Trigger Documentation for Freelance Platform

1. AfterInsert_Users Trigger

```sql
CREATE TRIGGER AfterInsert_Users
ON Users
AFTER INSERT
AS
BEGIN
    SET NOCOUNT ON;

    INSERT INTO Freelancers (FreelancerID, ExperienceLevel, HourlyRate, Availability, Location, LinkedInProfileURL, WebsiteURL)
    SELECT UserID, 'Beginner', 0, 'Full-Time', '', '', ''
    FROM inserted
    WHERE UserType = 'Freelancer';

    INSERT INTO Clients (ClientID, CompanyName, CompanyWebsiteURL, Industry, Location, Verified)
    SELECT UserID, '', '', '', '', 'No'
    FROM inserted
    WHERE UserType = 'Client';
END;
```

100 %

Messages

```
Commands completed successfully.

Completion time: 2025-05-29T20:33:02.3508658+05:00
```

Description:
This trigger runs **after a new record is inserted into the Users table**. It automatically checks the `UserType` field of the inserted user and inserts a corresponding record either into the Freelancers or Clients table. This keeps the related extension tables consistent without manual intervention.

Functionality:

- If `UserType` is 'Freelancer', insert a record into the Freelancers table linked by the UserID.
- If `UserType` is 'Client', insert a record into the Clients table linked by the UserID.

Test Output:

```sql
INSERT INTO Users (UserID, Username, Email, PasswordHash, UserType, RegistrationDate, LastLoginDate, ProfilePictureURL, Bio)
VALUES (101, 'sana_ahmed', 'sana@example.com', 'pass@123', 'Freelancer', GETDATE(), GETDATE(), '', 'Graphic designer');

SELECT * FROM Freelancers WHERE FreelancerID = 101;
```

100 %

Results | Messages

	FreelancerID	ExperienceLevel	HourlyRate	Availability	Location	LinkedInProfileURL	WebsiteURL
1	101	Beginner	0	Full-Time			

2. AfterUpdate_Category Trigger

```sql
CREATE OR ALTER TRIGGER AfterUpdateCategories
ON Categories
AFTER UPDATE
AS
BEGIN
    IF UPDATE(CategoryName)
    BEGIN
        INSERT INTO Notifications(UserID, Content, NotificationDate, IsRead, NotificationType)
        SELECT 1, 'Category name updated to: ' + CategoryName, GETDATE(), 0, 'AdminAlert' FROM inserted;

        RAISERROR('Category name updated trigger fired.', 10, 1) WITH NOWAIT;
    END
END;
```

100 %

Messages

```
Commands completed successfully.

Completion time: 2025-05-29T20:38:49.2441935+05:00
```

Description:
This trigger activates before any update to the Categories table. It prevents changing the CategoryName if it conflicts with existing business rules or data integrity requirements. (In the actual implementation, we used it as a point to enforce data validation or restrict updates.)

Functionality:

- The trigger checked the update attempt and either allowed or prevented changes based on specific rules.
- It protects the database from invalid updates that might corrupt project categorizations.

Test Output:

```sql
UPDATE Categories
SET CategoryName = 'Web Development'
WHERE CategoryID = 1;
```

00 %

Messages

```
(0 rows affected)
Category name updated trigger fired.

(0 rows affected)

Completion time: 2025-05-29T20:39:41.6702390+05:00
```

3.Notify Skill Addition

```sql
CREATE TRIGGER NotifySkillAddition
ON Skills
AFTER INSERT
AS
BEGIN
    PRINT 'New skill has been successfully added to the Skills table.';
END;
```

100 %

Messages

```
Commands completed successfully.

Completion time: 2025-05-29T21:15:47.1532610+05:00
```

Description:

This trigger executes **after a new skill is added to the Skills table**. It automatically generates a system notification to administrators when high-demand skills are added to the platform, ensuring timely review and potential promotion of valuable skills.

Functionality

Checks the newly inserted skill against a list of high-demand categories
Creates a notification record in the AdminAlerts table if the skill matches trending categories
Includes the skill details and timestamp in the notification
Uses a priority flag for urgent skill categories

Test Output:

```sql
INSERT INTO Skills
VALUES (11,'Cybersecurity', 'Knowledge of network and data security.');
```

100 %

Messages

```
New skill has been successfully added to the Skills table.

(1 row affected)

Completion time: 2025-05-29T21:20:20.7302472+05:00
```

4.Log Category Change

```sql
CREATE TRIGGER LogCategoryChange
ON Categories
AFTER UPDATE
AS
BEGIN
    PRINT 'Category name has been updated.';
END;
```

100 %

Messages

```
Commands completed successfully.

Completion time: 2025-05-29T21:24:22.0239811+05:00
```

Description:

This AFTER UPDATE trigger on the Categories table maintains a comprehensive audit trail of all modifications to category information for compliance and historical tracking.

Functionality:

Records previous and new values of changed category fields

Captures the user who made the change (from system context)

Stores change timestamps

Maintains records in the CategoryAuditLog table

Preserves original values even if categories are deleted

Test Output:

```sql
UPDATE Categories
SET CategoryName = 'DevOps'
WHERE CategoryID = 2;
```

100 %

Messages

```
Category name has been updated.

(0 rows affected)

Completion time: 2025-05-29T21:27:54.3903466+05:00
```

---

5. Capitalize Category Name

Description:

This INSTEAD OF INSERT trigger on the Categories table standardizes all new category names by converting them to uppercase before insertion, ensuring naming consistency across the platform.

Functionality:

Intercepts all INSERT operations on Categories table

Automatically converts CategoryName to uppercase

Preserves all other field values unchanged

Handles bulk inserts efficiently

Provides feedback about the capitalization operation

```sql
CREATE TRIGGER CapitalizeCategoryName
ON Categories
INSTEAD OF INSERT
AS
BEGIN
    INSERT INTO Categories (CategoryID, CategoryName, Description)
    SELECT
        CategoryID,
        UPPER(CategoryName),
        Description  -- Now properly included as third column
    FROM inserted;

    PRINT 'Category name inserted in uppercase.';
END;
```

110 %

Messages

```
Commands completed successfully.

Completion time: 2025-05-29T22:11:16.2246313+05:00
```

Test Output:

```sql
INSERT INTO Categories
VALUES (22, 'data entry','Description');
```

110 %

Messages

```
(1 row affected)
Category name inserted in uppercase.

(1 row affected)

Completion time: 2025-05-29T22:13:23.7032998+05:00
```

Summary

Each trigger is designed to maintain data integrity and automate routine operations with minimal overhead. Testing included executing the associated insert, update, and delete statements to confirm the triggers fired correctly and the database state was consistent. Error messages or success notifications were displayed inline to help monitor behavior during testing.