➢ Part 01 Stored Procedure

QUESTION 1

Create a stored procedure named **sp_GetRecentBadges** that retrieves all badges earned by users within the last **N days**.
The procedure should accept one input parameter **@DaysBack (INT)** to determine how many days back to search.
Test the procedure using different values for the number of days.

QUESTION 2

Create a stored procedure named **sp_GetUserSummary** that retrieves summary statistics for a specific user.
The procedure should accept **@UserId** as an input parameter and return the following values as **output parameters**:

Route Training center & innovation space

- Total number of posts created by the user
- Total number of badges earned by the user
- Average score of the user's posts

QUESTION 3

Create a stored procedure named **sp_SearchPosts** that searches for posts based on:

- A keyword found in the post title
- A minimum post score

The procedure should accept **@Keyword** as an input parameter and **@MinScore** as an optional parameter with a default value of 0.
The result should display matching posts ordered by score.

QUESTION 3

Create a stored procedure named **sp_GetUserOrError** that retrieves user details by user ID.
If the specified user does not exist, the procedure should raise a meaningful error.
Use **TRY…CATCH** for proper error handling.

QUESTION 4

Create a stored procedure named **sp_AnalyzeUserActivity** that:

- Calculates an **Activity Score** for a user using the formula:
  Reputation + (Number of Posts × 10)
- Returns the calculated Activity Score as an output parameter
- Returns a result set showing the user's top 5 posts ordered by score

Route
Training Center & Innovation Space

QUESTION 5

Create a stored procedure named **sp_GetReputationInOut** that uses a single **input/output parameter**.
The parameter should initially contain a **UserId** as input and return the corresponding **user reputation** as output.

QUESTION 6

Create a stored procedure named **sp_UpdatePostScore** that updates the score of a post.
The procedure should:

- Accept a post ID and a new score as input

- Validate that the post exists
- Use **transactions** and **TRY…CATCH** to ensure safe updates
- Roll back changes if an error occurs

QUESTION 7

Create a stored procedure named **sp_GetTopUsersByReputation** that retrieves the top **N users** whose reputation is above a specified minimum value.
Then create a permanent table named **TopUsersArchive** and insert the results returned by the procedure into this table.

QUESTION 8

Create a stored procedure named **sp_InsertUserLog** that inserts a new record into a **UserLog** table.
The procedure should:

- Accept user ID, action, and details as input
- Return the newly created log ID using an **output parameter**

Route
Training center & innovation space

QUESTION 9

Create a stored procedure named **sp_UpdateUserReputation** that updates a user's reputation.
The procedure should:

- Validate that the reputation value is not negative
- Validate that the user exists
- Return the number of rows affected
- Handle errors appropriately

QUESTION 10

Create a stored procedure named **sp_DeleteLowScorePosts** that deletes all posts with a score less than or equal to a given value.
The procedure should:

- Use transactions
- Return the number of deleted records as an output parameter
- Roll back changes if an error occurs

QUESTION 11

Create a stored procedure named **sp_BulkInsertBadges** that inserts multiple badge records for a user.
The procedure should:

- Accept a user ID
- Accept a badge count indicating how many badges to insert
- Insert multiple related records in a single operation

Route
Training center & innovation space

QUESTION 12

Create a stored procedure named **sp_GenerateUserReport** that generates a complete user report.
The procedure should:

➢ Call another stored procedure internally to retrieve user statistics

➢ Combine user profile data and statistics

➢ Return a formatted report including a calculated user level

➢ Part 02 Trigger

QUESTION 1

Create an **AFTER INSERT trigger** on the **Posts** table that logs every new post creation into a **ChangeLog** table.
The log should include:

- Table name
- Action type
- User ID of the post owner
- Post title stored as new data

QUESTION 2

Create an **AFTER UPDATE trigger** on the **Users** table that tracks changes to the **Reputation** column.
The trigger should:

- Log changes only when the reputation value actually changes
- Store both the old and new reputation values in the **ChangeLog** table

---

QUESTION 3

Create an **AFTER DELETE trigger** on the **Posts** table that archives deleted posts into a **DeletedPosts** table.
All relevant post information should be stored before the post is removed.

Route
Training center & innovation space

QUESTION 4

Create an **INSTEAD OF INSERT trigger** on a view named **vw_NewUsers** (based on the Users table).
The trigger should:

- Validate incoming data
- Prevent insertion if the **DisplayName** is NULL or empty

QUESTION 5

Create an **INSTEAD OF UPDATE trigger** on the **Posts** table that prevents updates to the **Id** column.
Any attempt to update the Id column should be:

- Blocked
- Logged in the **ChangeLog** table

Route
Training center & innovation space

QUESTION 6

Create an **INSTEAD OF DELETE trigger** on the **Comments** table that implements a **soft delete** mechanism.
Instead of deleting records:

- Add an **IsDeleted** flag
- Mark records as deleted
- Log the soft delete operation

QUESTION 7

Create a **DDL trigger** at the database level that prevents any table from being dropped.
All drop table attempts should be logged in the **ChangeLog** table.

QUESTION 8

Create a **DDL trigger** that logs all **CREATE TABLE** operations.
The trigger should record:

- The action type
- The full SQL command used to create the table

QUESTION 9

Create a **DDL trigger** that prevents any **ALTER TABLE** statement that attempts to drop a column.
All blocked attempts should be logged.

Route
Training Center & innovation space

QUESTION 10

Create a single trigger on the **Badges** table that tracks **INSERT**, **UPDATE**, and **DELETE** operations.
The trigger should:

- Detect the operation type using INSERTED and DELETED tables
- Log the action appropriately in the **ChangeLog** table

QUESTION 11

Create a trigger that maintains summary statistics in a **PostStatistics** table whenever posts are inserted, updated, or deleted.
The trigger should update:

- Total number of posts
- Total score
- Average score
  for the affected users.

QUESTION 12

Create an **INSTEAD OF DELETE trigger** on the **Posts** table that prevents deletion of posts with a score greater than 100.
Any prevented deletion should be logged.

Route
Training center & innovation space

QUESTION 13

Write the SQL commands required to:

1. Disable a specific trigger on the Posts table
2. Enable the same trigger again
3. Check whether the trigger is currently enabled or disabled