**Trigger Documentation for Freelance Platform**

**1. AfterInsert\_Users Trigger**

**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:**  
Upon inserting a new user, the trigger fired and automatically added the appropriate record into the Freelancers or Clients table. The success message was displayed (via RAISERROR) confirming the trigger execution.

**2. AfterUpdate\_Category Trigger**

**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:**  
When attempting to update a category name, the trigger fired and blocked or allowed the update accordingly. A confirmation message was shown verifying trigger activation.

**3. Delete Associated Bids, Contracts, and Messages Trigger**

**Description:**  
This trigger runs **after a project is deleted from the Projects table**. It cascades deletions by removing all related bids, contracts, and messages associated with the deleted project. This maintains referential integrity by cleaning up dependent records.

**Functionality:**

* Deletes all bids linked to the deleted project.
* Deletes contracts related to those bids.
* Deletes any messages associated with the project.

**Test Output:**  
Deleting a project caused the trigger to activate and remove all associated bids, contracts, and messages. Testing showed the dependent data was successfully cleaned without manual deletion.

**4. EmailValidation Trigger (Before Insert/Update on Users)**

**Description:**  
This trigger runs **before inserting or updating a user record**. It validates the email format to ensure only properly formatted email addresses are saved. If the email is invalid, it rolls back the transaction to prevent bad data entry.

**Functionality:**

* Checks if the email follows a valid format (e.g., contains '@' and '.').
* Prevents the insert or update if the email is invalid.

**Test Output:**  
When inserting or updating users, invalid emails triggered an error message and blocked the operation. Valid emails passed through successfully, confirming proper validation.

**5. Prevent Budget Change if Bids Exist Trigger (Before Update on Projects)**

**Description:**  
This trigger activates **before any update to the budget field in the Projects table**. It prevents changing the budget of a project if there are already bids submitted for that project, ensuring fairness and contract consistency.

**Functionality:**

* Checks if any bids exist for the project being updated.
* If bids exist, it blocks the budget change and raises an error.

**Test Output:**  
Testing updates to project budgets with existing bids caused the trigger to block the update and show an error message. Updates on projects without bids proceeded smoothly.

**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.