Triggers

- ✓ Trigger is a Custom Apex Code, which will get fired Automatically upon performing the DML Events.
- ✓ Trigger will get fired upon performing the DML operations (insert, update, delete, undelete) on the object records.
- ✓ By using Triggers, we can implement the complex validation rules, complex Business Logics, and complex Transactional flows inside the application.
- ✓ Triggers can fire on both **Before** and **After** performing the operations.
- ✓ By using Triggers, we can perform all the DML operations (like insert, Update, Delete, Upset, Undelete) on the same object or on any other object records. Triggers will get fired **Asynchronously.**
- ✓ Each trigger should be get associated with an object. But an object can have one or more associated triggers.

Syntax:

```
trigger TriggerName on ObjectName (trigger_events) {
    code_block
}
```

Trigger Events:

- ✓ Upon creating the Trigger, we need to specify, on what occasion to trigger should get fired.
- ✓ Apex provides the below 7 Events, can be used inside the trigger, to indicate when the trigger should get fired.

1. **Before Insert:** This event indicates the trigger should get fired **Before Inserting** the record into the object.

   Ex: Validating the data, upon eliminating the Duplicate records, upon maintaining the one – one association etc.

2. **Before Update:** This event will make the trigger to be get fired **Before Updating** a record into the object.

   Ex: Validating the Data, eliminating the duplicate records etc.

3. **Before Delete:** This event will make the trigger to be get fired **Before Deleting** the record from the associated object.

   Ex: Prevent the Deletion of specific records, Verify the Access Permissions Before Deletion, Removing the Child records before removing the parent etc.

4. **After Insert:** This event will make the trigger to be get fired **After Inserting** the record into the object.

   Ex: Sending the Email Alerts, Synchronizing the child records, performing post – execution logic, updating Rollup summary field etc.

5. **After Update:** This event will make, the trigger should get fired, after updating the record.

   Ex: Sending Email Alerts, Synchronize the records, etc.

6. **After Delete:** This event will make the trigger to be get fired after deleting the records from the object.

   Ex: Updating Rollup Summary fields, Removing the associated Child records etc.

7. **After Undelete:** This event will make the trigger should get fired after re-storing the deleted record back to the object.

   Ex: Updating Rollup Summary filed etc.

Note:

- A Trigger can get associated with one or more events. We need to specify the multiple events by separating a comma.

Example:

```
trigger TriggerName on Account (before insert, before update) {

    // Write the Business Logic for Before Insert..


    // Write the Business Logic for Before Update..

}
```

- Each Trigger should be associated with an object, but an object can have one or more Triggers associated with it.
- As a best practice, it is always recommended to have only one trigger per an object.
- If the object contains multiple triggers, which will fire on the same event, then we can't assure the order of execution. We don't have any options in salesforce to control the order of execution of triggers.
- To avoid this, we have to write the code inside a single trigger in the required order to perform the actions.

Types of Triggers:

Apex provides 2 types of Triggers as below.

1. **Before Triggers:** These triggers will get fired **Before** performing the operations on the specified object.

   Ex: By using Before Triggers we can achieve complex validation rules. Eliminating Duplicate Records, Validating the Conditions before updating / Deleting the records etc.

2. **After Triggers:** These triggers will get fired always **After** performing operations on the specified object.

Ex: By using After Triggers, we can implement the features to send Email Alerts, Update the Fields, Update Rollup Summary values, etc.

Trigger Context Variables:

✓ Upon writing the Business Logic inside the Trigger, we need to segregate the code blocks associated with **Multiple Event** with the help of Trigger Context Variables. These are used to get the Current status of the Trigger.

Apex Provides the below Trigger Context Variables.

1. **Trigger.isInsert**: It returns the TRUE, if the trigger has been fired because of an insert operation. Else it returns FALSE.

2. **Trigger.isUpdate**: It returns TRUE, if the trigger has been fired because of an Update operation. Else if returns FALSE.

3. **Trigger.isDelete**: it returns TRUE, if the trigger has been fired, because of a Delete operation. Else it returns FALSE.

4. **Trigger.isUpdate**: It returns TRUE, if the trigger has been fired because of an Undelete operation. Else it returns FALSE.

5. **Trigger.isBefore**: It returns TRUE, if the trigger is about to perform the operation. Else it returns FALSE.

6. **Trigger.isAfter**: It returns TRUE, if the trigger is already done with the operations. Else it returns FALSE.

Example:

```
trigger LeadTrigger on Lead (before insert,  after insert, before delete) {

    if(Trigger.isInsert && Trigger.isBefore){

        // Write the Business Logic to validate the record values..
    }
    if(Trigger.isInsert && Trigger.isAfter){

        // Write the Business Logic to replicate the Lead Record into prospect object..
        // Write the Business Logic, to send the Email Alert.

    }
    if(Trigger.isDelete && Trigger.isBefore){

        //Write the Business logic, to be find during Before Deleting the record.
    }
}
```

7. **Trigger.New:** Trigger.New is a List Collection. Which contains the Current context Records. It will hold the Newly inserting records into the collection, for the processing.

   Syntax:

   ```
   List<sobject> Trigger.New
   ```

   Note:

   - Trigger.New will be available in both Insert and Update Operations.
   - Trigger.New will not be available in Delete Operation.

8. **Trigger.Old:** It is a List Collection, which holds the collection of previous context records, which are holding the old values.

   Syntax:

   ```
   List<sobject> Trigger.Old
   ```

   Note:

   - Trigger.Old will not be available in Insert Operation.

- It will be available in Update and Delete Operations.

9. **Trigger.NewMap:** Trigger.NewMap is a collection, which holds the current context records in Map format.

Syntax:

```
Map<ID, SObject> Trigger.NewMap
```

Where:

- Key → Record ID will be the key.
- Value → Whole Record will the value.

Note:

- Trigger.NewMap will not be available in Before Insert Event.
- Trigger.NewMap will be available in **Update** and **After insert** Operations.

10. **Trigger.OldMap:** Trigger.OldMap is Map collection, which holds the previous context records in **Map** format.

Syntax:

```
Map<ID, SObject> Trigger.OldMap
```

Where:

- Key → Record ID will be the key.
- Value → Whole Record will be the value.

Note:

- Trigger.OldMap will be available in Update and Delete Operations.

Trigger Bulkification / Bulkify Trigger:

✓ Upon creating the Trigger, we have to write the Business in such a way, that it can able to handle the processing of either one / more records at a time.

✓ As a best practice, it is always recommended to make the Trigger Bulkify.

✓ We can achieve the Bulkify Trigger, by using FOR LOOP. Which can iterate all the records and process all the records at a time. By using Bulkification, we can make the code Generic.

Example:

```
trigger AccountsTrigger on Account (before insert) {
    if(Trigger.isInsert && Trigger.isBefore){
        for(Account acc : Trigger.New){ // Bulkify...
            if(acc.Rating == 'Hot'){

            }
        }
    }
}
```

Ways to Create the Trigger:

Apex provides 3 ways to create the Triggers.

1. By Using Standard Navigation:

- Click on Setup Menu.
- Goto Build Menu in Left Panel.
- Click on Develop and Expand it.
- Click on Apex Triggers Link
- Click on New Button, to Create a New Trigger.
- Write the Trigger code inside the Editor.
- Click on Save Button.

Note: Once the user click on Save button, it will send the code the Force.com Platform. It will compile the code and save the compiled code in Metadata Repository.

2. By Using Developer Console:

- Goto your name and expand it.
- Click on Developer console link.

- Goto the File Menu and Expand it.
- Click on New Menu item.
- Click on Apex Trigger from submenu.
- Click on Trigger Name inside the Text Box , to be get created.
- Select the Object Name from the Picklist.
- Click on Ok Button.
- Write the Trigger Business logic inside the Editor.
- Click on CTRL + S to save the Trigger code.

3. By Using Eclipse IDE:

Keep Learning | Keep Practicing

Sfdc Telugu

SFDC e-Book Store

https://www.sfdctelugu.in/store

Website | SFDC Telugu

https://www.sfdctelugu.in

For Free PDF Notes | Blogger

https://sfdctelugu.blogspot.com

Deals of the Day | Affiliate

https://sfdctelugu.blogspot.com/p/deals-of-day-shopping-now.html

Follow on : Instagram

https://www.instagram.com/sfdc_telugu/

Follow on: Telegram

https://t.me/sfdc_telugu

Follow on: WhatsApp Group

https://chat.whatsapp.com/CgQGDVLTCiME1fwKgvIA06

Follow on : Twitter

https://twitter.com/sfdc_telugu

Follow on : Facebook Page

https://www.facebook.com/people/Sfdc-Telugu/100087211670093/

Support us:

https://pages.razorpay.com/Buymeacoffe