Information Technology Institute

3/6/2025

Examination System

Supervisor: Eng. Marwa

Team Members:

1. **Santy Osama Mina**
2. **Yousry Essam Ayoub**

ITI Intake 45, Assiut Branch

Table Of Contents

Contents

Abstract

In the modern educational landscape, online examination systems have become essential for ensuring seamless and efficient assessment processes. This project aims to develop an **Automated Online Examination System** that facilitates the creation, execution, and evaluation of online exams. The system is supported by a **structured SQL database**, ensuring data integrity and efficient query performance.

To achieve this, an **Entity-Relationship Diagram (ERD)** was designed to define the relationships between various entities such as students, courses, exams, and questions. Additionally, a **Database Dictionary** was created to document table structures, attributes, and constraints.

The system also includes **stored procedures** to handle core functionalities:

- **Basic CRUD operations** (Select, Insert, Update, Delete) on all tables.
- **Exam Generation** to randomly create exams with different types of questions.
- **Exam Answers Management** to store students' responses.
- **Exam Correction** to automate grading based on predefined correct answers.
- This automated system enhances the efficiency of exam administration, reduces manual effort, and ensures fairness in assessment. Future enhancements may include **reporting tools (SSRS, Power BI)** and **integration with social media platforms** to further expand system capabilities.

Database Design

1.1 Entity-Relationship Diagram (ERD):

1.1.1 Overview:

The **Entity-Relationship Diagram (ERD)** represents the structure of the **Online Examination System** by illustrating the relationships between various tables/entities. This system manages **students, instructors, courses, exams, questions, answers, branches, and tracks** while ensuring **data integrity and normalization**.

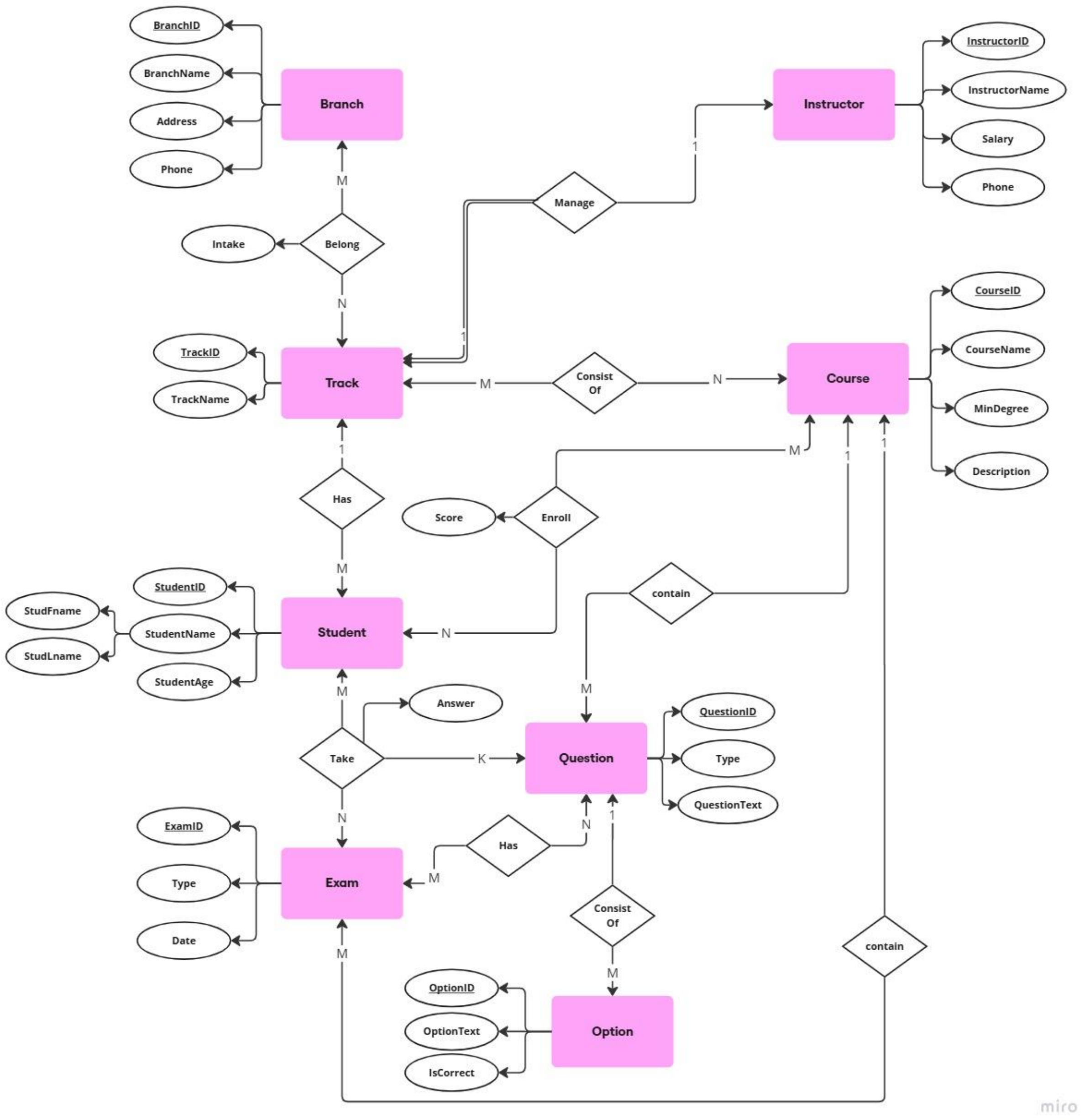
Key Entities & Relationships

1. **Branch**
   - A **Branch** offers multiple **Tracks**.
   - Relationship: **One-to-Many** (Branch → Track).
2. **Instructor**
   - An **Instructor** manages a **Track**.
   - Relationship: **One-to-One** (Instructor → Track).
3. **Track**
   - A **Track** contains multiple **Courses**.
   - Relationship: **One-to-Many** (Track → Course).
4. **Student**
   - A **Student** is assigned to one **Track**.
   - A **Student** enrolls in multiple **Courses**.
   - Relationship: **Many-to-One** (Student → Track), **Many-to-Many** (Student ↔ Course).
5. **Course**
   - A **Course** is associated with multiple **Exams**.
   - A **Course** contains multiple **Questions**.

- Relationship: **One-to-Many** (Course → Exam), **One-to-Many** (Course → Question).

6. **Exam**
   - An **Exam** consists of multiple **Questions**.
   - A **Student** takes multiple **Exams**.
   - Relationship: **Many-to-Many** (Exam ↔ Question), **Many-to-Many** (Student ↔ Exam).
7. **Question**
   - A **Question** has multiple **Options**.
   - Relationship: **One-to-Many** (Question → Option).
8. **Option**
   - Each **Option** belongs to one **Question** and indicates whether it is correct.
   - Relationship: **Many-to-One** (Option → Question).
9. **Student_Exam_Question**
   - Stores **student answers** to exam questions.
   - Relationship: **Many-to-Many** (Student ↔ Question in an Exam).

1.1.2 ERD Diagram Representation:



1.2 Database Schema Design

The **Database Schema Design** outlines the structure of the **Online Examination System** database. It includes tables, their attributes, relationships, and constraints to ensure data integrity and efficiency.

Schema Overview

The database consists of **13 tables**, each serving a specific purpose in managing **students, courses, exams, questions, and results**. Below is an overview of the schema design:

1. **Branch** – Stores branch details.
2. **Instructor** – Stores instructors' details.
3. **Track** – Defines different study tracks.
4. **Course** – Contains course information.
5. **Student** – Manages student records.
6. **Exam** – Stores exam details.
7. **Question** – Contains exam questions.
8. **Option_Table** – Holds multiple-choice question options.
9. **Branch_Track** – Manages branch and track relationships.
10. **Track_Course** – Defines courses assigned to tracks.
11. **Student_Course** – Tracks students' enrollment and scores in courses.
12. **Student_Exam_Question** – Stores students' answers to exam questions.
13. **Exam_Question** – Links exams to their questions.

Schema Constraints

- **Primary Keys (PK):** Ensure unique identification of each record.
- **Foreign Keys (FK):** Establish relationships between tables.
- **Unique Constraints:** Prevent duplicate values where necessary.
- **Check Constraints:** Maintain data validity (e.g., non-negative salaries, valid scores).

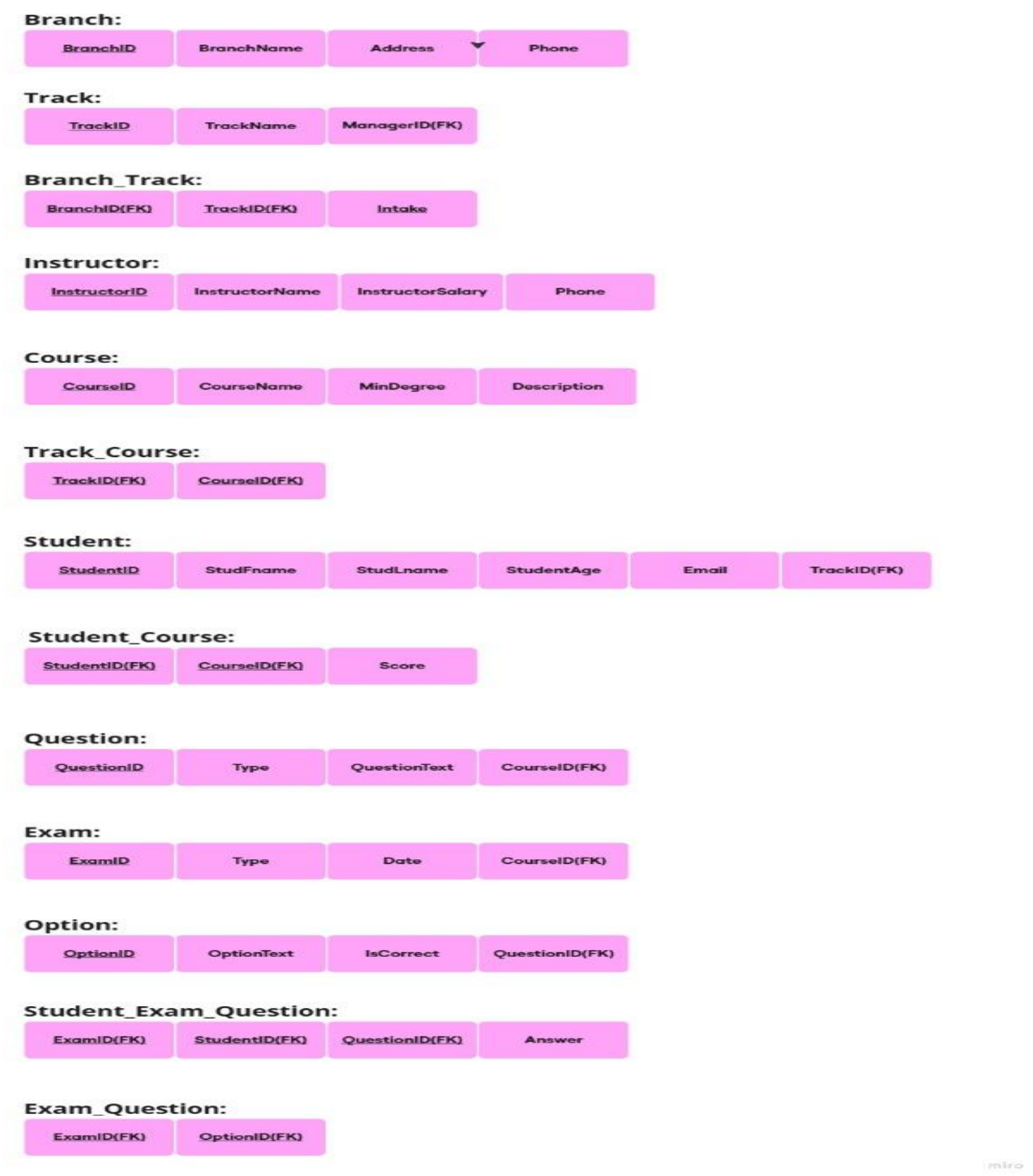
Relationships Between Tables

- **Branch ↔ Track**: A branch can have multiple tracks (Branch_Track table).
- **Track ↔ Course**: A track can offer multiple courses (Track_Course table).
- **Course ↔ Exam**: Each course can have multiple exams.
- **Exam ↔ Question**: Each exam consists of multiple questions.
- **Question ↔ Option_Table**: Each question has multiple options.
- **Student ↔ Track**: A student belongs to a specific track.
- **Student ↔ Course**: A student enrolls in multiple courses (Student_Course table).

- **Student ↔ Exam**: Students take exams and answer questions (Student_Exam_Question table).

This schema ensures a **normalized database** with **efficient data retrieval, consistency, and integrity** for the **Online Examination System**.

Database Mapping



Branch:

BranchID	BranchName	Address	Phone

Track:

TrackID	TrackName	ManagerID(FK)

Branch_Track:

BranchID(FK)	TrackID(FK)	Intake

Instructor:

InstructorID	InstructorName	InstructorSalary	Phone

Course:

CourseID	CourseName	MinDegree	Description

Track_Course:

TrackID(FK)	CourseID(FK)

Student:

StudentID	StudFname	StudLname	StudentAge	Email	TrackID(FK)

Student_Course:

StudentID(FK)	CourseID(FK)	Score

Question:

QuestionID	Type	QuestionText	CourseID(FK)

Exam:

ExamID	Type	Date	CourseID(FK)

Option:

OptionID	OptionText	IsCorrect	QuestionID(FK)

Student_Exam_Question:

ExamID(FK)	StudentID(FK)	QuestionID(FK)	Answer

Exam_Question:

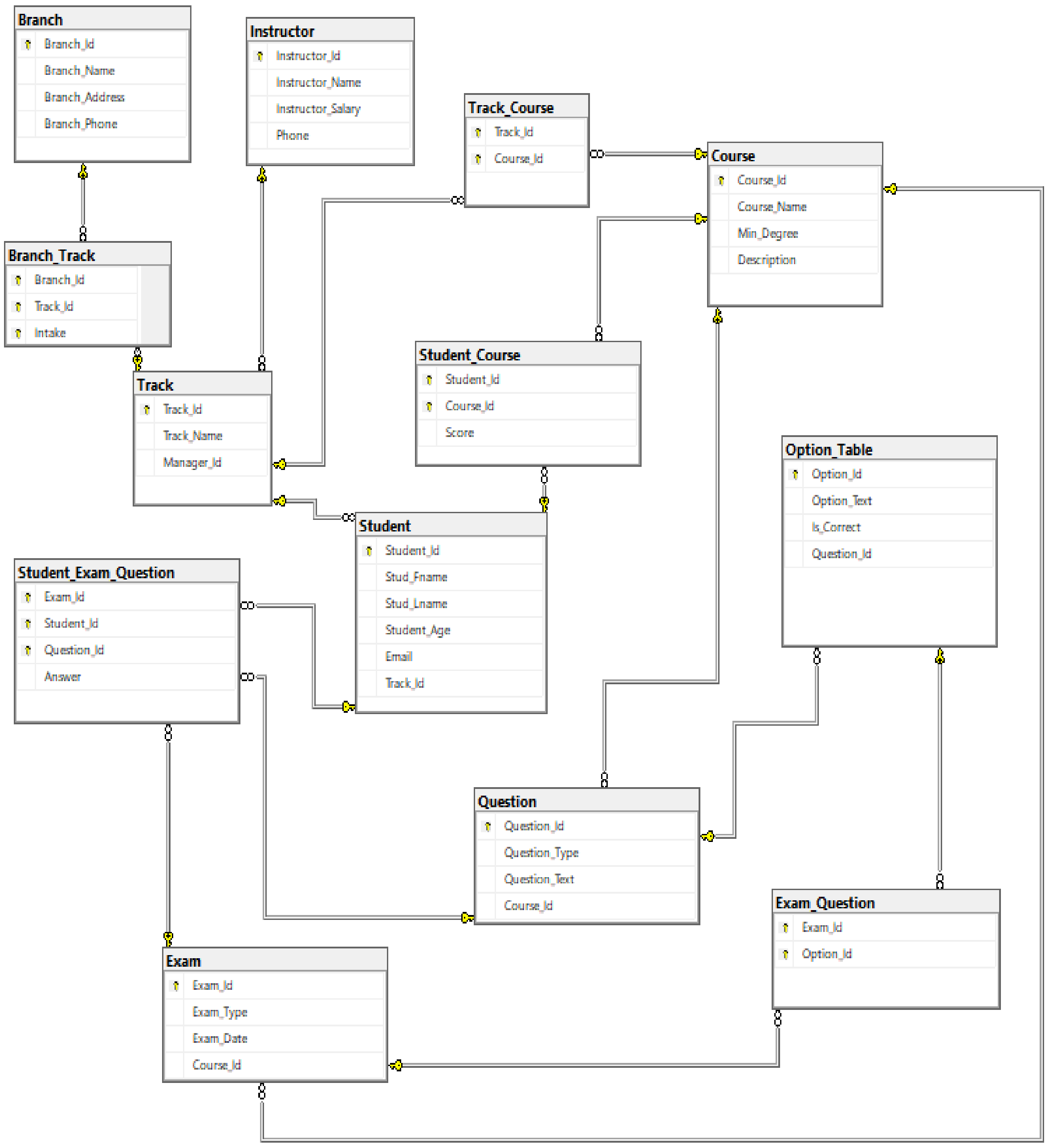
ExamID(FK)	OptionID(FK)

miro

Link Of ERD and Mapping:

Flowchart - Miro

Database Diagram:



1.3 Database Dictionary:

Overview:

The **Database Dictionary** defines the structure of the database by listing all tables along with their attributes, data types, constraints, and a brief description of each column. This ensures clarity in understanding the **schema design** and **data integrity constraints**.

1. Branch Table:

Stores information about different training branches.

Column Name	Data Type	Constraints	Description
branch_id	INT	PRIMARY KEY	Unique identifier for each branch
branch_name	NVARCHAR(50)	NOT NULL	Name of the branch
branch_address	NVARCHAR(100)	NOT NULL	Branch location
branch_phone	NVARCHAR(15)	NULLABLE	Contact number of the branch

2. Instructor Table:

Stores details of instructors.

Column Name	Data Type	Constraints	Description
instructor_id	INT	PRIMARY KEY	Unique identifier for instructors
instructor_name	VARCHAR(50)	NOT NULL	Instructor's full name
instructor_salary	DECIMAL(10,2)	CHECK (instructor_salary >= 0)	Salary of the instructor
phone	VARCHAR(15)	UNIQUE	Instructor’s contact number

3. Track Table:

Defines different study tracks in the system.

Column Name	Data Type	Constraints	Description
track_id	INT	PRIMARY KEY	Unique identifier for each track
track_name	VARCHAR(50)	NOT NULL	Name of the track
manager_id	INT	FOREIGN KEY	References Instructor(instructor_id)

4. Course Table:

Stores details of courses.

Column Name	Data Type	Constraints	Description
course_id	INT	PRIMARY KEY	Unique identifier for courses
course_name	VARCHAR(50)	NOT NULL	Name of the course
min_degree	INT	CHECK (min_degree >= 0)	Minimum passing grade
description	TEXT	NULLABLE	Course description

5. Student Table:

Stores student information.

Column Name	Data Type	Constraints	Description
student_id	INT	PRIMARY KEY	Unique identifier for students
stud_fname	VARCHAR(50)	NOT NULL	First name of the student
stud_lname	VARCHAR(50)	NOT NULL	Last name of the student
student_age	INT	CHECK (student_age > 0)	Age of the student
email	VARCHAR(100)	UNIQUE, NOT NULL	Student email for login
track_id	INT	FOREIGN KEY	References Track(track_id)

6. Exam Table:

Stores details about exams.

Column Name	Data Type	Constraints	Description
exam_id	INT	PRIMARY KEY	Unique identifier for each exam
exam_type	VARCHAR(50)	NOT NULL	Type of exam (e.g., Online, Written)
exam_date	DATE	DEFAULT GETDATE()	Date when the exam is scheduled
course_id	INT	FOREIGN KEY	References Course(course_id)

7. Question Table:

Stores questions for exams.

Column Name	Data Type	Constraints	Description
question_id	INT	PRIMARY KEY	Unique identifier for each question
question_type	VARCHAR(50)	CHECK (question_type IN ('MCQ', 'MMCQ', 'T/F'))	Type of question
question_text	TEXT	NOT NULL	Content of the question
course_id	INT	FOREIGN KEY	References Course(course_id)

8. Option_Table:

Stores answer choices for questions.

Column Name	Data Type	Constraints	Description
option_id	INT	PRIMARY KEY	Unique identifier for each option
option_text	TEXT	NOT NULL	Answer choice text
is_correct	BIT	NOT NULL	1 if correct, 0 if incorrect
question_id	INT	FOREIGN KEY	References Question(question_id)

9. Branch_Track Table:

Stores the relationship between branches and tracks.

Column Name	Data Type	Constraints	Description
branch_id	INT	PRIMARY KEY, FOREIGN KEY	References Branch(branch_id)
track_id	INT	PRIMARY KEY, FOREIGN KEY	References Track(track_id)
intake	INT	PRIMARY KEY	The intake number for this track

10. Track_Course Table

Defines courses assigned to each track.

Column Name	Data Type	Constraints	Description
track_id	INT	PRIMARY KEY, FOREIGN KEY	References Track(track_id)
course_id	INT	PRIMARY KEY, FOREIGN KEY	References Course(course_id)

11. Student_Course Table

Tracks student enrollment and scores in courses.

Column Name	Data Type	Constraints	Description
student_id	INT	PRIMARY KEY, FOREIGN KEY	References Student(student_id)
course_id	INT	PRIMARY KEY, FOREIGN KEY	References Course(course_id)
score	DECIMAL(5,2)	CHECK (score BETWEEN 0 AND 100)	Student's score

12. Student_Exam_Question Table:

Stores students' answers to exam questions.

Column Name	Data Type	Constraints	Description
exam_id	INT	PRIMARY KEY, FOREIGN KEY	References Exam(exam_id)
student_id	INT	PRIMARY KEY, FOREIGN KEY	References Student(student_id)
question_id	INT	PRIMARY KEY, FOREIGN KEY	References Question(question_id)
answer	TEXT	NOT NULL	Student's submitted answer

13. Exam_Question Table:

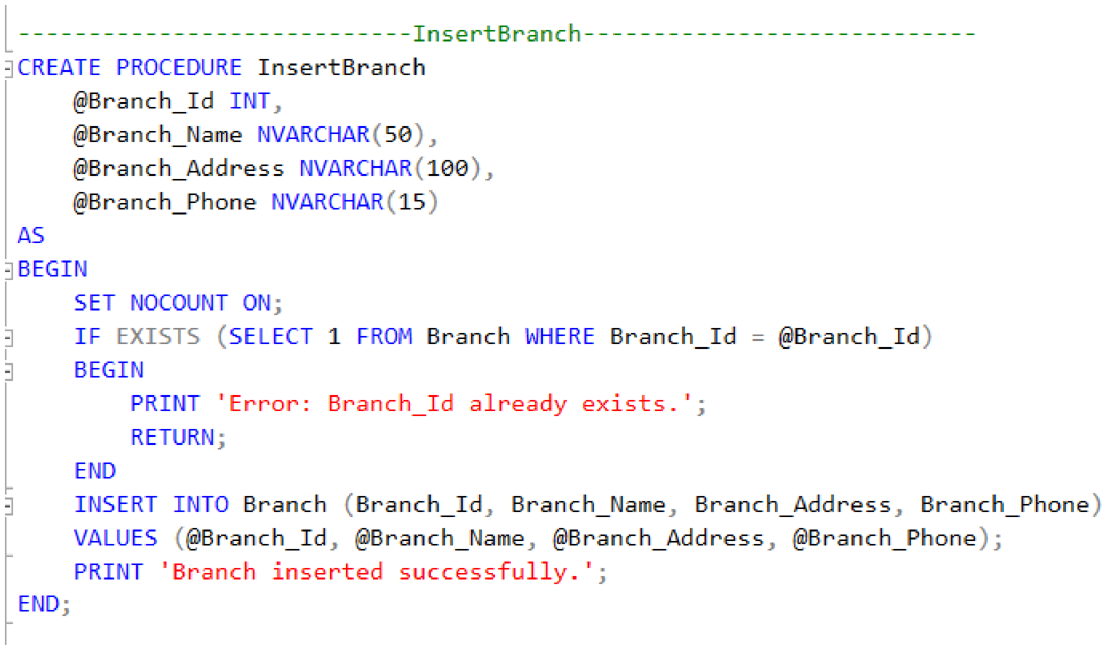
Links exams with their respective questions.

Column Name	Data Type	Constraints	Description
exam_id	INT	PRIMARY KEY, FOREIGN KEY	References Exam(exam_id)
option_id	INT	PRIMARY KEY, FOREIGN KEY	References Option_Table(option_id)

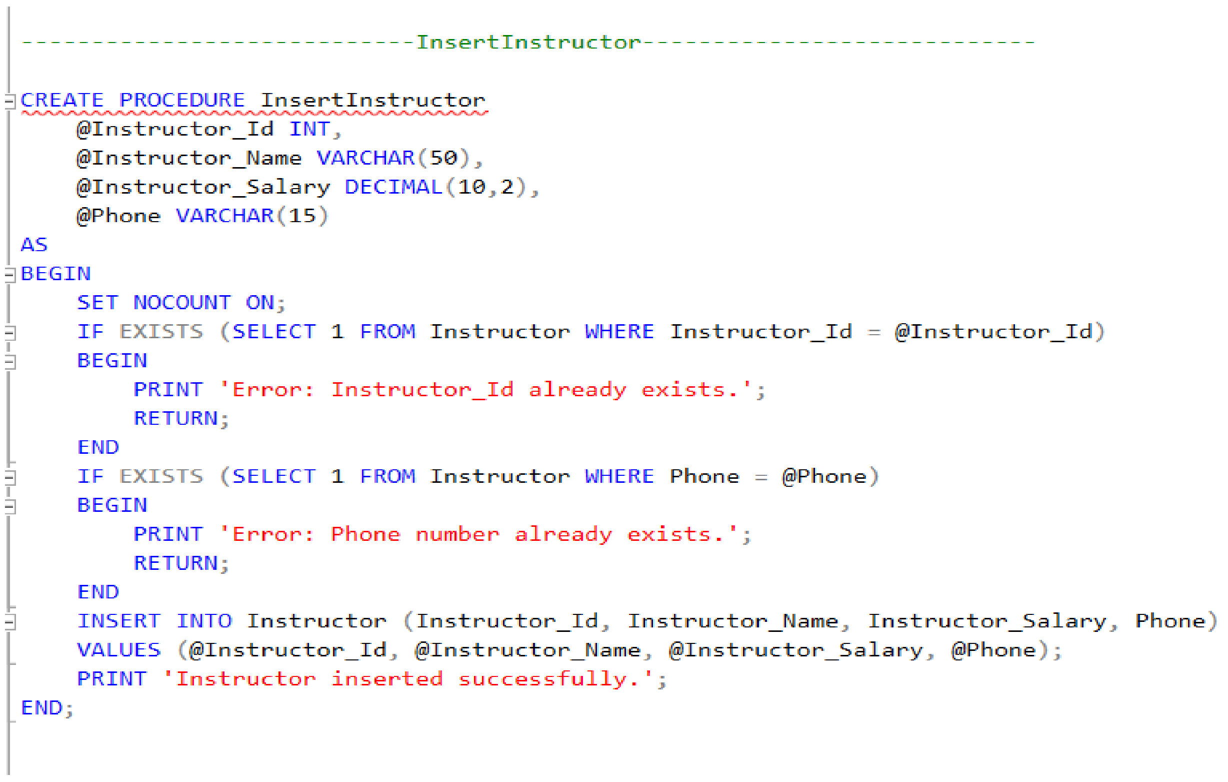
This **Database Dictionary** ensures a well-structured relational database design while maintaining **data integrity** and **efficiency**.

Stored Procedure:

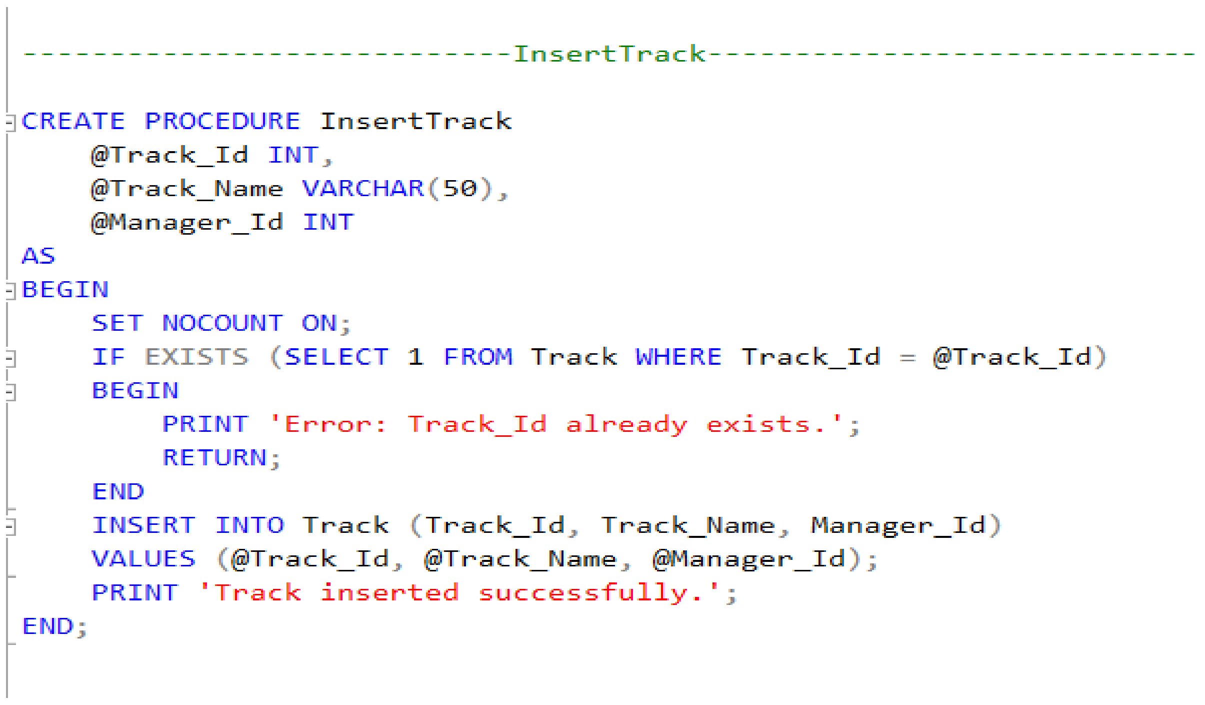
1.Insertion:



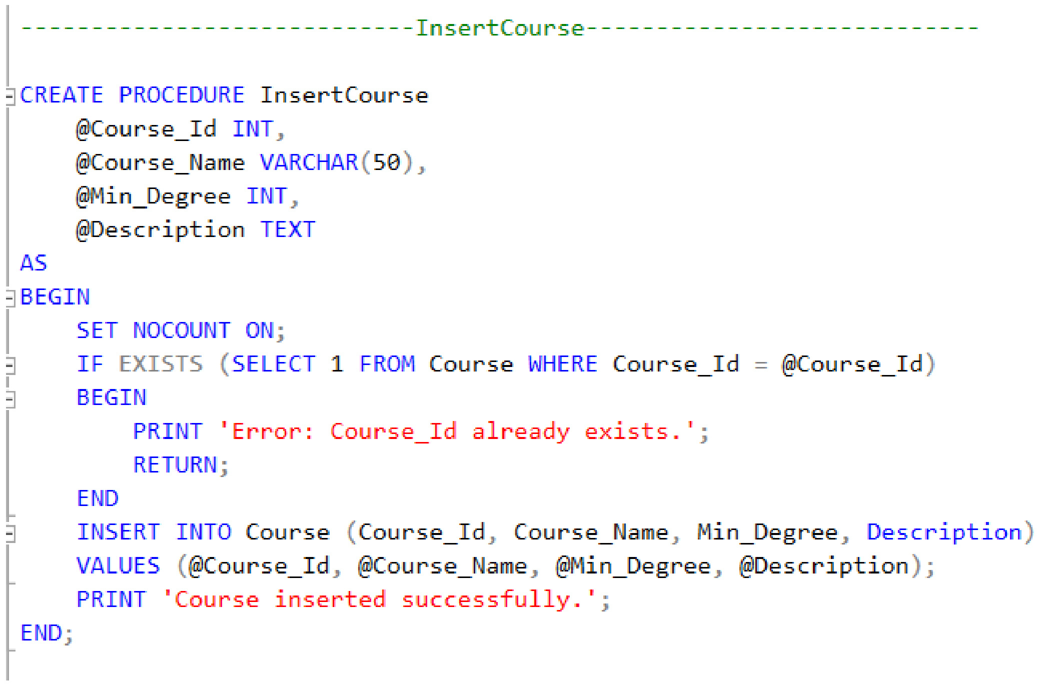
```
----------------------------InsertBranch---------------------------
CREATE PROCEDURE InsertBranch
    @Branch_Id INT,
    @Branch_Name NVARCHAR(50),
    @Branch_Address NVARCHAR(100),
    @Branch_Phone NVARCHAR(15)
AS
BEGIN
    SET NOCOUNT ON;
    IF EXISTS (SELECT 1 FROM Branch WHERE Branch_Id = @Branch_Id)
    BEGIN
        PRINT 'Error: Branch_Id already exists.';
        RETURN;
    END
    INSERT INTO Branch (Branch_Id, Branch_Name, Branch_Address, Branch_Phone)
    VALUES (@Branch_Id, @Branch_Name, @Branch_Address, @Branch_Phone);
    PRINT 'Branch inserted successfully.';
END;
```



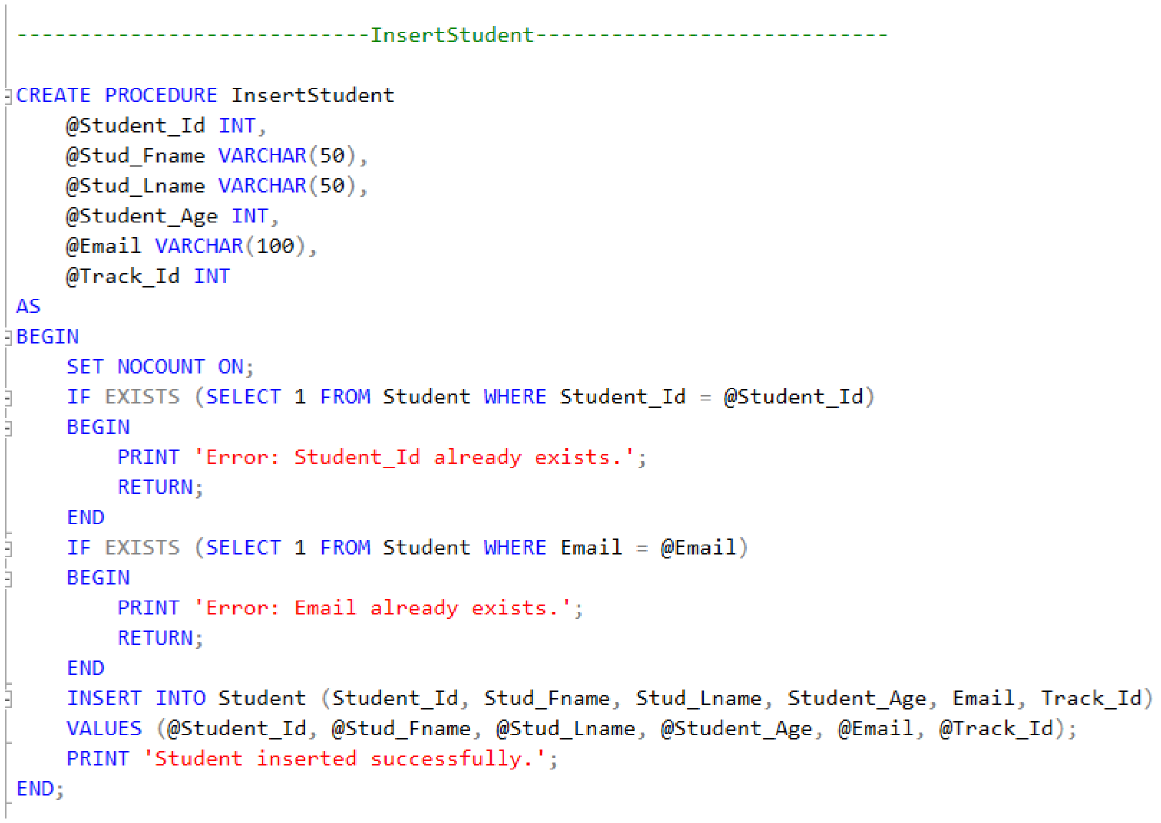
```
----------------------------InsertInstructor----------------------------

CREATE PROCEDURE InsertInstructor
    @Instructor_Id INT,
    @Instructor_Name VARCHAR(50),
    @Instructor_Salary DECIMAL(10,2),
    @Phone VARCHAR(15)
AS
BEGIN
    SET NOCOUNT ON;
    IF EXISTS (SELECT 1 FROM Instructor WHERE Instructor_Id = @Instructor_Id)
    BEGIN
        PRINT 'Error: Instructor_Id already exists.';
        RETURN;
    END
    IF EXISTS (SELECT 1 FROM Instructor WHERE Phone = @Phone)
    BEGIN
        PRINT 'Error: Phone number already exists.';
        RETURN;
    END
    INSERT INTO Instructor (Instructor_Id, Instructor_Name, Instructor_Salary, Phone)
    VALUES (@Instructor_Id, @Instructor_Name, @Instructor_Salary, @Phone);
    PRINT 'Instructor inserted successfully.';
END;
```



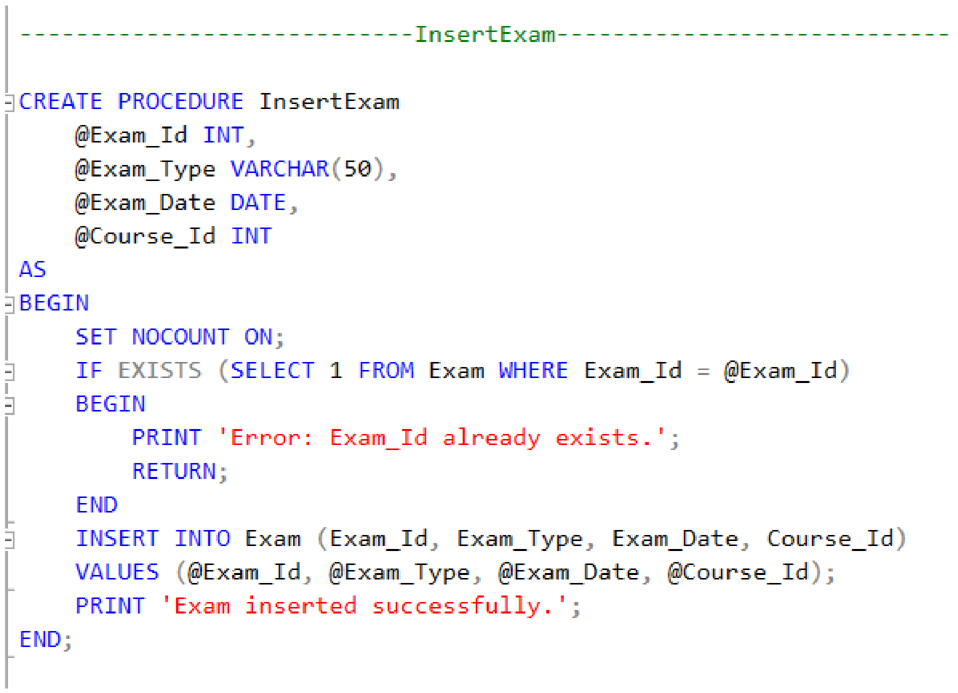
```
----------------------------InsertTrack----------------------------

CREATE PROCEDURE InsertTrack
    @Track_Id INT,
    @Track_Name VARCHAR(50),
    @Manager_Id INT
AS
BEGIN
    SET NOCOUNT ON;
    IF EXISTS (SELECT 1 FROM Track WHERE Track_Id = @Track_Id)
    BEGIN
        PRINT 'Error: Track_Id already exists.';
        RETURN;
    END
    INSERT INTO Track (Track_Id, Track_Name, Manager_Id)
    VALUES (@Track_Id, @Track_Name, @Manager_Id);
    PRINT 'Track inserted successfully.';
END;
```



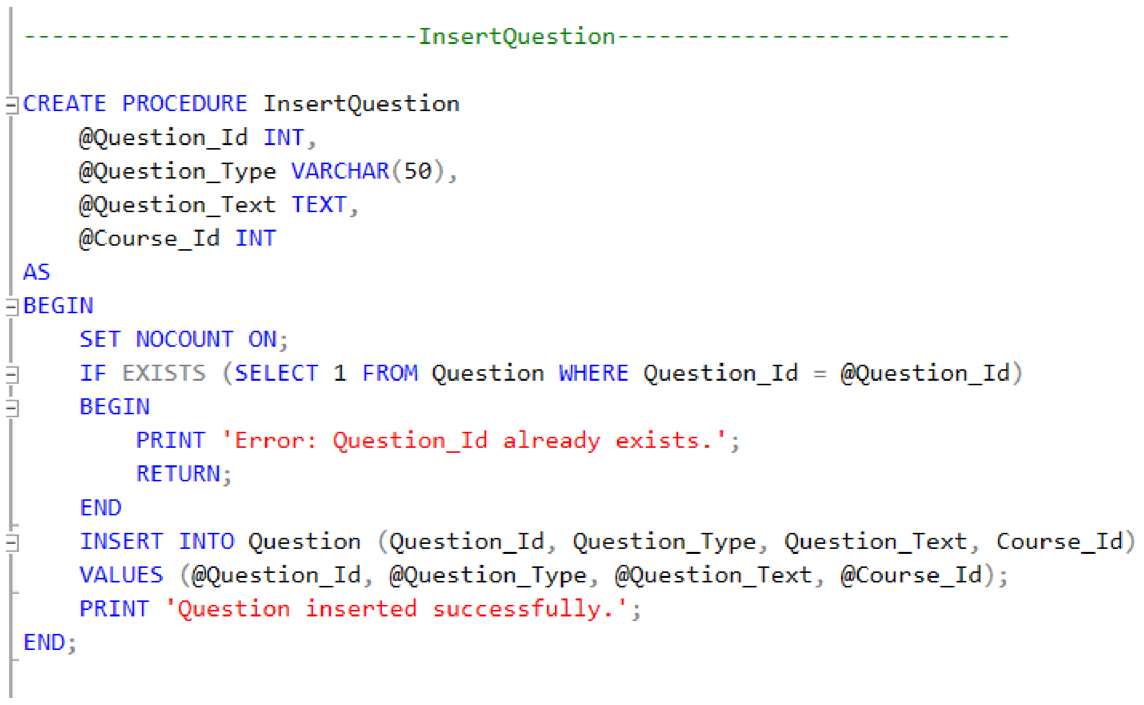
```
----------------------------InsertCourse----------------------------

CREATE PROCEDURE InsertCourse
    @Course_Id INT,
    @Course_Name VARCHAR(50),
    @Min_Degree INT,
    @Description TEXT
AS
BEGIN
    SET NOCOUNT ON;
    IF EXISTS (SELECT 1 FROM Course WHERE Course_Id = @Course_Id)
    BEGIN
        PRINT 'Error: Course_Id already exists.';
        RETURN;
    END
    INSERT INTO Course (Course_Id, Course_Name, Min_Degree, Description)
    VALUES (@Course_Id, @Course_Name, @Min_Degree, @Description);
    PRINT 'Course inserted successfully.';
END;
```



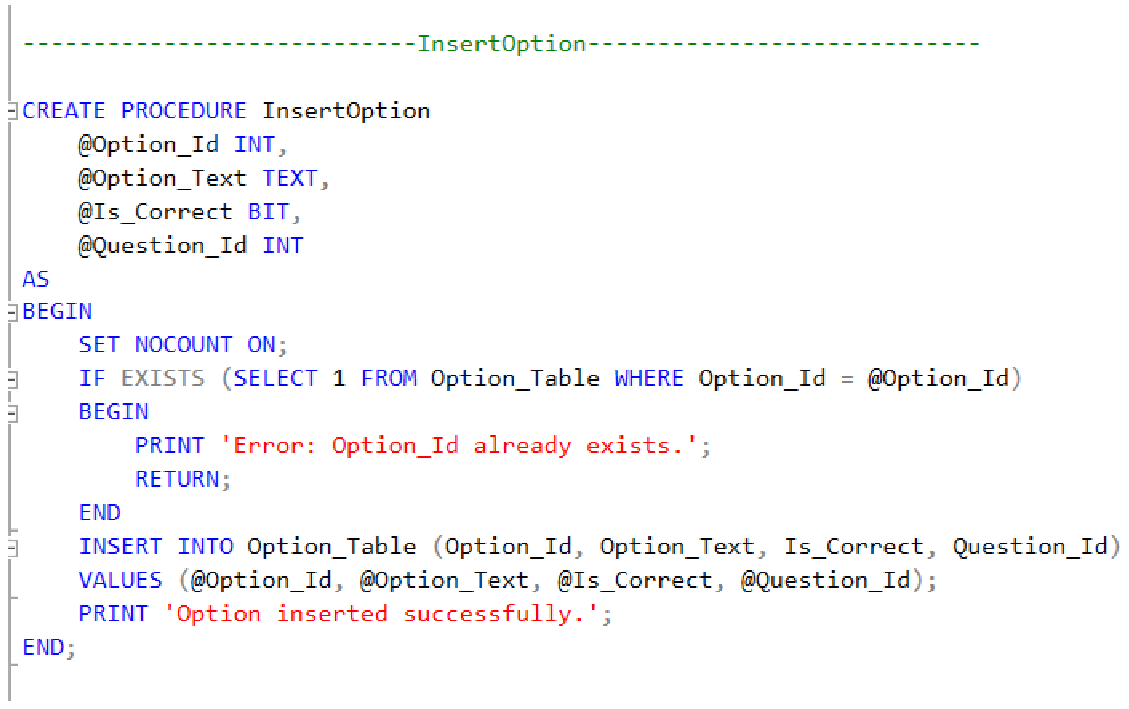
```sql
---------------------------InsertStudent---------------------------

CREATE PROCEDURE InsertStudent
    @Student_Id INT,
    @Stud_Fname VARCHAR(50),
    @Stud_Lname VARCHAR(50),
    @Student_Age INT,
    @Email VARCHAR(100),
    @Track_Id INT
AS
BEGIN
    SET NOCOUNT ON;
    IF EXISTS (SELECT 1 FROM Student WHERE Student_Id = @Student_Id)
    BEGIN
        PRINT 'Error: Student_Id already exists.';
        RETURN;
    END
    IF EXISTS (SELECT 1 FROM Student WHERE Email = @Email)
    BEGIN
        PRINT 'Error: Email already exists.';
        RETURN;
    END
    INSERT INTO Student (Student_Id, Stud_Fname, Stud_Lname, Student_Age, Email, Track_Id)
    VALUES (@Student_Id, @Stud_Fname, @Stud_Lname, @Student_Age, @Email, @Track_Id);
    PRINT 'Student inserted successfully.';
END;
```



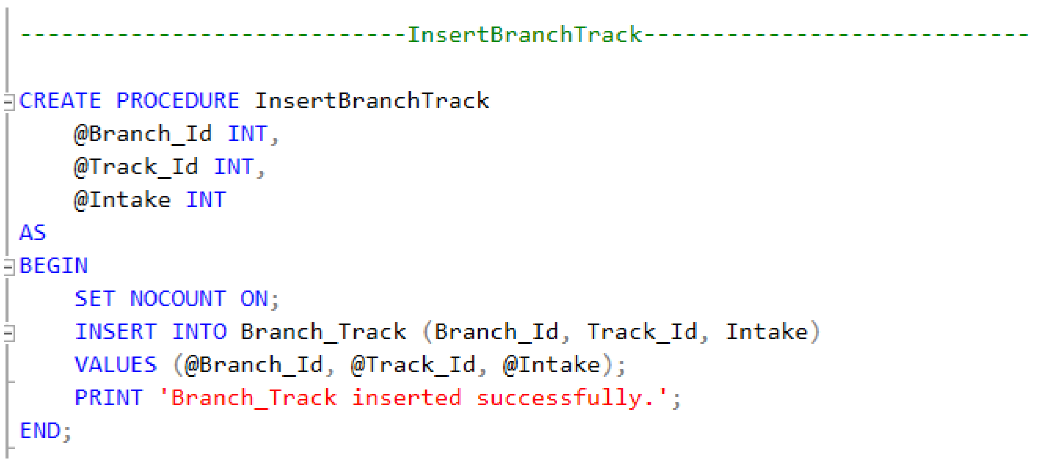
```sql
----------------------------InsertExam----------------------------

CREATE PROCEDURE InsertExam
    @Exam_Id INT,
    @Exam_Type VARCHAR(50),
    @Exam_Date DATE,
    @Course_Id INT
AS
BEGIN
    SET NOCOUNT ON;
    IF EXISTS (SELECT 1 FROM Exam WHERE Exam_Id = @Exam_Id)
    BEGIN
        PRINT 'Error: Exam_Id already exists.';
        RETURN;
    END
    INSERT INTO Exam (Exam_Id, Exam_Type, Exam_Date, Course_Id)
    VALUES (@Exam_Id, @Exam_Type, @Exam_Date, @Course_Id);
    PRINT 'Exam inserted successfully.';
END;
```



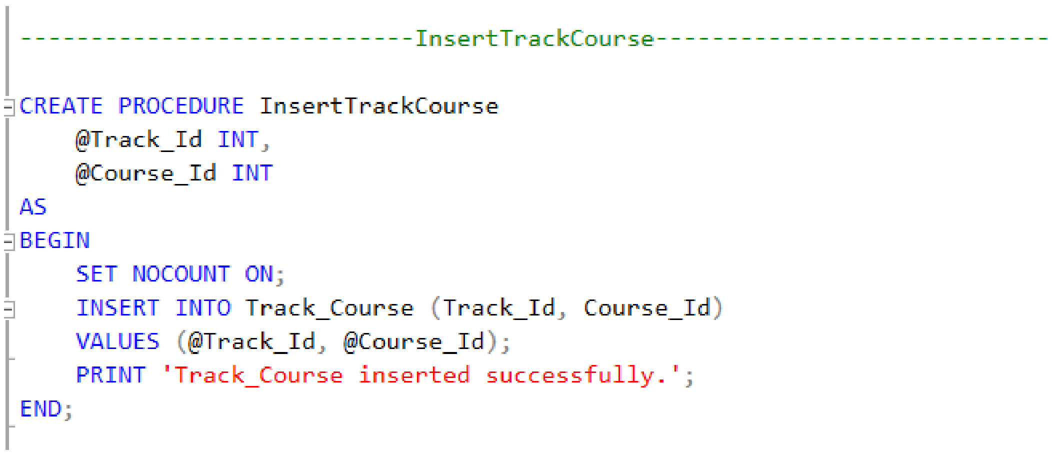
```sql
-----------------------------InsertQuestion----------------------------

CREATE PROCEDURE InsertQuestion
    @Question_Id INT,
    @Question_Type VARCHAR(50),
    @Question_Text TEXT,
    @Course_Id INT
AS
BEGIN
    SET NOCOUNT ON;
    IF EXISTS (SELECT 1 FROM Question WHERE Question_Id = @Question_Id)
    BEGIN
        PRINT 'Error: Question_Id already exists.';
        RETURN;
    END
    INSERT INTO Question (Question_Id, Question_Type, Question_Text, Course_Id)
    VALUES (@Question_Id, @Question_Type, @Question_Text, @Course_Id);
    PRINT 'Question inserted successfully.';
END;
```



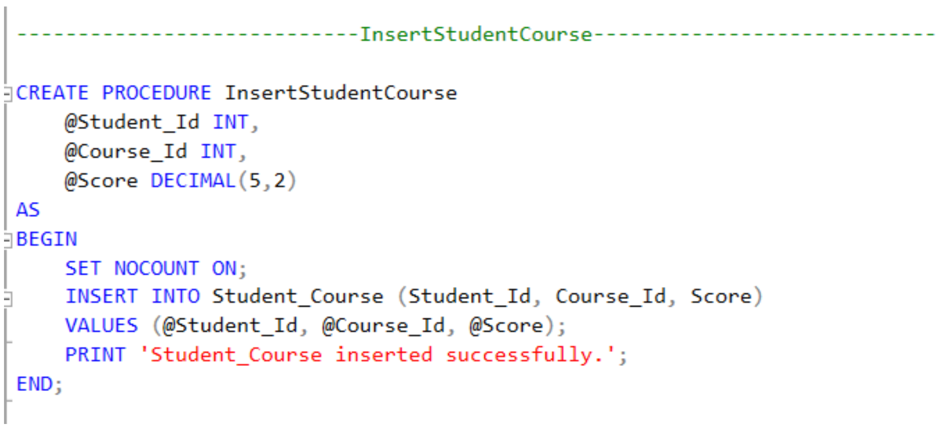
```sql
-----------------------------InsertOption----------------------------

CREATE PROCEDURE InsertOption
    @Option_Id INT,
    @Option_Text TEXT,
    @Is_Correct BIT,
    @Question_Id INT
AS
BEGIN
    SET NOCOUNT ON;
    IF EXISTS (SELECT 1 FROM Option_Table WHERE Option_Id = @Option_Id)
    BEGIN
        PRINT 'Error: Option_Id already exists.';
        RETURN;
    END
    INSERT INTO Option_Table (Option_Id, Option_Text, Is_Correct, Question_Id)
    VALUES (@Option_Id, @Option_Text, @Is_Correct, @Question_Id);
    PRINT 'Option inserted successfully.';
END;
```



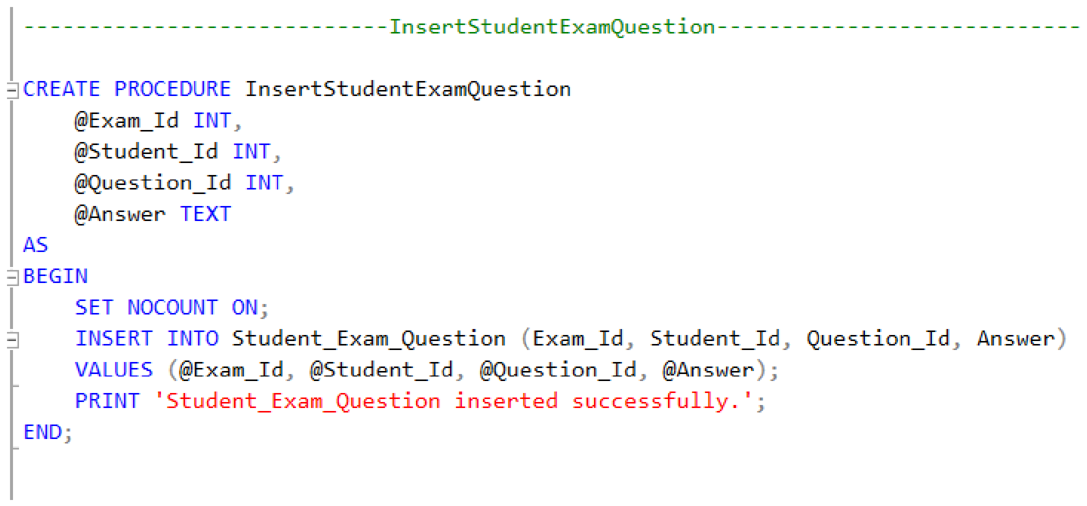
```sql
----------------------------InsertBranchTrack----------------------------

CREATE PROCEDURE InsertBranchTrack
    @Branch_Id INT,
    @Track_Id INT,
    @Intake INT
AS
BEGIN
    SET NOCOUNT ON;
    INSERT INTO Branch_Track (Branch_Id, Track_Id, Intake)
    VALUES (@Branch_Id, @Track_Id, @Intake);
    PRINT 'Branch_Track inserted successfully.';
END;
```



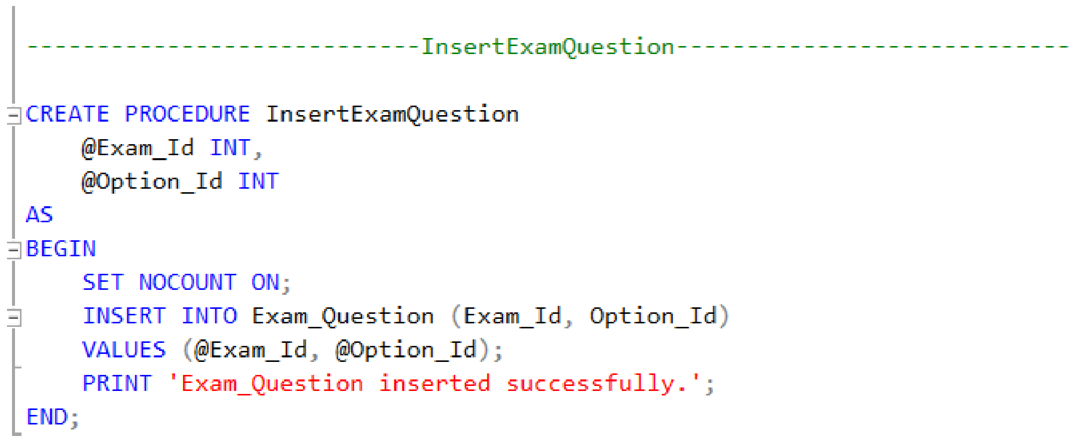
```sql
----------------------------InsertTrackCourse----------------------------

CREATE PROCEDURE InsertTrackCourse
    @Track_Id INT,
    @Course_Id INT
AS
BEGIN
    SET NOCOUNT ON;
    INSERT INTO Track_Course (Track_Id, Course_Id)
    VALUES (@Track_Id, @Course_Id);
    PRINT 'Track_Course inserted successfully.';
END;
```



```sql
---------------------------InsertStudentCourse---------------------------

CREATE PROCEDURE InsertStudentCourse
    @Student_Id INT,
    @Course_Id INT,
    @Score DECIMAL(5,2)
AS
BEGIN
    SET NOCOUNT ON;
    INSERT INTO Student_Course (Student_Id, Course_Id, Score)
    VALUES (@Student_Id, @Course_Id, @Score);
    PRINT 'Student_Course inserted successfully.';
END;
```

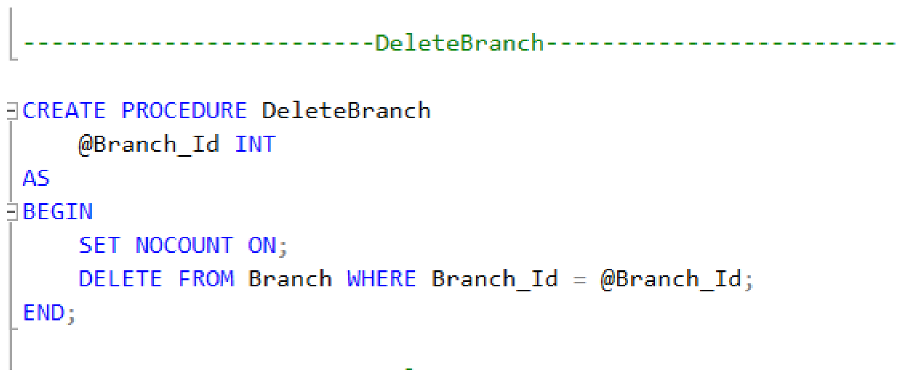


```sql
----------------------------InsertStudentExamQuestion----------------------------

CREATE PROCEDURE InsertStudentExamQuestion
    @Exam_Id INT,
    @Student_Id INT,
    @Question_Id INT,
    @Answer TEXT
AS
BEGIN
    SET NOCOUNT ON;
    INSERT INTO Student_Exam_Question (Exam_Id, Student_Id, Question_Id, Answer)
    VALUES (@Exam_Id, @Student_Id, @Question_Id, @Answer);
    PRINT 'Student_Exam_Question inserted successfully.';
END;
```

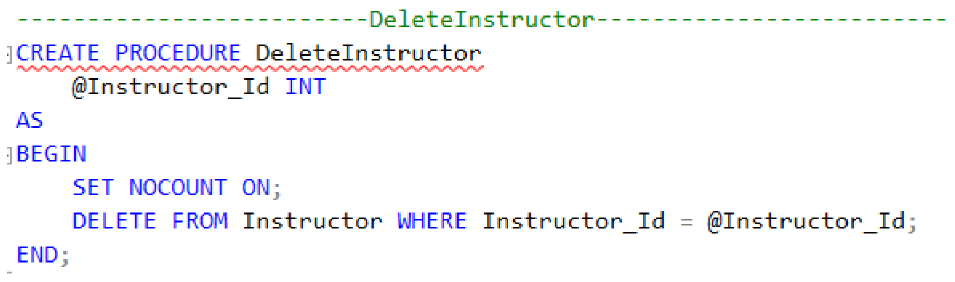


```sql
----------------------------InsertExamQuestion----------------------------

CREATE PROCEDURE InsertExamQuestion
    @Exam_Id INT,
    @Option_Id INT
AS
BEGIN
    SET NOCOUNT ON;
    INSERT INTO Exam_Question (Exam_Id, Option_Id)
    VALUES (@Exam_Id, @Option_Id);
    PRINT 'Exam_Question inserted successfully.';
END;
```

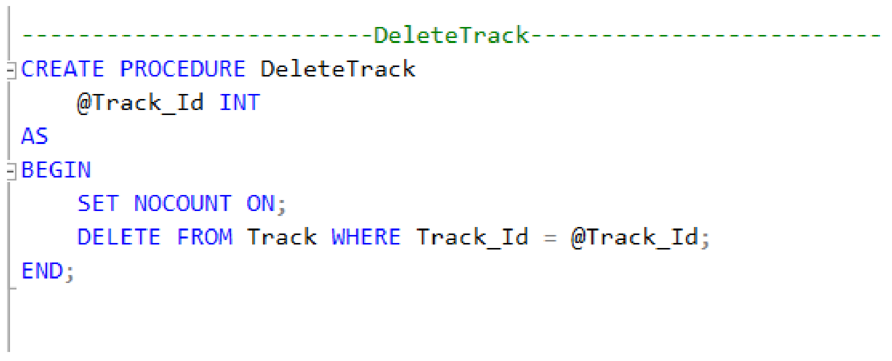
2.Deletion:



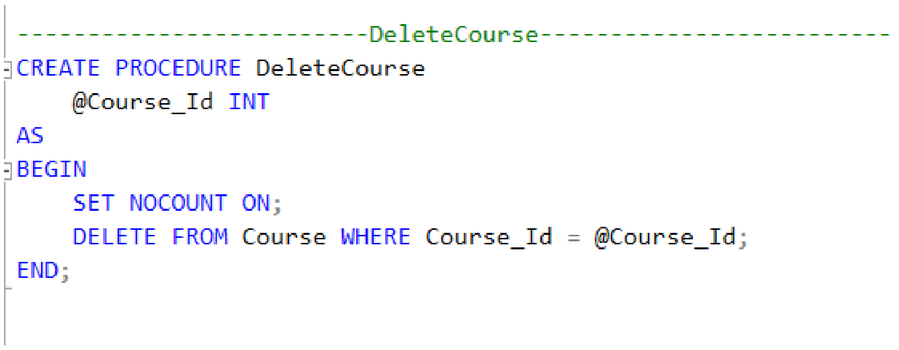
```
-------------------------DeleteBranch------------------------

CREATE PROCEDURE DeleteBranch
    @Branch_Id INT
AS
BEGIN
    SET NOCOUNT ON;
    DELETE FROM Branch WHERE Branch_Id = @Branch_Id;
END;
```



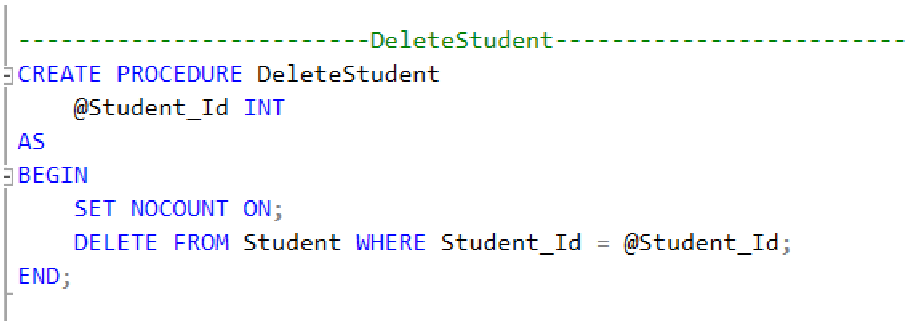
```
-------------------------DeleteInstructor-------------------------
CREATE PROCEDURE DeleteInstructor
    @Instructor_Id INT
AS
BEGIN
    SET NOCOUNT ON;
    DELETE FROM Instructor WHERE Instructor_Id = @Instructor_Id;
END;
```



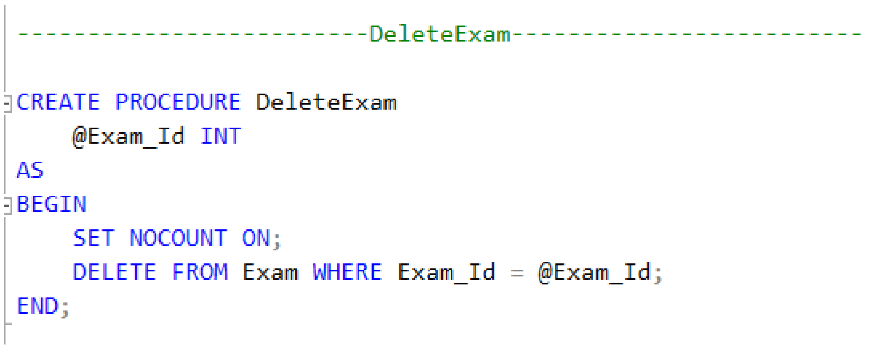
```
-------------------------DeleteTrack------------------------
CREATE PROCEDURE DeleteTrack
    @Track_Id INT
AS
BEGIN
    SET NOCOUNT ON;
    DELETE FROM Track WHERE Track_Id = @Track_Id;
END;
```



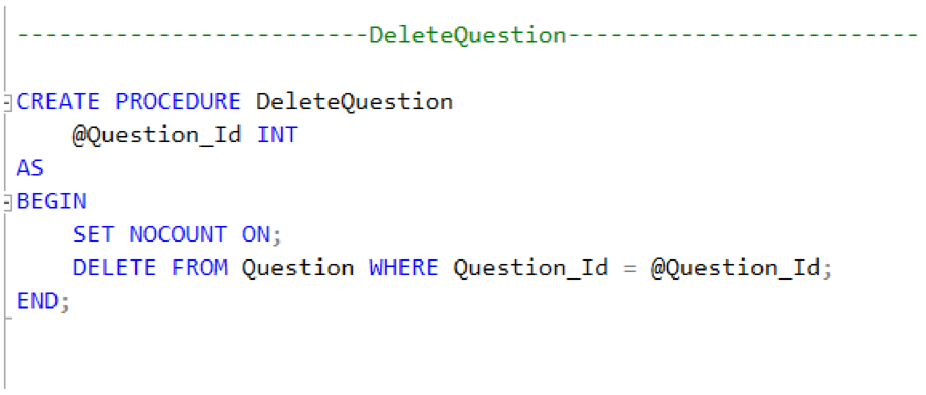
```sql
-------------------------DeleteCourse-------------------------
CREATE PROCEDURE DeleteCourse
    @Course_Id INT
AS
BEGIN
    SET NOCOUNT ON;
    DELETE FROM Course WHERE Course_Id = @Course_Id;
END;
```



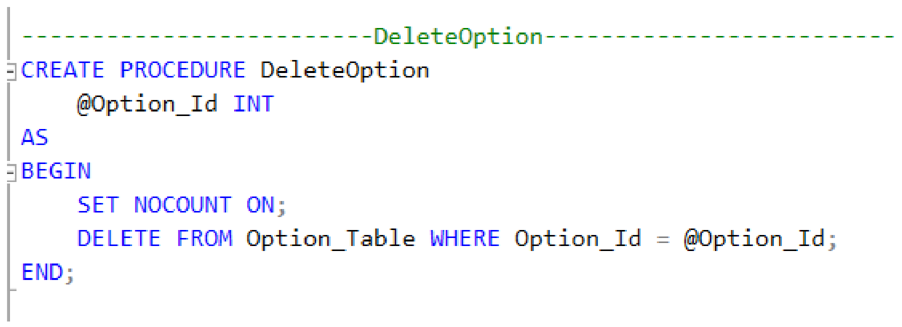
```sql
-------------------------DeleteStudent-------------------------
CREATE PROCEDURE DeleteStudent
    @Student_Id INT
AS
BEGIN
    SET NOCOUNT ON;
    DELETE FROM Student WHERE Student_Id = @Student_Id;
END;
```



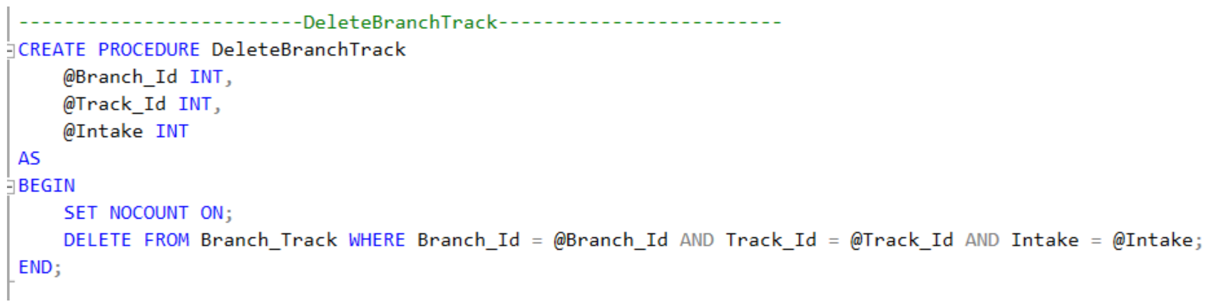
```sql
-------------------------DeleteExam-------------------------

CREATE PROCEDURE DeleteExam
    @Exam_Id INT
AS
BEGIN
    SET NOCOUNT ON;
    DELETE FROM Exam WHERE Exam_Id = @Exam_Id;
END;
```



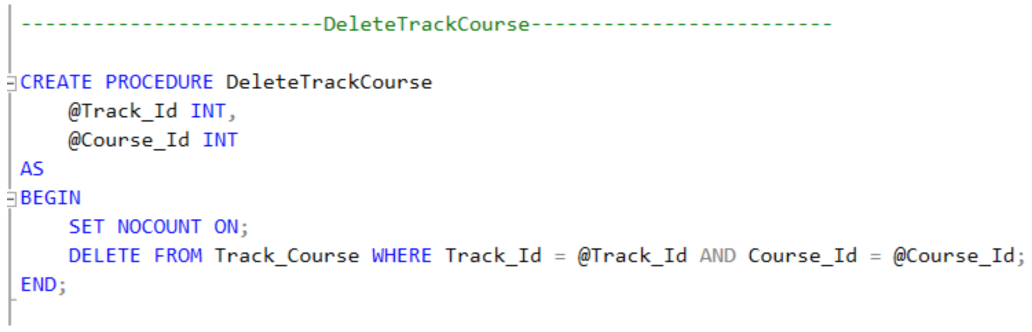
```sql
-------------------------DeleteQuestion-------------------------

CREATE PROCEDURE DeleteQuestion
    @Question_Id INT
AS
BEGIN
    SET NOCOUNT ON;
    DELETE FROM Question WHERE Question_Id = @Question_Id;
END;
```



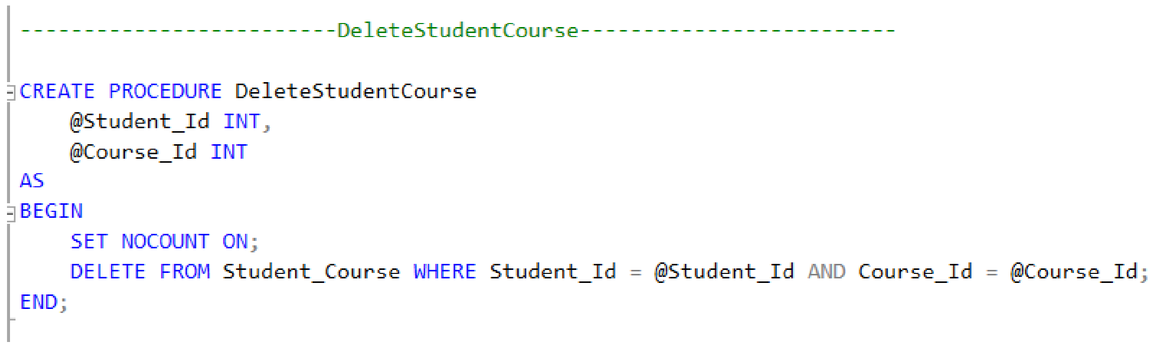
```sql
-------------------------DeleteOption------------------------
CREATE PROCEDURE DeleteOption
    @Option_Id INT
AS
BEGIN
    SET NOCOUNT ON;
    DELETE FROM Option_Table WHERE Option_Id = @Option_Id;
END;
```



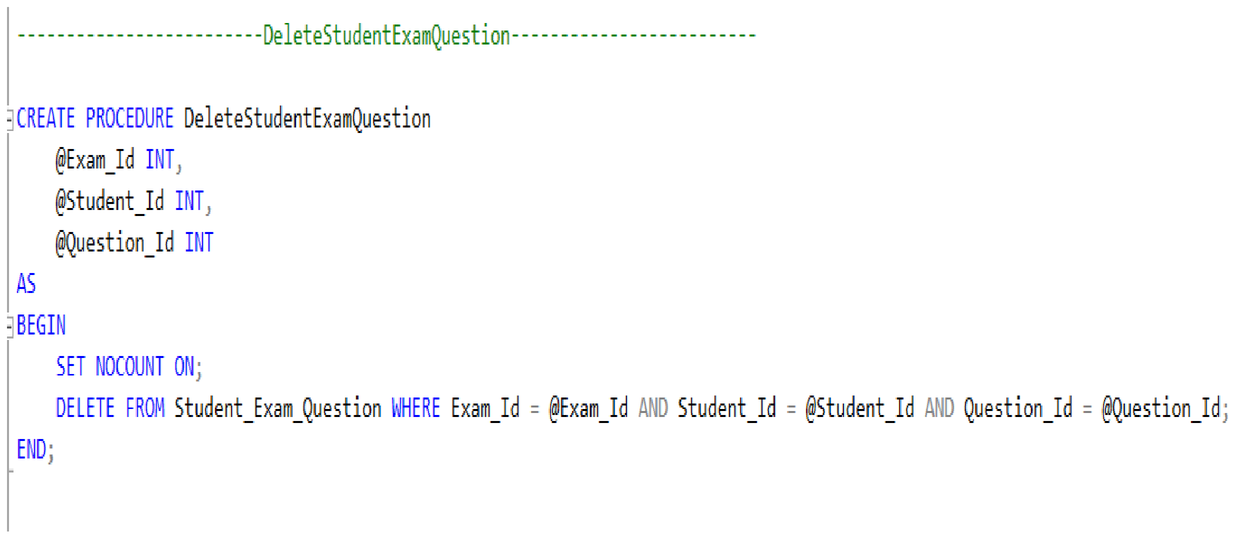
```sql
------------------------DeleteBranchTrack------------------------
CREATE PROCEDURE DeleteBranchTrack
    @Branch_Id INT,
    @Track_Id INT,
    @Intake INT
AS
BEGIN
    SET NOCOUNT ON;
    DELETE FROM Branch_Track WHERE Branch_Id = @Branch_Id AND Track_Id = @Track_Id AND Intake = @Intake;
END;
```



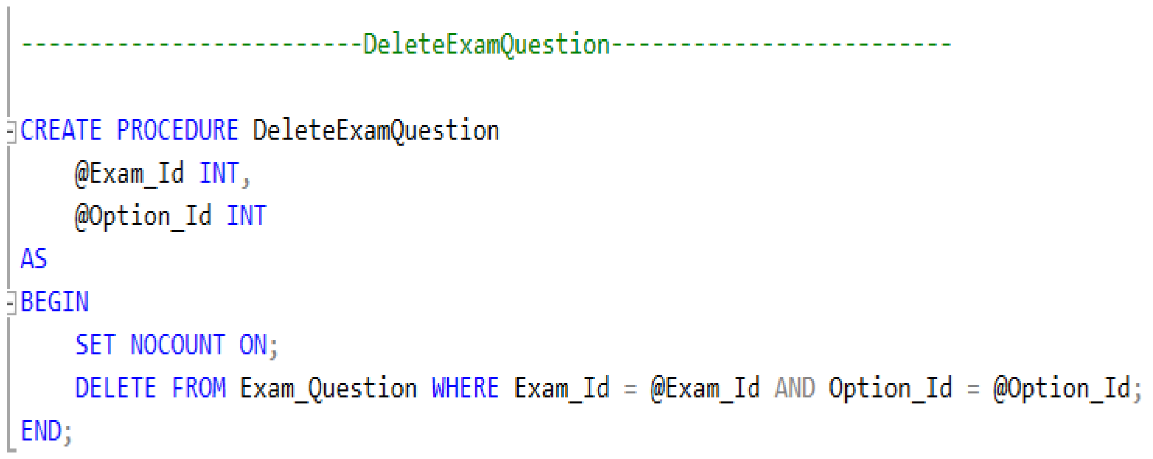
```sql
------------------------DeleteTrackCourse------------------------

CREATE PROCEDURE DeleteTrackCourse
    @Track_Id INT,
    @Course_Id INT
AS
BEGIN
    SET NOCOUNT ON;
    DELETE FROM Track_Course WHERE Track_Id = @Track_Id AND Course_Id = @Course_Id;
END;
```



```sql
-------------------------DeleteStudentCourse-------------------------

CREATE PROCEDURE DeleteStudentCourse
    @Student_Id INT,
    @Course_Id INT
AS
BEGIN
    SET NOCOUNT ON;
    DELETE FROM Student_Course WHERE Student_Id = @Student_Id AND Course_Id = @Course_Id;
END;
```

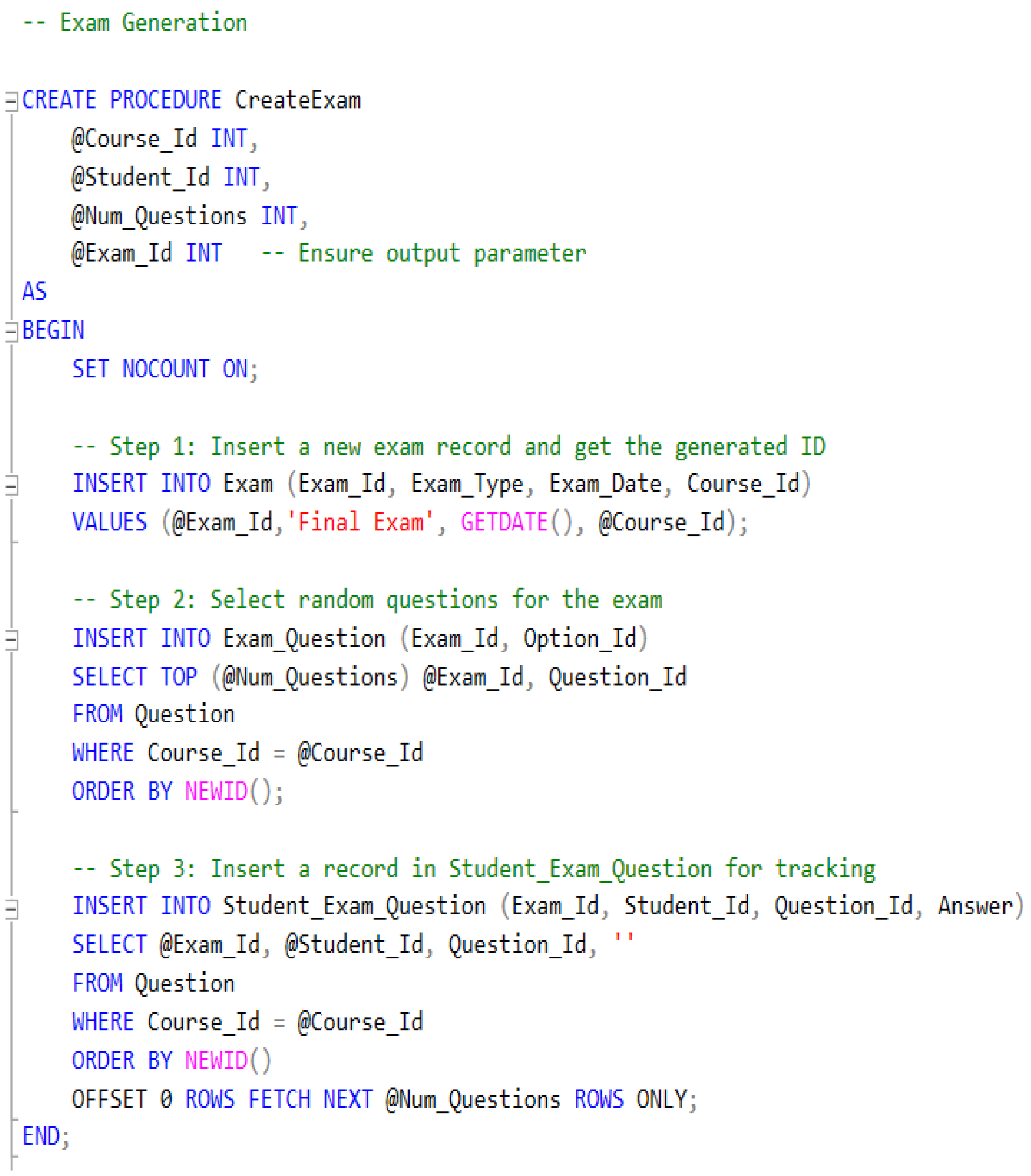


```
------------------------DeleteStudentExamQuestion------------------------

CREATE PROCEDURE DeleteStudentExamQuestion
    @Exam_Id INT,
    @Student_Id INT,
    @Question_Id INT
AS
BEGIN
    SET NOCOUNT ON;
    DELETE FROM Student_Exam_Question WHERE Exam_Id = @Exam_Id AND Student_Id = @Student_Id AND Question_Id = @Question_Id;
END;
```



```
-------------------------DeleteExamQuestion-------------------------

CREATE PROCEDURE DeleteExamQuestion
    @Exam_Id INT,
    @Option_Id INT
AS
BEGIN
    SET NOCOUNT ON;
    DELETE FROM Exam_Question WHERE Exam_Id = @Exam_Id AND Option_Id = @Option_Id;
END;
```

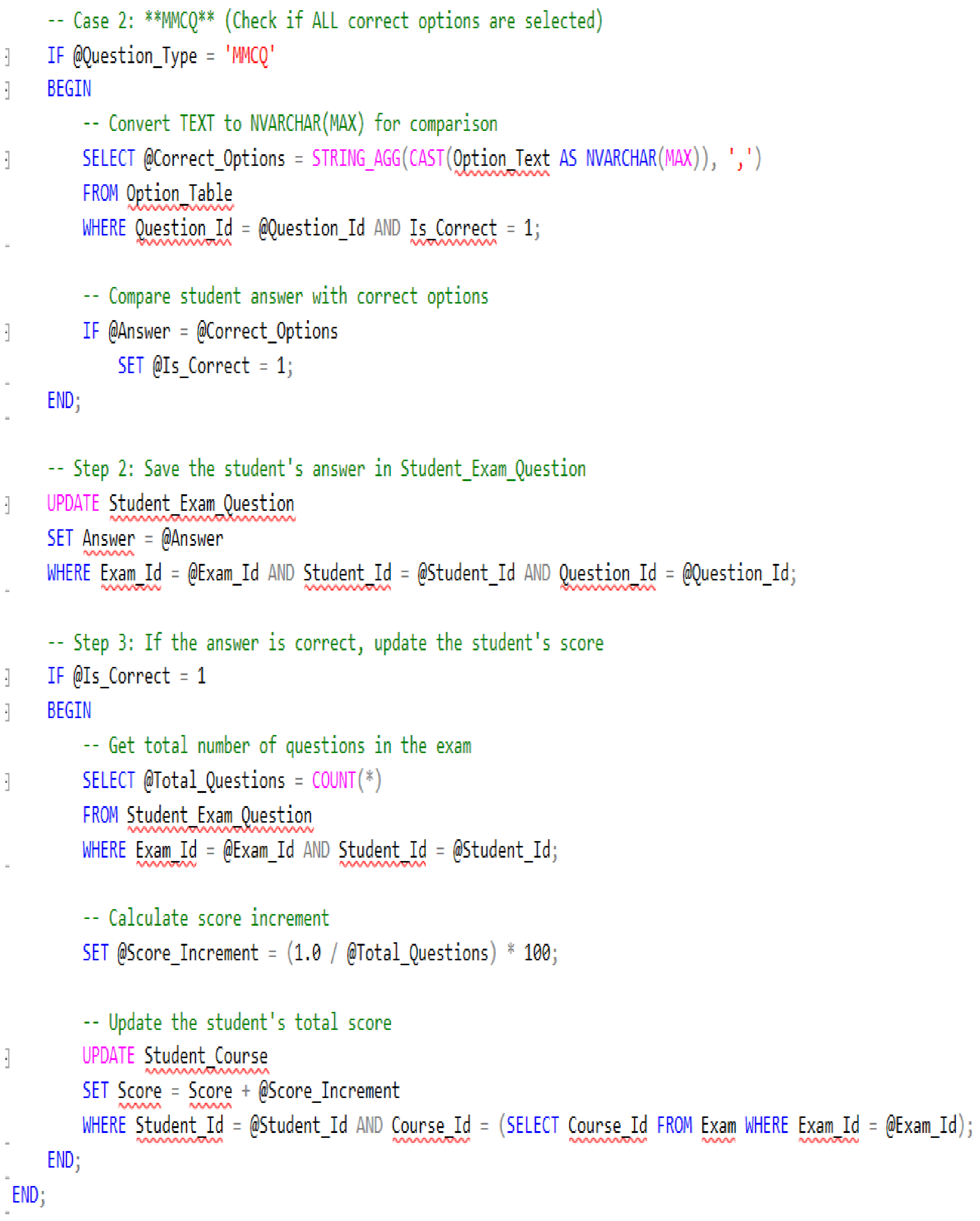
3.Exam Generation:



```sql
-- Exam Generation

CREATE PROCEDURE CreateExam
    @Course_Id INT,
    @Student_Id INT,
    @Num_Questions INT,
    @Exam_Id INT   -- Ensure output parameter
AS
BEGIN
    SET NOCOUNT ON;

    -- Step 1: Insert a new exam record and get the generated ID
    INSERT INTO Exam (Exam_Id, Exam_Type, Exam_Date, Course_Id)
    VALUES (@Exam_Id,'Final Exam', GETDATE(), @Course_Id);

    -- Step 2: Select random questions for the exam
    INSERT INTO Exam_Question (Exam_Id, Option_Id)
    SELECT TOP (@Num_Questions) @Exam_Id, Question_Id
    FROM Question
    WHERE Course_Id = @Course_Id
    ORDER BY NEWID();

    -- Step 3: Insert a record in Student_Exam_Question for tracking
    INSERT INTO Student_Exam_Question (Exam_Id, Student_Id, Question_Id, Answer)
    SELECT @Exam_Id, @Student_Id, Question_Id, ''
    FROM Question
    WHERE Course_Id = @Course_Id
    ORDER BY NEWID()
    OFFSET 0 ROWS FETCH NEXT @Num_Questions ROWS ONLY;
END;
```

4.Exam Correction:

```
-- submit question

-- If MMCQ, answers should be comma-separated (e.g., "A,B,D")

CREATE PROCEDURE CheckStudentAnswer
    @Student_Id INT,
    @Exam_Id INT,
    @Question_Id INT,
    @Answer NVARCHAR(MAX)  -- Changed to NVARCHAR(MAX) instead of TEXT
AS
BEGIN
    DECLARE @Question_Type VARCHAR(10);
    DECLARE @Is_Correct BIT = 0;
    DECLARE @Total_Questions INT;
    DECLARE @Score_Increment DECIMAL(5,2);
    DECLARE @Correct_Options NVARCHAR(MAX);

    -- Get the question type (MCQ, MMCQ, T/F)
    SELECT @Question_Type = Question_Type
    FROM Question
    WHERE Question_Id = @Question_Id;

    -- Case 1: **MCQ / T/F** (Check if the answer is correct)
    IF @Question_Type IN ('MCQ', 'T/F')
    BEGIN
        SELECT @Is_Correct = CASE
            WHEN EXISTS (
                SELECT 1 FROM Option_Table
                WHERE Question_Id = @Question_Id AND Option_Text = @Answer AND Is_Correct = 1
            )
            THEN 1 ELSE 0
        END;
    END;
```



```sql
    -- Case 2: **MMCQ** (Check if ALL correct options are selected)
    IF @Question_Type = 'MMCQ'
    BEGIN
        -- Convert TEXT to NVARCHAR(MAX) for comparison
        SELECT @Correct_Options = STRING_AGG(CAST(Option_Text AS NVARCHAR(MAX)), ',')
        FROM Option_Table
        WHERE Question_Id = @Question_Id AND Is_Correct = 1;

        -- Compare student answer with correct options
        IF @Answer = @Correct_Options
            SET @Is_Correct = 1;
    END;

    -- Step 2: Save the student's answer in Student_Exam_Question
    UPDATE Student_Exam_Question
    SET Answer = @Answer
    WHERE Exam_Id = @Exam_Id AND Student_Id = @Student_Id AND Question_Id = @Question_Id;

    -- Step 3: If the answer is correct, update the student's score
    IF @Is_Correct = 1
    BEGIN
        -- Get total number of questions in the exam
        SELECT @Total_Questions = COUNT(*)
        FROM Student_Exam_Question
        WHERE Exam_Id = @Exam_Id AND Student_Id = @Student_Id;

        -- Calculate score increment
        SET @Score_Increment = (1.0 / @Total_Questions) * 100;

        -- Update the student's total score
        UPDATE Student_Course
        SET Score = Score + @Score_Increment
        WHERE Student_Id = @Student_Id AND Course_Id = (SELECT Course_Id FROM Exam WHERE Exam_Id = @Exam_Id);
    END;
END;
```

Views:

```sql
-- Passed_Students
CREATE VIEW Passed_Students  AS
    SELECT CONCAT(Stud_Fname , Stud_Lname) AS 'Student Name'  , Course.Course_Name
    FROM Student
    INNER JOIN Student_Course ON Student.Student_Id = Student_Course.Student_Id
    INNER JOIN Course ON Course.Course_Id = Student_Course.Course_Id
    WHERE Score >= Course.Min_Degree;
```

```sql
-- Failed Students
CREATE VIEW Failed_Students  AS
    SELECT CONCAT(Stud_Fname , Stud_Lname) AS 'Student Name'  , Course.Course_Name
    FROM Student
    INNER JOIN Student_Course ON Student.Student_Id = Student_Course.Student_Id
    INNER JOIN Course ON Course.Course_Id = Student_Course.Course_Id
    WHERE Score <  Course.Min_Degree;
```

```sql
-- Get All managers
CREATE VIEW Tracks_Managers
    AS
    SELECT Instructor.Instructor_Name AS 'Manager Name'  , Track.Track_Name AS 'Track Name'
    FROM Instructor
    INNER JOIN Track ON Manager_Id = Instructor_Id;

SELECT * FROM Tracks_Managers;

CREATE VIEW Student_Total_Courses AS
    SELECT  CONCAT(Stud_Fname ,' ' ,Stud_Lname) AS 'Student_Name' , COUNT(Course_Id) AS 'Total_Courses'
    FROM Student
    INNER JOIN Student_Course ON Student.Student_Id = Student_Course.Student_Id
    GROUP BY CONCAT(Stud_Fname ,' ' ,Stud_Lname);

SELECT * FROM Student_Total_Courses;
```

Reports:

1. **Show_Track_Students**

   Description:

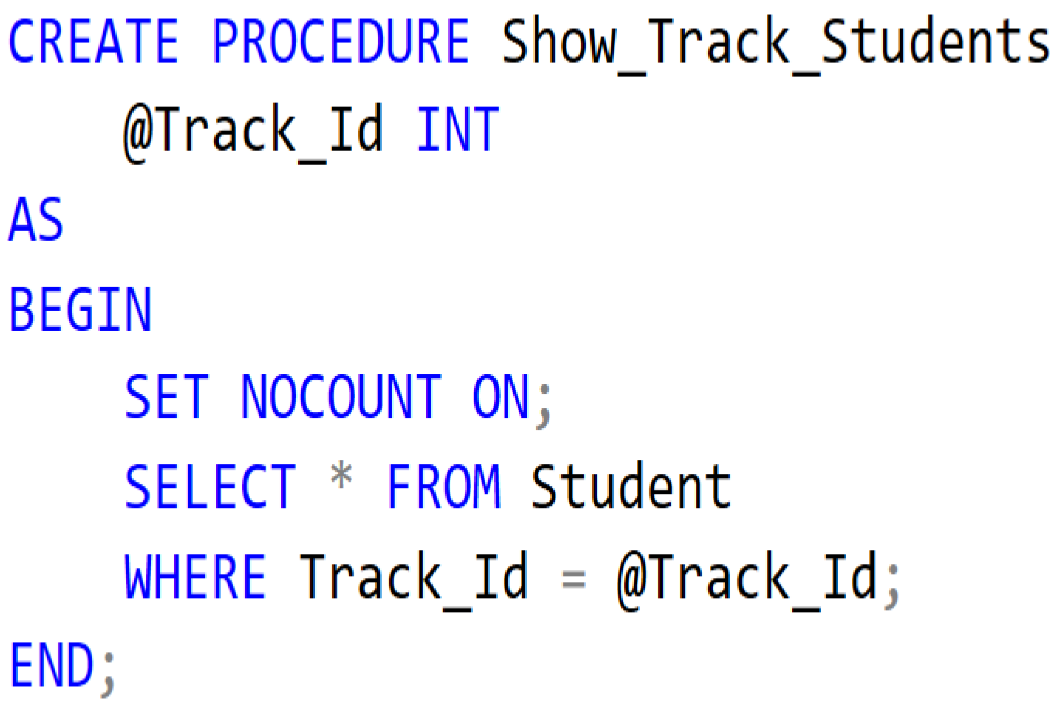
   This stored procedure retrieves all students enrolled in a specific track. It takes a Track_Id as input and returns a list of all students who belong to that track.

   Parameters:

   - @Track_Id (INT): The unique identifier of the track for which students will be retrieved.

Output:

A list of students with details corresponding to the specified Track_Id.



```
CREATE PROCEDURE Show_Track_Students
    @Track_Id INT
AS
BEGIN
    SET NOCOUNT ON;
    SELECT * FROM Student
    WHERE Track_Id = @Track_Id;
END;
```

2. **Show_Track_Students_At_Branch**

   Description:

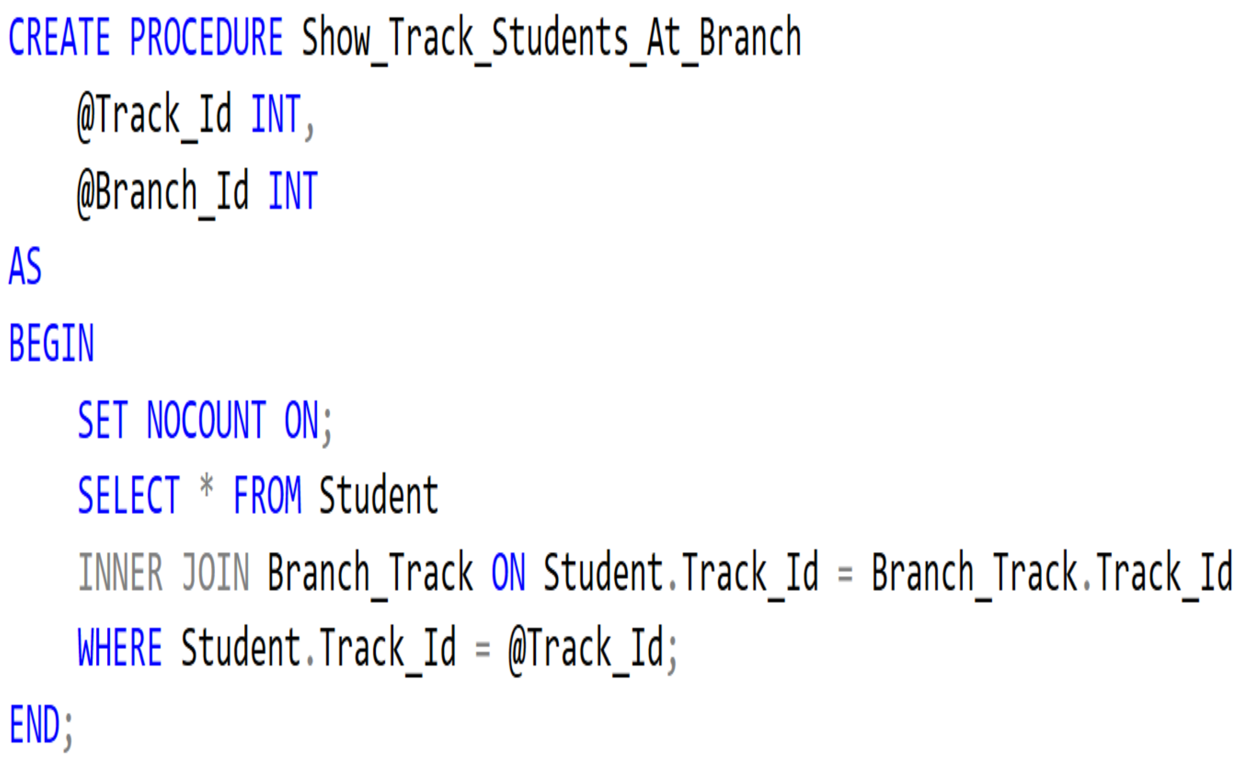
   This procedure retrieves students enrolled in a particular track and branch. It joins the Student table with the Branch_Track table to filter students based on their track and branch.

   Parameters:

   - @Track_Id (INT): The unique identifier for the track.
   - @Branch_Id (INT): The unique identifier for the branch.

Output:

A list of students who belong to the specified track at the specified branch.



```sql
CREATE PROCEDURE Show_Track_Students_At_Branch
    @Track_Id INT,
    @Branch_Id INT
AS
BEGIN
    SET NOCOUNT ON;
    SELECT * FROM Student
    INNER JOIN Branch_Track ON Student.Track_Id = Branch_Track.Track_Id
    WHERE Student.Track_Id = @Track_Id;
END;
```

3. **Show_Student_Grades**

   Description:

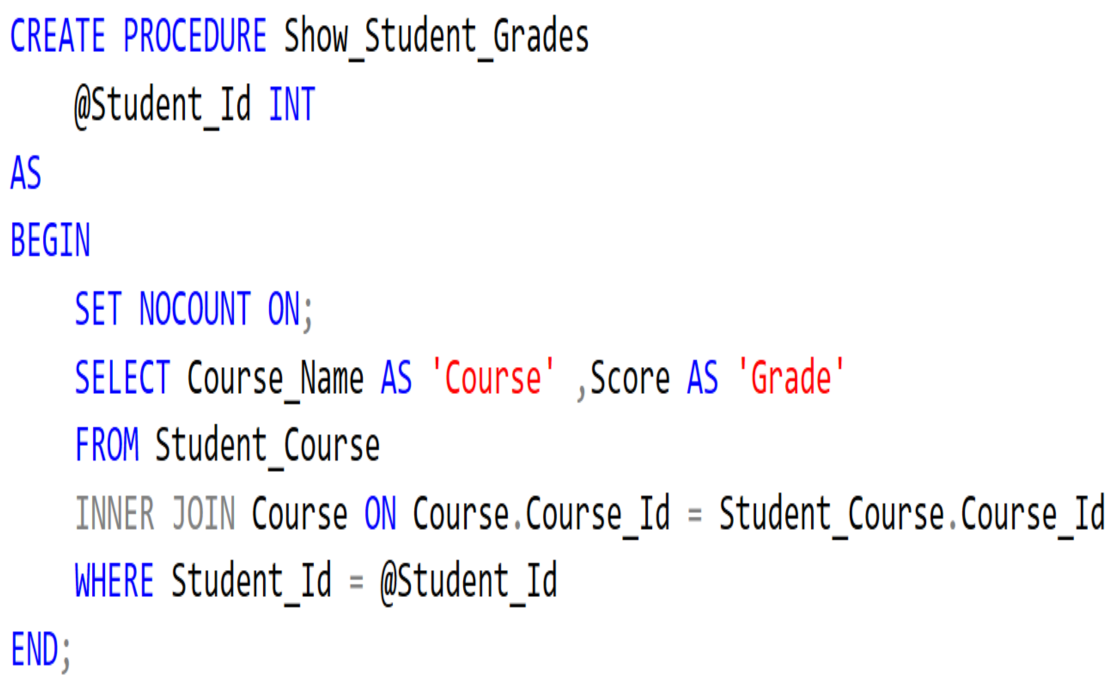
   This stored procedure returns the courses and grades for a specific student. It uses the Student_Course and Course tables to fetch the course names and corresponding scores for a given student.

   Parameters:

   - @Student_Id (INT): The unique identifier of the student.

   Output:

   A list of courses with their respective grades (score) for the specified student.



```sql
CREATE PROCEDURE Show_Student_Grades
    @Student_Id INT
AS
BEGIN
    SET NOCOUNT ON;
    SELECT Course_Name AS 'Course' ,Score AS 'Grade'
    FROM Student_Course
    INNER JOIN Course ON Course.Course_Id = Student_Course.Course_Id
    WHERE Student_Id = @Student_Id
END;
```

4. **Course_Topics**

Description:

This procedure provides details about a specific course, including its name and description. It filters the Course table using the provided Course_Id.

Parameters:

- @Course_Id (INT): The unique identifier of the course for which details will be returned.

Output:

The name and description of the specified course.

```
CREATE PROCEDURE  Course_Topics
    @Course_Id INT
AS
BEGIN
    SET NOCOUNT ON;
    SELECT Course_Name AS 'Course' ,Course.Description
    FROM Course
    WHERE Course_Id = @Course_Id;
END;
```

5. **Number_Of_Questions_In_Exam**

   Description:

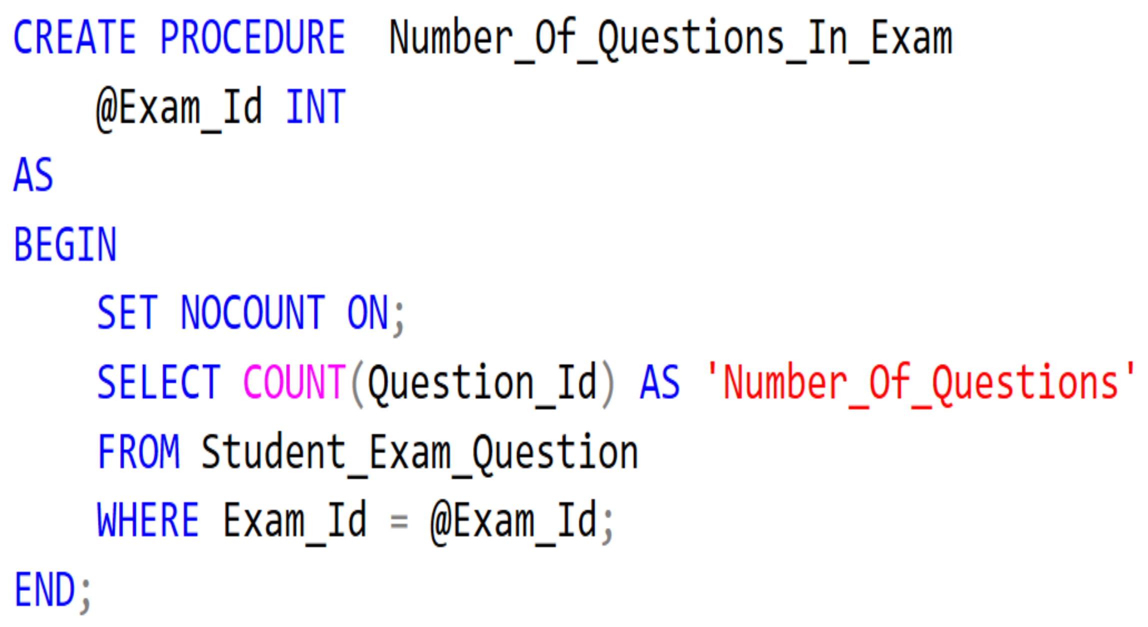
   This stored procedure counts the total number of questions in a given exam by querying the Student_Exam_Question table. It returns the number of questions associated with the specified Exam_Id.

   Parameters:

   - @Exam_Id (INT): The unique identifier of the exam.

Output:

A count of the number of questions associated with the specified exam.



```sql
CREATE PROCEDURE  Number_Of_Questions_In_Exam
    @Exam_Id INT
AS
BEGIN
    SET NOCOUNT ON;
    SELECT COUNT(Question_Id) AS 'Number_Of_Questions'
    FROM Student_Exam_Question
    WHERE Exam_Id = @Exam_Id;
END;
```

6. **Student_Exam_Answers**

   Description:

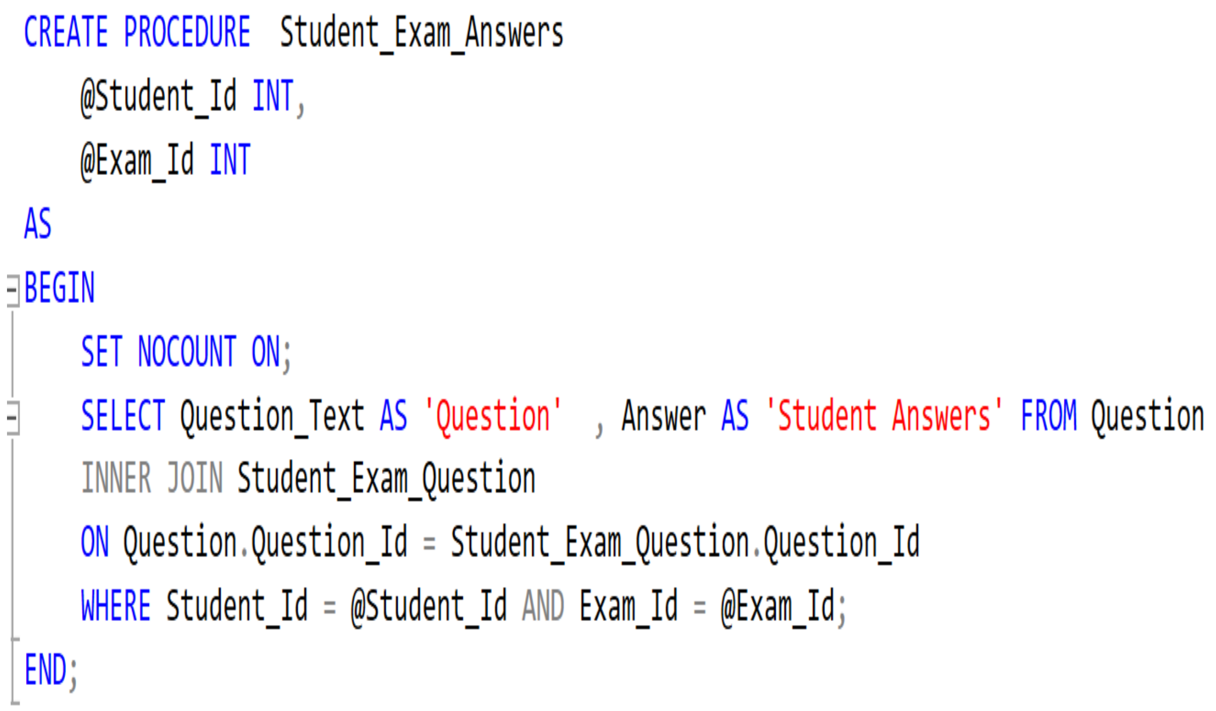
   This procedure retrieves the questions and corresponding answers provided by a student during a specific exam. It joins the Question table with the Student_Exam_Question table to fetch the question text and the student's answer for the specified exam.

   Parameters:

   - @Student_Id (INT): The unique identifier of the student.
   - @Exam_Id (INT): The unique identifier of the exam.

Output:

A list of questions and the student's answers for the specified exam.



```sql
CREATE PROCEDURE  Student_Exam_Answers
    @Student_Id INT,
    @Exam_Id INT
AS
BEGIN
    SET NOCOUNT ON;
    SELECT Question_Text AS 'Question'  , Answer AS 'Student Answers' FROM Question
    INNER JOIN Student_Exam_Question
    ON Question.Question_Id = Student_Exam_Question.Question_Id
    WHERE Student_Id = @Student_Id AND Exam_Id = @Exam_Id;
END;
```

Data Base Backup:

githubLink:

YousryEssam/ExaminationSystem: Database for Examination System

Thank You!