@LastName	varchar(255)	255
@Email	varchar(255)	255
@HireDate	date	3

SQL Script

```
SET QUOTED_IDENTIFIER OFF
GO
CREATE PROCEDURE [dbo].[UpdateInstructor]
    @InstructorId INT,
    @FirstName VARCHAR(255),
    @LastName VARCHAR(255),
    @Email VARCHAR(255),
    @HireDate DATE
AS
BEGIN
    SET NOCOUNT ON;
    BEGIN TRANSACTION;
    BEGIN TRY
        -- Validate InstructorId
        IF @InstructorId IS NULL OR @InstructorId <= 0
        BEGIN
            RAISERROR('Invalid Instructor ID', 16, 1);
            ROLLBACK TRANSACTION;
            RETURN;
        END

        -- Validate input parameters
        IF @FirstName IS NULL OR LTRIM(RTRIM(@FirstName)) = ''
        BEGIN
            RAISERROR('First name cannot be empty', 16, 1);
            ROLLBACK TRANSACTION;
            RETURN;
        END

        IF @LastName IS NULL OR LTRIM(RTRIM(@LastName)) = ''
        BEGIN
            RAISERROR('Last name cannot be empty', 16, 1);
            ROLLBACK TRANSACTION;
            RETURN;
        END

        IF @Email IS NULL OR LTRIM(RTRIM(@Email)) = ''
        BEGIN
            RAISERROR('Email cannot be empty', 16, 1);
            ROLLBACK TRANSACTION;
            RETURN;
        END

        IF @HireDate IS NULL
        BEGIN
            RAISERROR('Hire date cannot be empty', 16, 1);
            ROLLBACK TRANSACTION;
            RETURN;
        END

        -- Validate hire date is not in the future
        IF @HireDate > GETDATE()
        BEGIN
            RAISERROR('Hire date cannot be in the future', 16, 1);
            ROLLBACK TRANSACTION;
            RETURN;
        END

        -- Check if instructor exists
        IF NOT EXISTS (SELECT 1 FROM [instructor] WHERE [instructor_id] = @InstructorId)
        BEGIN
            RAISERROR('Instructor not found', 16, 1);
            ROLLBACK TRANSACTION;
            RETURN;
        END

        -- Check if email already exists for another person
        IF EXISTS (SELECT 1 FROM [person] WHERE [email] = @Email AND [person_id] != @Instructor-
Id)
        BEGIN
            RAISERROR('Email already exists for another user', 16, 1);
            ROLLBACK TRANSACTION;
            RETURN;
        END

        -- Update person table
        UPDATE [person]
        SET [first_name] = @FirstName,
            [last_name] = @LastName,
            [email] = @Email
        WHERE [person_id] = @InstructorId;

        -- Update instructor table
        UPDATE [instructor]
        SET [hire_date] = @HireDate
        WHERE [instructor_id] = @InstructorId;

        COMMIT TRANSACTION;

        SELECT 'Instructor updated successfully' AS Message;
    END TRY
    BEGIN CATCH
        ROLLBACK TRANSACTION;
        DECLARE @ErrorMessage NVARCHAR(4000) = ERROR_MESSAGE();
        DECLARE @ErrorSeverity INT = ERROR_SEVERITY();
        DECLARE @ErrorState INT = ERROR_STATE();
        RAISERROR(@ErrorMessage, @ErrorSeverity, @ErrorState);
    END CATCH
END
GO
```

Uses

[dbo].[instructor]
[dbo].[person]

[dbo].[UpdateQuestion]

Properties

Property	Value
ANSI Nulls On	True
Quoted Identifier On	False

Parameters

Name	Data Type	Max Length (Bytes)
@QuestionID	int	4
@QuestionText	varchar(255)	255
@QuestionDifficulty	varchar(50)	50

SQL Script

```
SET QUOTED_IDENTIFIER OFF
GO
CREATE PROCEDURE [dbo].[UpdateQuestion]
    @QuestionID INT,
    @QuestionText VARCHAR(255) = NULL,
    @QuestionDifficulty VARCHAR(50) = NULL
AS
BEGIN
    SET NOCOUNT ON;

    IF @QuestionID IS NULL
        RETURN -1;

    IF @QuestionText IS NULL AND @QuestionDifficulty IS NULL
        RETURN -2;

    IF @QuestionText IS NOT NULL AND LTRIM(RTRIM(@QuestionText)) = ''
        RETURN -3;

    IF @QuestionDifficulty IS NOT NULL
        AND @QuestionDifficulty NOT IN ('easy', 'medium', 'hard')
        RETURN -4;

    IF NOT EXISTS (SELECT 1 FROM question WHERE question_id = @QuestionID)
        RETURN -5;

    BEGIN TRY
        UPDATE question
        SET
            question_text = CASE
                WHEN @QuestionText IS NOT NULL
                THEN LTRIM(RTRIM(@QuestionText))
                ELSE question_text
            END,
            question_difficulty = CASE
                WHEN @QuestionDifficulty IS NOT NULL
                THEN @QuestionDifficulty
                ELSE question_difficulty
            END
        WHERE question_id = @QuestionID;

        RETURN 0;
    END TRY
    BEGIN CATCH
        RETURN -6;
    END CATCH
END
GO
```

Uses

[dbo].[question]

[dbo].[UpdateQuestionCorrectAnswer]

Properties

Property	Value
ANSI Nulls On	True
Quoted Identifier On	False

Parameters

Name	Data Type	Max Length (Bytes)
@QuestionID	int	4
@NewCorrectChoiceID	int	4

SQL Script

```
SET QUOTED_IDENTIFIER OFF
GO
CREATE PROCEDURE [dbo].[UpdateQuestionCorrectAnswer]
    @QuestionID INT,
    @NewCorrectChoiceID INT
AS
BEGIN
    SET NOCOUNT ON;

    IF @QuestionID IS NULL
        RETURN -1;

    IF @NewCorrectChoiceID IS NULL
        RETURN -2;

    IF NOT EXISTS (SELECT 1 FROM question WHERE question_id = @QuestionID)
        RETURN -3;

    DECLARE @QuestionType VARCHAR(50);

    SELECT @QuestionType = question_type
    FROM question
    WHERE question_id = @QuestionID;

    IF @QuestionType = 'True_False'
    BEGIN
        DECLARE @TrueChoiceID INT;
        DECLARE @FalseChoiceID INT;
        SELECT @TrueChoiceID = choice_id FROM quesiton_choice WHERE LOWER(choice_text) = 'true';
        SELECT @FalseChoiceID = choice_id FROM quesiton_choice WHERE LOWER(choice_text) = 
'false';

        IF @TrueChoiceID IS NULL OR @FalseChoiceID IS NULL
            RETURN -4; -- True/False choice entries missing or not configured

        IF @NewCorrectChoiceID <> @TrueChoiceID AND @NewCorrectChoiceID <> @FalseChoiceID
            RETURN -4;
    END
    ELSE IF @QuestionType = 'MCQ'
    BEGIN
        IF NOT EXISTS (SELECT 1 FROM quesiton_choice WHERE choice_id = @NewCorrectChoiceID)
            RETURN -5;

        IF NOT EXISTS (
            SELECT 1
            FROM question_choise_bridge
            WHERE question_id = @QuestionID
            AND choice_id = @NewCorrectChoiceID
        )
            RETURN -6;
    END
    ELSE
    BEGIN
        RETURN -7;
    END

    BEGIN TRY
        UPDATE question
        SET correct_ans_id = @NewCorrectChoiceID
        WHERE question_id = @QuestionID;

        RETURN 0;
    END TRY
    BEGIN CATCH
        RETURN -8;
    END CATCH
END
GO
```

Uses

[dbo].[quesiton_choice]
[dbo].[question]
[dbo].[question_choise_bridge]

[dbo].[UpdateStudent]

Properties

Property	Value
ANSI Nulls On	True
Quoted Identifier On	False

Parameters

Name	Data Type	Max Length (Bytes)
@StudentId	int	4
@FirstName	varchar(255)	255
@LastName	varchar(255)	255
@Email	varchar(255)	255

SQL Script

```
SET QUOTED_IDENTIFIER OFF
GO
CREATE PROCEDURE [dbo].[UpdateStudent]
    @StudentId INT,
    @FirstName VARCHAR(255),
    @LastName VARCHAR(255),
    @Email VARCHAR(255)
AS
BEGIN
    SET NOCOUNT ON;
    BEGIN TRANSACTION;
    BEGIN TRY
        -- Validate StudentId
        IF @StudentId IS NULL OR @StudentId <= 0
        BEGIN
            RAISERROR('Invalid Student ID', 16, 1);
            ROLLBACK TRANSACTION;
            RETURN;
        END

        -- Validate input parameters
        IF @FirstName IS NULL OR LTRIM(RTRIM(@FirstName)) = ''
        BEGIN
            RAISERROR('First name cannot be empty', 16, 1);
            ROLLBACK TRANSACTION;
            RETURN;
        END
        IF @LastName IS NULL OR LTRIM(RTRIM(@LastName)) = ''
        BEGIN
            RAISERROR('Last name cannot be empty', 16, 1);
            ROLLBACK TRANSACTION;
            RETURN;
        END

        IF @Email IS NULL OR LTRIM(RTRIM(@Email)) = ''
        BEGIN
            RAISERROR('Email cannot be empty', 16, 1);
            ROLLBACK TRANSACTION;
            RETURN;
        END

        -- Check if student exists
        IF NOT EXISTS (SELECT 1 FROM [student] WHERE [student_id] = @StudentId)
        BEGIN
            RAISERROR('Student not found', 16, 1);
            ROLLBACK TRANSACTION;
            RETURN;
        END

        -- Check if email already exists for another person
        IF EXISTS (SELECT 1 FROM [person] WHERE [email] = @Email AND [person_id] != @StudentId)
        BEGIN
            RAISERROR('Email already exists for another user', 16, 1);
            ROLLBACK TRANSACTION;
            RETURN;
        END

        -- Update person table
        UPDATE [person]
        SET [first_name] = @FirstName,
            [last_name] = @LastName,
            [email] = @Email
        WHERE [person_id] = @StudentId;

        COMMIT TRANSACTION;

        SELECT 'Student updated successfully' AS Message;
    END TRY
    BEGIN CATCH
        ROLLBACK TRANSACTION;
        DECLARE @ErrorMessage NVARCHAR(4000) = ERROR_MESSAGE();
        DECLARE @ErrorSeverity INT = ERROR_SEVERITY();
        DECLARE @ErrorState INT = ERROR_STATE();
        RAISERROR(@ErrorMessage, @ErrorSeverity, @ErrorState);
    END CATCH
END
GO
```

Uses

[dbo].[person]
[dbo].[student]

[dbo].[UpdateTopic]

Properties

Property	Value
ANSI Nulls On	True
Quoted Identifier On	False

Parameters

Name	Data Type	Max Length (Bytes)
@topic_id	int	4
@topic_order	int	4
@topic_duration	varchar(255)	255
@topic_title	varchar(255)	255

SQL Script

```
SET QUOTED_IDENTIFIER OFF
GO
CREATE   PROCEDURE [dbo].[UpdateTopic]
    @topic_id INT,
    @topic_order INT,
    @topic_duration VARCHAR(255),
    @topic_title VARCHAR(255)
AS
BEGIN
    BEGIN TRY
        BEGIN TRANSACTION
            IF EXISTS (SELECT 1 FROM topic WHERE topic_id = @topic_id)
            BEGIN
                UPDATE topic
                SET topic_order = @topic_order,
                    topic_duration = @topic_duration,
                    topic_title = @topic_title
                WHERE topic_id = @topic_id;
            END
            ELSE
            BEGIN
                 DECLARE @error_message VARCHAR(100) = 'Topic with ID ' + CAST(@topic_id AS
VARCHAR(20)) + ' does not exist!';
                 THROW 50005, @error_message, 1;
            END
        COMMIT TRANSACTION;
    END TRY
    BEGIN CATCH
        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION;
        THROW;
    END CATCH
END
GO
```

Uses

[dbo].[topic]

User-Defined Table Types

Objects

Name
dbo.StudentAnswers

[dbo].[StudentAnswers]

Properties

Property	Value
Heap	True

Columns

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

SQL Script

```
CREATE TYPE [dbo].[StudentAnswers] AS TABLE
(
[student_answer] [int] NULL,
[question_id] [int] NULL
)
GO
```

Used By

[dbo].[StudentSubmitAnswers]

Users

Objects

Name
dbo
guest

dbo

Properties

Property	Value
Type	WindowsUser
Login Name	DESKTOP-BC5Q7G5\Electronica Care
Default Schema	dbo

Database Level Permissions

Type	Action
CONNECT	Grant

SQL Script

```
GO
```

guest

Properties

Property	Value
Type	SqlUser
Default Schema	guest

SQL Script

```
GO
```

Database Roles

Objects

Name
db_accessadmin
db_backupoperator
db_datareader
db_datawriter
db_ddladmin
db_denydatareader
db_denydatawriter
db_owner
db_securityadmin
public

db_accessadmin

Properties

Property	Value
Owner	dbo

db_backupoperator

Properties

Property	Value
Owner	dbo

db_datareader

Properties

Property	Value
Owner	dbo

db_datawriter

Properties

Property	Value
Owner	dbo

db_ddladmin

Properties

Property	Value
Owner	dbo

db_denydatareader

Properties

Property	Value
Owner	dbo

db_denydatawriter

Properties

Property	Value
Owner	dbo

db_owner

Properties

Property	Value
Owner	dbo

db_securityadmin

Properties

Property	Value
Owner	dbo

public

Properties

Property	Value
Owner	dbo