Examination System

Database Project

Team Members

- Ahmed Mohamed Allam
- Islam Mosad Adly
- Shady Ayman Maher
- Fatma Said
- Gehad Nabel

Supervised by:

Dr. Rami Nagy

Contents

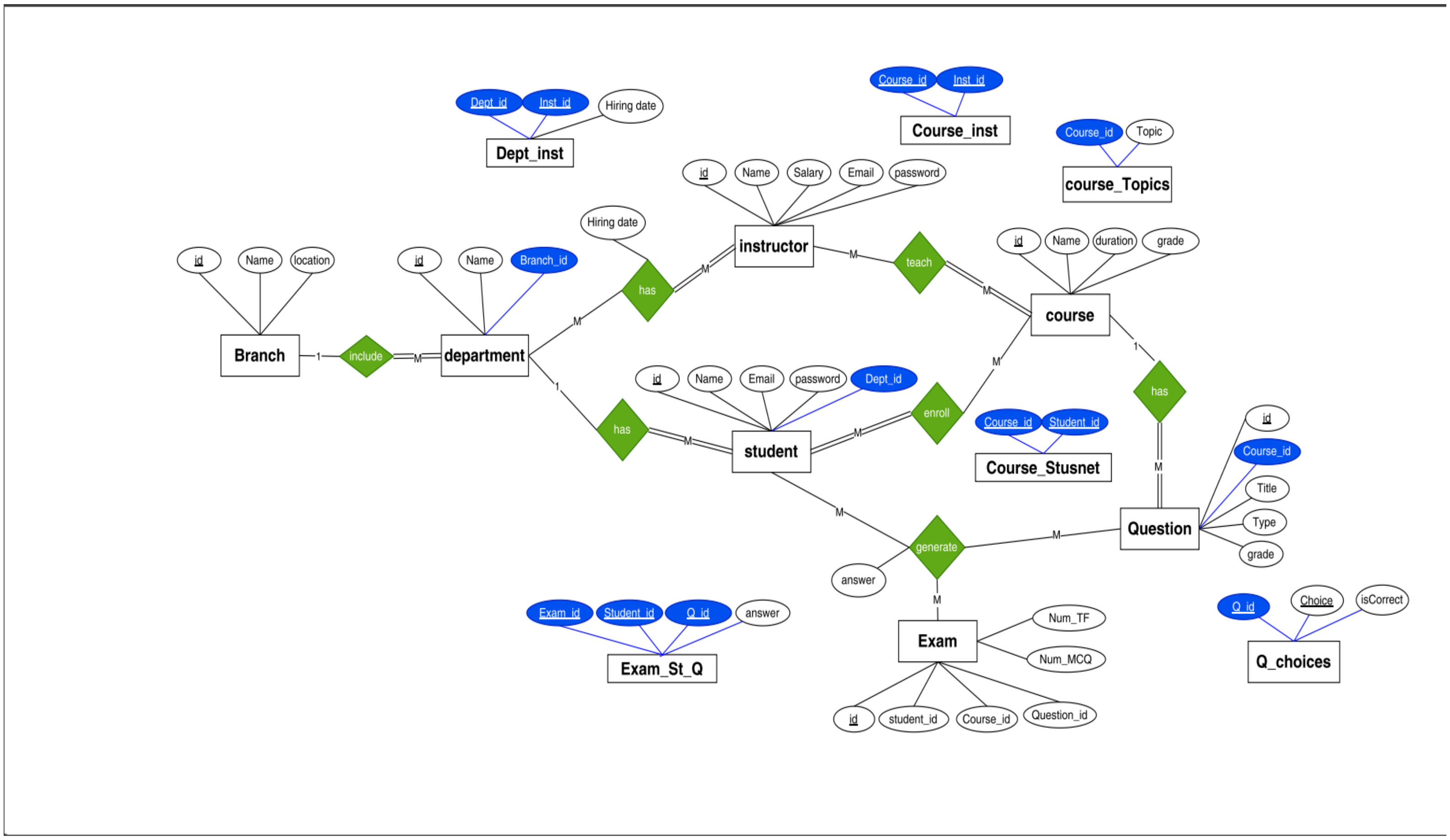
Introduction

This project, named "ExaminationSystem," is a comprehensive application developed as part of the ITI Schooler Ship 9-month program, focusing on the professional web and business intelligence (BI) track. The system is designed to manage various aspects of an examination system, including data storage, desktop application functionalities, and reporting features. The technologies used in this project include SQL Server, C#, LINQ, Entity Framework 8, and Report Builder.

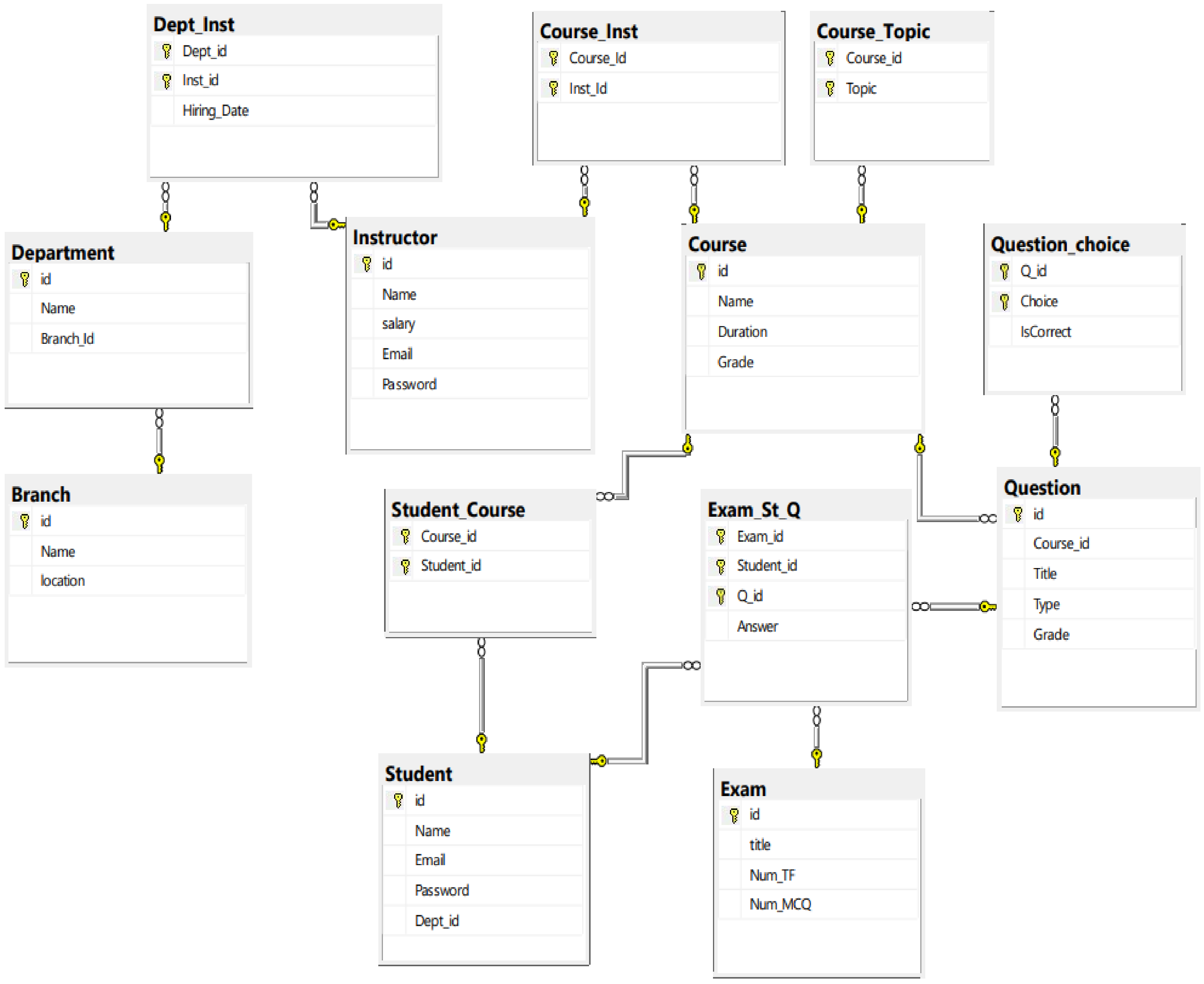
Project Structure

The project consists of two main components: a database backend and a desktop application frontend. The database plays a crucial role in storing and managing the data required for the examination system. The desktop application provides a user-friendly interface for interacting with the syste

ERD


Dept_id
Inst_id
Hiring date
Dept_inst
Course_id
Inst_id
Course_inst
Course_id
Topic
course_Topics
id
Name
Salary
Email
password
instructor
Hiring date
has
teach
id
Name
duration
grade
course
id
Name
location
Branch
include
id
Name
Branch_id
department
has
id
Name
Email
password
Dept_id
student
enroll
Course_id
Student_id
Course_Stusnet
has
id
Course_id
Title
Type
grade
Question
generate
answer
Exam_id
Student_id
Q_id
answer
Exam_St_Q
Exam
Num_TF
Num_MCQ
id
student_id
Course_id
Question_id
Q_id
Choice
isCorrect
Q_choices
1
M

Mapping


Dept_Inst
Dept_id
Inst_id
Hiring_Date
Course_Inst
Course_Id
Inst_Id
Course_Topic
Course_id
Topic
Department
id
Name
Branch_Id
Instructor
id
Name
salary
Email
Password
Course
id
Name
Duration
Grade
Question_choice
Q_id
Choice
IsCorrect
Branch
id
Name
location
Student_Course
Course_id
Student_id
Exam_St_Q
Exam_id
Student_id
Q_id
Answer
Question
id
Course_id
Title
Type
Grade
Student
id
Name
Email
Password
Dept_id
Exam
id
title
Num_TF
Num_MCQ

Tables in Data Base

Branch Table

Description:

The "Branch" table within the "ExaminationSystem" database serves as a repository for information about different branches. It stores details such as the branch's unique identifier, name, and location. This table is designed to support the effective organization and management of branch-related data within the Examination System.

Attributes:

1. id (integer): Represents the unique identifier for each branch, serving as the primary key for the "Branch" table.

2. Name (varchar(50)): Stores the name of the branch, providing a concise label for identification.

3. Location (varchar(50)): Captures the location or address of the branch, facilitating geographical categorization.

Functionality:

The "Branch" table offers the following functionalities:

1. Record Storage: Enables the storage of information about various branches, including their names and locations.

2. Identification: The "id" column serves as a unique identifier for each branch, ensuring individual records can be easily distinguished.

3. Data Organization: Supports the organization and categorization of branch-related data, aiding administrators in efficiently managing information.

4. Referential Integrity: Utilizes the primary key constraint on the "id" column to maintain referential integrity within the table and uphold data consistency.

Department Table

Description:

The "Department" table, residing within the "ExaminationSystem" database, serves as a structured repository for department-related information. It enables the organization and management of distinct departments associated with different branches.

Attributes:

1. id (integer): Represents the unique identifier for each department, functioning as the primary key of the "Department" table.

2. Name (varchar(50)): Stores the name of the department, providing a clear label for identification.

3. Branch_Id (integer): Establishes a relationship with the "Branch" table by referencing the unique identifier (id) of the associated branch. This column serves as a foreign key to maintain a connection between departments and branches.

Functionality:

The "Department" table offers the following functionalities:

1. Record Storage: Facilitates the storage of information about various departments, including their names and associated branches.

2. Identification: The "id" column serves as a unique identifier for each department, ensuring individual records can be easily distinguished.

3. Relationship Establishment: Utilizes the "Branch_Id" column as a foreign key to establish a relationship with the "Branch" table. This linkage enables the association of each department with a specific branch.

4. Referential Integrity: The foreign key constraint ([FK_Department_Branch]) maintains referential integrity, ensuring that valid relationships are maintained between records in the "Department" and "Branch" tables.

Department Instructor Table

Description:

The "Dept_Inst" table, residing within the "ExaminationSystem" database, is designed to establish relationships between departments and instructors, incorporating information about the hiring date of instructors within specific departments.

Attributes:

1. Dept_id (integer): Represents the unique identifier for each department, functioning as part of the composite primary key of the "Dept_Inst" table. This column serves as a foreign key referencing the "id" column in the "Department" table.

2. Inst_id (integer): Denotes the unique identifier for each instructor, constituting part of the composite primary key of the "Dept_Inst" table. This column serves as a foreign key referencing the "id" column in the "Instructor" table.

3. Hiring_Date (date): Captures the date when an instructor was hired within a specific department.

Functionality:

The "Dept_Inst" table offers the following functionalities:

1. Relationship Establishment: Utilizes the "Dept_id" and "Inst_id" columns as composite primary keys to establish relationships between departments and instructors.

2. Hiring Date Recording: The "Hiring_Date" column records the date when an instructor was hired within a particular department, providing historical data for human resources and administrative purposes.

3. Referential Integrity: Foreign key constraints ([FK_Dept_Inst_Department], [FK_Dept_Inst_Instructor]) ensure referential integrity by linking records in the "Dept_Inst" table to corresponding records in the "Department" and "Instructor" tables. Additionally, the "ON DELETE CASCADE" option ensures that related records are automatically deleted when the corresponding department or instructor is deleted.

Instructor Table

Description:

The " Instructor " table within the "ExaminationSystem" database serves as a repository for information about different Instructors. It stores details such as the instructor 's unique identifier, email, name, password, and salary. This table is designed to support the effective organization and management of instructor-related data within the Examination System.

Attributes:

1. id (integer): Represents the unique identifier for each instructor, serving as the primary key for the " instructor " table.

2. Name (varchar (50)): Stores the name of the instructor, providing a concise label for identification.

3. Email (varchar (50)): Stores the email of the instructor, providing a concise label for identification and this attribute has index to not repeated.

4. Salary (int): Stores the salary of the instructor.

Functionality:

The " instructor" table offers the following functionalities:

1. Record Storage: Enables the storage of information about various instructor, including their names, emails, passwords and salary.

2. Identification: The "Id" column serves as a unique identifier for each instructor, ensuring individual records can be easily distinguished.

3. Data Organization: Supports the organization and categorization of instructor -related data, aiding administrators in efficiently managing information.

4. Referential Integrity: Utilizes the primary key constraint on the "id" column to maintain referential integrity within the table and uphold data consistency.

Course Table

Description:

The " Course" table within the "ExaminationSystem" database serves as a repository for information about different Instructors. It stores details such as the Course's unique identifier, name, duration, and grade. This table is designed to support the effective organization and management of instructor-related data within the Examination System.

Attributes:

1. id (integer): Represents the unique identifier for each instructor, serving as the primary key for the "Course" table.

2. Name (varchar (50)): Stores the name of the Course, providing a concise label for identification.

3. Duration (int): Stores the duration of the Course, providing a concise label for identification.

4. Grade (int): Stores the final grade of the Course.

Functionality:

The " Course" table offers the following functionalities:

1. Record Storage: Enables the storage of information about various Course, including their names, duration and grade.

2. Identification: The "Id" column serves as a unique identifier for each Course, ensuring individual records can be easily distinguished.

3. Data Organization: Supports the organization and categorization of Course-related data, aiding administrators in efficiently managing information.

4. Referential Integrity: Utilizes the primary key constraint on the "id" column to maintain referential integrity within the table and uphold data consistency.

Course Instructor Table

Description:

The "Course_Inst" table, residing within the "ExaminationSystem" database, is designed to establish relationships between Courses and instructors.

Attributes:

1. Course_id (integer): Represents the unique identifier for each Course, functioning as part of the composite primary key of the " Course _Inst" table. This column serves as a foreign key referencing the "id" column in the " Course " table.

2. Inst_id (integer): Denotes the unique identifier for each instructor, constituting part of the composite primary key of the " Course _Inst " table. This column serves as a foreign key referencing the "id" column in the "Instructor" table.

Functionality:

The " Course_Inst" table offers the following functionalities:

1. Relationship Establishment: Utilizes the " Course_id" and "Inst_id" columns as composite primary keys to establish relationships between departments and instructors.

2. Referential Integrity: Foreign key constraints ([FK_Dept_Inst_Course], [FK_Dept_Inst_Instructor]) ensure referential integrity by linking records in the " Course_Inst" table to corresponding records in the " Course " and "Instructor" tables. Additionally, the "ON DELETE CASCADE" option ensures that related records are automatically deleted when the corresponding Course or instructor is deleted.

Course Topic Table

Description:

The "Course_Topic" table, residing within the "ExaminationSystem" database, is designed to establish relationships between Courses and instructors.

Attributes:

1. Course_id (integer): Represents the unique identifier for each Course, functioning as part of the composite primary key of the " Course_Topic" table. This column serves as a foreign key referencing the "id" column in the " Course " table.

2. Topic (varchar (50)): Denotes Topic For each Course, constituting part of the composite primary key of the " Course_Topic" table.

Functionality:

The " Course_Topic" table offers the following functionalities:

1. Relationship Establishment: Utilizes the " Course_id" and "Topic" columns as composite primary keys to establish relationships between departments and instructors.

2. Referential Integrity: Foreign key constraints ([FK_Dept_Inst_Course] ensure referential integrity by linking records in the " Course_Topic" table to corresponding records in the " Course " table. Additionally, the "ON DELETE CASCADE" option ensures that related records are automatically deleted when the corresponding Course is deleted.

Exam Table

Description:
The "Exam" table is a fundamental component of the Examination System project, designed to store information related to exams generated within the system. It serves as a repository for organizing and managing various aspects of exams, including their titles and the number of true/false (TF) and multiple-choice questions (MCQ) associated with each exam.

Attributes:

1. id (integer): Represents the unique identifier assigned to each exam within the system.
2. title (varchar(50)): Denotes the title or name of the exam, providing a brief description of its content or focus.
3. Num_TF (integer): Indicates the number of true/false questions included in the exam, contributing to its overall structure and content.
4. Num_MCQ (integer): Specifies the number of multiple-choice questions incorporated into the exam, offering a variety of question formats to assess student knowledge and understanding.

Functionality:
The "generate_exam" stored procedure, when invoked by a student within the Examination System, dynamically generates exam questions based on predefined criteria. Upon execution, the procedure generates a set of true/false and multiple-choice questions, and the generated questions are subsequently inserted into the "Exam" table, with appropriate counts updated in the "Num_TF" and "Num_MCQ" columns.

Exam_St_Q Table

Description:

The "Exam_St_Q" table is designed to store interaction data between exams, students, and questions. It serves as a relational junction table, facilitating the recording and management of students' responses to specific questions within exams.

Attributes:

1. Exam_id (integer): Represents the exam ID associated with the interaction data, serving as a foreign key referencing the "id" column in the "Exam" table.
2. Student_id (integer): Denotes the student ID associated with the interaction data, serving as a foreign key referencing the "id" column in the "Student" table.
3. Q_id (integer): Specifies the question ID associated with the interaction data, serving as a foreign key referencing the "id" column in the "Question" table.
4. Answer (varchar(500)): Stores the student's answer to the corresponding question, allowing for the recording of textual responses.

Functionality:

The "Exam_St_Q" table plays a pivotal role in capturing student interactions within exams, including their responses to individual questions. It facilitates the following functionalities:

1. Recording Responses: Allows for the storage of students' answers to specific questions within exams, enabling detailed tracking of their performance.
2. Data Management: Facilitates efficient organization and management of interaction data, ensuring that responses are correctly associated with the corresponding exams, students, and questions.

Referential Integrity: Enforces referential integrity through foreign key constraints, guaranteeing that valid relationships are maintained between records in the "Exam_St_Q" table and related tables

Question Table

Description:

The “Question” table within the “ExaminationSystem” database serves as a repository for storing various exam questions. It captures essential details related to each question, including its unique identifier, title, type, and associated grade.

Attributes:

1. **id** (integer): Represents the unique identifier for each question. This column serves as the primary key for the “Question” table.
2. **Course_id** (integer): Stores the foreign key reference to the course to which the question belongs. It links each question to a specific course.
3. **Title** (varchar(500)): Contains the text of the question itself. This field provides the actual content of the question.
4. **Type** (tinyint): Indicates the type of the question (e.g., multiple-choice, true/false, etc.). It is a required field.
5. **Grade** (int): Represents the assigned grade or score for the question. It provides information about the question’s weight in the overall assessment.

Functionality:

The “Question” table offers the following functionalities:

1. **Storage**: Allows the storage of various exam questions, ensuring that each question is uniquely identified.
2. **Association**: The “Course_id” column establishes a relationship between questions and their corresponding courses.
3. **Content**: The “Title” field contains the actual question text, making it essential for understanding the content.
4. **Type Differentiation**: The “Type” field categorizes questions based on their format (e.g., multiple-choice, true/false).
5. **Scoring**: The “Grade” field provides information about the question’s weight in the overall assessment.

Question_choice Table

Description:

The “Question_choice” table within the “ExaminationSystem” database stores the available choices (options) for each exam question. It captures details related to each choice, including the question it belongs to, the choice text, and whether it is the correct answer.

Attributes:

1. **Q_id** (integer): Represents the unique identifier of the question to which the choice belongs. This column serves as part of the composite primary key for the “Question_choice” table.
2. **Choice** (varchar(500)): Contains the text of the choice (e.g., option A, option B, etc.). It provides the various options for answering the question.
3. **IsCorrect** (bit): Indicates whether the choice is the correct answer (1) or not (0). It is a required field.

Functionality:

The “Question_choice” table offers the following functionalities:

1. **Choice Storage**: Allows the storage of various choices associated with each question.
2. **Question Association**: The “Q_id” column establishes a relationship between choices and their corresponding questions.
3. **Content**: The “Choice” field contains the actual text of each choice, providing the available options for answering the question.
4. **Correctness**: The “IsCorrect” field indicates whether a choice is the correct answer (1) or not (0).

Students Table

Overview

The Students table within the Examy database is designed to meticulously manage information pertaining to enrolled students. It serves as a pivotal repository for crucial student details, facilitating efficient administration and tracking within the educational system.

Columns

1. Id (Primary Key):

   - Data Type: Integer (int)

   - Description: This attribute uniquely identifies each student within the system. It is automatically generated and strictly enforced as non-null.

2. Name:

   - Data Type: Unicode string (nvarchar(50))

   - Description: Denotes the name of the student, aiding in personalized identification and communication.

   - Constraints: Non-null.

3. Age:

   - Data Type: Integer (int)

   - Description: Represents the age of the student, providing demographic insights.

   - Constraints: Nullable.

4. Email:

- Data Type: Unicode string (nvarchar(100))

- Description: Captures the email address of the student, serving as a crucial contact point.

- Constraints: Non-null.

5. Password:

- Data Type: Unicode string (nvarchar(50))

- Description: Safeguards the student's account with a secure password.

- Constraints: Non-null.

6. Address:

- Data Type: Unicode string (nvarchar(100))

- Description: Records the residential address of the student, aiding in logistical correspondence.

- Constraints: Nullable.

7. DeptId (Foreign Key):

- Data Type: Integer (int)

- Description: Establishes a link to the department to which the student is affiliated.

- Constraints: Nullable.

- Foreign Key Constraint: Enforces referential integrity with the Id column of the Departments table, facilitating seamless navigation across related data.

  - On deletion of a department, associated student records are automatically cascaded for consistency.

Constraints

- Primary Key Constraint: Imposed on the Id column to ensure uniqueness and expedite efficient retrieval operations.

- Foreign Key Constraint: Guarantees integrity between the Students and Departments tables, fortifying relational coherence.

Indexes

No supplementary indexes are explicitly defined in the table schema.

Relationships

- One-to-Many Relationship: Each student is affiliated with a single department, as indicated by the DeptId foreign key.

Script Date

The script for table creation was executed on 3/6/2024 at 9:35:25 PM, ensuring traceability and version control.

Student_Course Table

Description:

The **Student_Course** table maintains records of students' course enrollments. It establishes a relationship between students and the courses they are registered for. Each entry in this table represents a student's enrollment in a particular course.

Attributes:

1. **Course_id** (integer): A unique identifier for each course. This column serves as part of the composite primary key for the **Student_Course** table.
2. **Student_id** (integer): A unique identifier for each student. This column also serves as part of the composite primary key.

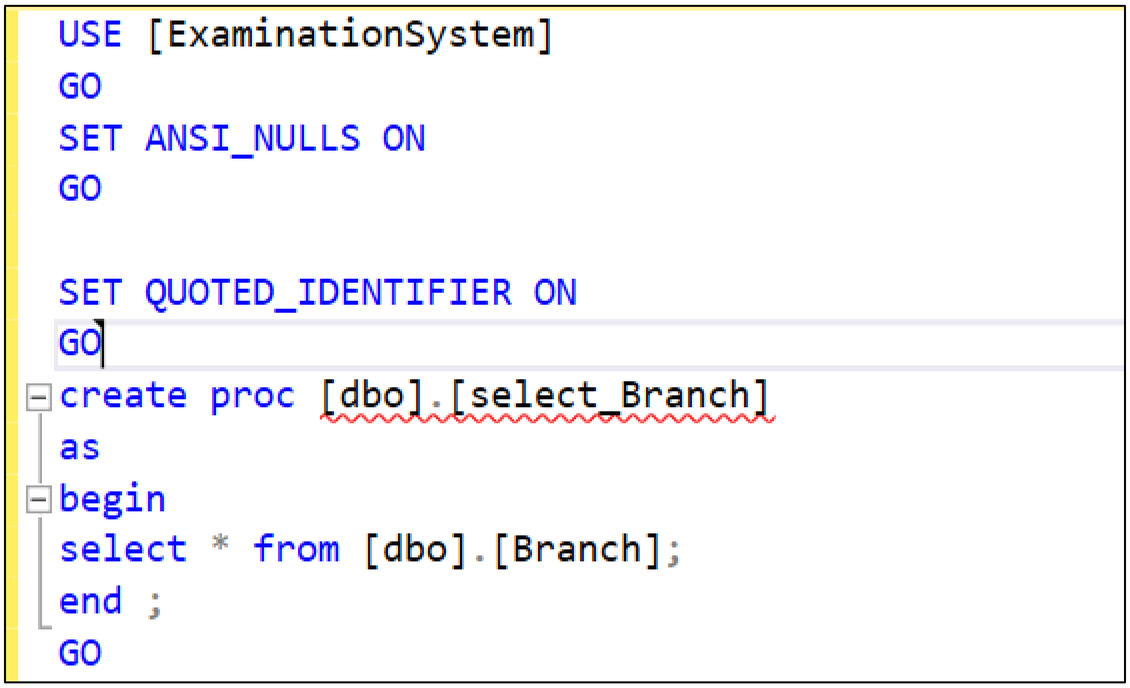
Functionality:

The **Student_Course** table provides the following functionalities:

1. **Enrollment Storage**: Allows the storage of student enrollments in specific courses.
2. **Course-Student Association**: The **Course_id** and **Student_id** columns establish a relationship between students and the courses they are taking.
3. **Content**: The table contains information about which students are enrolled in which courses.
4. **Primary Key**: The composite primary key ensures that each combination of **Course_id** and **Student_id** is unique.

Stored Procedures

- **Select Branch Stored Procedure**



```
USE [ExaminationSystem]
GO
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO
create proc [dbo].[select_Branch]
as
begin
select * from [dbo].[Branch];
end ;
GO
```

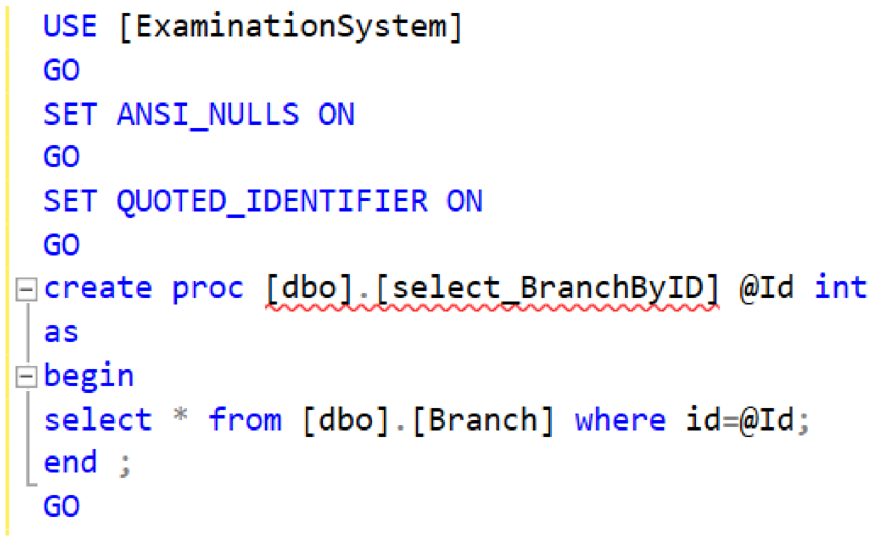
Description:

This stored procedure, named [dbo].[select_Branch], does not have any parameters. It retrieves all records from the [dbo].[Branch] table without any filtering criteria.

Functionality:

Upon execution, the procedure selects all columns from the [dbo].[Branch] table, providing a comprehensive snapshot of all data stored in the table. This stored procedure is designed for situations where a complete view of the branch information is required, without the need for specific filtering based on parameters. It serves as a convenient tool for obtaining an overview of all available data in the [dbo].[Branch] table within the Examination System database.

- **Select Branch By ID Stored Procedure**



```sql
USE [ExaminationSystem]
GO
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
create proc [dbo].[select_BranchByID] @Id int
as
begin
select * from [dbo].[Branch] where id=@Id;
end ;
GO
```

<u>Description:</u>

The stored procedure named select_BranchByID retrieves data from the [dbo].[Branch] table within the ExaminationSystem database. It focuses on a specific branch identified by its unique id. By passing the @Id parameter, users can obtain detailed information about a particular branch.

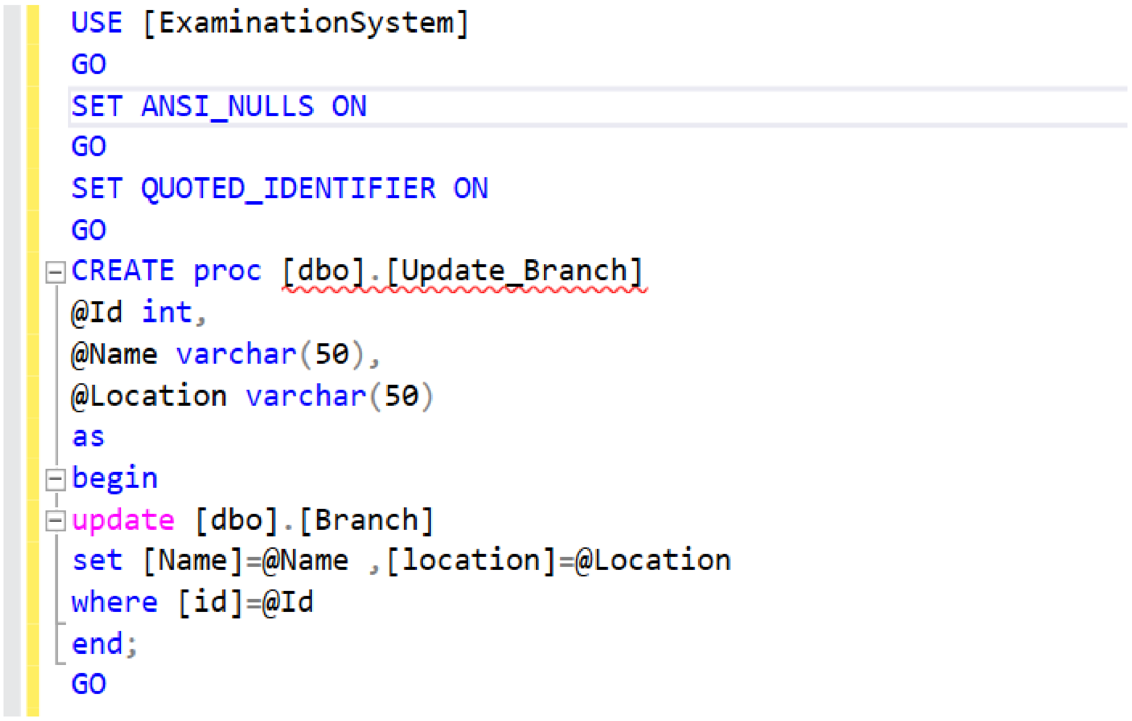
<u>Parameters:</u>

1. @Id (integer): Represents the unique identifier (ID) of the branch for data retrieval.

<u>Functionality:</u>

Upon invocation, this procedure selects records from the `[dbo].[Branch]` table where the `id` matches the provided `@Id` parameter.

- **Update Branch Stored Procedure**



```
USE [ExaminationSystem]
GO
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE proc [dbo].[Update_Branch]
@Id int,
@Name varchar(50),
@Location varchar(50)
as
begin
update [dbo].[Branch]
set [Name]=@Name ,[location]=@Location
where [id]=@Id
end;
GO
```

Description:

The stored procedure named Update_Branch is designed to update data within the [dbo].[Branch] table in the ExaminationSystem database. It allows modifications to the branch name and its corresponding location based on the provided parameters.

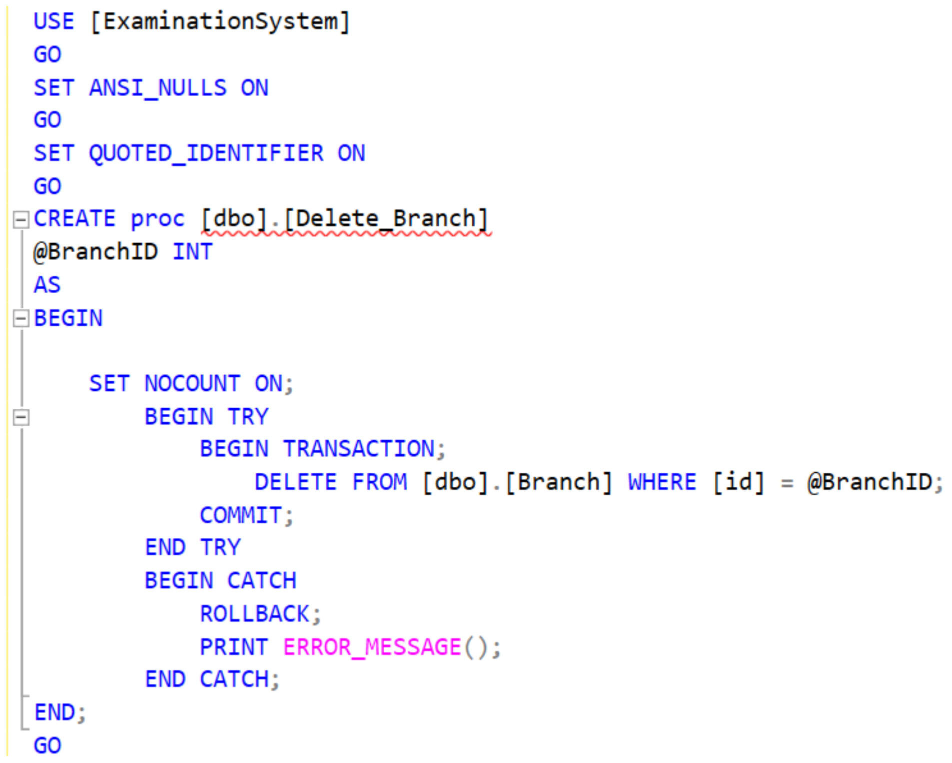
Parameters:

1. @Id (integer): Represents the unique identifier (ID) of the branch to be updated.
2. @Name (varchar, max length 50): Denotes the new name for the branch.
3. @Location (varchar, max length 50): Specifies the updated location associated with the branch.

Functionality:

Upon invocation, this procedure modifies the existing record in the [dbo].[Branch] table where the id matches the provided @Id parameter.

- **Delete Branch Stored Procedure**



```sql
USE [ExaminationSystem]
GO
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE proc [dbo].[Delete_Branch]
@BranchID INT
AS
BEGIN

    SET NOCOUNT ON;
        BEGIN TRY
            BEGIN TRANSACTION;
                DELETE FROM [dbo].[Branch] WHERE [id] = @BranchID;
            COMMIT;
        END TRY
        BEGIN CATCH
            ROLLBACK;
            PRINT ERROR_MESSAGE();
        END CATCH;
END;
GO
```

Description:

The stored procedure named Delete_Branch is designed to delete records from the [dbo].[Branch] table within the ExaminationSystem database. It allows removal of a branch based on the provided @BranchID parameter.

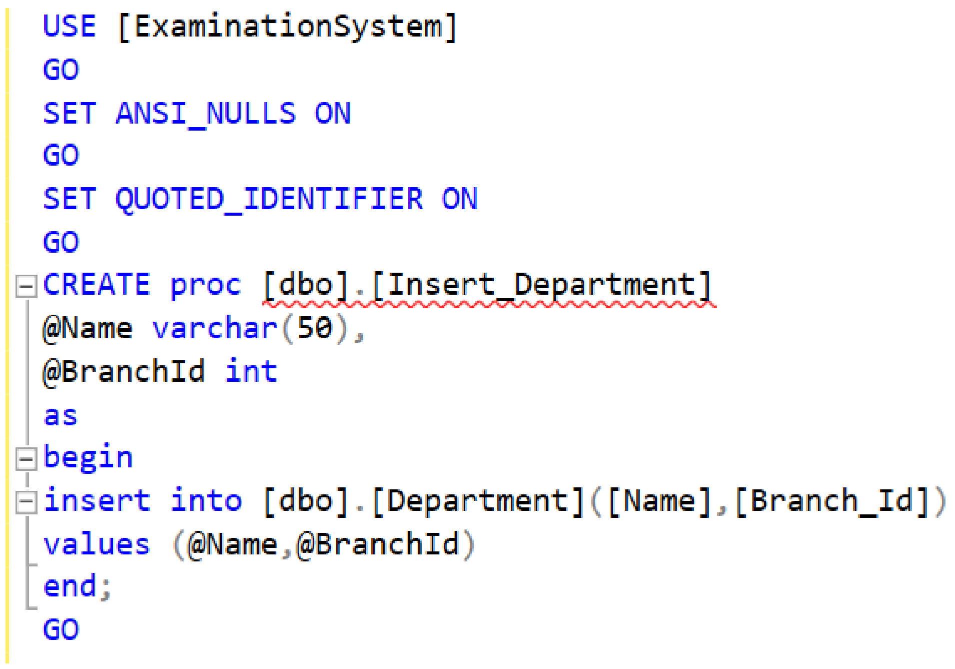
Parameters:

1. @BranchID (integer): Represents the unique identifier (ID) of the branch to be deleted.

Functionality:

Upon invocation, this procedure initiates a transaction to delete the branch record from the [dbo].[Branch] table where the id matches the provided @BranchID parameter.

- **Insert Department Stored Procedure**



```sql
USE [ExaminationSystem]
GO
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE proc [dbo].[Insert_Department]
@Name varchar(50),
@BranchId int
as
begin
insert into [dbo].[Department]([Name],[Branch_Id])
values (@Name,@BranchId)
end;
GO
```

<u>Description</u>:

The stored procedure named Insert_Department is designed to insert data into the [dbo].[Department] table within the ExaminationSystem database. It allows the addition of new department information, including the department name and its corresponding branch ID.

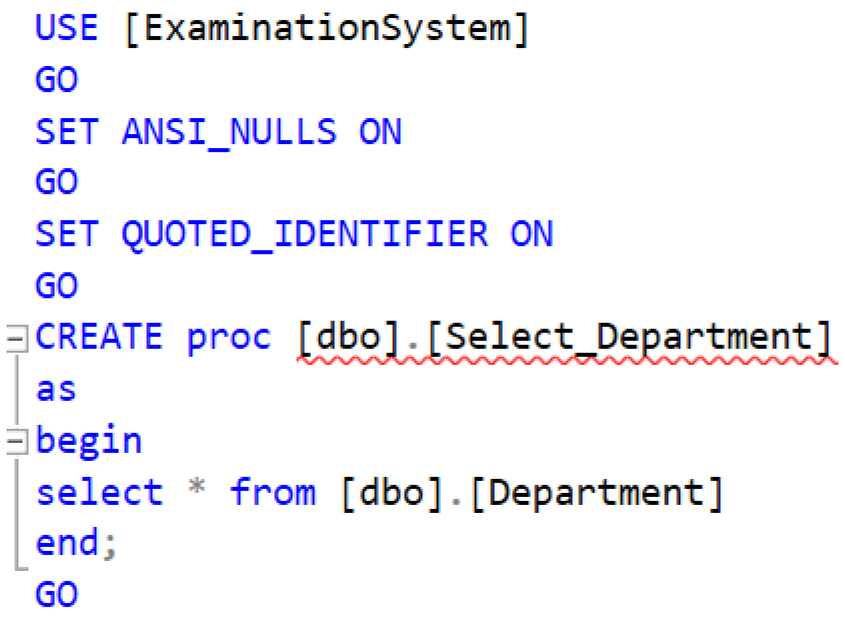
<u>Parameters:</u>

1. @Name (varchar, max length 50): Represents the name of the department to be inserted.
2. @BranchId (integer): Denotes the unique identifier (ID) of the branch associated with the department.

<u>Functionality:</u>

Upon invocation, this procedure inserts a new record into the [dbo].[Department] table, populating the Name and Branch_Id columns with the provided values.

- **Select Department Stored Procedure**



```
USE [ExaminationSystem]
GO
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE proc [dbo].[Select_Department]
as
begin
select * from [dbo].[Department]
end;
GO
```

<u>Description:</u>

The stored procedure named Select_Department retrieves all records from the [dbo].[Department] table within the ExaminationSystem database. It provides a comprehensive view of department-related information.

<u>Parameters:</u>

This procedure does not require any input parameters.

<u>Functionality:</u>

Upon invocation, this procedure performs a simple SELECT query, fetching all rows from the [dbo].[Department] table.

- **Select Department By ID Stored Procedure**

```
USE [ExaminationSystem]
GO
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
create proc [dbo].[Select_DepartmentByid] @id int
as
begin
select * from [dbo].[Department] where id=@id
end;
GO
```

Description:

The stored procedure named Select_DepartmentByid retrieves data from the [dbo].[Department] table within the ExaminationSystem database. It focuses on a specific department identified by its unique id. By passing the @id parameter, users can obtain detailed information about a particular department.

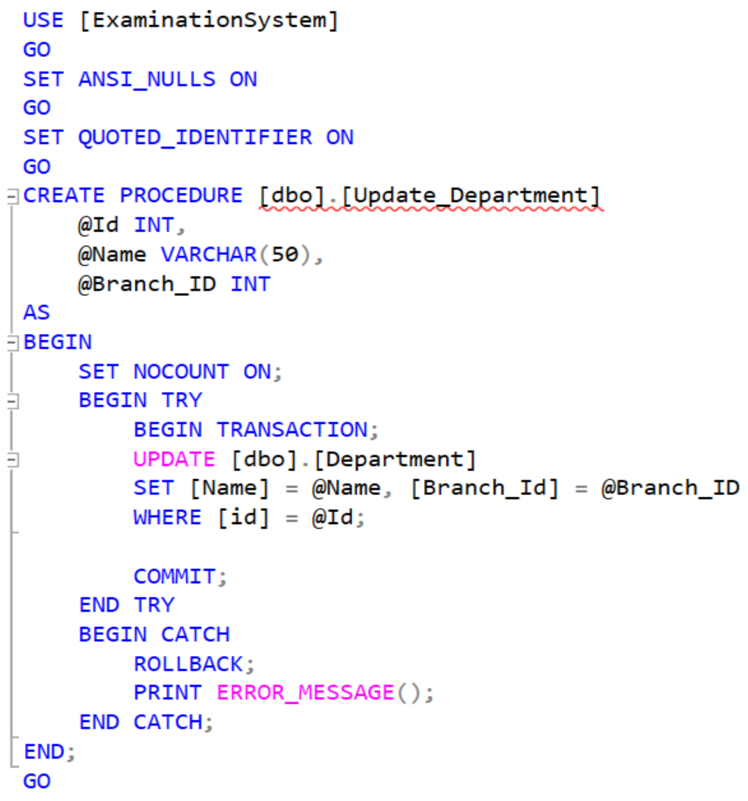
Parameters:

1. @id (integer): Represents the unique identifier (ID) of the department for data retrieval.

Functionality:

Upon invocation, this procedure selects records from the [dbo].[Department] table where the id matches the provided @id parameter.

- **Update Department Stored Procedure**



```
USE [ExaminationSystem]
GO
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE PROCEDURE [dbo].[Update_Department]
    @Id INT,
    @Name VARCHAR(50),
    @Branch_ID INT
AS
BEGIN
    SET NOCOUNT ON;
    BEGIN TRY
        BEGIN TRANSACTION;
        UPDATE [dbo].[Department]
        SET [Name] = @Name, [Branch_Id] = @Branch_ID
        WHERE [id] = @Id;

        COMMIT;
    END TRY
    BEGIN CATCH
        ROLLBACK;
        PRINT ERROR_MESSAGE();
    END CATCH;
END;
GO
```

Description:

The stored procedure named Update_Department is designed to update data within the [dbo].[Department] table in the ExaminationSystem database. It allows modifications to the branch name and its corresponding location based on the provided parameters.

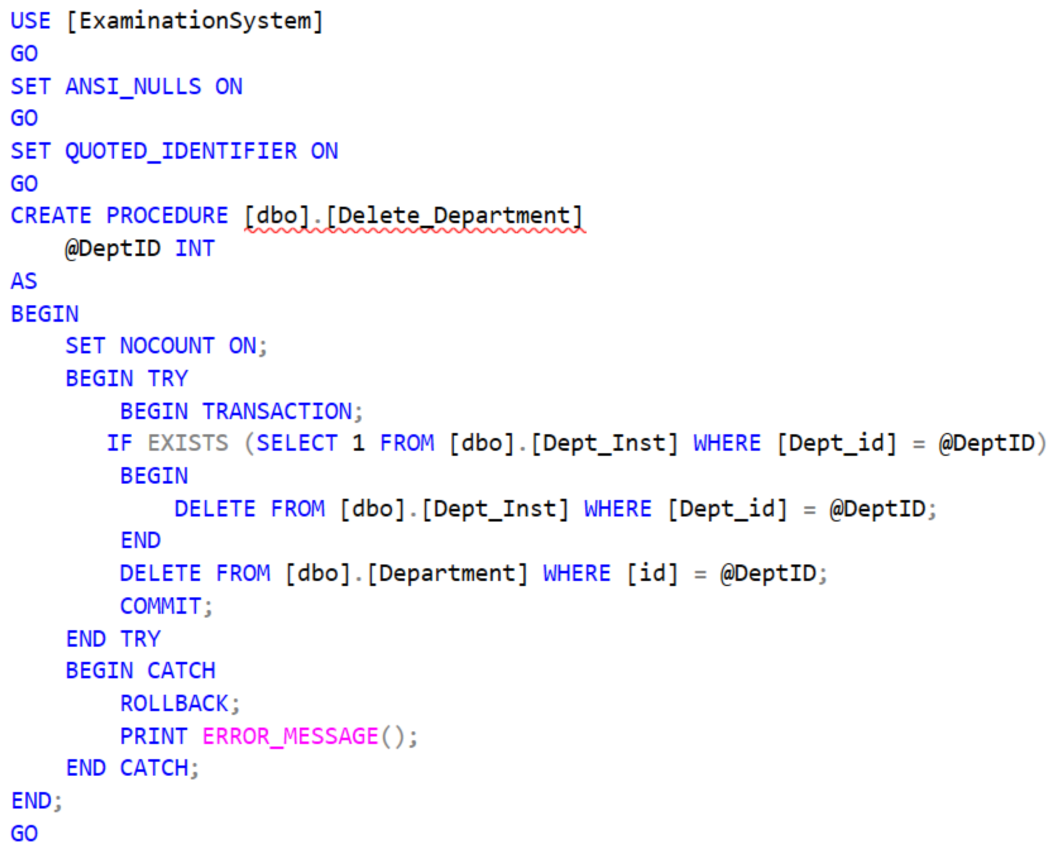
Parameters:

1. @Id (integer): Represents the unique identifier (ID) of the department to be updated.
2. @Name (varchar, max length 50): Denotes the new name for the department.
3. @Branch_ID (integer): Specifies the updated branch ID associated with the department.

Functionality:

Upon invocation, this procedure modifies the existing record in the [dbo].[Department] table where the id matches the provided @Id parameter.

- **Delete Department Stored Procedure**



```
USE [ExaminationSystem]
GO
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE PROCEDURE [dbo].[Delete_Department]
    @DeptID INT
AS
BEGIN
    SET NOCOUNT ON;
    BEGIN TRY
        BEGIN TRANSACTION;
       IF EXISTS (SELECT 1 FROM [dbo].[Dept_Inst] WHERE [Dept_id] = @DeptID)
        BEGIN
            DELETE FROM [dbo].[Dept_Inst] WHERE [Dept_id] = @DeptID;
        END
        DELETE FROM [dbo].[Department] WHERE [id] = @DeptID;
        COMMIT;
    END TRY
    BEGIN CATCH
        ROLLBACK;
        PRINT ERROR_MESSAGE();
    END CATCH;
END;
GO
```

Description:

The stored procedure named Delete_Department is designed to delete records from the [dbo].[Department] table within the ExaminationSystem database. It allows removal of a department based on the provided @DeptID parameter.

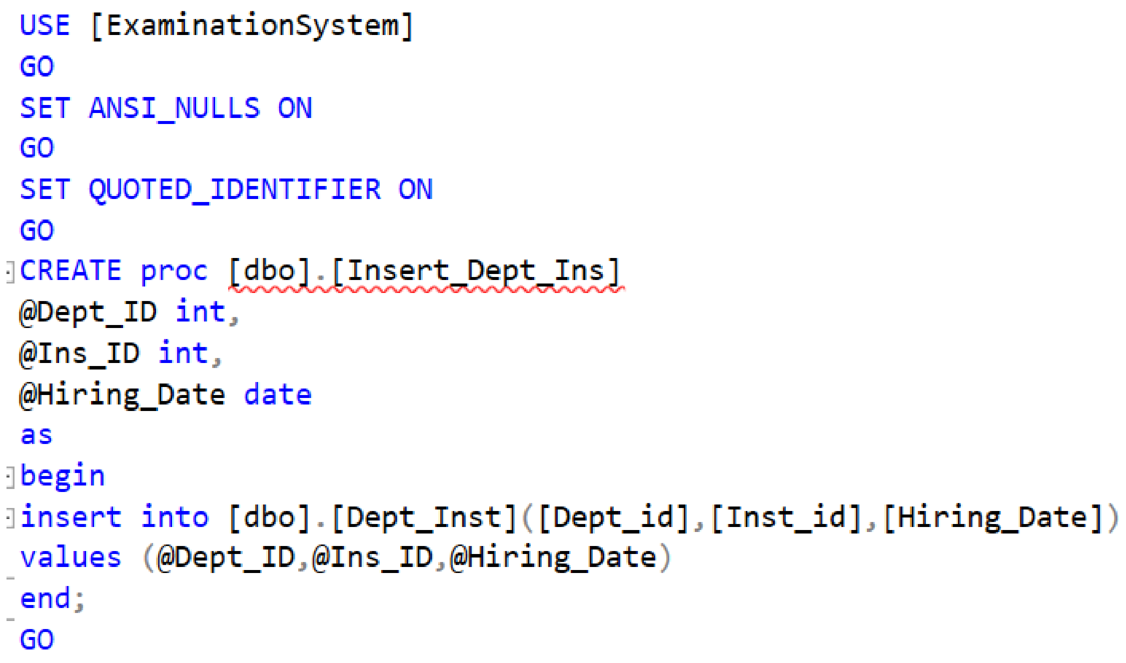
Parameters:

1. @DeptID (integer): Represents the unique identifier (ID) of the department to be deleted.

Functionality:

Upon invocation, this procedure initiates a transaction to delete the department record from the [dbo].[Department] table where the id matches the provided @DeptID parameter.

- **Insert Department Instructor Stored Procedure**



```
USE [ExaminationSystem]
GO
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE proc [dbo].[Insert_Dept_Ins]
@Dept_ID int,
@Ins_ID int,
@Hiring_Date date
as
begin
insert into [dbo].[Dept_Inst]([Dept_id],[Inst_id],[Hiring_Date])
values (@Dept_ID,@Ins_ID,@Hiring_Date)
end;
GO
```

Description:

The stored procedure named `Insert_Dept_Ins` is designed to insert data into the `[dbo]`.`[Dept_Inst]` table within the `ExaminationSystem` database. It allows the addition of new records representing the association between departments and instructors.

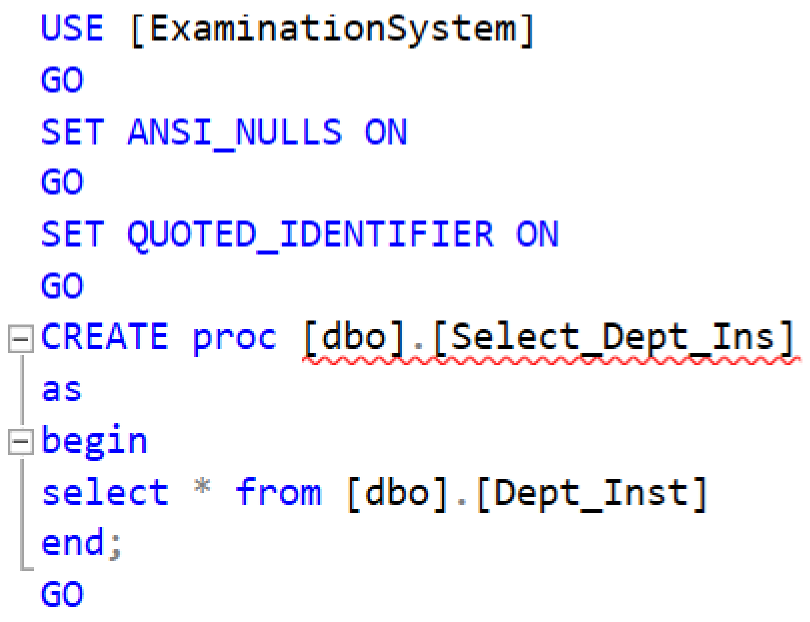
Parameters:

1. @Dept_ID (integer): Represents the unique identifier (ID) of the department.
2. @Ins_ID (integer): Denotes the unique identifier (ID) of the instructor.
3. @Hiring_Date (date): Specifies the date when the instructor was hired.

Functionality:

Upon invocation, this procedure inserts a new record into the `[dbo]`.`[Dept_Inst]` table, associating the specified department (`@Dept_ID`) with the corresponding instructor (`@Ins_ID`).

- **Select Department Instructor Stored Procedure**



```
USE [ExaminationSystem]
GO
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE proc [dbo].[Select_Dept_Ins]
as
begin
select * from [dbo].[Dept_Inst]
end;
GO
```

<u>Description</u>:

The stored procedure named Select_Dept_Ins retrieves all records from the [dbo].[Dept_Inst] table within the ExaminationSystem database. It provides a comprehensive view of department-instructor relationships.

<u>Parameters</u>:

This procedure does not require any input parameters.

<u>Functionality</u>:

Upon invocation, this procedure performs a simple SELECT query, fetching all rows from the [dbo].[Dept_Inst] table.

- **Select Department Instructor By ID Stored Procedure**

```
USE [ExaminationSystem]
GO
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
create proc [dbo].[Select_Dept_InsByDept_Id] @dept_Id int
as
begin
select * from [dbo].[Dept_Inst] where Dept_id=@dept_Id
end;
GO
```

Description:

The stored procedure named Select_Dept_InsByDept_Id retrieves data from the [dbo].[Dept_Inst] table within the ExaminationSystem database. It focuses on a specific department identified by its unique id. By passing the @dept_Id parameter, users can obtain detailed information about a particular department's instructors.

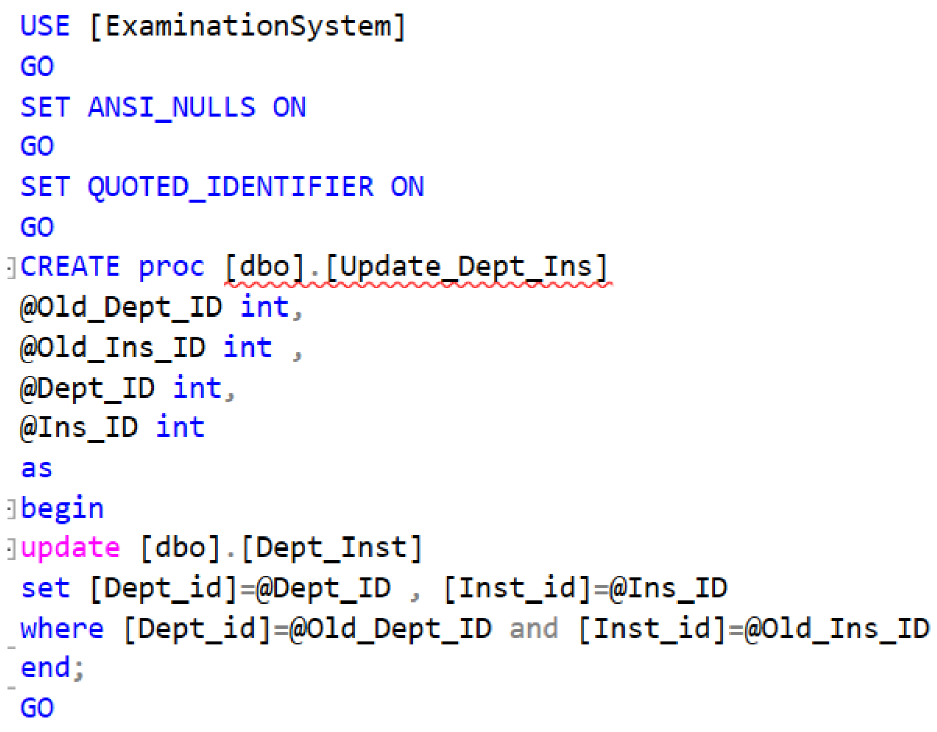
Parameters:

1. @dept_Id (integer): Represents the unique identifier (ID) of the department for data retrieval.

Functionality:

Upon invocation, this procedure selects records from the [dbo].[Dept_Inst] table where the Dept_id matches the provided @dept_Id parameter.

- **Update Department Instructor Stored Procedure**



```sql
USE [ExaminationSystem]
GO
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE proc [dbo].[Update_Dept_Ins]
@Old_Dept_ID int,
@Old_Ins_ID int ,
@Dept_ID int,
@Ins_ID int
as
begin
update [dbo].[Dept_Inst]
set [Dept_id]=@Dept_ID , [Inst_id]=@Ins_ID
where [Dept_id]=@Old_Dept_ID and [Inst_id]=@Old_Ins_ID
end;
GO
```

<u>**Description**</u>:

The stored procedure named `Update_Dept_Ins` is designed to update data within the `[dbo].[Dept_Inst]` table in the `ExaminationSystem` database. It allows modifications to the department ID and instructor ID associated with a specific record based on the provided parameters.

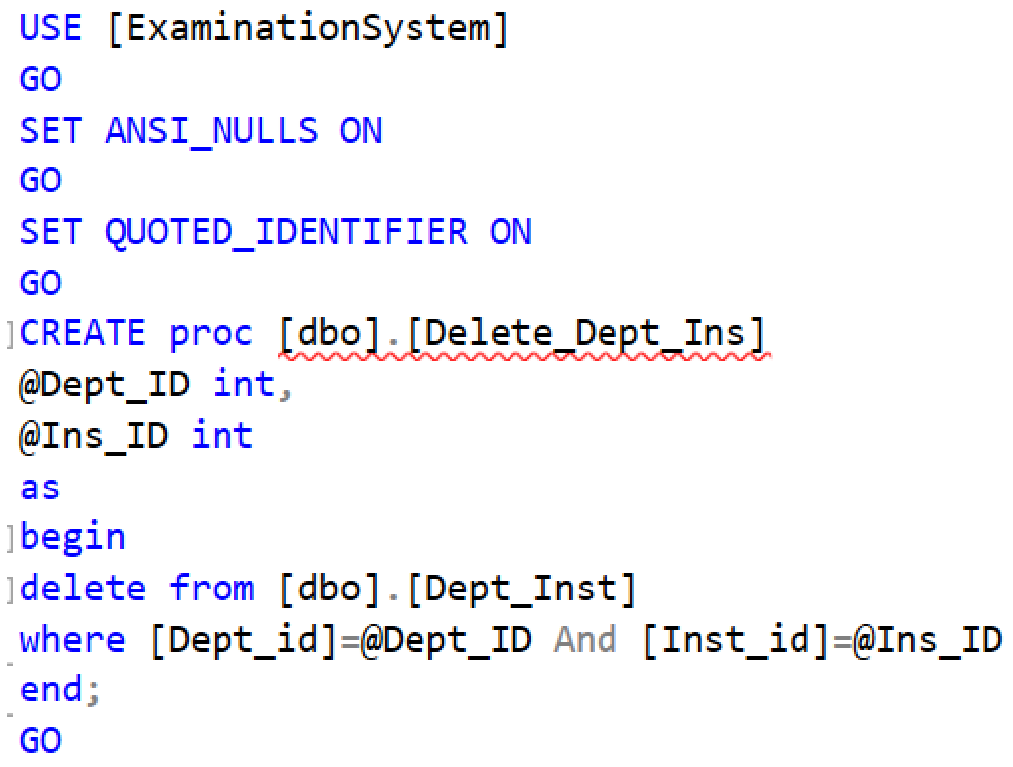
<u>**Parameters:**</u>

1. `@Old_Dept_ID` (integer): Represents the unique identifier (ID) of the existing department associated with the instructor.
2. `@Old_Ins_ID` (integer): Denotes the unique identifier (ID) of the existing instructor.
3. `@Dept_ID` (integer): Specifies the new department ID to be associated with the instructor.
4. `@Ins_ID` (integer): Specifies the new instructor ID.

<u>**Functionality**</u>:

Upon invocation, this procedure updates the existing record in the `[dbo].[Dept_Inst]` table where the `Dept_id` matches the provided `@Old_Dept_ID` parameter and the `Inst_id` matches the provided `@Old_Ins_ID` parameter.

- **Delete Department Instructor Stored Procedure**



```
USE [ExaminationSystem]
GO
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE proc [dbo].[Delete_Dept_Ins]
@Dept_ID int,
@Ins_ID int
as
begin
delete from [dbo].[Dept_Inst]
where [Dept_id]=@Dept_ID And [Inst_id]=@Ins_ID
end;
GO
```

Description:

The stored procedure named Delete_Dept_Ins is designed to delete records from the [dbo].[Dept_Inst] table within the ExaminationSystem database. It allows removal of a department-instructor relationship based on the provided @Dept_ID and @Ins_ID parameters.

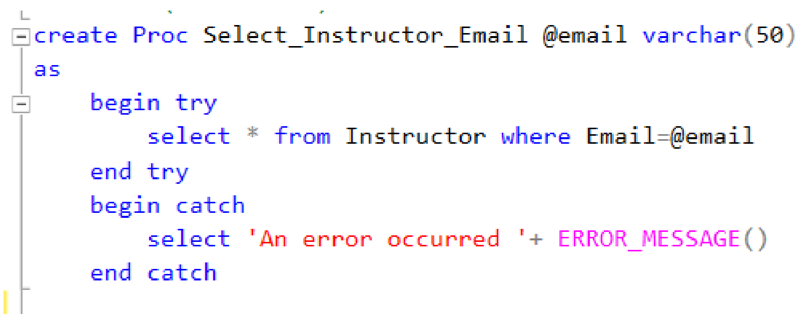
Parameters:

1. @Dept_ID (integer): Represents the unique identifier (ID) of the department associated with the instructor.
2. @Ins_ID (integer): Denotes the unique identifier (ID) of the instructor to be removed from the department.

Functionality:

Upon invocation, this procedure deletes the corresponding record from the [dbo].[Dept_Inst] table where the Dept_id matches the provided @Dept_ID parameter and the Inst_id matches the provided @Ins_ID parameter.

- **Select_Instructor_Email**



```
create Proc Select_Instructor_Email @email varchar(50)
as
    begin try
        select * from Instructor where Email=@email
    end try
    begin catch
        select 'An error occurred '+ ERROR_MESSAGE()
    end catch
```

Description:

The stored procedure named Select_Instructor_Email is designed to select records from the instructor table within the ExaminationSystem database. Based on Instructor Email.

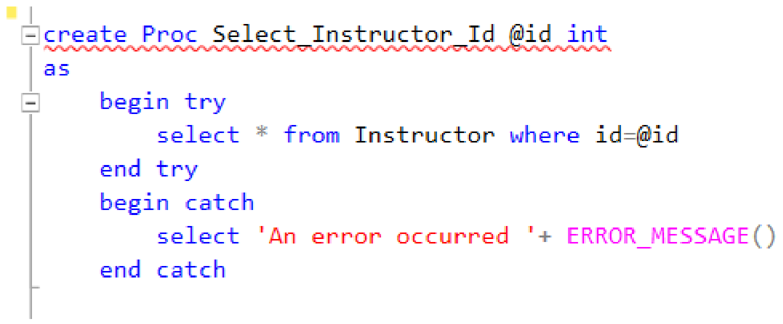
Parameters:

1. @email (varchar (50)): Email address of the instructor.

Functionality:

Retrieves instructor details matching the provided email address.

- **Select_Instructor_Id**



```sql
create Proc Select_Instructor_Id @id int
as
    begin try
        select * from Instructor where id=@id
    end try
    begin catch
        select 'An error occurred '+ ERROR_MESSAGE()
    end catch
```

Description:

The stored procedure named Select_Instructor_Id is designed to select records from the instructor table within the ExaminationSystem database. Based on Instructor Id.

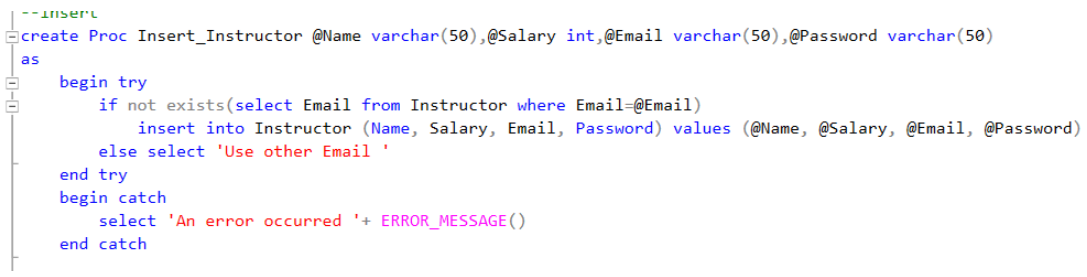
Parameters:

1. @id (int): Id of the instructor.

Functionality:

Retrieves instructor details matching the provided ID.

- **Insert_Instructor**



```
--insert
create Proc Insert_Instructor @Name varchar(50),@Salary int,@Email varchar(50),@Password varchar(50)
as
    begin try
        if not exists(select Email from Instructor where Email=@Email)
            insert into Instructor (Name, Salary, Email, Password) values (@Name, @Salary, @Email, @Password)
        else select 'Use other Email '
    end try
    begin catch
        select 'An error occurred '+ ERROR_MESSAGE()
    end catch
```

Description:

The stored procedure named Insert_Instructor is designed to add record into the instructor table within the ExaminationSystem database.

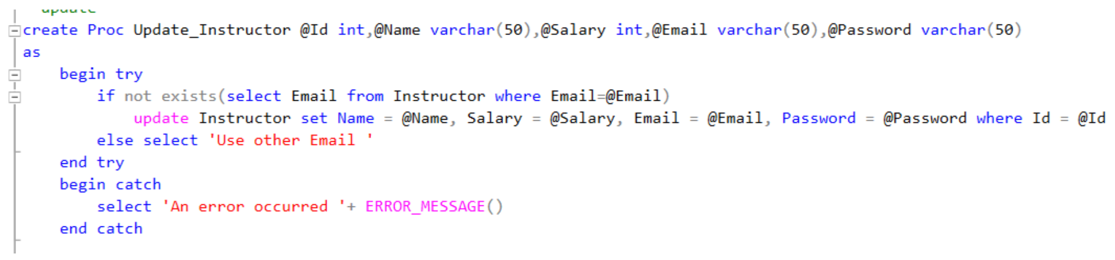
Parameters:

1. @Name (varchar (50)): Name of the instructor.
2. @Salary (int): Salary of the instructor.
3. @Email (varchar (50)): Email address of the instructor.
4. @Password (varchar (50)): Password of the instructor.

Functionality:

Inserts a new instructor if the provided email is not already in use.

- **Update_Instructor**



```
create Proc Update_Instructor @Id int,@Name varchar(50),@Salary int,@Email varchar(50),@Password varchar(50)
as
    begin try
        if not exists(select Email from Instructor where Email=@Email)
            update Instructor set Name = @Name, Salary = @Salary, Email = @Email, Password = @Password where Id = @Id
        else select 'Use other Email '
    end try
    begin catch
        select 'An error occurred '+ ERROR_MESSAGE()
    end catch
```

Description:

The stored procedure named Update_Instructor is designed to update record into the instructor table within the ExaminationSystem database based on Id.

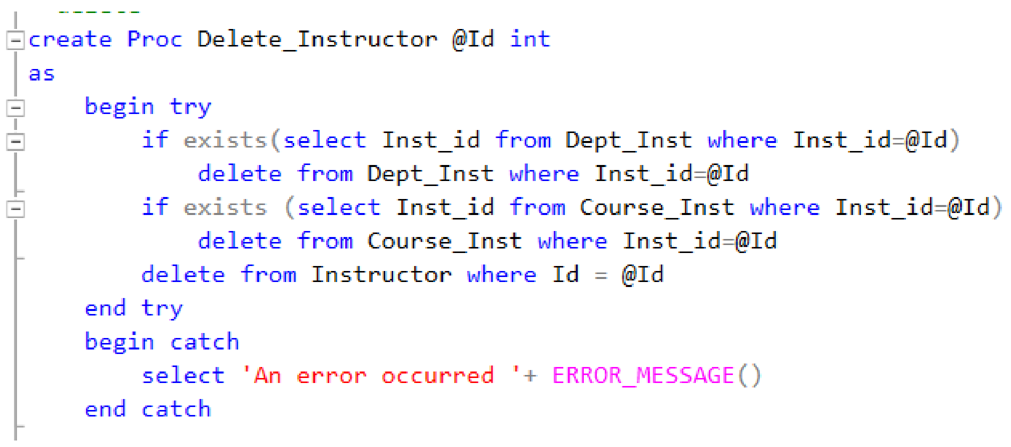
Parameters:

1. **@Id** (int): ID of the instructor.
2. **@Name** (varchar (50)): New name of the instructor.
3. **@Salary** (int): New salary of the instructor.
4. **@Email** (varchar (50)): New email address of the instructor.
5. **@Password** (varchar (50)): New password of the instructor.

Functionality:

Updates information of an existing instructor.

- **Delete_Instructor**



```sql
------
create Proc Delete_Instructor @Id int
as
    begin try
        if exists(select Inst_id from Dept_Inst where Inst_id=@Id)
            delete from Dept_Inst where Inst_id=@Id
        if exists (select Inst_id from Course_Inst where Inst_id=@Id)
            delete from Course_Inst where Inst_id=@Id
        delete from Instructor where Id = @Id
    end try
    begin catch
        select 'An error occurred '+ ERROR_MESSAGE()
    end catch
```

Description:

The stored procedure named Delete_Instructor is designed to Delete record From the instructor table within the ExaminationSystem database based on Id.

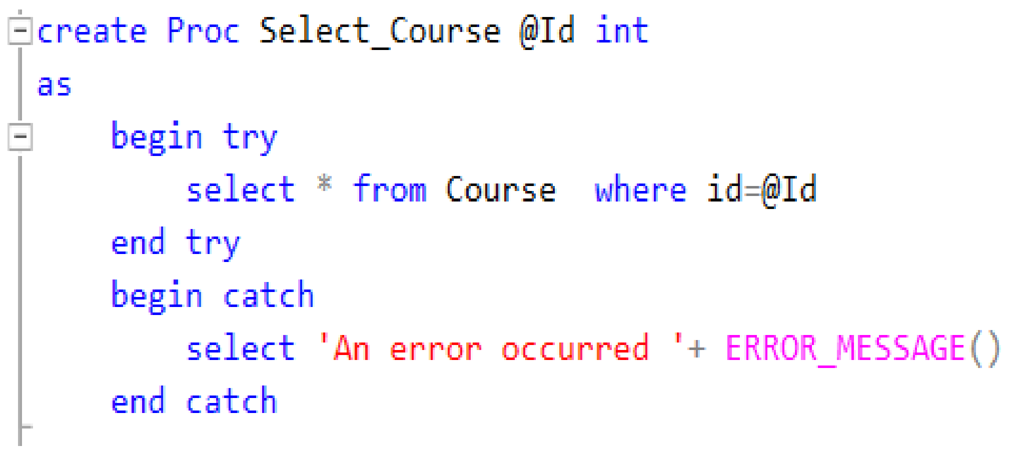
Parameters:

1. **@Id** (int): ID of the instructor.

Functionality:

Deletes an instructor from the database.

- **Select_Course**



```
create Proc Select_Course @Id int
as
    begin try
        select * from Course  where id=@Id
    end try
    begin catch
        select 'An error occurred '+ ERROR_MESSAGE()
    end catch
```

Description:

The stored procedure named Select_Course is designed to select record From the course table within the ExaminationSystem database based on Id.

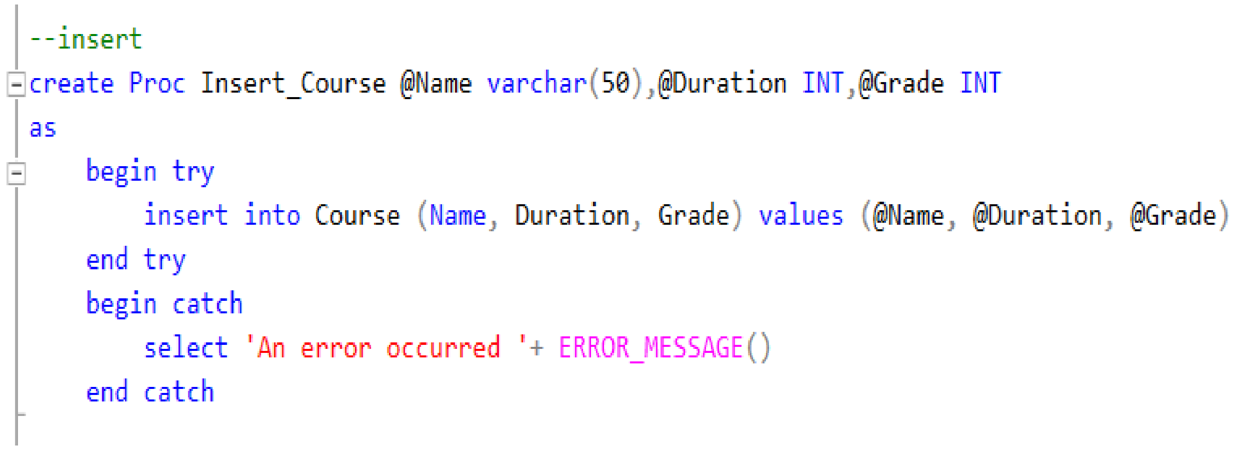
Parameters:

1. **@Id** (int): ID of the instructor.

Functionality:

Retrieves course details matching the provided ID.

- **Insert_Course**



```
--insert
create Proc Insert_Course @Name varchar(50),@Duration INT,@Grade INT
as
    begin try
        insert into Course (Name, Duration, Grade) values (@Name, @Duration, @Grade)
    end try
    begin catch
        select 'An error occurred '+ ERROR_MESSAGE()
    end catch
```

<u>Description</u>:

The stored procedure named Insert_Course is designed to add records into the course table within the ExaminationSystem database.

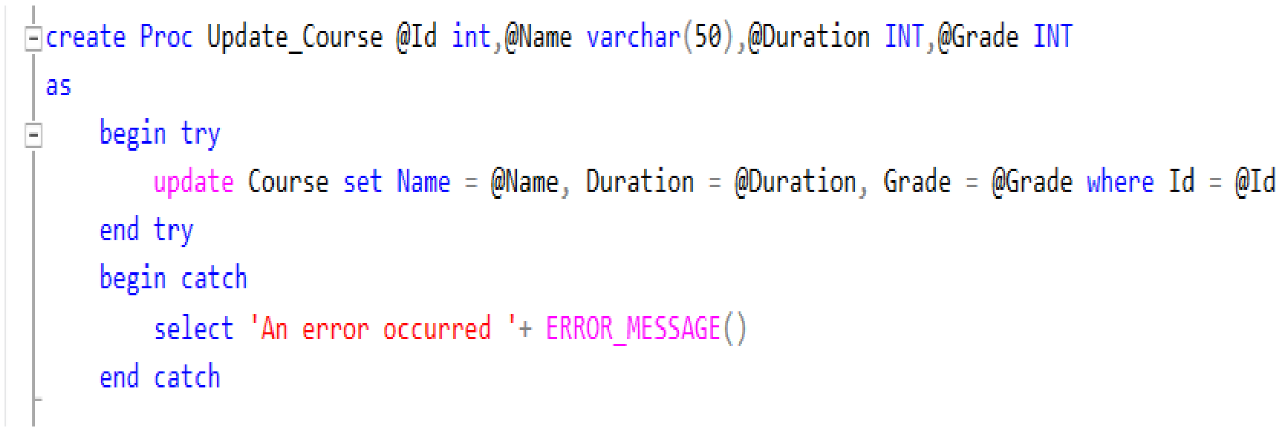
<u>Parameters</u>:

1. **@Name (varchar (50)):** Name of the course.
2. **@Duration (int):** Duration of the course.
3. **@Grade (int):** Grade of the course.

<u>Functionality</u>:

Inserts a new course with provided details.

- **Update_Course**



```sql
create Proc Update_Course @Id int,@Name varchar(50),@Duration INT,@Grade INT
as
    begin try
        update Course set Name = @Name, Duration = @Duration, Grade = @Grade where Id = @Id
    end try
    begin catch
        select 'An error occurred '+ ERROR_MESSAGE()
    end catch
```

Description:

The stored procedure named Update_Course is designed to Update records From the course table within the ExaminationSystem database Based On Id.

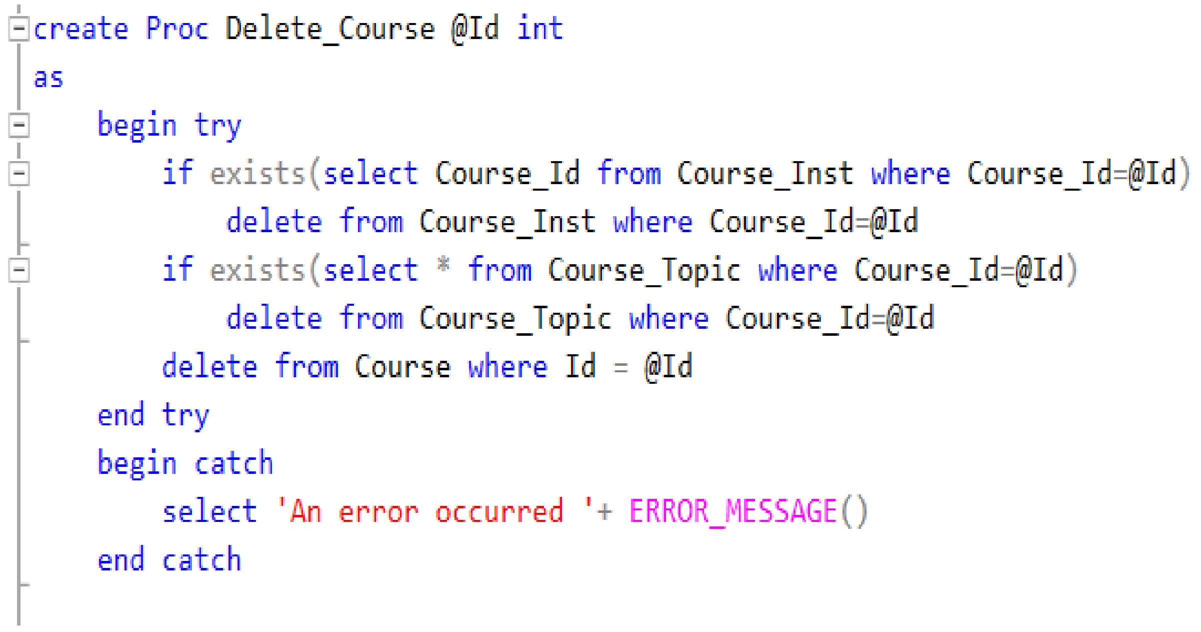
Parameters:

1. **@Id (int):** ID of the course.
2. **@Name (varchar (50)):** New name of the course.
3. **@Duration (int):** New duration of the course.
4. **@Grade (int):** New grade of the course.

Functionality:

Updates the details of an existing course.

- **Delete_Course**



```
create Proc Delete_Course @Id int
as
    begin try
        if exists(select Course_Id from Course_Inst where Course_Id=@Id)
            delete from Course_Inst where Course_Id=@Id
        if exists(select * from Course_Topic where Course_Id=@Id)
            delete from Course_Topic where Course_Id=@Id
        delete from Course where Id = @Id
    end try
    begin catch
        select 'An error occurred '+ ERROR_MESSAGE()
    end catch
```

Description:

The stored procedure named Delete_Course is designed to Delete records From the course table within the ExaminationSystem database Based On Id.

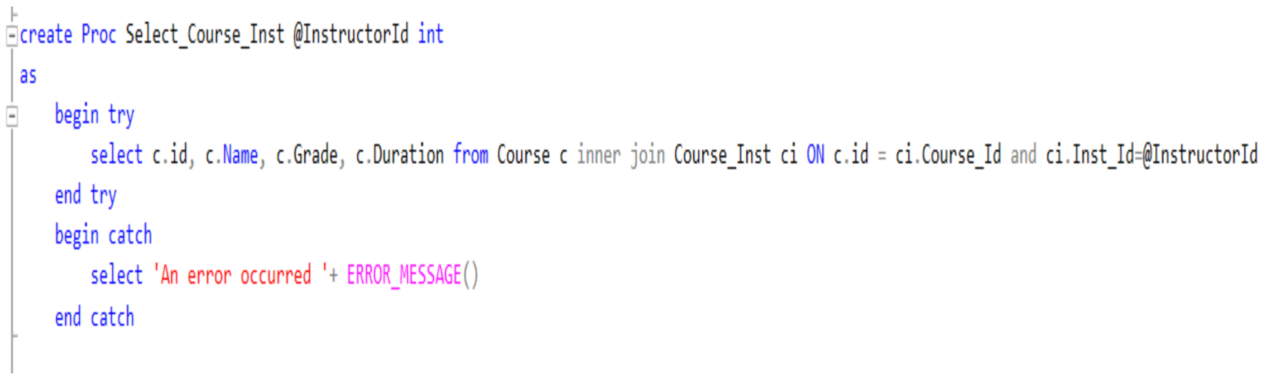
Parameters:

1. **@Id (int):** ID of the course.

Functionality:

Updates the details of an existing course.

- **Select_Course_Inst**



```sql
create Proc Select_Course_Inst @InstructorId int
as
    begin try
        select c.id, c.Name, c.Grade, c.Duration from Course c inner join Course_Inst ci ON c.id = ci.Course_Id and ci.Inst_Id=@InstructorId
    end try
    begin catch
        select 'An error occurred '+ ERROR_MESSAGE()
    end catch
```

<u>Description</u>:

The stored procedure named Select_Course_Inst is designed to Select records From the Course_Inst table within the ExaminationSystem database Based On Id.

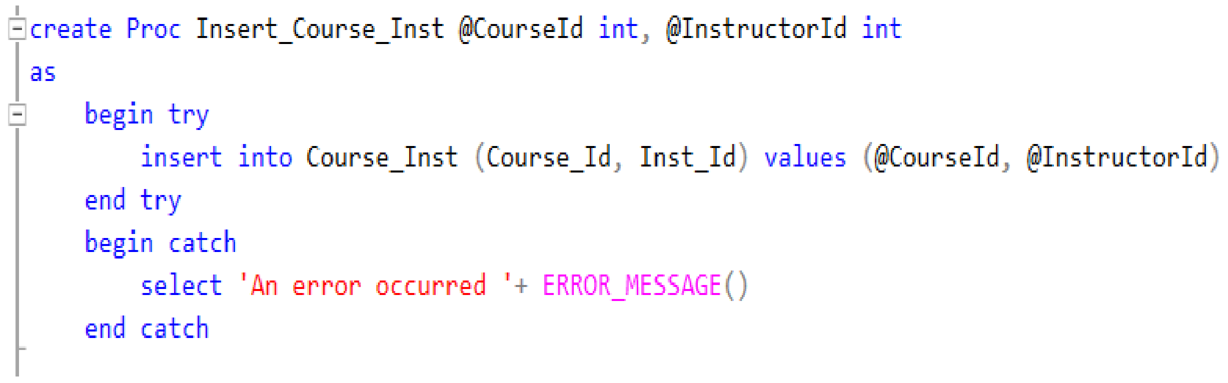
<u>Parameters</u>:

1. **@ InstructorId (int):** ID of the Instructor.

<u>Functionality</u>:

Retrieval of All Courses Data That Associated with the instructor.

- **Insert_Course_Inst**



```sql
create Proc Insert_Course_Inst @CourseId int, @InstructorId int
as
    begin try
        insert into Course_Inst (Course_Id, Inst_Id) values (@CourseId, @InstructorId)
    end try
    begin catch
        select 'An error occurred '+ ERROR_MESSAGE()
    end catch
```

Description:

The stored procedure named Insert_Course_Inst is designed to Add New records into the Course_Inst table within the ExaminationSystem database.

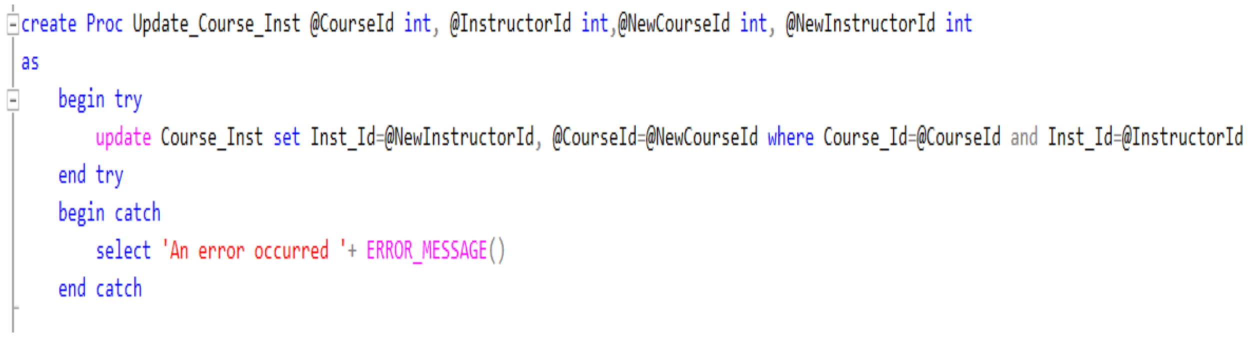
Parameters:

1. **@CourseId (int):** ID of the course.
2. **@InstructorId (int):** ID of the instructor.

Functionality:

Associates a course with an instructor.

- **Update_Course_Inst**



```
create Proc Update_Course_Inst @CourseId int, @InstructorId int,@NewCourseId int, @NewInstructorId int
as
    begin try
        update Course_Inst set Inst_Id=@NewInstructorId, @CourseId=@NewCourseId where Course_Id=@CourseId and Inst_Id=@InstructorId
    end try
    begin catch
        select 'An error occurred '+ ERROR_MESSAGE()
    end catch
```

Description:

The stored procedure named Update_Course_Inst is designed to Update records into the Course_Inst table within the ExaminationSystem database.

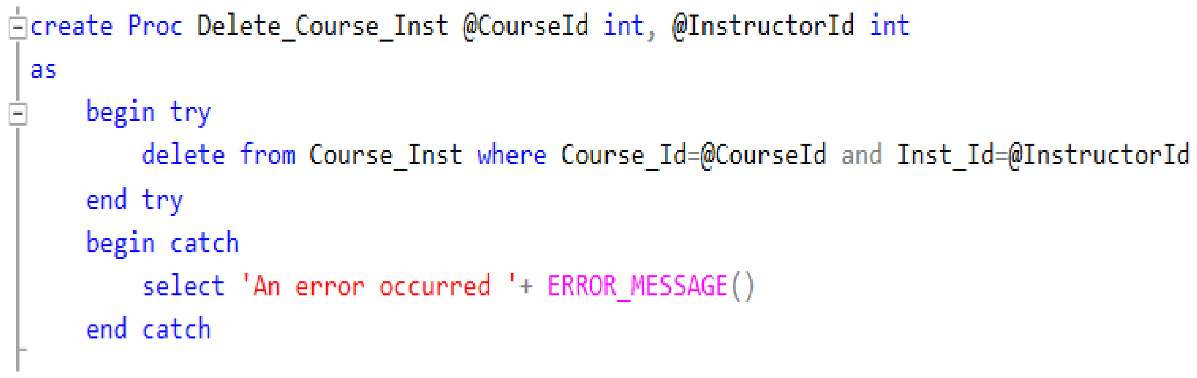
Parameters:

1. **@CourseId (int):** ID of the course.
2. **@InstructorId (int):** ID of the instructor.
3. **@NewCourseId (int):** New ID of the course.
4. **@NewInstructorId (int):** New ID of the instructor.

Functionality:

Updates course-instructor association.

- **Delete_Course_Inst**



```
create Proc Delete_Course_Inst @CourseId int, @InstructorId int
as
    begin try
        delete from Course_Inst where Course_Id=@CourseId and Inst_Id=@InstructorId
    end try
    begin catch
        select 'An error occurred '+ ERROR_MESSAGE()
    end catch
```

Description:

The stored procedure named Delete_Course_Inst is designed to Delete records from the Course_Inst table within the ExaminationSystem database.

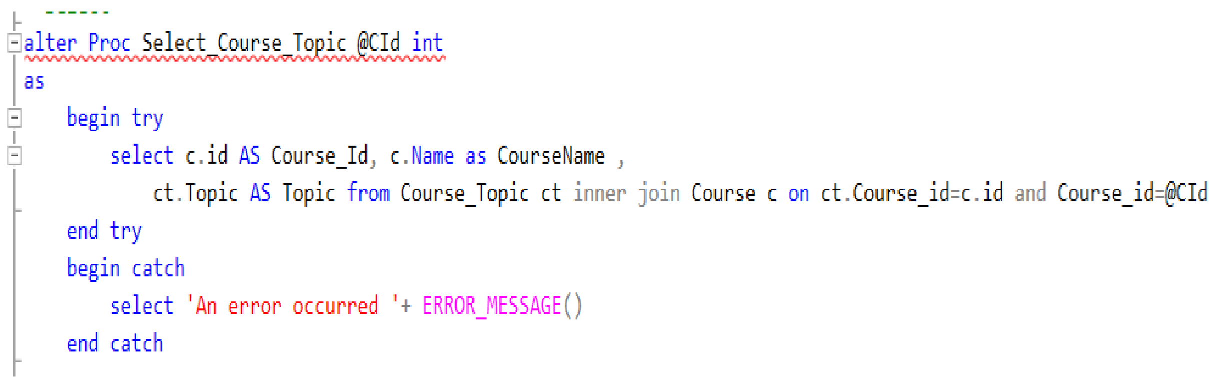
Parameters:

1. **@CourseId (int):** ID of the course.
2. **@InstructorId (int):** ID of the instructor.

Functionality:

Deletes course-instructor association.

- **Select_Course_Topic**



```
 ------
alter Proc Select_Course_Topic @CId int
as
    begin try
        select c.id AS Course_Id, c.Name as CourseName ,
            ct.Topic AS Topic from Course_Topic ct inner join Course c on ct.Course_id=c.id and Course_id=@CId
    end try
    begin catch
        select 'An error occurred '+ ERROR_MESSAGE()
    end catch
```

Description:

The stored procedure named Select_Course_Topic is designed to select records of topics for specific course from the Course_Topic table within the ExaminationSystem database.

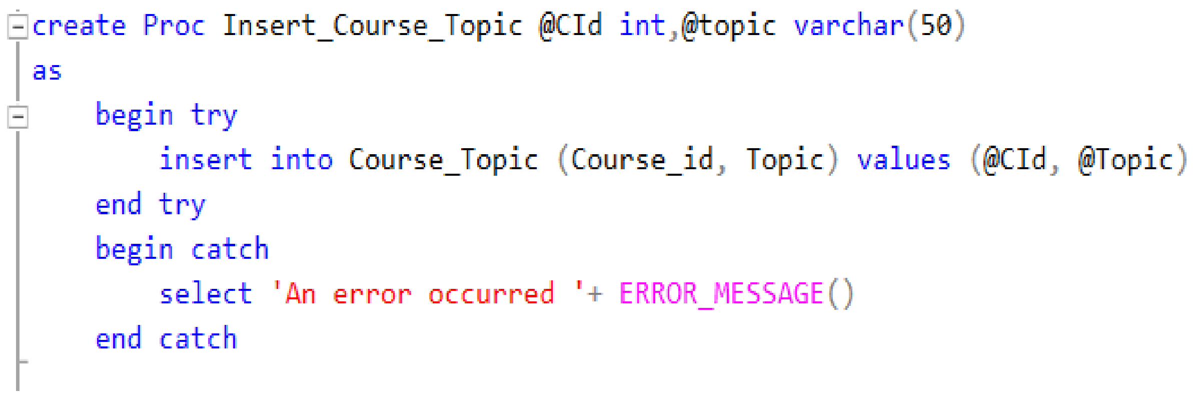
Parameters:

1. **@CId (int):** ID of the course.

Functionality:

Selects topics of a course.

- **Insert_Course_Topic**



```
create Proc Insert_Course_Topic @CId int,@topic varchar(50)
as
    begin try
        insert into Course_Topic (Course_id, Topic) values (@CId, @Topic)
    end try
    begin catch
        select 'An error occurred '+ ERROR_MESSAGE()
    end catch
```

Description:

The stored procedure named Insert_Course_Topic is designed to Add records of topics for specific course from the Course_Topic table within the ExaminationSystem database.

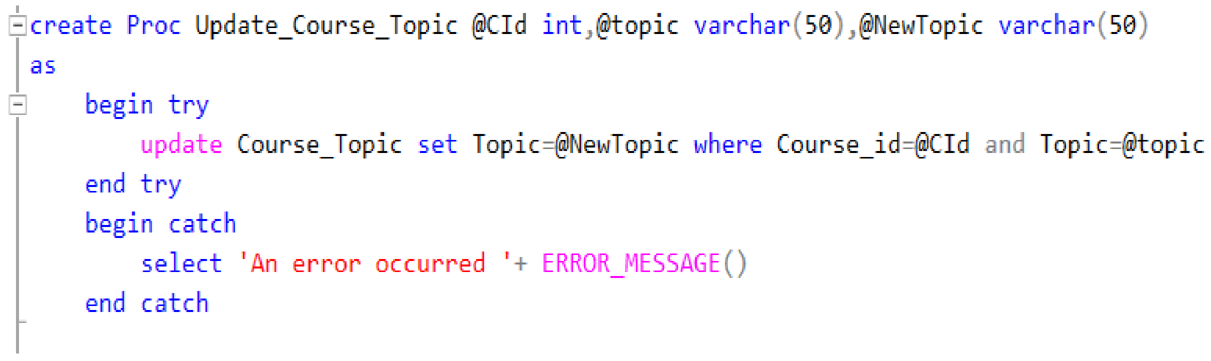
Parameters:

1. **@CId (int):** ID of the course.
2. **@topic (varchar (50)):** Topic for Course.

Functionality:

Inserts a topic for a course.

- **Update_Course_Topic**



```
create Proc Update_Course_Topic @CId int,@topic varchar(50),@NewTopic varchar(50)
as
    begin try
        update Course_Topic set Topic=@NewTopic where Course_id=@CId and Topic=@topic
    end try
    begin catch
        select 'An error occurred '+ ERROR_MESSAGE()
    end catch
```

<u>Description</u>:

The stored procedure named Update_Course_Topic is designed to Update records of topics for specific course from the Course_Topic table within the ExaminationSystem database.

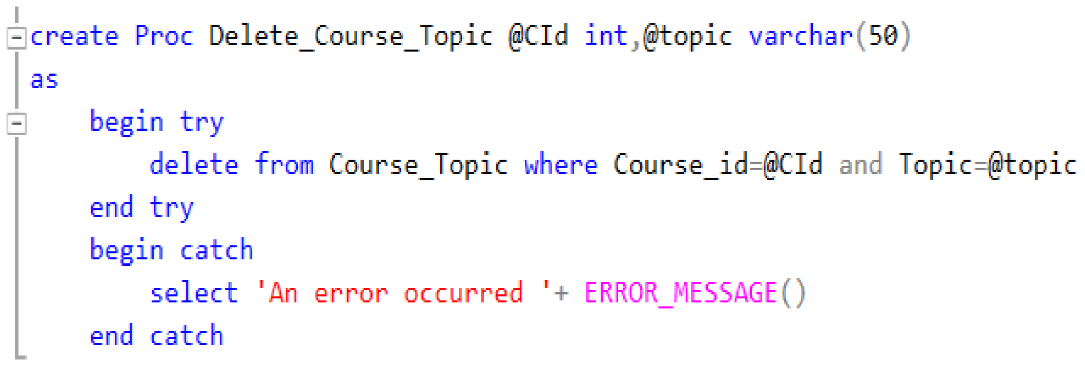
<u>Parameters</u>:

1. **@CId (int):** ID of the course.
2. **@topic (varchar (50)):** Topic of the course.
3. **@NewTopic (varchar (50)):** New topic of the course.

<u>Functionality</u>:

Updates topic of a course.

- **Delete_Course_Topic**



```
create Proc Delete_Course_Topic @CId int,@topic varchar(50)
as
    begin try
        delete from Course_Topic where Course_id=@CId and Topic=@topic
    end try
    begin catch
        select 'An error occurred '+ ERROR_MESSAGE()
    end catch
```

Description:

The stored procedure named Delete_Course_Topic is designed to Delete records of topics for specific course from the Course_Topic table within the ExaminationSystem database.

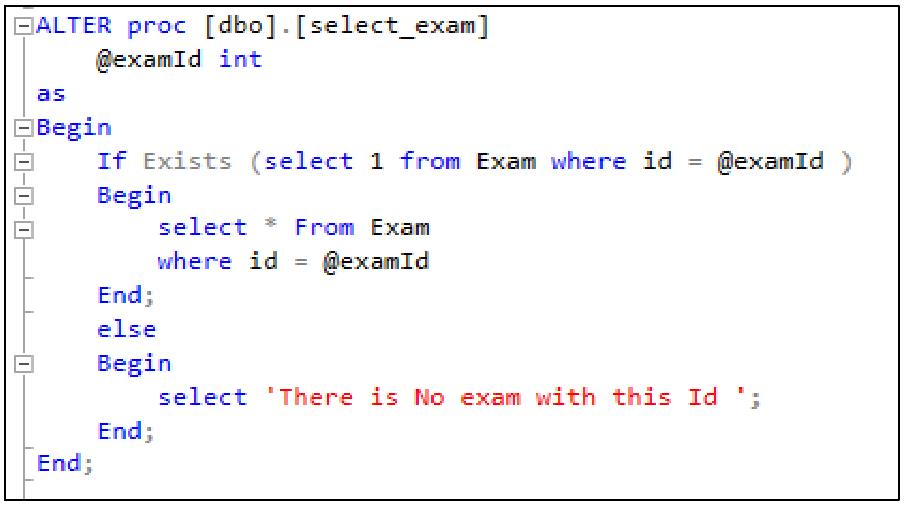
Parameters:

1. **@CId (int):** ID of the course.
2. **@topic (varchar (50)):** Topic of the course.

Functionality:

Deletes topic of a course.

- **Select From Exam Stored Procedure**



```
ALTER proc [dbo].[select_exam]
    @examId int
as
Begin
    If Exists (select 1 from Exam where id = @examId )
    Begin
        select * From Exam
        where id = @examId
    End;
    else
    Begin
        select 'There is No exam with this Id ';
    End;
End;
```

Description:

The select_exam stored procedure is designed to retrieve information about a specific exam based on the provided exam ID.

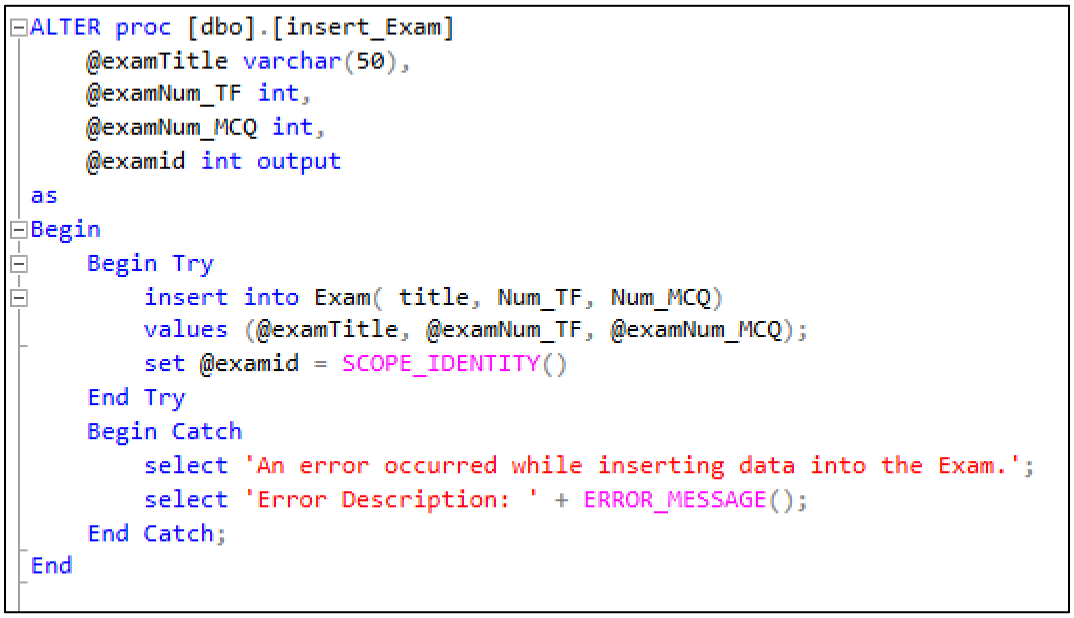
Parameters:

@examId (integer): This parameter represents the unique identifier of the exam for which information is to be retrieved. It is an input parameter required by the stored procedure to execute successfully.

Functionality:

Upon receiving the exam ID as input, the stored procedure first checks whether an exam with the provided ID exists in the database. If an exam with the specified ID is found, the stored procedure proceeds to fetch and return all details associated with that particular exam. However, if no exam is found matching the provided ID, the stored procedure returns a custom message indicating the absence of such an exam.

- **Insert in Exam Stored Procedure**



```
ALTER proc [dbo].[insert_Exam]
    @examTitle varchar(50),
    @examNum_TF int,
    @examNum_MCQ int,
    @examid int output
as
Begin
    Begin Try
        insert into Exam( title, Num_TF, Num_MCQ)
        values (@examTitle, @examNum_TF, @examNum_MCQ);
        set @examid = SCOPE_IDENTITY()
    End Try
    Begin Catch
        select 'An error occurred while inserting data into the Exam.';
        select 'Error Description: ' + ERROR_MESSAGE();
    End Catch;
End
```

Description:

The insert_Exam stored procedure facilitates the insertion of new exam records into the database. It is designed to add details of a new exam, such as title, number of True/False questions (Num_TF), and number of Multiple Choice Questions (Num_MCQ). Additionally, it provides functionality to output the generated exam ID for reference or further operations.

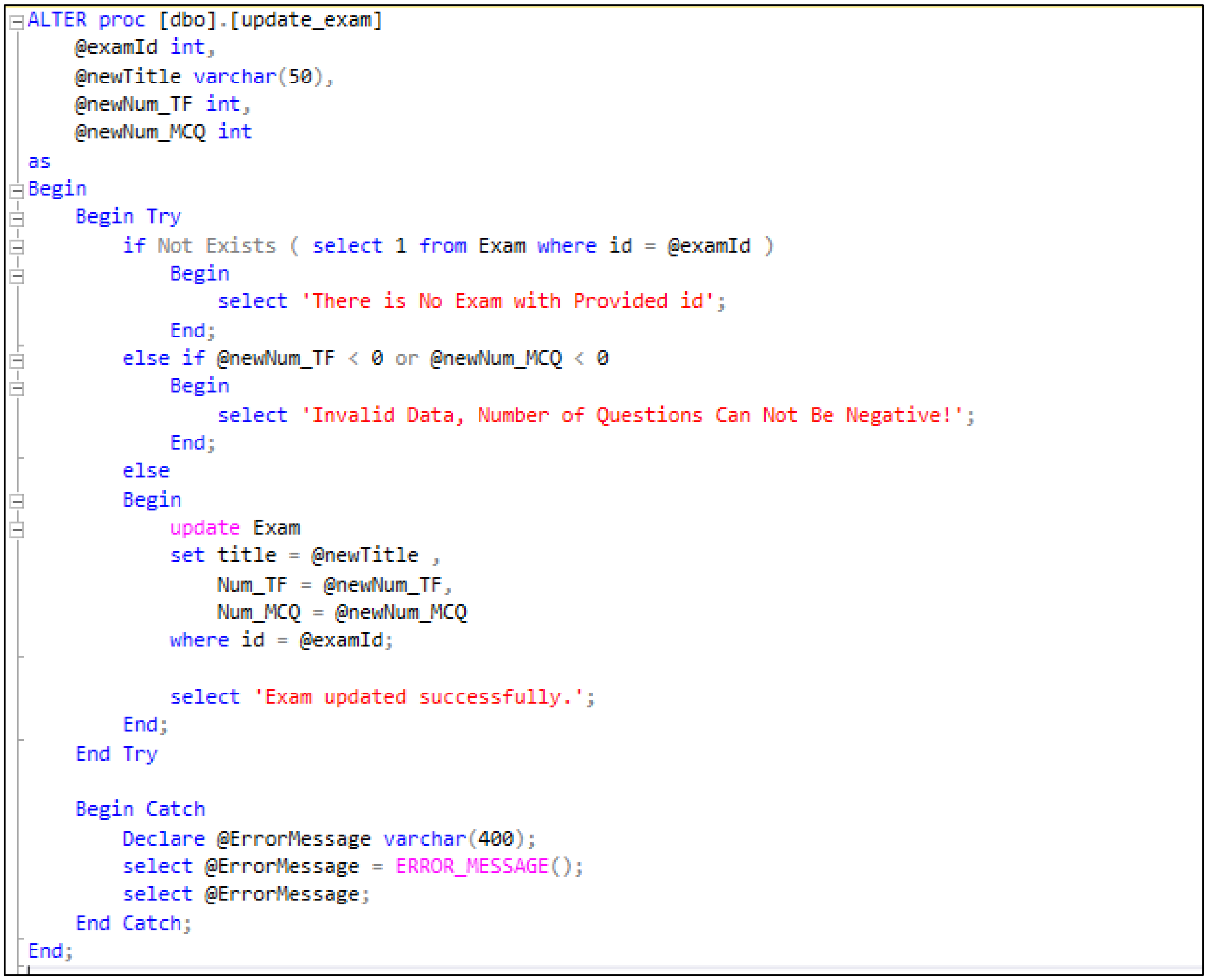
Parameters:

1. @examTitle (varchar(50)): This parameter represents the title or name of the exam being inserted. It accepts a string value of up to 50 characters.
2. @examNum_TF (integer): This parameter specifies the number of True/False questions included in the exam.
3. @examNum_MCQ (integer): This parameter denotes the number of Multiple Choice Questions (MCQs) present in the exam.
4. @examid (integer output): An output parameter that captures and returns the unique identifier (ID) assigned to the newly inserted exam. It allows for easy retrieval and reference of the generated exam ID.

Functionality:

Upon invocation, the insert_Exam stored procedure attempts to insert a new exam record into the database using the provided parameters. It utilizes a Try-Catch block to handle any potential errors that may occur during the insertion process. If the insertion is successful, the stored procedure retrieves the generated exam ID using the SCOPE_IDENTITY() function and assigns it to the output parameter @examid. In case of an error during insertion, the Catch block captures and displays an error message along with a description of the error encountered.

- **Update Exam Stored Procedure**



```sql
ALTER proc [dbo].[update_exam]
    @examId int,
    @newTitle varchar(50),
    @newNum_TF int,
    @newNum_MCQ int
as
Begin
    Begin Try
        if Not Exists ( select 1 from Exam where id = @examId )
            Begin
                select 'There is No Exam with Provided id';
            End;
        else if @newNum_TF < 0 or @newNum_MCQ < 0
            Begin
                select 'Invalid Data, Number of Questions Can Not Be Negative!';
            End;
        else
        Begin
            update Exam
            set title = @newTitle ,
                Num_TF = @newNum_TF,
                Num_MCQ = @newNum_MCQ
            where id = @examId;

            select 'Exam updated successfully.';
        End;
    End Try

    Begin Catch
        Declare @ErrorMessage varchar(400);
        select @ErrorMessage = ERROR_MESSAGE();
        select @ErrorMessage;
    End Catch;
End;
```

Description:

The update_exam stored procedure facilitates the modification of existing exam records within the database. It allows authorized users to update specific attributes of an exam, such as its title, the number of True/False questions (Num_TF), and the number of Multiple Choice Questions (Num_MCQ), based on the provided exam ID.

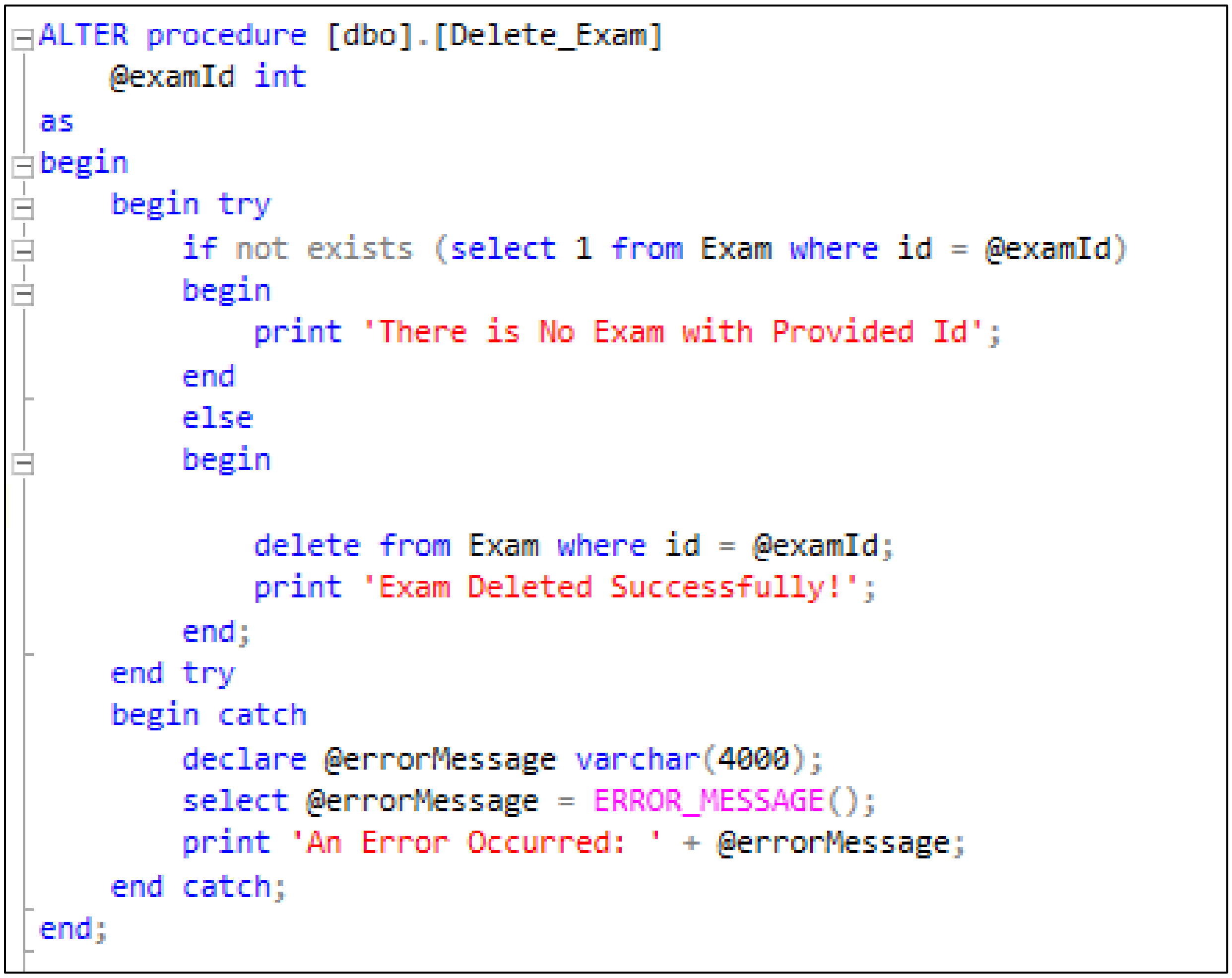
Parameters:

1. @examId (integer): This parameter represents the unique identifier of the exam to be updated. It serves as a reference to locate the exam record within the database.
2. @newTitle (varchar(50)): This parameter indicates the new title or name to be assigned to the exam. It accepts a string value of up to 50 characters.
3. @newNum_TF (integer): This parameter specifies the updated number of True/False questions for the exam.
4. @newNum_MCQ (integer): This parameter denotes the updated number of Multiple Choice Questions (MCQs) for the exam.

Functionality:

Upon invocation, the update_exam stored procedure begins by verifying the existence of an exam with the provided exam ID. If no exam is found matching the specified ID, the stored procedure returns a message indicating the absence of such an exam.
If the exam ID is valid, the stored procedure checks whether the provided values for the number of True/False questions (@newNum_TF) and the number of Multiple Choice Questions (@newNum_MCQ) are non-negative. If either of these values is negative, indicating invalid input, the stored procedure returns an appropriate error message.
If all input parameters are valid, the stored procedure proceeds to update the exam record in the database with the new title and question counts provided. Upon successful update, it returns a confirmation message indicating that the exam has been updated successfully.
In the event of an error during the update process, such as database connectivity issues or data validation failures, the stored procedure catches and displays the corresponding error message.

- **Delete Exam Stored Procedure**



```
ALTER procedure [dbo].[Delete_Exam]
    @examId int
as
begin
    begin try
        if not exists (select 1 from Exam where id = @examId)
        begin
            print 'There is No Exam with Provided Id';
        end
        else
        begin

            delete from Exam where id = @examId;
            print 'Exam Deleted Successfully!';
        end;
    end try
    begin catch
        declare @errorMessage varchar(4000);
        select @errorMessage = ERROR_MESSAGE();
        print 'An Error Occurred: ' + @errorMessage;
    end catch;
end;
```

Description:

The Delete_Exam stored procedure is designed to remove an existing exam record from the database based on the provided exam ID. It provides a mechanism for authorized users to delete unwanted or obsolete exam records, thereby ensuring the integrity and relevance of exam-related data within the database.

Parameters:

@examId (integer): This parameter represents the unique identifier of the exam to be deleted. It serves as a reference to locate and remove the specific exam record from the database.

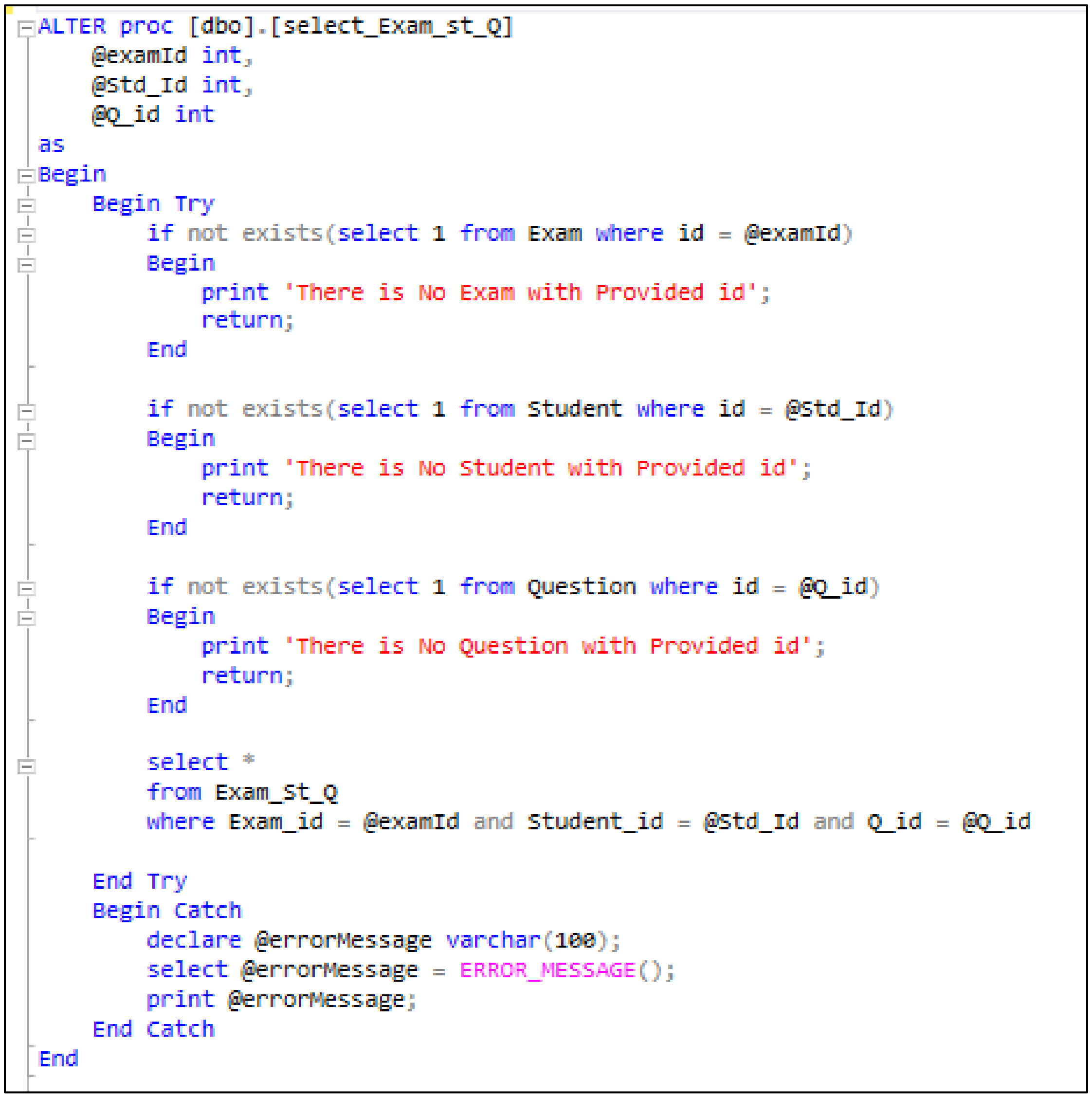
Functionality:

Upon invocation, the Delete_Exam stored procedure begins by verifying the existence of an exam with the provided exam ID. If no exam is found matching the specified ID, the stored procedure prints a message indicating the absence of such an exam.

If the exam ID is valid and an exam record with the specified ID exists in the database, the stored procedure proceeds to delete the exam record using a SQL DELETE statement. Once the deletion is successful, it prints a confirmation message indicating that the exam has been deleted successfully.

In case of any errors encountered during the deletion process, such as database connectivity issues or unexpected errors, the stored procedure catches and prints the corresponding error message to alert users about the issue**.**

- **Select From Exam_St_Q Table Stored Procedure**



```
ALTER proc [dbo].[select_Exam_st_Q]
    @examId int,
    @Std_Id int,
    @Q_id int
as
Begin
    Begin Try
        if not exists(select 1 from Exam where id = @examId)
        Begin
            print 'There is No Exam with Provided id';
            return;
        End

        if not exists(select 1 from Student where id = @Std_Id)
        Begin
            print 'There is No Student with Provided id';
            return;
        End

        if not exists(select 1 from Question where id = @Q_id)
        Begin
            print 'There is No Question with Provided id';
            return;
        End

        select *
        from Exam_St_Q
        where Exam_id = @examId and Student_id = @Std_Id and Q_id = @Q_id

    End Try
    Begin Catch
        declare @errorMessage varchar(100);
        select @errorMessage = ERROR_MESSAGE();
        print @errorMessage;
    End Catch
End
```

Description:

The select_Exam_st_Q stored procedure is designed to retrieve specific data from the Exam_St_Q table based on the provided parameters, including exam ID, student ID, and question ID. This stored procedure serves as a means to fetch information related to the interaction between exams, students, and questions stored within the database.

Parameters:

1. @examId (integer): This parameter represents the unique identifier of the exam for which data is to be retrieved from the Exam_St_Q table.
2. @Std_Id (integer): This parameter denotes the unique identifier of the student whose interaction data with the exam is to be retrieved from the Exam_St_Q table.
3. @Q_id (integer): This parameter specifies the unique identifier of the question for which interaction data with the student and exam is to be retrieved from the Exam_St_Q table.

Functionality:

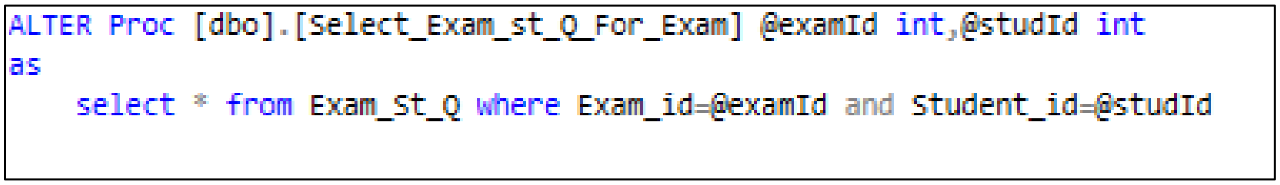
Upon invocation, the select_Exam_st_Q stored procedure begins by verifying the existence of the referenced exam, student, and question within their respective tables (Exam, Student, and Question). If any of these referenced entities are not found, the stored procedure prints an appropriate message indicating their absence and returns from the procedure.

If all referenced entities are found to exist, the stored procedure proceeds to select data from the Exam_St_Q table based on the provided parameters. It retrieves records from the Exam_St_Q table where the Exam_id matches the provided @examId, the Student_id matches the provided @Std_Id, and the Q_id matches the provided @Q_id.

The selected data typically represents the interaction or association between the specified exam, student, and question stored within the Exam_St_Q table.

In case of any errors encountered during the selection process, such as database connectivity issues or unexpected errors, the stored procedure catches and prints the corresponding error message.

- **Select Exam for Specific Student**



```sql
ALTER Proc [dbo].[Select_Exam_st_Q_For_Exam] @examId int,@studId int
as
    select * from Exam_St_Q where Exam_id=@examId and Student_id=@studId
```

Description:

This stored procedure retrieves interaction data from the Exam_St_Q table, focusing on a specific exam and student. It serves as a tool to analyze and evaluate student performance on a particular exam.

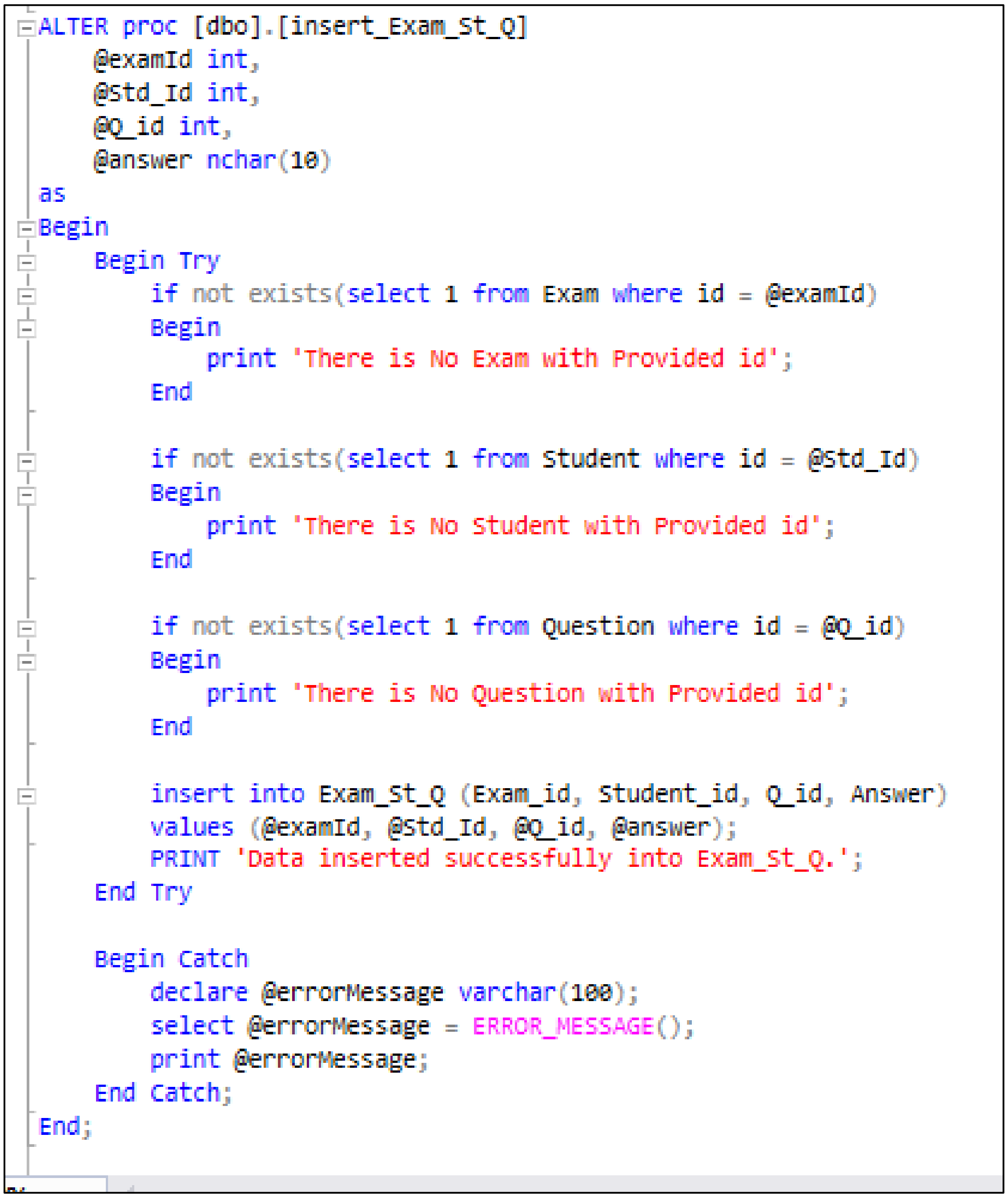
Parameters:

1. @examId (integer): Represents the exam ID for data retrieval.
2. @studId (integer): Denotes the student ID for data retrieval.

Functionality:

Upon invocation, this procedure selects records from Exam_St_Q where the Exam_id matches the provided @examId parameter and the Student_id matches the provided @studId parameter. It gathers information about the interaction between the specified exam and student, such as responses to questions and associated details.

- **Insert in Exam_St_Q Stored Procedure**



```sql
ALTER proc [dbo].[insert_Exam_St_Q]
    @examId int,
    @Std_Id int,
    @Q_id int,
    @answer nchar(10)
as
Begin
    Begin Try
        if not exists(select 1 from Exam where id = @examId)
        Begin
            print 'There is No Exam with Provided id';
        End

        if not exists(select 1 from Student where id = @Std_Id)
        Begin
            print 'There is No Student with Provided id';
        End

        if not exists(select 1 from Question where id = @Q_id)
        Begin
            print 'There is No Question with Provided id';
        End

        insert into Exam_St_Q (Exam_id, Student_id, Q_id, Answer)
        values (@examId, @Std_Id, @Q_id, @answer);
        PRINT 'Data inserted successfully into Exam_St_Q.';
    End Try

    Begin Catch
        declare @errorMessage varchar(100);
        select @errorMessage = ERROR_MESSAGE();
        print @errorMessage;
    End Catch;
End;
```

<u>**Description:**</u>

The insert_Exam_St_Q stored procedure facilitates the insertion of interaction data into the Exam_St_Q table. It allows for the recording of students' answers to specific questions in a particular exam.

Parameters:

1. @examId (integer): Represents the exam ID associated with the interaction data.
2. @Std_Id (integer): Denotes the student ID linked to the interaction data.
3. @Q_id (integer): Specifies the question ID for which the interaction data is recorded.
4. @answer (nchar(10)): Indicates the student's answer to the question.

Functionality:

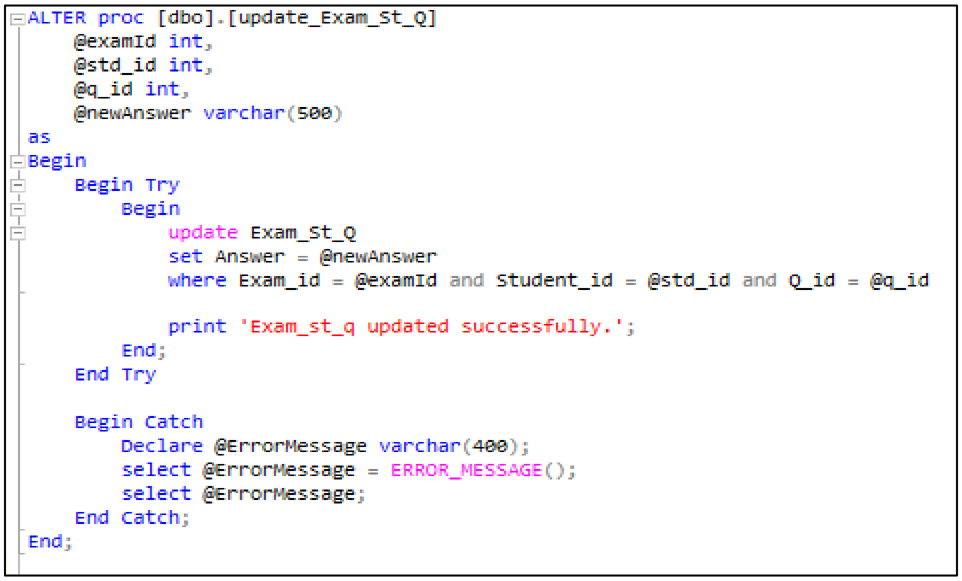
Upon invocation, the insert_Exam_St_Q stored procedure begins by verifying the existence of the referenced exam, student, and question within their respective tables (Exam, Student, and Question). If any of these referenced entities are not found, appropriate messages are printed.

If all referenced entities exist, the stored procedure proceeds to insert a new record into the Exam_St_Q table. The record includes the exam ID, student ID, question ID, and the student's answer provided as parameters.

Upon successful insertion of the data into the Exam_St_Q table, a confirmation message is printed.

In case of any errors during the insertion process, such as database connectivity issues or data validation failures, the stored procedure catches and prints the corresponding error message.

- **Update Exam_St_Q Stored Procedure**



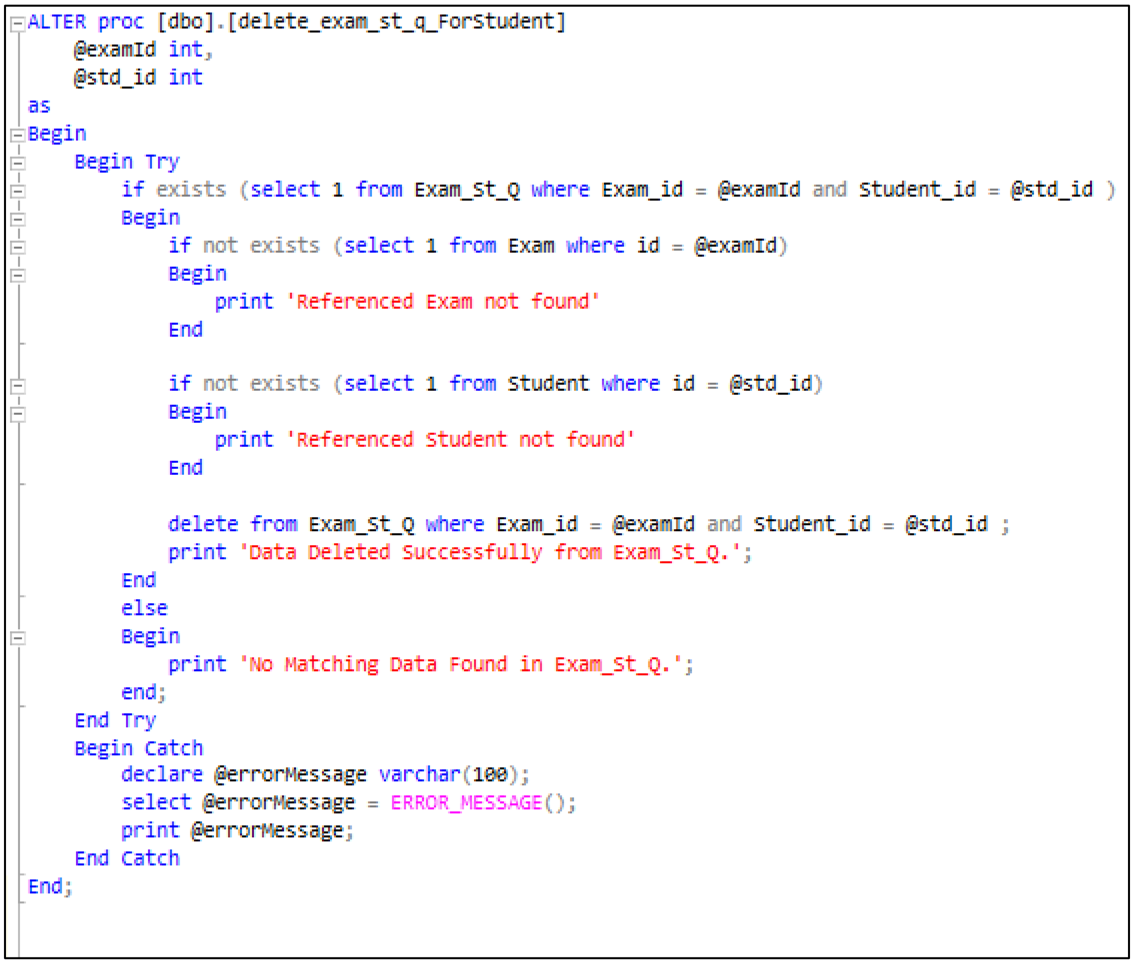
```
ALTER proc [dbo].[update_Exam_St_Q]
    @examId int,
    @std_id int,
    @q_id int,
    @newAnswer varchar(500)
as
Begin
    Begin Try
        Begin
            update Exam_St_Q
            set Answer = @newAnswer
            where Exam_id = @examId and Student_id = @std_id and Q_id = @q_id

            print 'Exam_st_q updated successfully.';
        End;
    End Try

    Begin Catch
        Declare @ErrorMessage varchar(400);
        select @ErrorMessage = ERROR_MESSAGE();
        select @ErrorMessage;
    End Catch;
End;
```

Description:
The update_Exam_St_Q stored procedure is designed to modify interaction data within the Exam_St_Q table. It allows for updating the answer provided by a student to a specific question in a particular exam.

Parameters:
1. @examId (integer): Represents the exam ID associated with the interaction data.
2. @std_id (integer): Denotes the student ID linked to the interaction data.
3. @q_id (integer): Specifies the question ID for which the interaction data is recorded.
4. @newAnswer (varchar(500)): Indicates the updated answer provided by the student.

Functionality:
Upon invocation, the update_Exam_St_Q stored procedure attempts to update the answer field in the Exam_St_Q table. The update is performed for the record where the exam ID, student ID, and question ID match the provided parameters.
Upon successful update, a confirmation message is printed. In case of any errors during the update process, such as database connectivity issues or data validation failures, the stored procedure catches and prints the corresponding error message.

Delete From Exam_St_Q Table Stored Procedure



```sql
ALTER proc [dbo].[delete_exam_st_q_ForStudent]
    @examId int,
    @std_id int
as
Begin
    Begin Try
        if exists (select 1 from Exam_St_Q where Exam_id = @examId and Student_id = @std_id )
        Begin
            if not exists (select 1 from Exam where id = @examId)
            Begin
                print 'Referenced Exam not found'
            End

            if not exists (select 1 from Student where id = @std_id)
            Begin
                print 'Referenced Student not found'
            End

            delete from Exam_St_Q where Exam_id = @examId and Student_id = @std_id ;
            print 'Data Deleted Successfully from Exam_St_Q.';
        End
        else
        Begin
            print 'No Matching Data Found in Exam_St_Q.';
        end;
    End Try
    Begin Catch
        declare @errorMessage varchar(100);
        select @errorMessage = ERROR_MESSAGE();
        print @errorMessage;
    End Catch
End;
```

<u>Description:</u>

The delete_exam_st_q_ForStudent stored procedure is designed to delete interaction data from the Exam_St_Q table associated with a specific exam and student. It allows for the removal of all interaction records pertaining to a particular student's participation in a given exam.

<u>Parameters:</u>

1. @examId (integer): Represents the exam ID associated with the interaction data to be deleted.
2. @std_id (integer): Denotes the student ID associated with the interaction data to be deleted.

Functionality:

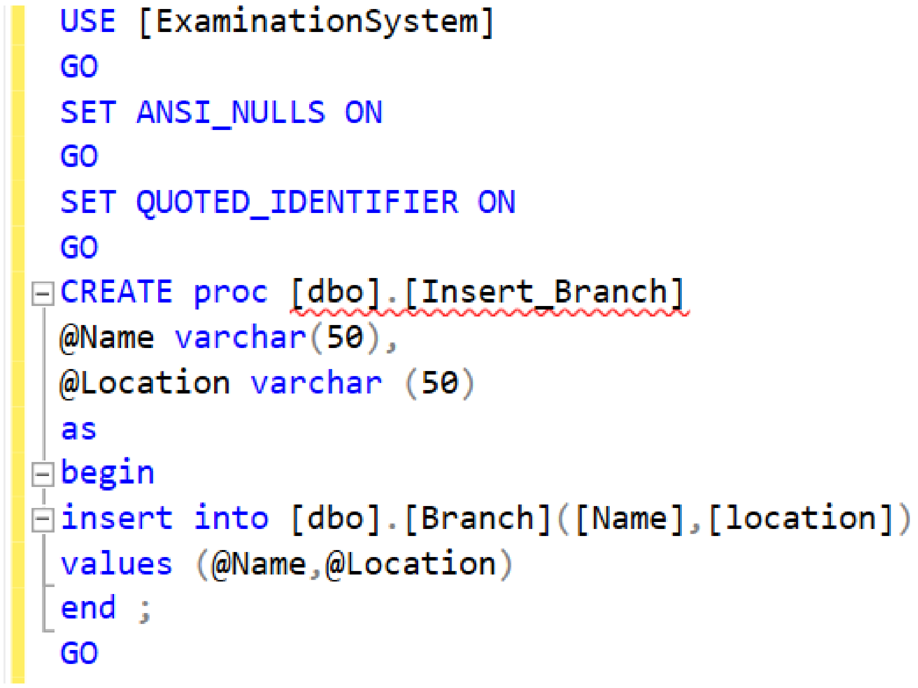
Upon invocation, the delete_exam_st_q_ForStudent stored procedure checks for the existence of interaction data in the Exam_St_Q table based on the provided exam ID and student ID parameters. If matching data is found, the procedure verifies the existence of the referenced exam and student in their respective tables (Exam and Student).

If all referenced entities exist, the procedure proceeds to delete all interaction records associated with the specified exam and student from the Exam_St_Q table. Upon successful deletion, a confirmation message is printed.

If no matching data is found in the Exam_St_Q table based on the provided parameters, a message indicating the absence of matching data is printed.

In case of any errors during the deletion process, such as database connectivity issues or unexpected errors, the procedure catches and prints the corresponding error message.

- **Insert in Exam_St_Q Stored Procedure**



```
USE [ExaminationSystem]
GO
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE proc [dbo].[Insert_Branch]
@Name varchar(50),
@Location varchar (50)
as
begin
insert into [dbo].[Branch]([Name],[location])
values (@Name,@Location)
end ;
GO
```

Description:

This stored procedure, named Insert_Branch, is designed to insert data into the [dbo].[Branch] table within the ExaminationSystem database. It allows the addition of new branch information, including the branch name and its corresponding location.

Parameters:

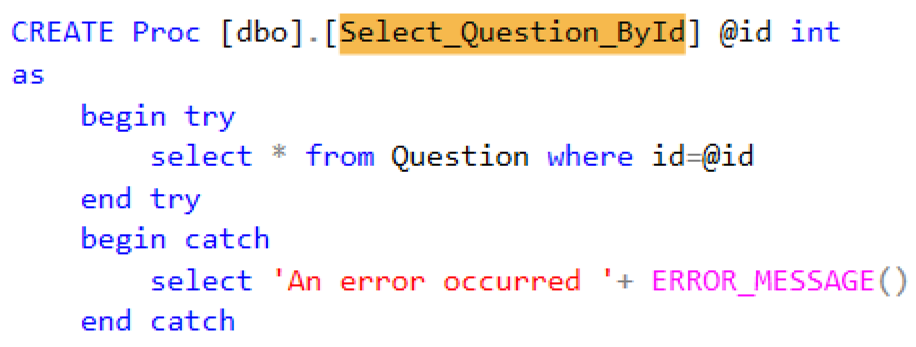
1. @Name (varchar, max length 50): Represents the name of the branch to be inserted.
2. @Location (varchar, max length 50): Denotes the location associated with the branch.

Functionality:

Upon invocation, this procedure inserts a new record into the [dbo].[Branch] table, populating the Name and location columns with the provided values. It effectively maintains the branch data within the system.

Stored Procedures for table Question

Select_Question_ById

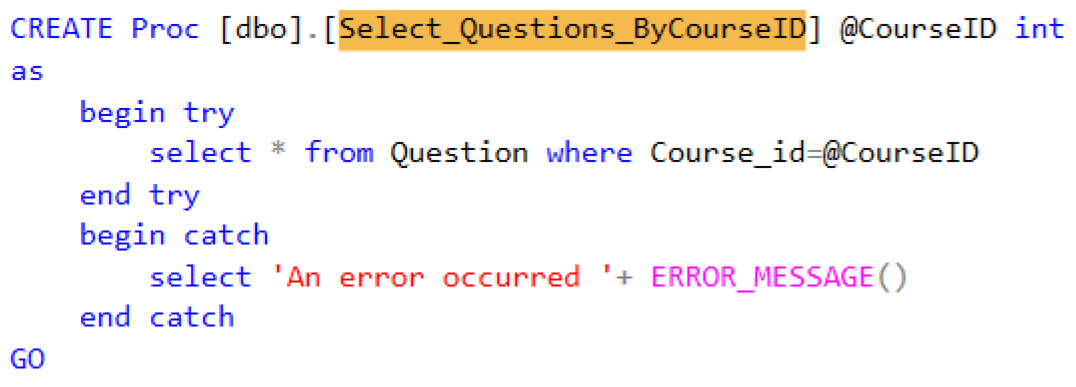


```
CREATE Proc [dbo].[Select_Question_ById] @id int
as
    begin try
        select * from Question where id=@id
    end try
    begin catch
        select 'An error occurred '+ ERROR_MESSAGE()
    end catch
```

Purpose: This stored procedure is designed to retrieve information about a specific question based on the provided question ID.

1. **Parameters**:
   - @id (integer): Represents the unique identifier of the question for which information is to be retrieved. It serves as an input parameter required for the successful execution of the stored procedure.
2. **Functionality**:
   - Upon receiving the question ID as input, the stored procedure performs the following steps:
     - Tries to execute a SELECT query to retrieve all details related to the question with the specified ID.
     - If successful (i.e., the question exists in the database), it returns the result set containing the question details.
     - If an error occurs during execution (e.g., invalid ID or database issue), it catches the error and returns a custom message indicating the error along with the error message.

Select_Questions_ByCourseID



```
CREATE Proc [dbo].[Select_Questions_ByCourseID] @CourseID int
as
    begin try
        select * from Question where Course_id=@CourseID
    end try
    begin catch
        select 'An error occurred '+ ERROR_MESSAGE()
    end catch
GO
```

1. **Purpose**: This stored procedure is designed to retrieve information about questions related to a specific course based on the provided course ID.
2. **Parameters**:
   - @CourseID (integer): Represents the unique identifier of the course for which questions are to be retrieved. It serves as an input parameter required for the successful execution of the stored procedure.
3. **Functionality**:
   - Upon receiving the course ID as input, the stored procedure performs the following steps:
     - Tries to execute a SELECT query to retrieve all details related to questions associated with the specified course ID.
     - If successful (i.e., questions exist in the database for that course), it returns the result set containing the question details.
     - If an error occurs during execution (e.g., invalid ID or database issue), it catches the error and returns a custom message indicating the error along with the error message.

Select_Questions_ByCourseID_and_Type

```
CREATE Proc [dbo].[Select_Questions_ByCourseID_and_Type] @CourseID int ,@type tinyint
as
    begin try
        select * from Question where Course_id=@CourseID and Type=@type
    end try
    begin catch
        select 'An error occurred '+ ERROR_MESSAGE()
    end catch
GO
```

- **Purpose**: This stored procedure is designed to retrieve information about questions related to a specific course based on the provided course ID and question type.

- **Parameters**:

  - @CourseID (integer): Represents the unique identifier of the course for which questions are to be retrieved. It serves as an input parameter required for the successful execution of the stored procedure.

  - @type (tinyint): Represents the type of questions to be retrieved (e.g., multiple-choice, essay, etc.). It is another input parameter essential for the execution of this stored procedure.

- **Functionality**:

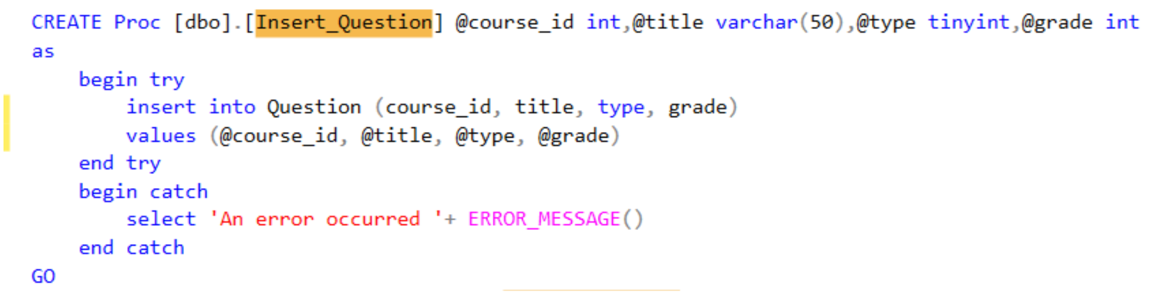
  - Upon receiving the course ID and question type as input, the stored procedure performs the following steps:

    - Tries to execute a SELECT query to retrieve all details related to questions associated with the specified course ID and matching the provided question type.

    - If successful (i.e., questions exist in the database for that course and type), it returns the result set containing the question details.

If an error occurs during execution (e.g., invalid ID or database issue), it catches the error and returns a custom message indicating the error along with the error message

Insert_Question



```
CREATE Proc [dbo].[Insert_Question] @course_id int,@title varchar(50),@type tinyint,@grade int
as
    begin try
        insert into Question (course_id, title, type, grade)
        values (@course_id, @title, @type, @grade)
    end try
    begin catch
        select 'An error occurred '+ ERROR_MESSAGE()
    end catch
GO
```

1. **Purpose**: This stored procedure is designed to insert a new question into the database.

2. **Parameters**:

○ @course_id (integer): Represents the unique identifier of the course to which the question belongs. It serves as an input parameter required for the successful execution of the stored procedure.

○ @title (varchar(50)): Represents the title or description of the question. It is another input parameter essential for the execution of this stored procedure.

○ @type (tinyint): Represents the type of the question (e.g., multiple-choice, essay, etc.). It is an input parameter required for the successful execution of the stored procedure.

○ @grade (int): Represents the grade or score associated with the question. It serves as an input parameter required for the successful execution of the stored procedure.

3. **Functionality**:

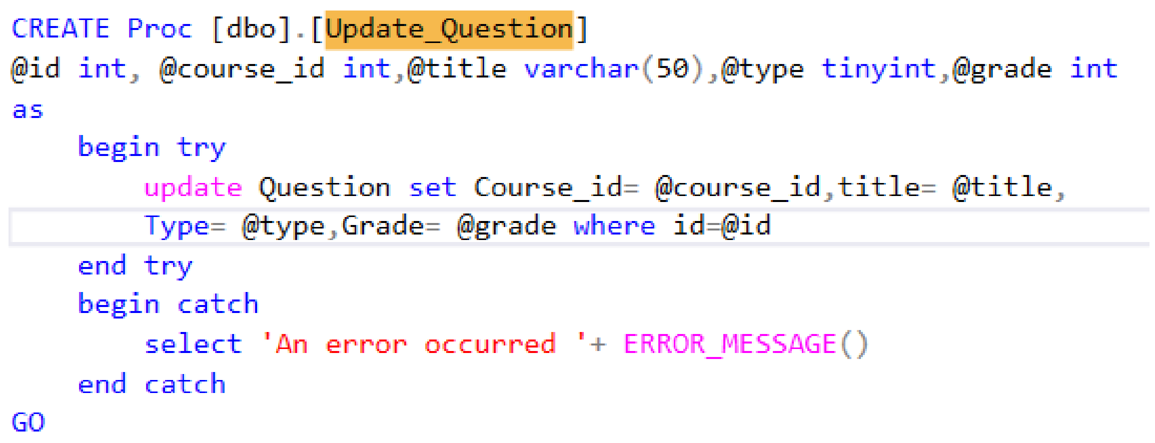
○ Upon receiving the necessary input parameters, the stored procedure performs the following steps:

▪ Attempts to insert a new record into the Question table with the provided course ID, title, type, and grade.

▪ If the insertion is successful, the question is added to the database.

▪ If an error occurs during execution (e.g., invalid parameters or database issue), it catches the error and returns a custom message indicating the error along with the error message.

Update_Question



```
CREATE Proc [dbo].[Update_Question]
@id int, @course_id int,@title varchar(50),@type tinyint,@grade int
as
    begin try
        update Question set Course_id= @course_id,title= @title,
        Type= @type,Grade= @grade where id=@id
    end try
    begin catch
        select 'An error occurred '+ ERROR_MESSAGE()
    end catch
GO
```

- **Purpose**: This stored procedure is designed to update the details of an existing question in the database.

- **Parameters**:

○ @id (integer): Represents the unique identifier of the question to be updated. It serves as an input parameter required for the successful execution of the stored procedure.

○ @course_id (integer): Represents the unique identifier of the course to which the question belongs. It is another input parameter essential for the execution of this stored procedure.

○ @title (varchar(50)): Represents the updated title or description of the question. It is an input parameter required for the successful execution of the stored procedure.

○ @type (tinyint): Represents the updated type of the question (e.g., multiple-choice, essay, etc.). It is an input parameter required for the successful execution of the stored procedure.

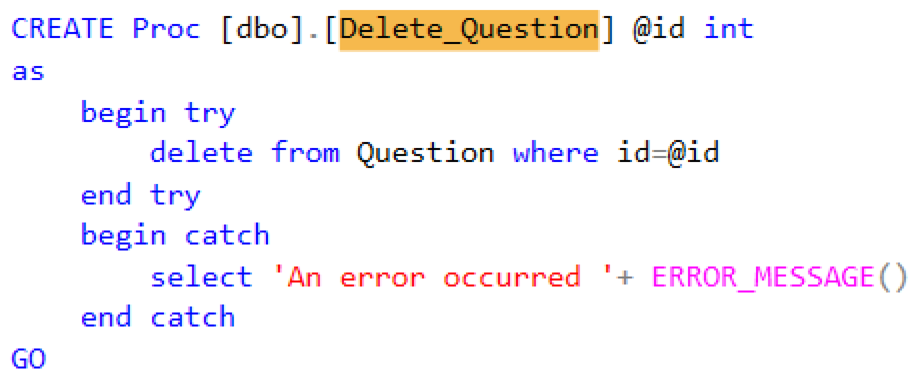
○ @grade (int): Represents the updated grade or score associated with the question. It serves as an input parameter required for the successful execution of the stored procedure.

- **Functionality**:

○ Upon receiving the necessary input parameters, the stored procedure performs the following steps:

- Attempts to update the existing record in the Question table with the provided course ID, title, type, and grade, where the question ID matches the specified @id.
- If the update is successful, the question details are modified in the database.
- If an error occurs during execution (e.g., invalid parameters or database issue), it catches the error and returns a custom message indicating the error along with the error message.

Delete_Question



```
CREATE Proc [dbo].[Delete_Question] @id int
as
    begin try
        delete from Question where id=@id
    end try
    begin catch
        select 'An error occurred '+ ERROR_MESSAGE()
    end catch
GO
```

1. **Purpose**: This stored procedure is designed to delete an existing question from the database based on the provided question ID.

2. **Parameters**:

○ @id (integer): Represents the unique identifier of the question to be deleted. It serves as an input parameter required for the successful execution of the stored procedure.

3. **Functionality**:

○ Upon receiving the question ID as input, the stored procedure performs the following steps:

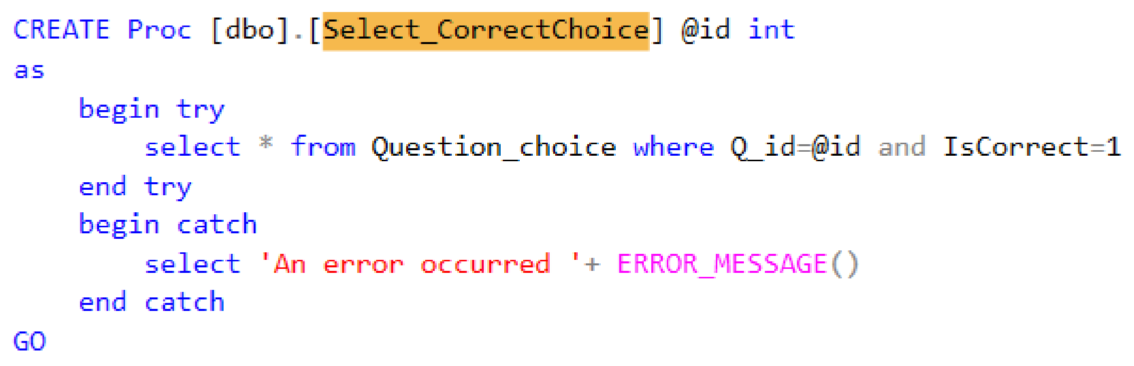
▪ Attempts to delete the record from the Question table where the question ID matches the specified @id.

▪ If the deletion is successful, the question is removed from the database.

▪ If an error occurs during execution (e.g., invalid ID or database issue), it catches the error and returns a custom message indicating the error along with the error message.

Stored Procedures for table Question_Choice

Select_CorrectChoice



```
CREATE Proc [dbo].[Select_CorrectChoice] @id int
as
    begin try
        select * from Question_choice where Q_id=@id and IsCorrect=1
    end try
    begin catch
        select 'An error occurred '+ ERROR_MESSAGE()
    end catch
GO
```

1. **Purpose**: This stored procedure is designed to retrieve the correct choice (answer) for a specific question based on the provided question ID.

2. **Parameters**:

○ @id (integer): Represents the unique identifier of the question for which the correct choice needs to be retrieved. It serves as an input parameter required for the successful execution of the stored procedure.

3. **Functionality**:

○ Upon receiving the question ID as input, the stored procedure performs the following steps:

▪ Tries to execute a SELECT query on the Question_choice table.

▪ Retrieves the Choice (answer) where the Q_id matches the specified question ID and the IsCorrect flag is set to 1 (indicating the correct choice).

▪ If successful, it returns the correct choice.

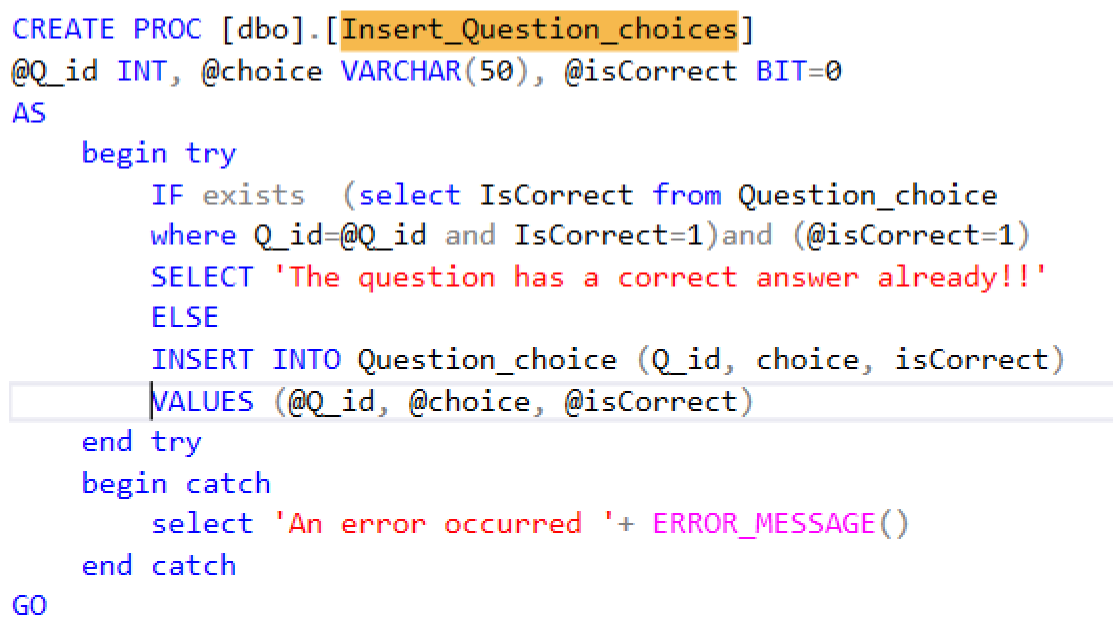
▪ If an error occurs during execution (e.g., invalid ID or database issue), it catches the error and returns a custom message indicating the error along with the error message.

Select_Question_choices_ById

```
CREATE Proc [dbo].[Select_Question_choices_ById] @id int
as
    begin try
        select * from Question_choice where Q_id=@id
    end try
    begin catch
        select 'An error occurred '+ ERROR_MESSAGE()
    end catch
GO
```

- **Purpose**: This stored procedure is designed to retrieve all available choices (options) for a specific question based on the provided question ID.
- **Parameters**:
  - @id (integer): Represents the unique identifier of the question for which choices need to be retrieved. It serves as an input parameter required for the successful execution of the stored procedure.
- **Functionality**:
  - Upon receiving the question ID as input, the stored procedure performs the following steps:
    - Executes a SELECT query on the Question_choice table.
    - Retrieves all rows where the Q_id matches the specified question ID.
    - Returns the result set containing all available choices for that particular question.
    - If an error occurs during execution (e.g., invalid ID or database issue), it catches the error and returns a custom message indicating the error along with the error message.

Insert_Question_choices



```
CREATE PROC [dbo].[Insert_Question_choices]
@Q_id INT, @choice VARCHAR(50), @isCorrect BIT=0
AS
    begin try
        IF exists  (select IsCorrect from Question_choice
        where Q_id=@Q_id and IsCorrect=1)and (@isCorrect=1)
        SELECT 'The question has a correct answer already!!'
        ELSE
        INSERT INTO Question_choice (Q_id, choice, isCorrect)
        VALUES (@Q_id, @choice, @isCorrect)
    end try
    begin catch
        select 'An error occurred '+ ERROR_MESSAGE()
    end catch
GO
```

- **Purpose**: This stored procedure is designed to insert a new choice (option) for a specific question into the database. It allows for specifying whether the inserted choice is the correct answer or not.

- **Parameters**:

  - @Q_id (integer): Represents the unique identifier of the question to which the choice belongs. It serves as an input parameter required for the successful execution of the stored procedure.

  - @choice (varchar(50)): Represents the text of the choice (e.g., option A, option B, etc.). It is another input parameter essential for the execution of this stored procedure.

  - @isCorrect (bit): Indicates whether the choice is the correct answer (1) or not (0). It is an optional input parameter; if not provided, the default value is 0 (not correct).

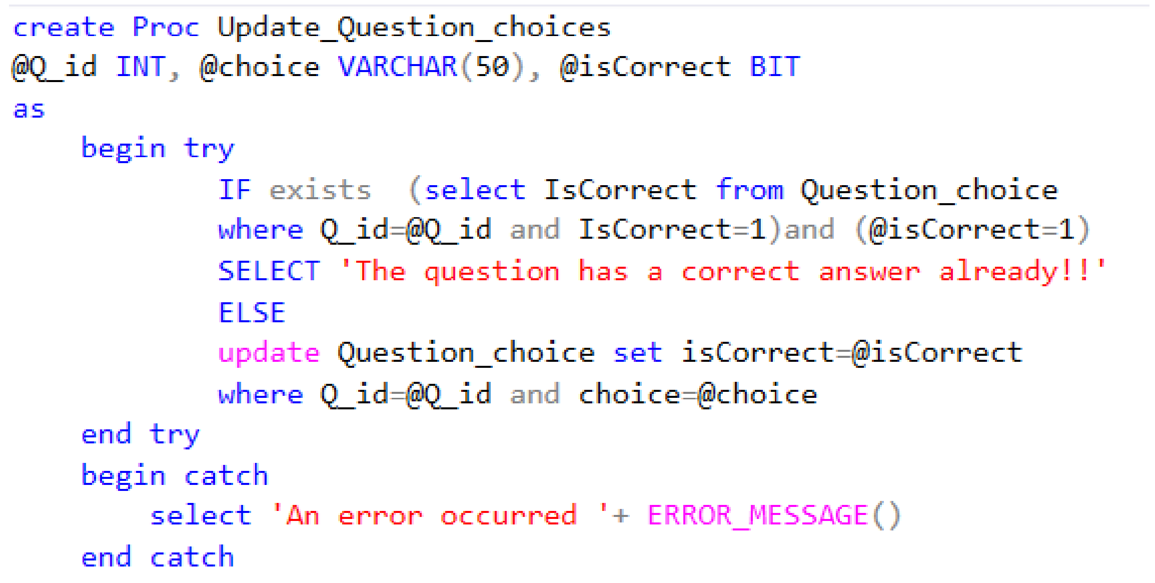
- **Functionality**:

  - Upon receiving the necessary input parameters, the stored procedure performs the following steps:

    - Checks if there is already a correct answer (IsCorrect = 1) associated with the specified question (@Q_id).

- If a correct answer exists and the new choice is marked as correct (@isCorrect = 1), it returns a message indicating that the question already has a correct answer.
- Otherwise, it inserts the new choice into the Question_choice table with the provided question ID, choice text, and correctness flag.
- If an error occurs during execution (e.g., invalid parameters or database issue), it catches the error and returns a custom message indicating the error along with the error message.

Update_Question_choices



```
create Proc Update_Question_choices
@Q_id INT, @choice VARCHAR(50), @isCorrect BIT
as
    begin try
            IF exists  (select IsCorrect from Question_choice
            where Q_id=@Q_id and IsCorrect=1)and (@isCorrect=1)
            SELECT 'The question has a correct answer already!!'
            ELSE
            update Question_choice set isCorrect=@isCorrect
            where Q_id=@Q_id and choice=@choice
    end try
    begin catch
        select 'An error occurred '+ ERROR_MESSAGE()
    end catch
```

- **Purpose**: This stored procedure is designed to update the correctness status of an existing choice (option) for a specific question in the database.

- **Parameters**:

  - @Q_id (integer): Represents the unique identifier of the question to which the choice belongs. It serves as an input parameter required for the successful execution of the stored procedure.

  - @choice (varchar(50)): Represents the text of the choice (e.g., option A, option B, etc.). It is another input parameter essential for the execution of this stored procedure.

  - @isCorrect (bit): Indicates whether the choice should be marked as the correct answer (1) or not (0). It is an input parameter required for the successful execution of the stored procedure.

- **Functionality**:

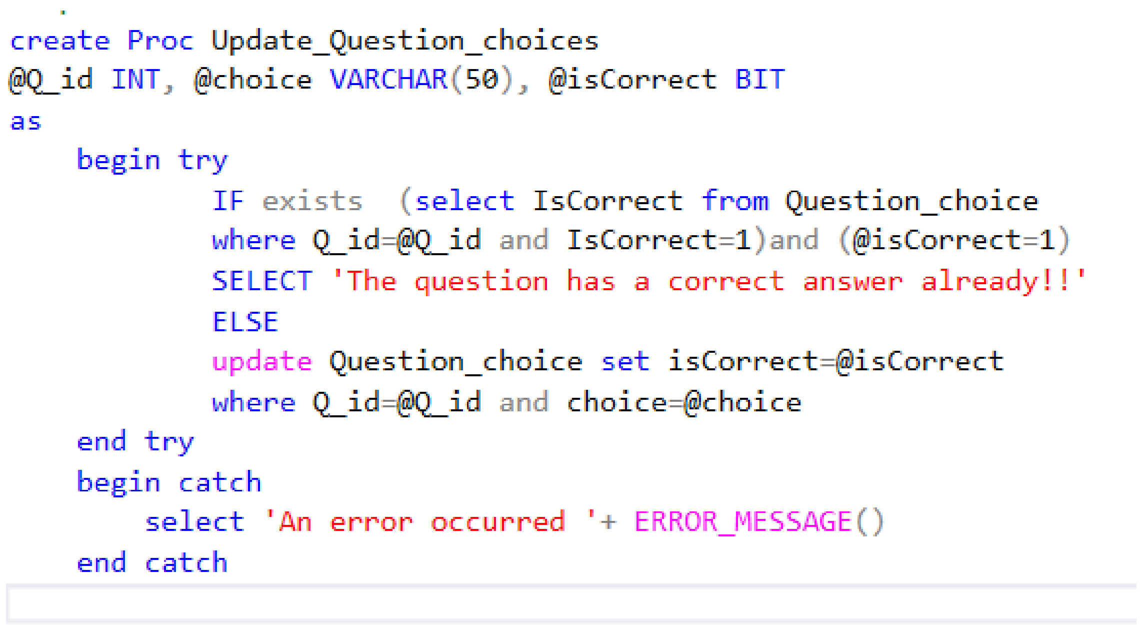
  - Upon receiving the necessary input parameters, the stored procedure performs the following steps:

    - Checks if there is already a correct answer (IsCorrect = 1) associated with the specified question (@Q_id).

    - If a correct answer exists and the new choice is marked as correct (@isCorrect = 1), it returns a message indicating that the question already has a correct answer.

- Otherwise, it updates the correctness status of the specified choice in the Question_choice table based on the provided question ID and choice text.
- If an error occurs during execution (e.g., invalid parameters or database issue), it catches the error and returns a custom message indicating the error along with the error message.

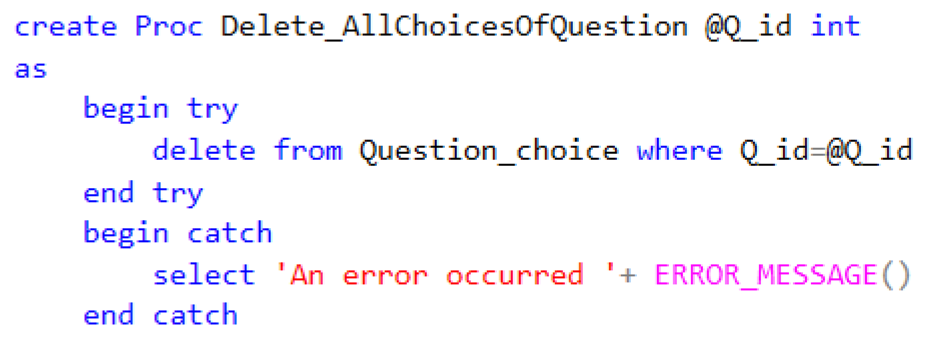
Update_Question_choices



```
create Proc Update_Question_choices
@Q_id INT, @choice VARCHAR(50), @isCorrect BIT
as
    begin try
            IF exists  (select IsCorrect from Question_choice
            where Q_id=@Q_id and IsCorrect=1)and (@isCorrect=1)
            SELECT 'The question has a correct answer already!!'
            ELSE
            update Question_choice set isCorrect=@isCorrect
            where Q_id=@Q_id and choice=@choice
    end try
    begin catch
        select 'An error occurred '+ ERROR_MESSAGE()
    end catch
```

1. **Purpose**: This stored procedure is designed to update the correctness status of an existing choice (option) for a specific question in the database.
2. **Parameters**:
   - `@Q_id` (integer): Represents the unique identifier of the question to which the choice belongs. It serves as an input parameter required for the successful execution of the stored procedure.
   - `@choice` (varchar(50)): Represents the text of the choice (e.g., option A, option B, etc.). It is another input parameter essential for the execution of this stored procedure.
   - `@isCorrect` (bit): Indicates whether the choice should be marked as the correct answer (1) or not (0). It is an input parameter required for the successful execution of the stored procedure.
3. **Functionality**:
   - Upon receiving the necessary input parameters, the stored procedure performs the following steps:
     - Checks if there is already a correct answer (IsCorrect = 1) associated with the specified question (`@Q_id`).
     - If a correct answer exists and the new choice is marked as correct (`@isCorrect = 1`), it returns a message indicating that the question already has a correct answer.
     - Otherwise, it updates the correctness status of the specified choice in the `Question_choice` table based on the provided question ID and choice text.
     - If an error occurs during execution (e.g., invalid parameters or database issue), it catches the error and returns a custom message indicating the error along with the error message.

`Delete_AllChoicesOfQuestion`

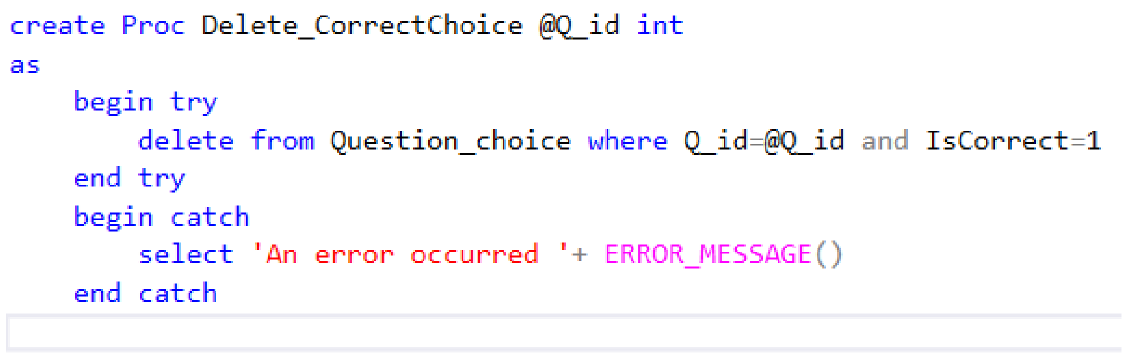


```
create Proc Delete_AllChoicesOfQuestion @Q_id int
as
    begin try
        delete from Question_choice where Q_id=@Q_id
    end try
    begin catch
        select 'An error occurred '+ ERROR_MESSAGE()
    end catch
```

1. **Purpose**: This stored procedure is designed to delete all choices (options) associated with a specific question based on the provided question ID.
2. **Parameters**:
   - `@Q_id` (integer): Represents the unique identifier of the question for which all choices need to be deleted. It serves as an input parameter required for the successful execution of the stored procedure.
3. **Functionality**:
   - Upon receiving the question ID as input, the stored procedure performs the following steps:
     - Deletes all records from the `Question_choice` table where the `Q_id` matches the specified question ID.
     - If the deletion is successful, all choices related to that question are removed from the database.
     - If an error occurs during execution (e.g., invalid ID or database issue), it catches the error and returns a custom message indicating the error along with the error message.

▪

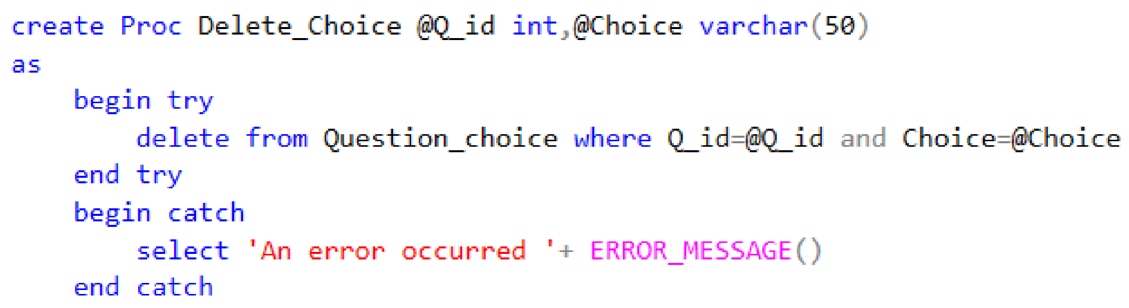
Delete_CorrectChoice



```
create Proc Delete_CorrectChoice @Q_id int
as
    begin try
        delete from Question_choice where Q_id=@Q_id and IsCorrect=1
    end try
    begin catch
        select 'An error occurred '+ ERROR_MESSAGE()
    end catch
```

1. **Purpose**: This stored procedure is designed to delete the correct choice (answer) associated with a specific question based on the provided question ID.
2. **Parameters**:
   - `@Q_id` (integer): Represents the unique identifier of the question for which the correct choice needs to be deleted. It serves as an input parameter required for the successful execution of the stored procedure.
3. **Functionality**:
   - Upon receiving the question ID as input, the stored procedure performs the following steps:
     - Deletes the record from the `Question_choice` table where the `Q_id` matches the specified question ID and the `IsCorrect` flag is set to 1 (indicating the correct choice).
     - If the deletion is successful, the correct choice associated with that question is removed from the database.
     - If an error occurs during execution (e.g., invalid ID or database issue), it catches the error and returns a custom message indicating the error along with the error message.

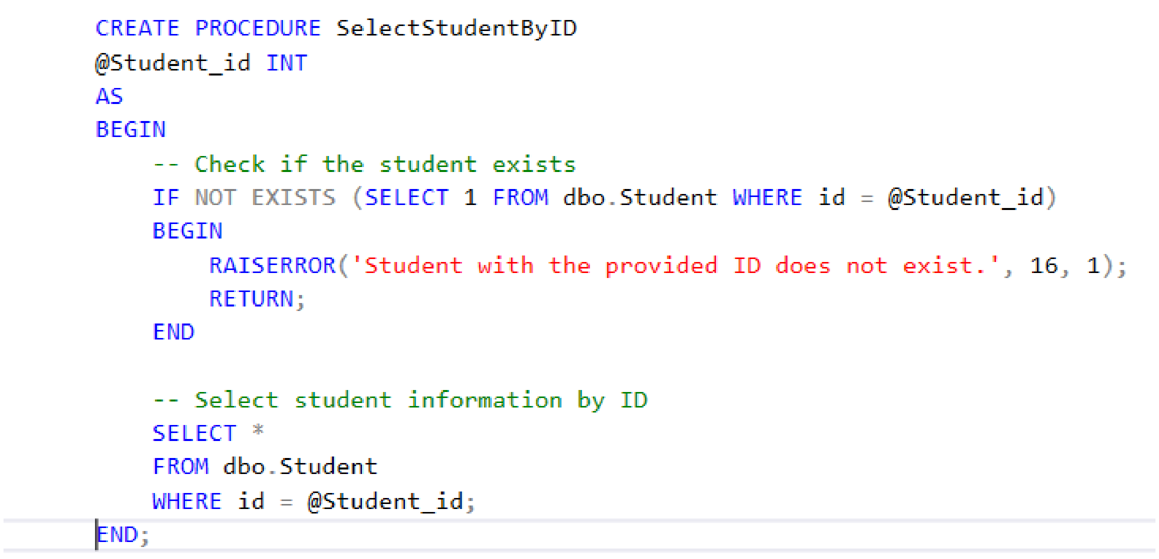
Delete_Choice



```
create Proc Delete_Choice @Q_id int,@Choice varchar(50)
as
    begin try
        delete from Question_choice where Q_id=@Q_id and Choice=@Choice
    end try
    begin catch
        select 'An error occurred '+ ERROR_MESSAGE()
    end catch
```

1. **Purpose**: This stored procedure is designed to delete a specific choice (option) associated with a particular question based on the provided question ID and choice text.
2. **Parameters**:
   - `@Q_id` (integer): Represents the unique identifier of the question from which the choice needs to be deleted. It serves as an input parameter required for the successful execution of the stored procedure.
   - `@Choice` (varchar(50)): Represents the text of the choice (e.g., option A, option B, etc.) that should be removed. It is another input parameter essential for the execution of this stored procedure.
3. **Functionality**:
   - Upon receiving the necessary input parameters, the stored procedure performs the following steps:
     - Deletes the record from the `Question_choice` table where the `Q_id` matches the specified question ID and the `Choice` matches the provided choice text.
     - If the deletion is successful, the specified choice is removed from the database for that particular question.
     - If an error occurs during execution (e.g., invalid parameters or database issue), it catches the error and returns a custom message indicating the error along with the error message.

Select Student By ID Stored Procedure



```sql
CREATE PROCEDURE SelectStudentByID
@Student_id INT
AS
BEGIN
    -- Check if the student exists
    IF NOT EXISTS (SELECT 1 FROM dbo.Student WHERE id = @Student_id)
    BEGIN
        RAISERROR('Student with the provided ID does not exist.', 16, 1);
        RETURN;
    END

    -- Select student information by ID
    SELECT *
    FROM dbo.Student
    WHERE id = @Student_id;
END;
```

Description:

Retrieves information about a student from the database based on the provided student ID.

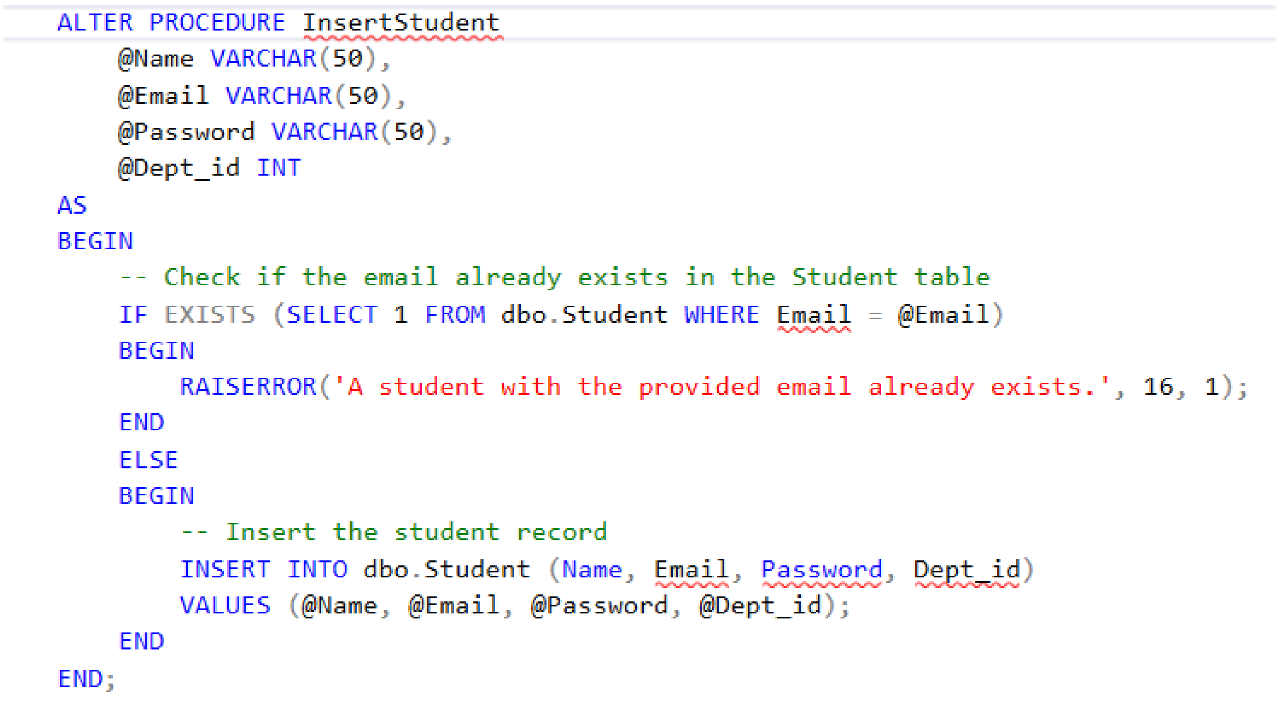
Parameters:

@Student_id (INT): ID of the student whose information is to be retrieved.

Functionality:

1. Checks if the student with the provided ID exists.
2. Retrieves and returns all information about the student from the Student table.

Insert Student Stored Procedure



```
ALTER PROCEDURE InsertStudent
    @Name VARCHAR(50),
    @Email VARCHAR(50),
    @Password VARCHAR(50),
    @Dept_id INT
AS
BEGIN
    -- Check if the email already exists in the Student table
    IF EXISTS (SELECT 1 FROM dbo.Student WHERE Email = @Email)
    BEGIN
        RAISERROR('A student with the provided email already exists.', 16, 1);
    END
    ELSE
    BEGIN
        -- Insert the student record
        INSERT INTO dbo.Student (Name, Email, Password, Dept_id)
        VALUES (@Name, @Email, @Password, @Dept_id);
    END
END;
```

Description:

Inserts a new student record into the database.

.

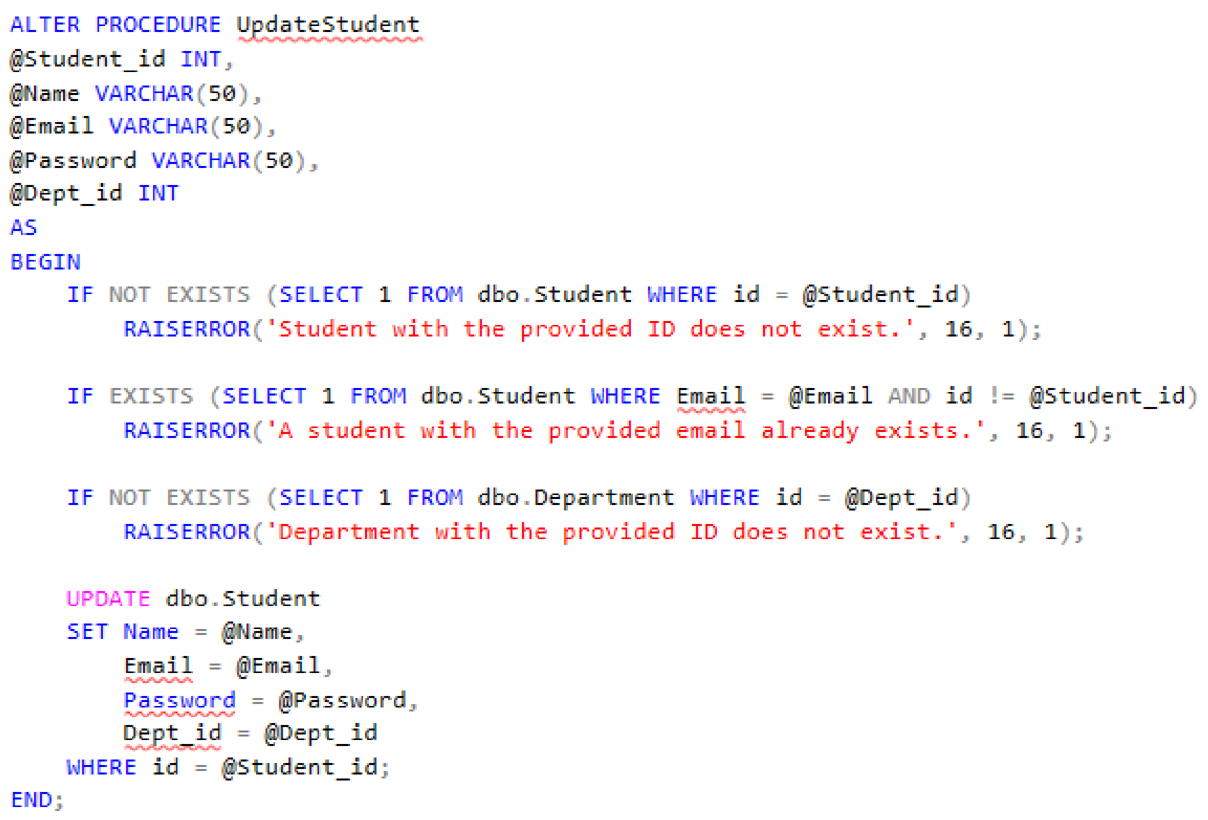
Parameters:

1) @Name (VARCHAR(50)): Name of the student to be inserted.
2) @Email (VARCHAR(50)): Email of the student to be inserted. It must be unique.
3) @Password (VARCHAR(50)): Password of the student to be inserted.
4) @Dept_id (INT): Department ID of the student to be inserted.

Functionality:

1) Checks if the provided email already exists in the Student table.
2) Inserts the new student record if the email is unique.
3) Raises an error if the email already exists, providing a user-friendly error message.

Update Student Stored Procedure



```
ALTER PROCEDURE UpdateStudent
@Student_id INT,
@Name VARCHAR(50),
@Email VARCHAR(50),
@Password VARCHAR(50),
@Dept_id INT
AS
BEGIN
    IF NOT EXISTS (SELECT 1 FROM dbo.Student WHERE id = @Student_id)
        RAISERROR('Student with the provided ID does not exist.', 16, 1);

    IF EXISTS (SELECT 1 FROM dbo.Student WHERE Email = @Email AND id != @Student_id)
        RAISERROR('A student with the provided email already exists.', 16, 1);

    IF NOT EXISTS (SELECT 1 FROM dbo.Department WHERE id = @Dept_id)
        RAISERROR('Department with the provided ID does not exist.', 16, 1);

    UPDATE dbo.Student
    SET Name = @Name,
        Email = @Email,
        Password = @Password,
        Dept_id = @Dept_id
    WHERE id = @Student_id;
END;
```

Description:

Updates an existing student record in the database with new information, including name, email, password, and department.

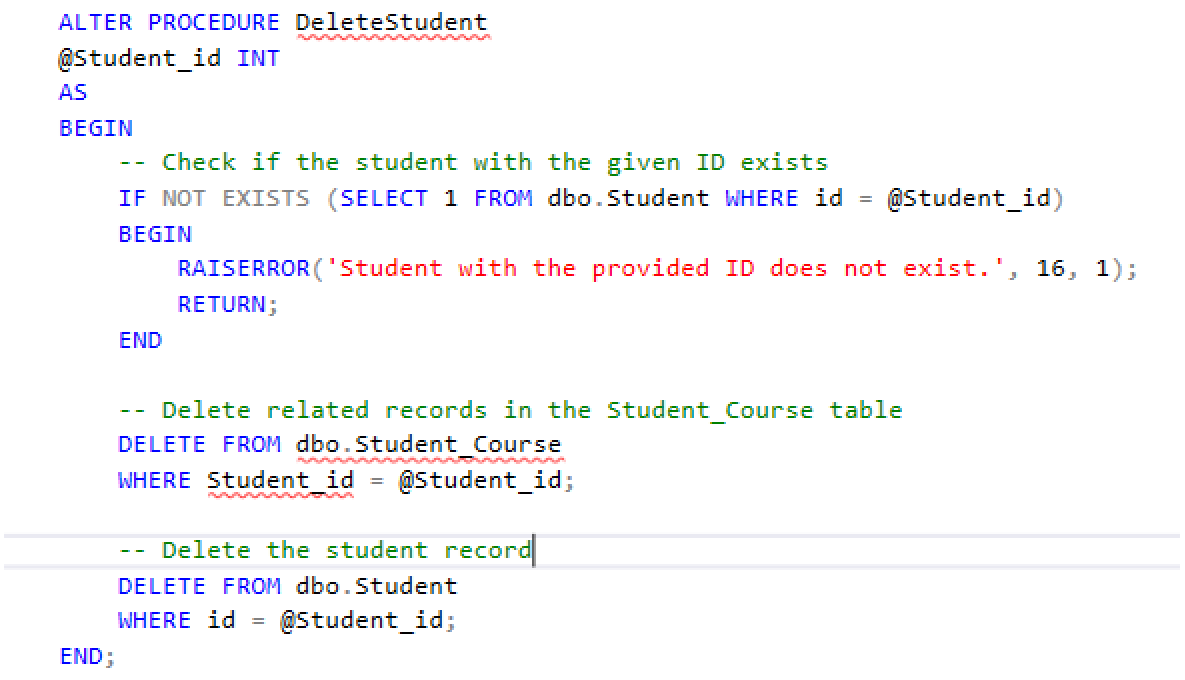
Parameters:

5. @Student_id (INT): ID of the student to be updated.
6. @Name (VARCHAR(50)): New name of the student.
7. @Email (VARCHAR(50)): New email of the student.
8. @Password (VARCHAR(50)): New password of the student.
9. @Dept_id (INT): New department ID of the student.

Functionality:

1) Checks if the student with the provided ID exists.
2) Verifies the uniqueness of the new email (excluding the current student).
3) Validates the existence of the new department ID.
4) Updates the student record with the provided details if all checks pass.

DeleteStudent Stored Procedure



```
ALTER PROCEDURE DeleteStudent
@Student_id INT
AS
BEGIN
    -- Check if the student with the given ID exists
    IF NOT EXISTS (SELECT 1 FROM dbo.Student WHERE id = @Student_id)
    BEGIN
        RAISERROR('Student with the provided ID does not exist.', 16, 1);
        RETURN;
    END

    -- Delete related records in the Student_Course table
    DELETE FROM dbo.Student_Course
    WHERE Student_id = @Student_id;

    -- Delete the student record
    DELETE FROM dbo.Student
    WHERE id = @Student_id;
END;
```

Description:

Deletes an existing student record from the database along with any related records in the Student_Course table.

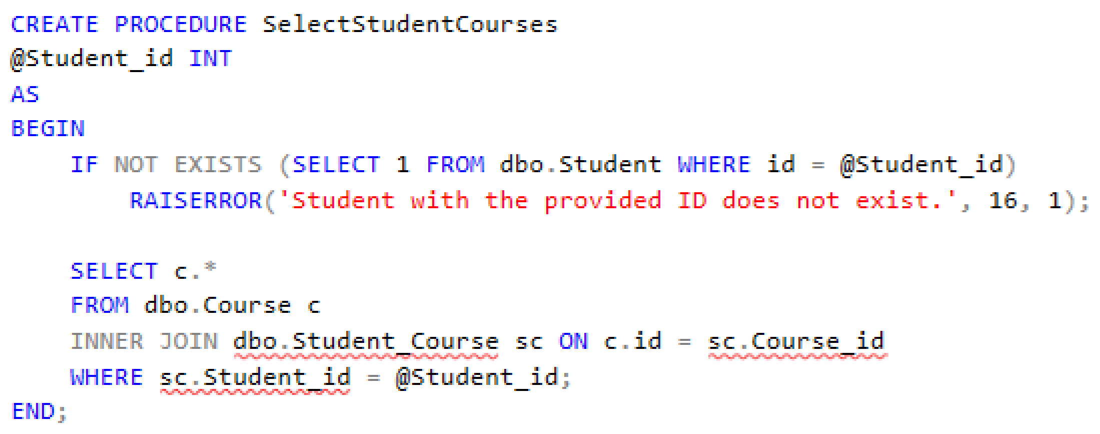
Parameters:

1) @Student_id (INT): ID of the student to be deleted.

Functionality:

1) Checks if the student with the provided ID exists.
2) Deletes any related records in the Student_Course table associated with the student.
3) Deletes the student record from the Student table.

Select All StudentCourses Stored Procedure



```
CREATE PROCEDURE SelectStudentCourses
@Student_id INT
AS
BEGIN
    IF NOT EXISTS (SELECT 1 FROM dbo.Student WHERE id = @Student_id)
        RAISERROR('Student with the provided ID does not exist.', 16, 1);

    SELECT c.*
    FROM dbo.Course c
    INNER JOIN dbo.Student_Course sc ON c.id = sc.Course_id
    WHERE sc.Student_id = @Student_id;
END;
```

Description:

Selects all courses associated with a given student from the database.

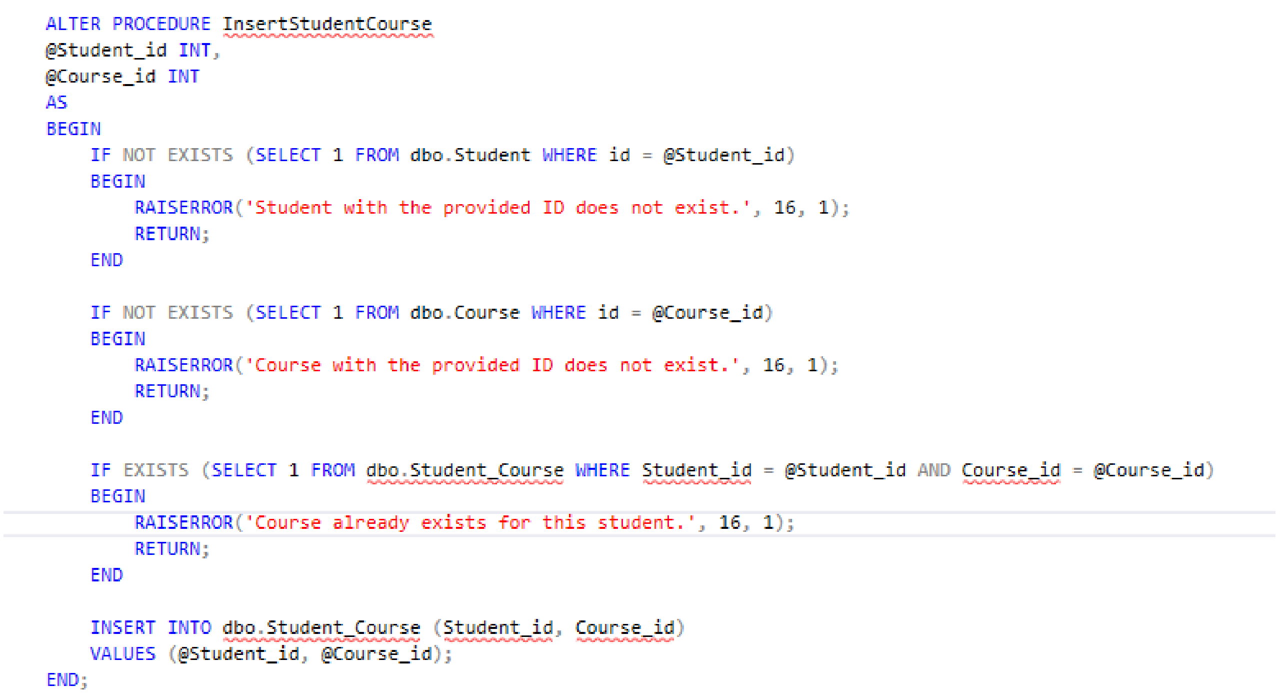
Parameters:

4) @Student_id (INT): ID of the student whose courses are to be retrieved.

Functionality:

5) Checks if the student exists.
6) Selects courses for the given student.

Insert Student Course Stored Procedure



```sql
ALTER PROCEDURE InsertStudentCourse
@Student_id INT,
@Course_id INT
AS
BEGIN
    IF NOT EXISTS (SELECT 1 FROM dbo.Student WHERE id = @Student_id)
    BEGIN
        RAISERROR('Student with the provided ID does not exist.', 16, 1);
        RETURN;
    END

    IF NOT EXISTS (SELECT 1 FROM dbo.Course WHERE id = @Course_id)
    BEGIN
        RAISERROR('Course with the provided ID does not exist.', 16, 1);
        RETURN;
    END

    IF EXISTS (SELECT 1 FROM dbo.Student_Course WHERE Student_id = @Student_id AND Course_id = @Course_id)
    BEGIN
        RAISERROR('Course already exists for this student.', 16, 1);
        RETURN;
    END

    INSERT INTO dbo.Student_Course (Student_id, Course_id)
    VALUES (@Student_id, @Course_id);
END;
```

Description:

Inserts a new student-course relationship into the database.

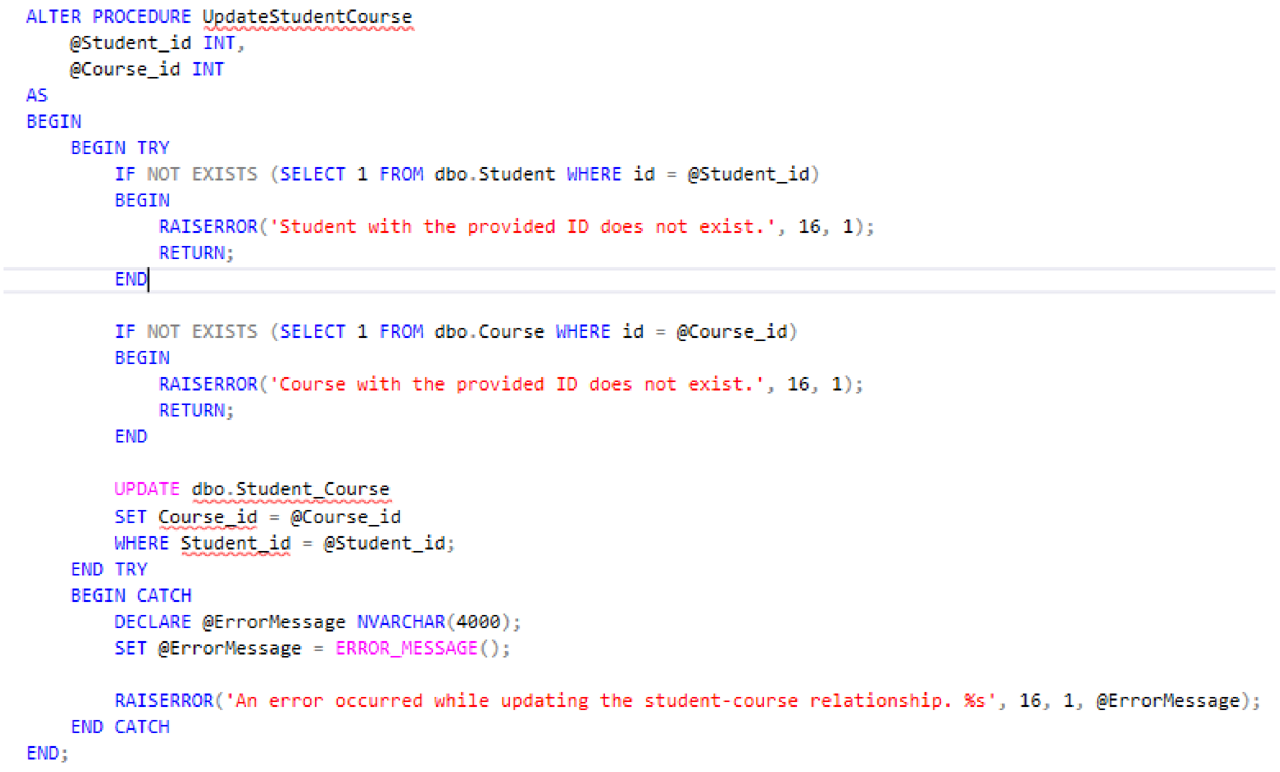
Parameters:

1) @Student_id (INT): ID of the student to associate with the course.
2) @Course_id (INT): ID of the course to associate with the student.

Functionality:

1) Checks if the student with the provided ID exists.
2) Checks if the course with the provided ID exists.
3) Checks if the student-course relationship already exists.
4) Inserts the student-course relationship if it doesn't already exist.

Update Student Course



```sql
ALTER PROCEDURE UpdateStudentCourse
    @Student_id INT,
    @Course_id INT
AS
BEGIN
    BEGIN TRY
        IF NOT EXISTS (SELECT 1 FROM dbo.Student WHERE id = @Student_id)
        BEGIN
            RAISERROR('Student with the provided ID does not exist.', 16, 1);
            RETURN;
        END

        IF NOT EXISTS (SELECT 1 FROM dbo.Course WHERE id = @Course_id)
        BEGIN
            RAISERROR('Course with the provided ID does not exist.', 16, 1);
            RETURN;
        END

        UPDATE dbo.Student_Course
        SET Course_id = @Course_id
        WHERE Student_id = @Student_id;
    END TRY
    BEGIN CATCH
        DECLARE @ErrorMessage NVARCHAR(4000);
        SET @ErrorMessage = ERROR_MESSAGE();

        RAISERROR('An error occurred while updating the student-course relationship. %s', 16, 1, @ErrorMessage);
    END CATCH
END;
```

Description:

This stored procedure updates the course associated with a student in the database.

Parameters:

1) @Student_id (INT): The ID of the student whose course is to be updated.
2) @Course_id (INT): The new ID of the course to associate with the student.

Functionality:

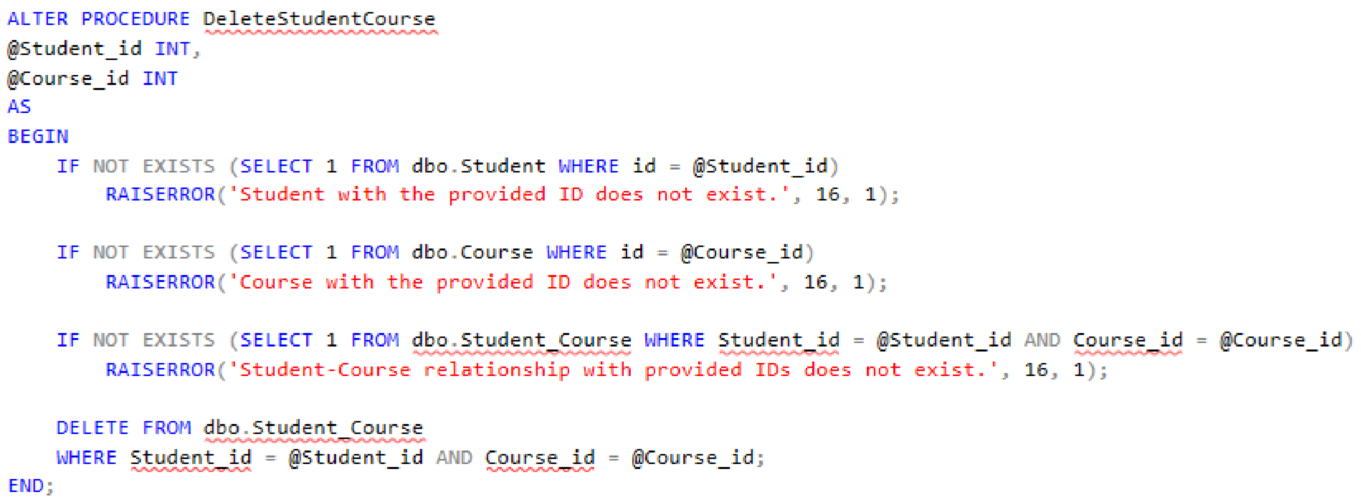
1) Checks if the student exists in the Student table.
2) Checks if the course exists in the Course table.

3) Updates the student-course relationship in the Student_Course table.

Error Handling:

1) If the provided student ID does not exist, the procedure raises an error indicating that the student does not exist.
2) If the provided course ID does not exist, the procedure raises an error indicating that the course does not exist.
3) If an error occurs during the update process, it is caught and an appropriate error message is raised.

Delete Student Course Stored Procedure



```sql
ALTER PROCEDURE DeleteStudentCourse
@Student_id INT,
@Course_id INT
AS
BEGIN
    IF NOT EXISTS (SELECT 1 FROM dbo.Student WHERE id = @Student_id)
        RAISERROR('Student with the provided ID does not exist.', 16, 1);

    IF NOT EXISTS (SELECT 1 FROM dbo.Course WHERE id = @Course_id)
        RAISERROR('Course with the provided ID does not exist.', 16, 1);

    IF NOT EXISTS (SELECT 1 FROM dbo.Student_Course WHERE Student_id = @Student_id AND Course_id = @Course_id)
        RAISERROR('Student-Course relationship with provided IDs does not exist.', 16, 1);

    DELETE FROM dbo.Student_Course
    WHERE Student_id = @Student_id AND Course_id = @Course_id;
END;
```

Description:

This stored procedure deletes a student-course relationship from the database.

Parameters:

5. @Student_id (INT): The ID of the student whose course relationship is to be deleted.
6. @Course_id (INT): The ID of the course to be disassociated from the student.student.

Functionality:

1. Checks if the student exists in the Student table.
2. Checks if the course exists in the Course table.
3. Checks if the student-course relationship exists in the Student_Course table.
4. Deletes the student-course relationship from the Student_Course table.

Stored Procedures for Reporting

Stored Procedure: DisplayExam

```
create proc DisplayExam @examID int , @StudentID int
as
select
    q.id AS Question_ID,
    q.Title,
    qc.choice
FROM
    question q
LEFT JOIN
    Question_choice qc ON qc.Q_id = q.id
INNER JOIN
    Exam_St_Q esq ON esq.Q_id = q.id
WHERE
    esq.Exam_id = @examID
    AND esq.Student_id = @StudentID
ORDER BY q.id;
```

Description:

The “DisplayExam” stored procedure retrieves information related to an exam for a specific student. It combines data from the “question,” “Question_choice,” and “Exam_St_Q” tables to present a detailed view of the exam questions and their available choices.

Parameters:

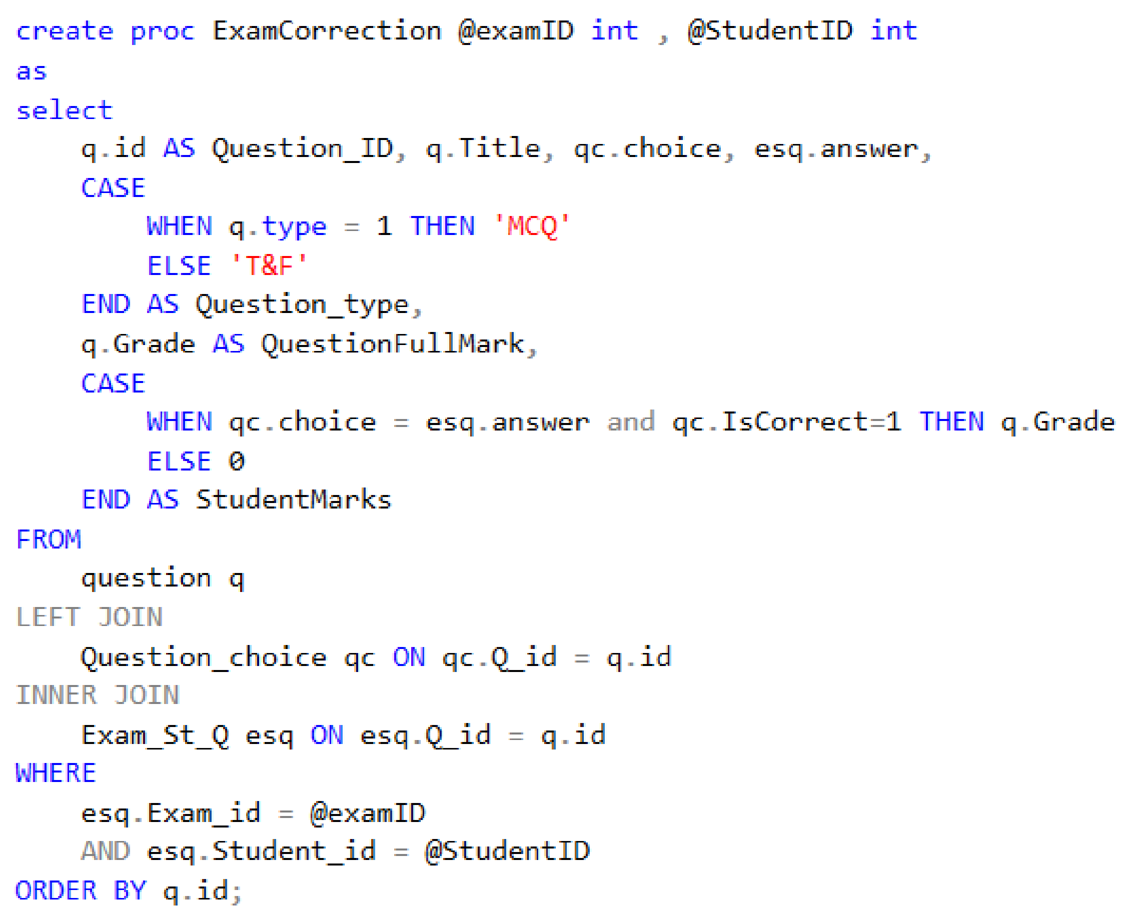
1. **@examID** (integer): Represents the unique identifier of the exam for which details are to be displayed.
2. **@StudentID** (integer): Represents the unique identifier of the student for whom the exam details are retrieved.

Functionality:

The “DisplayExam” stored procedure provides the following functionalities:

1. **Data Retrieval**: Retrieves information about each question in the specified exam, including its ID and title.
2. **Choice Details**: Joins the “Question_choice” table to include the available choices (options) for each question.
3. **Student-Specific**: Filters the results based on the provided student ID to display only relevant exam details.
4. **Ordering**: Orders the questions by their IDs for consistent presentation.

Stored Procedure: ExamCorrection



```sql
create proc ExamCorrection @examID int , @StudentID int
as
select
    q.id AS Question_ID, q.Title, qc.choice, esq.answer,
    CASE
        WHEN q.type = 1 THEN 'MCQ'
        ELSE 'T&F'
    END AS Question_type,
    q.Grade AS QuestionFullMark,
    CASE
        WHEN qc.choice = esq.answer and qc.IsCorrect=1 THEN q.Grade
        ELSE 0
    END AS StudentMarks
FROM
    question q
LEFT JOIN
    Question_choice qc ON qc.Q_id = q.id
INNER JOIN
    Exam_St_Q esq ON esq.Q_id = q.id
WHERE
    esq.Exam_id = @examID
    AND esq.Student_id = @StudentID
ORDER BY q.id;
```

Description:

The “ExamCorrection” stored procedure retrieves and calculates exam-related information for a specific student. It combines data from the “question,” “Question_choice,” and “Exam_St_Q” tables to determine the student’s marks based on their answers.

Parameters:

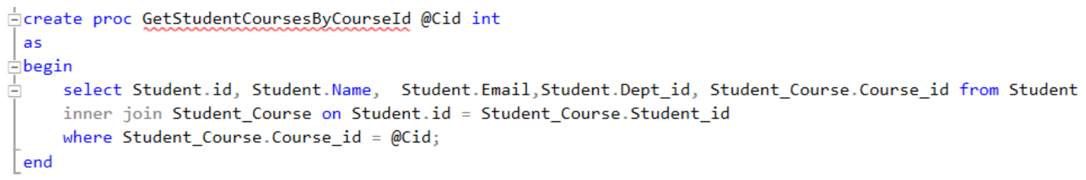
1. **@examID** (integer): Represents the unique identifier of the exam for which correction details are to be calculated.
2. **@StudentID** (integer): Represents the unique identifier of the student whose exam is being corrected.

Functionality:

The “ExamCorrection” stored procedure provides the following functionalities:

1. **Data Retrieval**: Retrieves information about each question in the specified exam, including its ID, title, and available choices.
2. **Answer Comparison**: Compares the student's answers (stored in the "Exam_St_Q" table) with the correct choices (from the "Question_choice" table).
3. **Scoring**: Calculates the student's marks based on the correctness of their answers and the assigned question grades.
4. **Ordering**: Orders the questions by their IDs for consistent presentation.

- **GetStudentCoursesByCourseId**



```
create proc GetStudentCoursesByCourseId @Cid int
as
begin
    select Student.id, Student.Name,  Student.Email,Student.Dept_id, Student_Course.Course_id from Student
    inner join Student_Course on Student.id = Student_Course.Student_id
    where Student_Course.Course_id = @Cid;
end
```

Description:

The stored procedure named GetStudentCoursesByCourseId retrieves course student information from student and course tables within the ExaminationSystem database. It focuses on a specific course identified by its unique CourseId. By passing the @CId parameter, users can obtain a list of students associated with that course.

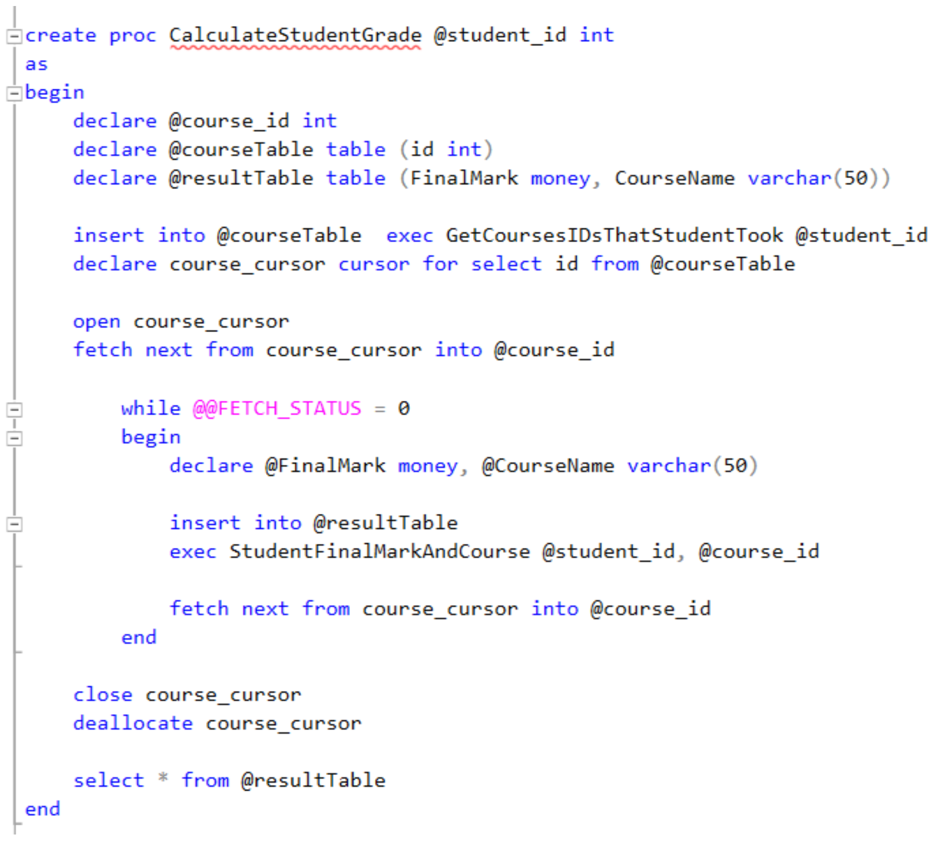
Parameters:

1. @CId (integer): Represents the unique identifier (ID) of the course.

Functionality:

Upon invocation, this procedure selects records from the student table where the Course_id matches the provided @CId parameter.

- **CalculateStudentGrade**



```
create proc CalculateStudentGrade @student_id int
as
begin
    declare @course_id int
    declare @courseTable table (id int)
    declare @resultTable table (FinalMark money, CourseName varchar(50))

    insert into @courseTable  exec GetCoursesIDsThatStudentTook @student_id
    declare course_cursor cursor for select id from @courseTable

    open course_cursor
    fetch next from course_cursor into @course_id

        while @@FETCH_STATUS = 0
        begin
            declare @FinalMark money, @CourseName varchar(50)

            insert into @resultTable
            exec StudentFinalMarkAndCourse @student_id, @course_id

            fetch next from course_cursor into @course_id
        end

    close course_cursor
    deallocate course_cursor

    select * from @resultTable
end
```

Description:

The stored procedure named `CalculateStudentGrade` retrieves course name and student degree in this course. It focuses on a specific student identified by its unique StudentID. By passing the @student_id parameter, users can obtain a list of Courses associated with that Student With his degree in each course.

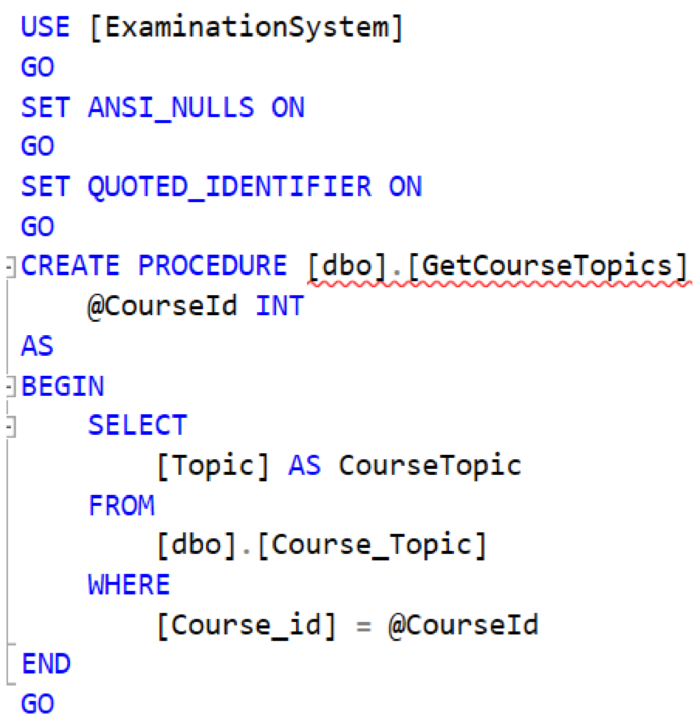
Parameters:

1. @student_id (int): Represents the unique identifier (ID) of the student.

Functionality:

Upon invocation, this procedure selects records from more table to calculate degree on each course and retrieve it where the StudentId matches the provided @student_id parameter.

- **Get Course Topics Stored Procedure Report**



```
USE [ExaminationSystem]
GO
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE PROCEDURE [dbo].[GetCourseTopics]
    @CourseId INT
AS
BEGIN
    SELECT
        [Topic] AS CourseTopic
    FROM
        [dbo].[Course_Topic]
    WHERE
        [Course_id] = @CourseId
END
GO
```

Description:

The stored procedure named GetCourseTopics retrieves course topics from the [dbo].[Course_Topic] table within the ExaminationSystem database. It focuses on a specific course identified by its unique CourseId. By passing the @CourseId parameter, users can obtain a list of topics associated with that course.

Parameters:

2. @CourseId (integer): Represents the unique identifier (ID) of the course for which topics are to be retrieved.

Functionality:

Upon invocation, this procedure selects records from the [dbo].[Course_Topic] table where the Course_id matches the provided @CourseId parameter.

- **Get Instructor Courses And Students Stored Procedure Report**

```
USE [ExaminationSystem]
GO
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE PROCEDURE [dbo].[GetInstructorCoursesAndStudents]
    @InstructorId INT
AS
BEGIN
    SELECT
        C.[Name] AS CourseName,
        COUNT(SC.[Student_id]) AS NumberOfStudents
    FROM
        [dbo].[Course_Inst] CI
    INNER JOIN
        [dbo].[Course] C ON CI.[Course_Id] = C.[id]
    LEFT JOIN
        [dbo].[Student_Course] SC ON CI.[Course_Id] = SC.[Course_id]
    WHERE
        CI.[Inst_Id] = @InstructorId
    GROUP BY
        C.[Name]
END
GO
```

Description:

The stored procedure named `GetInstructorCoursesAndStudents` retrieves information related to courses taught by a specific instructor. It focuses on the instructor identified by the unique `@InstructorId` parameter. By invoking this procedure, users can obtain a list of course names along with the corresponding number of students enrolled in each course.

Parameters:

1. @InstructorId (integer): Represents the unique identifier (ID) of the instructor for whom course information is to be retrieved.

Functionality:

Upon invocation, this procedure performs the following steps:

1. Joins the `[dbo].[Course_Inst]` table with the `[dbo].[Course]` table based on the `Course_Id`.
2. Retrieves the course name (`C.[Name]`) and calculates the count of students (`COUNT(SC.[Student_id])`) associated with each course.
3. Filters the results to include only courses where the instructor's ID (`CI.[Inst_Id]`) matches the provided `@InstructorId`.
4. Groups the results by course name (`C.[Name]`).

Conclusion:

The "ExaminationSystem" project, developed as part of the ITI Schooler Ship program, is a robust application focusing on managing various aspects of an examination system. It comprises a database backend and a desktop application frontend, utilizing technologies like SQL Server, C#, LINQ, Entity Framework 8, and Report Builder.

The project's database efficiently stores data related to students, courses, exams, and more, with stored procedures aiding in complex operations. The desktop application offers a user-friendly interface for administrators, instructors, and students, facilitating tasks such as exam creation and student assessment.

Powered by Report Builder, the system provides reporting capabilities for generating custom reports on exam results and student performance.

Overall, the "ExaminationSystem" project represents a successful collaboration, demonstrating expertise in web development and business intelligence gained through the ITI Schooler Ship program.