1-Stored Procedure for Creating an Exam

Procedure Name: CreateExam

Description:

This stored procedure is used to create an exam, validate the instructor's credentials, ensure sufficient questions are available, and insert the selected questions into the exam.

Parameters:

Parameter	**Type**	**Description**
@Name	VARCHAR(50)	Name of the exam
@StartTime	DATETIME	Exam start time
@EndTime	DATETIME	Exam end time
@Type	VARCHAR(50)	Type of the exam
@TotalGradeOfExam	INT	Total grade for the exam
@CourseId	INT	Course ID to which the exam belongs
@MCQCount	INT	Number of Multiple Choice Questions (MCQ)
@TrueFalseCount	INT	Number of True/False questions
@WrittenCount	INT	Number of Open-Ended questions
@MCQMark	INT	Marks per MCQ question
@TrueFalseMark	INT	Marks per True/False question
@WrittenMark	INT	Marks per Open-Ended question
@UserName	VARCHAR(50)	Instructor's username
@Password	VARCHAR(40)	Instructor's password

Process Flow:

1. **Authenticate Instructor:**
   - Checks if the provided username and password match an existing account.
   - Ensures the user is an instructor.
   - Confirms that the instructor is assigned to the specified course.
2. **Validate Exam Data:**
   - Ensures that the total exam grade equals the sum of marks assigned to different question types.
   - Verifies that there are enough questions available in the database for the exam.
3. **Create the Exam:**
   - Inserts a new exam record into ExamManagement.Exam.
   - Retrieves the newly created ExamId.
4. **Assign Questions to the Exam:**
   - Randomly selects and inserts the required number of questions of each type (MCQ, True/False, Open-Ended) from the question bank.
5. **Completion Message:**
   - If all steps are successful, a success message is printed.
   - If any validation fails, an appropriate error message is displayed.

2-Stored Procedure: AddStudentsToExam

Description

This stored procedure adds a student to an exam by inserting their record into the StudentListExam table. It first validates the exam and student existence before proceeding with the insertion.

Parameters

- @ExamId (INT): The ID of the exam the student will be added to.
- @StudentId (INT): The ID of the student being registered for the exam.

Logic

1. **Check if the exam exists**:
    - If the exam does not exist in the ExamManagement.Exam table, it prints 'Exam does not exist' and exits.
2. **Check if the student exists**:
    - If the student does not exist in the AcademicRecords.Student table, it prints 'Student does not exist' and exits.
3. **Check if the student is already registered for the exam**:
    - If a record already exists in ExamManagement.StudentListExam, it prints 'Student is already registered for this exam' and exits.
4. **Insert the student into the exam**:
    - If all checks pass, it inserts a new record into ExamManagement.StudentListExam and prints 'Student added to the exam successfully'.

3-Stored Procedure: GetExamQuestionsWithChoices

Description

This stored procedure retrieves all questions for a given exam along with their respective choices (if applicable). It supports multiple question types, including multiple-choice questions (MCQs).

Parameters

- @ExamId (INT): The ID of the exam for which the questions and choices should be retrieved.

Output Columns

- ExamId: The ID of the exam.
- Exam_QuestionId: The ID of the question in the exam.
- QuestionText: The text of the question.
- QuestionMark: The mark assigned to the question.
- QuestionType: The type of question (e.g., Multiple Choice, True/False, Open-Ended).
- MCQID: The ID of the multiple-choice question (if applicable).
- Choices: A concatenated string of choices for MCQs, formatted as "ChoiceId - ChoiceText | ChoiceId - ChoiceText".

Logic

1. **Retrieve Exam Questions**
    - Joins the ExamManagement.Exam_Question table to get the questions associated with the given @ExamId.
2. **Retrieve Question Details**
    - Joins the ExamManagement.Question table to get the question text, type, and mark.
3. **Retrieve Choices for MCQs**
    - If the question is a multiple-choice question, retrieves choices from ExamManagement.Choices.
    - Uses STRING_AGG to concatenate choices into a single string.
4. **Filter by Exam ID**
    - Ensures only questions for the specified exam are retrieved.

4-Stored Procedure: InsertStudentAnswers

Description

This stored procedure allows students to submit their answers for a specific exam question. It supports multiple question types, including Multiple Choice (MCQ), True/False, and Open-Ended questions.

Parameters

- @ExamId (INT): The ID of the exam.
- @Exam_QuestionId (INT): The ID of the specific question in the exam.
- @StudentAnswerOpenEnded (VARCHAR(300)): The student's answer for an open-ended question (default is an empty string).
- @StudentAnswerTrueFalse (VARCHAR(6)): The student's answer for a True/False question (default is an empty string).
- @McqId (INT): The ID of the multiple-choice question (optional, default is NULL).
- @ChoiceId (INT): The ID of the selected choice for an MCQ (optional, default is NULL).
- @UserName (VARCHAR(50)): The username of the student submitting the answer.
- @Password (VARCHAR(40)): The password of the student.

Logic

1. **User Authentication & Validation**
    - Retrieves AccountId, StudentId, and QuestionType based on @UserName and @Password.
    - Ensures the user is a valid student and has access to the exam.
    - Checks if the exam has ended and prevents submission if the time has passed.
2. **Exam & Question Validation**
    - Ensures the exam and question exist.
    - Checks if the student is registered for the exam.
    - Ensures the student hasn't already answered the question.
3. **Insert Answer Data**
    - If the student hasn't submitted any answers yet for the exam, creates a new record in AcademicRecords.Student_Exam with an initial score of 0.

- Inserts a new record in ExamManagement.Student_Answer_Details to track the student's response.
- Based on the QuestionType, inserts the response into the appropriate table:
  - **MCQ:** Inserts into Answer_MCQ and Answer_Choice.
  - **True/False:** Inserts into Answer_True_False.
  - **Open-Ended:** Inserts into Answer_Written.

4. **Success Message**
   - Prints "Answer inserted successfully" when the submission is recorded.

5-Stored Procedure: CalcGradeOfExamForStudent

This stored procedure calculates the total score of a student for a specific exam by evaluating multiple-choice, true/false, and open-ended questions.

Parameters

Parameter	Data Type	Description
@ExamId	INT	The ID of the exam being graded.
@StudentId	INT	The ID of the student whose exam is being graded.
@UserName	VARCHAR(50)	The username of the instructor.
@Password	VARCHAR(40)	The password of the instructor.

Procedure Logic

1. **Instructor Authentication**
    - Retrieves the InstructorId by checking the provided username and password.
    - If authentication fails, the procedure prints 'Invalid instructor credentials' and exits.
2. **Authorization Check**
    - Retrieves the CourseId associated with the exam.
    - Ensures the instructor is authorized to grade the exam.
    - If unauthorized, prints 'You are not the instructor of this course' and exits.
3. **Student Verification**
    - Ensures that the student is registered for the exam.
    - If the student is not found, prints 'Student is not registered for this exam' and exits.
    - Checks if the student has submitted any answers.
    - If no answers are found, prints 'No answers found for this student' and exits.
4. **Scoring Multiple-Choice Questions**
    - Joins Exam_Question, Question, Student_Answer_Details, Answer_MCQ, Answer_Choice, and Choices tables.
    - Adds the mark of each correctly answered multiple-choice question to @TotalScore.

5. **Scoring True/False Questions**
    - Joins Exam_Question, Question, True_False, Student_Answer_Details, and Answer_True_False tables.
    - Adds the mark of each correctly answered true/false question to @TotalScore.
6. **Scoring Open-Ended Questions**
    - Uses a cursor to iterate through all open-ended questions.
    - Splits the keywords stored in WrittenQuestion into separate entries.
    - Counts how many of these keywords are found in the student's answer.
    - Calculates a proportional score based on the number of matched keywords and updates @TotalScore.
7. **Updating the Student's Exam Score**
    - Updates the Score field in the Student_Exam table for the given student and exam.
8. **Completion Message**
    - Prints 'Exam graded successfully. Total Score: [score]'.

6- View Name: StudentExamResultsWithRanking

Purpose:

This view retrieves student exam results order by score with rank(), including student details, exam information, course details, and scores. It consolidates data from multiple tables to provide a comprehensive view of student performance in exams and courses.

Columns Explained:

Column Name	**Description**
StudentId	Unique identifier for the student.
FName	First name of the student.
ExamId	Unique identifier for the exam.
ExamName	Name/title of the exam.
CourseId	Unique identifier for the course related to the exam.
Crs_Name	Name of the course associated with the exam.
ExamGrade	The score the student achieved in the exam.
TotalGradesOfExam	The total maximum score possible for the exam.

1-Stored Procedure for Creating an Exam

Procedure Name: CreateExam

Description:

This stored procedure is used to create an exam, validate the instructor's credentials, ensure sufficient questions are available, and insert the selected questions into the exam.

Parameters:

Parameter	Type	Description
@Name	VARCHAR(50)	Name of the exam
@StartTime	DATETIME	Exam start time
@EndTime	DATETIME	Exam end time
@Type	VARCHAR(50)	Type of the exam
@TotalGradeOfExam	INT	Total grade for the exam
@CourseId	INT	Course ID to which the exam belongs
@MCQCount	INT	Number of Multiple Choice Questions (MCQ)
@TrueFalseCount	INT	Number of True/False questions
@WrittenCount	INT	Number of Open-Ended questions
@MCQMark	INT	Marks per MCQ question
@TrueFalseMark	INT	Marks per True/False question
@WrittenMark	INT	Marks per Open-Ended question
@UserName	VARCHAR(50)	Instructor's username
@Password	VARCHAR(40)	Instructor's password

Process Flow:

1. **Authenticate Instructor:**
    - Checks if the provided username and password match an existing account.
    - Ensures the user is an instructor.
    - Confirms that the instructor is assigned to the specified course.
2. **Validate Exam Data:**
    - Ensures that the total exam grade equals the sum of marks assigned to different question types.

   - Verifies that there are enough questions available in the database for the exam.
3. **Create the Exam:**
   - Inserts a new exam record into ExamManagement.Exam.
   - Retrieves the newly created ExamId.
4. **Assign Questions to the Exam:**
   - Randomly selects and inserts the required number of questions of each type (MCQ, True/False, Open-Ended) from the question bank.
5. **Completion Message:**
   - If all steps are successful, a success message is printed.
   - If any validation fails, an appropriate error message is displayed.

7-Stored Procedure: AddStudentsToExam

Description

This stored procedure adds a student to an exam by inserting their record into the StudentListExam table. It first validates the exam and student existence before proceeding with the insertion.

Parameters

- @ExamId (INT): The ID of the exam the student will be added to.
- @StudentId (INT): The ID of the student being registered for the exam.

Logic

1. **Check if the exam exists**:
   - If the exam does not exist in the ExamManagement.Exam table, it prints 'Exam does not exist' and exits.
2. **Check if the student exists**:
   - If the student does not exist in the AcademicRecords.Student table, it prints 'Student does not exist' and exits.
3. **Check if the student is already registered for the exam**:
   - If a record already exists in ExamManagement.StudentListExam, it prints 'Student is already registered for this exam' and exits.
4. **Insert the student into the exam**:

  - If all checks pass, it inserts a new record into ExamManagement.StudentListExam and prints 'Student added to the exam successfully'.

8-Stored Procedure: GetExamQuestionsWithChoices

Description

This stored procedure retrieves all questions for a given exam along with their respective choices (if applicable). It supports multiple question types, including multiple-choice questions (MCQs).

Parameters

- @ExamId (INT): The ID of the exam for which the questions and choices should be retrieved.

Output Columns

- ExamId: The ID of the exam.
- Exam_QuestionId: The ID of the question in the exam.
- QuestionText: The text of the question.
- QuestionMark: The mark assigned to the question.
- QuestionType: The type of question (e.g., Multiple Choice, True/False, Open-Ended).
- MCQID: The ID of the multiple-choice question (if applicable).
- Choices: A concatenated string of choices for MCQs, formatted as "ChoiceId - ChoiceText | ChoiceId - ChoiceText".

Logic

1. **Retrieve Exam Questions**
   - Joins the ExamManagement.Exam_Question table to get the questions associated with the given @ExamId.
2. **Retrieve Question Details**
   - Joins the ExamManagement.Question table to get the question text, type, and mark.
3. **Retrieve Choices for MCQs**
   - If the question is a multiple-choice question, retrieves choices from ExamManagement.Choices.
   - Uses STRING_AGG to concatenate choices into a single string.
4. **Filter by Exam ID**

    - Ensures only questions for the specified exam are retrieved.

9-Stored Procedure: InsertStudentAnswers

Description

This stored procedure allows students to submit their answers for a specific exam question. It supports multiple question types, including Multiple Choice (MCQ), True/False, and Open-Ended questions.

Parameters

- @ExamId (INT): The ID of the exam.
- @Exam_QuestionId (INT): The ID of the specific question in the exam.
- @StudentAnswerOpenEnded (VARCHAR(300)): The student's answer for an open-ended question (default is an empty string).
- @StudentAnswerTrueFalse (VARCHAR(6)): The student's answer for a True/False question (default is an empty string).
- @McqId (INT): The ID of the multiple-choice question (optional, default is NULL).
- @ChoiceId (INT): The ID of the selected choice for an MCQ (optional, default is NULL).
- @UserName (VARCHAR(50)): The username of the student submitting the answer.
- @Password (VARCHAR(40)): The password of the student.

Logic

1. **User Authentication & Validation**
    - Retrieves AccountId, StudentId, and QuestionType based on @UserName and @Password.
    - Ensures the user is a valid student and has access to the exam.
    - Checks if the exam has ended and prevents submission if the time has passed.
2. **Exam & Question Validation**
    - Ensures the exam and question exist.
    - Checks if the student is registered for the exam.
    - Ensures the student hasn't already answered the question.
3. **Insert Answer Data**

   - If the student hasn't submitted any answers yet for the exam, creates a new record in AcademicRecords.Student_Exam with an initial score of 0.
   - Inserts a new record in ExamManagement.Student_Answer_Details to track the student's response.
   - Based on the QuestionType, inserts the response into the appropriate table:
     - **MCQ:** Inserts into Answer_MCQ and Answer_Choice.
     - **True/False:** Inserts into Answer_True_False.
     - **Open-Ended:** Inserts into Answer_Written.
4. **Success Message**
   - Prints "Answer inserted successfully" when the submission is recorded.

<u>10-Stored Procedure: CalcGradeOfExamForStudent</u>

This stored procedure calculates the total score of a student for a specific exam by evaluating multiple-choice, true/false, and open-ended questions.

Parameters

Parameter	Data Type	Description
@ExamId	INT	The ID of the exam being graded.
@StudentId	INT	The ID of the student whose exam is being graded.
@UserName	VARCHAR(50)	The username of the instructor.
@Password	VARCHAR(40)	The password of the instructor.

Procedure Logic

1. **Instructor Authentication**
   - Retrieves the InstructorId by checking the provided username and password.
   - If authentication fails, the procedure prints 'Invalid instructor credentials' and exits.
2. **Authorization Check**
   - Retrieves the CourseId associated with the exam.
   - Ensures the instructor is authorized to grade the exam.

- If unauthorized, prints 'You are not the instructor of this course' and exits.

3. **Student Verification**
    - Ensures that the student is registered for the exam.
    - If the student is not found, prints 'Student is not registered for this exam' and exits.
    - Checks if the student has submitted any answers.
    - If no answers are found, prints 'No answers found for this student' and exits.
4. **Scoring Multiple-Choice Questions**
    - Joins Exam_Question, Question, Student_Answer_Details, Answer_MCQ, Answer_Choice, and Choices tables.
    - Adds the mark of each correctly answered multiple-choice question to @TotalScore.
5. **Scoring True/False Questions**
    - Joins Exam_Question, Question, True_False, Student_Answer_Details, and Answer_True_False tables.
    - Adds the mark of each correctly answered true/false question to @TotalScore.
6. **Scoring Open-Ended Questions**
    - Uses a cursor to iterate through all open-ended questions.
    - Splits the keywords stored in WrittenQuestion into separate entries.
    - Counts how many of these keywords are found in the student's answer.
    - Calculates a proportional score based on the number of matched keywords and updates @TotalScore.
7. **Updating the Student's Exam Score**
    - Updates the Score field in the Student_Exam table for the given student and exam.
8. **Completion Message**
    - Prints 'Exam graded successfully. Total Score: [score]'.

11- View Name: StudentExamResultsWithRanking

Purpose:

This view retrieves student exam results order by score with rank(), including student details, exam information, course details, and scores. It consolidates data from multiple tables to provide a comprehensive view of student performance in exams and courses.

Columns Explained:

Column Name	**Description**
StudentId	Unique identifier for the student.
FName	First name of the student.
ExamId	Unique identifier for the exam.
ExamName	Name/title of the exam.
CourseId	Unique identifier for the course related to the exam.
Crs_Name	Name of the course associated with the exam.
ExamGrade	The score the student achieved in the exam.
TotalGradesOfExam	The total maximum score possible for the exam.

12. SP_AddNewStudent

Description:
This stored procedure adds a new student to the AcademicRecords.Student table. It ensures that the student does not already exist and that the manager account ID is valid.

Parameters:

- @fname (nvarchar(50)): First name of the student.
- @Lname (nvarchar(50)): Last name of the student.
- @City (nvarchar(50)): City of the student.
- @zipcode (int): Zip code of the student.
- @phone (int): Phone number of the student.
- @Gender (char(1)): Gender of the student (e.g., 'M' or 'F').
- @BirthDate (date): Birth date of the student.
- @manager_Acc_Id (int): Account ID of the training manager adding the student.

Logic:

1. Checks if the manager account ID exists in the UserManagement.TrainingManager table.
2. Verifies that the student does not already exist in the AcademicRecords.Student table using their first and last name.
3. Inserts the new student into the AcademicRecords.Student table if all checks pass.

Success Message:
'Student Inserted Successfully'

Failure Messages:

- 'There is a student with firstname: [@fname] and last name: [@Lname]' (if the student already exists).
- 'You are not a manager' (if the manager account ID is invalid).
- 'There is an error in your Structure' (if an exception occurs).

---

13. sp_AddAccountToStudent

Description:
This stored procedure adds an account to a student in the UserManagement.Account table and links it to the student in the AcademicRecords.Student table.

Parameters:

- @username (nvarchar(200)): Username for the account.
- @Password (nvarchar(20)): Password for the account.
- @std_Id (int): ID of the student to whom the account will be linked.

Logic:

1. Checks if the student exists in the AcademicRecords.Student table.
2. Verifies that the student does not already have an account.
3. Inserts a new account into the UserManagement.Account table.
4. Updates the AcademicRecords.Student table to link the new account to the student.

Success Message:
'Account added successfully to Student'

Failure Messages:

- 'This Student already has an Account' (if the student already has an account).
- 'Student does not exist in your DB' (if the student ID is invalid).
- 'There is an error in your Structure' (if an exception occurs).

---

14. Sp_AddInstructor

Description:
This stored procedure adds a new instructor to the UserManagement.Instructor table.

Parameters:

- @fname (nvarchar(50)): First name of the instructor.
- @Lname (nvarchar(50)): Last name of the instructor.
- @City (nvarchar(50)): City of the instructor.
- @zipcode (int): Zip code of the instructor.
- @phone (int): Phone number of the instructor.
- @Gender (char(1)): Gender of the instructor (e.g., 'M' or 'F').
- @BirthDate (date): Birth date of the instructor.
- @salary (float): Salary of the instructor.

Logic:

1. Checks if the instructor does not already exist in the UserManagement.Instructor table using their first and last name.
2. Inserts the new instructor into the UserManagement.Instructor table if all checks pass.

Success Message:
'Instructor added successfully'

Failure Messages:

- 'There is an instructor with firstname: [@fname] and last name: [@Lname]' (if the instructor already exists).
- 'There is an error in your Structure' (if an exception occurs).

---

15. sp_AddAccountToInstructor

Description:
This stored procedure adds an account to an instructor in the UserManagement.Account table and links it to the instructor in the UserManagement.Instructor table.

Parameters:

- @username (nvarchar(200)): Username for the account.
- @Password (nvarchar(20)): Password for the account.
- @Inst_Id (int): ID of the instructor to whom the account will be linked.

Logic:

1. Checks if the instructor exists in the UserManagement.Instructor table.
2. Verifies that the instructor does not already have an account.

3. Inserts a new account into the UserManagement.Account table.
4. Updates the UserManagement.Instructor table to link the new account to the instructor.

Success Message:
'Account added successfully'

Failure Messages:

- 'This Instructor already has an Account' (if the instructor already has an account).
- 'Instructor does not exist in your DB' (if the instructor ID is invalid).
- 'There is an error in your Structure' (if an exception occurs).

---

16. sp_ADDTraining_Manager

Description:
This stored procedure adds a new training manager to the UserManagement.TrainingManager table.

Parameters:

- @fname (nvarchar(50)): First name of the training manager.
- @lname (nvarchar(50)): Last name of the training manager.

Logic:

1. Checks if the training manager does not already exist in the UserManagement.TrainingManager table using their first and last name.
2. Inserts the new training manager into the UserManagement.TrainingManager table if all checks pass.

Success Message:
'Training Manager added successfully'

Failure Messages:

- 'There is a Training Manager with firstname: [@fname] and last name: [@lname]' (if the training manager already exists).
- 'There is an error in your Structure' (if an exception occurs).

---

17. sp_AddAccountToTraining_manager

Description:
This stored procedure adds an account to a training manager in the UserManagement.Account table and links it to the training manager in the UserManagement.TrainingManager table.

Parameters:

- @username (nvarchar(200)): Username for the account.
- @Password (nvarchar(20)): Password for the account.
- @tr_m_Id (int): ID of the training manager to whom the account will be linked.

Logic:

1. Checks if the training manager exists in the UserManagement.TrainingManager table.
2. Verifies that the training manager does not already have an account.
3. Inserts a new account into the UserManagement.Account table.
4. Updates the UserManagement.TrainingManager table to link the new account to the training manager.

Success Message:
'Account added successfully'

Failure Messages:

- 'This Training Manager already has an Account' (if the training manager already has an account).
- 'Training Manager does not exist in your DB' (if the training manager ID is invalid).
- 'There is an error in your Structure' (if an exception occurs).

---

18. sp_TrMager_Add_Branch

Description:
This stored procedure allows a training manager to add a new branch to the AcademicRecords.Branch table.

Parameters:

- @managerAccount_ID (int): Account ID of the training manager.
- @Name (nvarchar(50)): Name of the branch.
- @Location (nvarchar(50)): Location of the branch.

Logic:

1. Checks if the training manager account ID is valid.
2. Verifies that the branch does not already exist in the AcademicRecords.Branch table.
3. Inserts the new branch into the AcademicRecords.Branch table if all checks pass.

Success Message:
'Branch added successfully'

Failure Messages:

- 'This branch was added before' (if the branch already exists).
- 'You are not a Manager' (if the training manager account ID is invalid).
- 'Error in your structure' (if an exception occurs).

---

19. sp_TrMager_Add_Track

Description:
This stored procedure allows a training manager to add a new track to the AcademicRecords.Track table.

Parameters:

- @managerAccount_ID (int): Account ID of the training manager.
- @Name (nvarchar(50)): Name of the track.
- @Description (nvarchar(200)): Description of the track.

Logic:

1. Checks if the training manager account ID is valid.
2. Verifies that the track does not already exist in the AcademicRecords.Track table.
3. Inserts the new track into the AcademicRecords.Track table if all checks pass.

Success Message:
'Track added successfully'

Failure Messages:

- 'This Track was added before' (if the track already exists).
- 'You are not a Manager' (if the training manager account ID is invalid).
- 'Error in your structure' (if an exception occurs).

---

20. sp_TrMager_Add_Intake

Description:
This stored procedure allows a training manager to add a new intake to the AcademicRecords.Intake table.

Parameters:

- @managerAccount_ID (int): Account ID of the training manager.
- @Name (nvarchar(50)): Name of the intake.
- @Intake_Year (int): Year of the intake.

Logic:

1. Checks if the training manager account ID is valid.
2. Verifies that the intake does not already exist in the AcademicRecords.Intake table.
3. Inserts the new intake into the AcademicRecords.Intake table if all checks pass.

Success Message:
'Intake added successfully'

Failure Messages:

- 'This Intake was added before' (if the intake already exists).
- 'You are not a Manager' (if the training manager account ID is invalid).
- 'Error in your structure' (if an exception occurs).

---

21. sp_TrMager_Edit_Branch

Description:
This stored procedure allows a training manager to edit an existing branch in the AcademicRecords.Branch table.

Parameters:

- @managerAccount_ID (int): Account ID of the training manager.
- @branch_Id (int): ID of the branch to edit.
- @Name (nvarchar(50)): New name for the branch (optional).
- @Location (nvarchar(50)): New location for the branch (optional).

Logic:

1. Checks if the training manager account ID is valid.
2. Verifies that the branch exists in the AcademicRecords.Branch table.
3. Updates the branch's name and/or location if provided.

Success Message:
'Edit Done Successfully'

Failure Messages:

- 'This Branch Does not exist; go add it first' (if the branch ID is invalid).
- 'You are not a Manager' (if the training manager account ID is invalid).
- 'There is an error in your Structure' (if an exception occurs).

---

22. sp_TrMager_Edit_Track

- Description**:**
  This stored procedure allows a training manager to edit an existing track in the AcademicRecords.Track table.

Parameters:

- @managerAccount_ID (int): Account ID of the training manager.
- @Track_Id (int): ID of the track to edit.
- @Name (nvarchar(50)): New name for the track (optional).
- @Description (nvarchar(200)): New description for the track (optional).

Logic:

1. Checks if the training manager account ID is valid.
2. Verifies that the track exists in the AcademicRecords.Track table.
3. Updates the track's name and/or description if provided.

Success Message:
'Edit Done Successfully'

Failure Messages:

- 'This Track Does not exist; go add it first' (if the track ID is invalid).
- 'You are not a Manager' (if the training manager account ID is invalid).
- 'There is an error in your Structure' (if an exception occurs).

---

23. sp_TrMager_Edit_Intake

Description:
This stored procedure allows a training manager to edit an existing intake in the AcademicRecords.Intake table.

Parameters:

- @managerAccount_ID (int): Account ID of the training manager.

- @Intake_Id (int): ID of the intake to edit.
- @Name (nvarchar(50)): New name for the intake (optional).
- @Intake_Year (int): New year for the intake (optional).

Logic:

1. Checks if the training manager account ID is valid.
2. Verifies that the intake exists in the AcademicRecords.Intake table.
3. Updates the intake's name and/or year if provided.

Success Message:
'Edit Done Successfully'

Failure Messages:

- 'This Intake Does not exist; go add it first' (if the intake ID is invalid).
- 'You are not a Manager' (if the training manager account ID is invalid).
- 'There is an error in your Structure' (if an exception occurs).

---

24. sp_TrMager_Set_studentinTrackandintackandbranch

Description:
This stored procedure assigns a student to a specific track, branch, and intake combination in the AcademicRecords.Student table.

Parameters:

- @std_id (int): ID of the student.
- @track_id (int): ID of the track.
- @branch_id (int): ID of the branch.
- @intack_id (int): ID of the intake.

Logic:

1. Checks if the combination of branch, track, and intake exists in the AcademicRecords.Branch_Track_Intake table.
2. Verifies that the student is not already assigned to a track, branch, and intake.
3. Updates the AcademicRecords.Student table to link the student to the specified track, branch, and intake.

Success Message:
'Student Enrolled Successfully'

Failure Messages:

- 'This student is already assigned to a track_intake_branch' (if the student is already assigned).
- 'This branch, intake, or track does not exist' (if the combination is invalid).
- 'Error in your structure' (if an exception occurs).

---

25. sp_CreateAccount

Description:
This stored procedure creates a new account in the UserManagement.Account table with strong username and password validation.

Parameters:

- @username (nvarchar(200)): Username for the account.
- @Password (nvarchar(20)): Password for the account.
- @AccountId (int output): Output parameter to return the newly created account ID.

Logic:

1. Validates that the username does not already exist in the UserManagement.Account table.
2. Ensures the username and password meet strong validation criteria (e.g., length, special characters).
3. Inserts the new account into the UserManagement.Account table.

Success Message:
'Account Added successfully'

Failure Messages:

- 'Your username or password is not strong' (if validation fails).
- 'This username already exists' (if the username is already taken).
- 'There is an error in your Structure' (if an exception occurs).

---

26. sp_ValidateUserLogin

Description:
This stored procedure validates a user's login credentials against the UserManagement.Account table.

Parameters:

- @username (nvarchar(200)): Username for the account.
- @password (nvarchar(20)): Password for the account.
- @isvalid (bit output): Output parameter to indicate if the login is valid (1) or invalid (0).

Logic:

1. Checks if the username and password match a record in the UserManagement.Account table.
2. Sets the @isvalid output parameter to 1 if valid, otherwise 0.

Success Message:
'Valid username and password'

Failure Messages:

- 'Invalid username or password' (if the credentials do not match).
- 'There is an error in your Structure' (if an exception occurs).

---

27. sp_AddCoursetoInstructor

Description:
This stored procedure assigns a course to an instructor in the AcademicRecords.Course table.

Parameters:

- @inst_id (int): ID of the instructor.
- @crc_id (int): ID of the course.

Logic:

1. Checks if the course exists in the AcademicRecords.Course table.
2. Updates the course's InstructorId to the specified instructor ID.

Success Message:
'Course assigned to instructor successfully'

Failure Messages:

- 'This course is already assigned to an instructor' (if the course is already assigned).
- 'Error in your structure' (if an exception occurs).

---

28. sp_CreateCourseByTRM

Description:
This stored procedure allows a training manager to create a new course in the AcademicRecords.Course table.

Parameters:

- @trmAccount_id (int): Account ID of the training manager.
- @crs_Name (nvarchar(50)): Name of the course.
- @Description (nvarchar(500)): Description of the course.
- @MaxDegree (int): Maximum degree for the course.
- @minDegre (int): Minimum degree for the course.
- @DurationHour (int): Duration of the course in hours.

Logic:

1. Checks if the training manager account ID is valid.
2. Verifies that the course does not already exist in the AcademicRecords.Course table.
3. Inserts the new course into the AcademicRecords.Course table.

Success Message:
'Course Added Successfully'

Failure Messages:

- 'This course already exists' (if the course already exists).
- 'You are not a manager' (if the training manager account ID is invalid).
- 'There is an error in your Structure' (if an exception occurs).

---

29. sp_EditCourseByTRM

Description:
This stored procedure allows a training manager to edit an existing course in the AcademicRecords.Course table.

Parameters:

- @trmAccount_id (int): Account ID of the training manager.
- @Crc_ID (int): ID of the course to edit.
- @crs_Name (nvarchar(50)): New name for the course (optional).
- @Description (nvarchar(500)): New description for the course (optional).

- @MaxDegree (int): New maximum degree for the course (optional).
- @minDegre (int): New minimum degree for the course (optional).
- @DurationHour (int): New duration for the course in hours (optional).

Logic:

1. Checks if the training manager account ID is valid.
2. Verifies that the course exists in the AcademicRecords.Course table.
3. Updates the course's details if provided.

Success Message:
'Course Edit Successfully'

Failure Messages:

- 'This course does not exist' (if the course ID is invalid).
- 'You are not a manager' (if the training manager account ID is invalid).
- 'There is an error in your Structure' (if an exception occurs).

---

30. verificationAccount (Trigger)

Description:
This trigger ensures that any new account added to the UserManagement.Account table meets strong username and password validation criteria.

Logic:

1. Validates the username and password for strength (e.g., length, special characters).
2. Ensures the username is unique.
3. Rolls back the insertion if validation fails.

Success Message:
'Account Added successfully'

Failure Messages:

- 'Your username or password is not strong' (if validation fails).
- 'This username already exists' (if the username is already taken).

31. *AllCourses*

- Displays all data from the Course table.

select * from AllCources

32. *CourseWithPassingRate*

- Displays courses with the calculated passing rate (MinDegree/MaxDegree * 100).

34. *LongCourses*

- Displays courses with a duration greater than 30 hours.

35. *CourseMinMaxDuration*

- Displays the longest and shortest course based on duration.

36. *AvgCourseDuration*

- Displays the average duration of all courses in the table.

37. *CoursesNeedingReview*

- Displays courses where the minimum degree is greater than 50% of the maximum degree, indicating the need for review.

38. *CoursesOrderedByDuration*

- Displays courses ordered from longest to shortest based on duration.

39. *CoursesWithoutDescription*

- Displays courses that have no description (NULL).

40. **CoursesWithStudents

- View to view the students enrolled in each course

41. **CourseDetails

-View It displays details of the courses,

with information about the teachers who taught them, and the classes in which they were taught.

42. *StudentCoursesDetails*

-This view displays data for students registered in training courses,

along with course information and the instructor responsible for teaching.

/Synonyms/

18) AcademicRecords.Branch ===>ACREBR

```
create synonym ACREBR
for AcademicRecords.Branch

select* from AcademicRecords.Branch
select* from ACREBR
```

----

43) AcademicRecords.Branch_Track_Intake ===>ACREBRTRIN

```
create synonym ACREBRTRIN
for AcademicRecords.Branch_Track_Intake

select* from AcademicRecords.Branch_Track_Intake
select* from ACREBRTRIN
```

----

44) AcademicRecords.Class ===>ACRECL

```
create synonym ACRECL
for AcademicRecords.Class

select* from AcademicRecords.Class
select* from ACRECL
```

```
----------------------------------------------
45) AcademicRecords.Course ===>ACRECO
create synonym ACRECO
for AcademicRecords.Course
drop synonym ACRECo

select* from AcademicRecords.Course
select* from ACRECO
----------------------------------------------
46) AcademicRecords.Instructor_Course_History ===>ACREINCOHI
create synonym ACREINCOHI
for AcademicRecords.Instructor_Course_History

select* from AcademicRecords.Instructor_Course_History
select* from ACREINCOHI
----------------------------------------------
47) AcademicRecords.Student ===>ACREINCOST
create synonym ACREINCOST
for AcademicRecords.Student

select* from AcademicRecords.Student
select* from ACREINCOST
----------------------------------------------
48) AcademicRecords.Intake ===>ACREIN
create synonym ACREIN
for AcademicRecords.Intake

select* from AcademicRecords.Intake
select* from ACREIN
```

----

49) AcademicRecords.Student ===>ACREST

```
create synonym ACREST
for AcademicRecords.Student

select* from AcademicRecords.Student
select* from ACREST
```

----

50) AcademicRecords.Student_Exam ===>ACRESTEXAM

```
create synonym ACRESTEXAM
for AcademicRecords.Student_Exam

select* from AcademicRecords.Student_Exam
select* from ACRESTEXAM
```

----

51) AcademicRecords.Track ===>ACRETR

```
create synonym ACRETR
for AcademicRecords.Track

select* from AcademicRecords.Track
select* from ACRETR
```

----

52) ExamManagement.Answer_Choice ===>EXMGANCH

```
create synonym EXMGANCH
for ExamManagement.Answer_Choice

select* from ExamManagement.Answer_Choice
select* from EXMGANCH
```

----

53) ExamManagement.Answer_MCQ ===>EXMGANMCQ

```
create synonym EXMGANMCQ
for ExamManagement.Answer_MCQ

select* from ExamManagement.Answer_MCQ
select* from EXMGANMCQ
```

----

54) ExamManagement.Answer_True_False ===>EXMGANTF

```
create synonym EXMGANTF
for ExamManagement.Answer_True_False

select* from ExamManagement.Answer_True_False
select* from EXMGANTF
```

----

55) ExamManagement.Answer_Written ===>EXMGANWRITTEN

```
create synonym EXMGANWRITTEN
for ExamManagement.Answer_Written

select* from ExamManagement.Answer_Written
select* from EXMGANWRITTEN
```

----

56) ExamManagement.Choices ===>EXMGCHOICES

```
create synonym EXMGCHOICES
for ExamManagement.Choices

select* from ExamManagement.Choices
select* from EXMGCHOICES
```

57) ExamManagement.Exam ===>EXMGEXAM

create synonym EXMGEXAM

for ExamManagement.Exam

select* from ExamManagement.Exam

select* from EXMGEXAM

----

58) ExamManagement.Exam_Question ===>EXMGEXAMQUE

create synonym EXMGEXAMQUE

for ExamManagement.Exam_Question

select* from ExamManagement.Exam_Question

select* from EXMGEXAMQUE

----

59) ExamManagement.MCQ ===>EXMGMCQ

create synonym EXMGMCQ

for ExamManagement.MCQ

select* from ExamManagement.MCQ

select* from EXMGMCQ

----

60) ExamManagement.Question ===>EXMGQUE

create synonym EXMGQUE

for ExamManagement.Question

select* from ExamManagement.Question

select* from EXMGQUE

----

61) ExamManagement.Student_Answer_Details ===>EXMGSTANDETAILS

create synonym EXMGSTANDETAILS

for ExamManagement.Student_Answer_Details

select* from ExamManagement.Student_Answer_Details

select* from EXMGSTANDETAILS

----

62) ExamManagement.StudentListExam ===>EXMGSTLIEXAM

create synonym EXMGSTLIEXAM

for ExamManagement.StudentListExam

select* from ExamManagement.StudentListExam

select* from EXMGSTLIEXAM

----

63) ExamManagement.True_False ===>EXMGTF

create synonym EXMGTF

for ExamManagement.True_False

select* from ExamManagement.True_False

select* from EXMGTF

----

64) ExamManagement.WrittenQuestion ===>EXMGWRQUE

create synonym EXMGWRQUE

for ExamManagement.WrittenQuestion

select* from ExamManagement.WrittenQuestion

select* from EXMGWRQUE

----

65) UserManagement.Account ===>USERMACC

create synonym USERMACC

for UserManagement.Account

select* from UserManagement.Account

select* from USERMACC

----

66) UserManagement.Instructor ===>USERMAINS

create synonym USERMAINS

for UserManagement.Instructor

select* from UserManagement.Instructor

select* from USERMAINS

----

67) UserManagement.TrainingManager ===>USERMATRMANGER

create synonym USERMATRMANGER

for UserManagement.TrainingManager

select* from UserManagement.TrainingManager

select* from USERMATRMANGER