Silver microsystems
QUALITY. SERVICE. ASSURANCE

Triggers in Salesforce

[1] What is Trigger in Salesforce?
Ans: The trigger is piece of code that executes before or after event on the following operations such as insert, update, delete, undelete etc. trigger enables to perform custom actions before and after modifications to the records of Salesforce.

[2] Write trigger syntax
Ans: **trigger** trigger_name **on** SobjectName (event operation1, event operation 2){
}

Example:

```
trigger prefixName on Account (before Insert, before Update){
}
```

[3] What are different event of trigger
Ans: before event and after event

[4] What are different operations in trigger
Ans: Insert, update, delete, undelete etc.

[5] What is context variables and which are they?
Ans: Context variable is used to bifurcate different operation logics.
OR
All triggers define implicit variables that allow developers to access run-time context. These variables are contained in the **System.Trigger** class.

Here is List of all Trigger Context Variables

- **Trigger.isExecuting**: Returns true if the current context for the Apex code is a trigger, not a Visualforce page, a Web service, or an executeanonymous() API call.
- **Trigger.isInsert**: Returns true if this trigger was fired due to an insert operation, from the Salesforce user interface, Apex, or the API.
- **Trigger.isUpdate**: Returns true if this trigger was fired due to an update operation, from the Salesforce user interface, Apex, or the API.
- **Trigger.isDelete**: Returns true if this trigger was fired due to a delete operation, from the Salesforce user interface, Apex, or the API.
- **Trigger.isBefore**: Returns true if this trigger was fired before any record was saved.
- **Trigger.isAfter**: Returns true if this trigger was fired after all records were saved.
- **Trigger.isUndelete**: Returns true if this trigger was fired after a record is recovered from the Recycle Bin (that is, after an undelete operation from the Salesforce user interface, Apex, or the API.)
- **Trigger.new**: Returns a list of the new versions of the sObject records. This sObject list is only available in insert, update, and undelete triggers, and the records can only be modified in before triggers.
- **Trigger.newMap**: A map of IDs to the new versions of the sObject records. This map is only available in before update, after insert, after update, and after undelete triggers.
- **Trigger.old** : Returns a list of the old versions of the sObject records. This sObject list is only available in update and delete triggers.
- **Trigger.oldMap**: A map of IDs to the old versions of the sObject records. This map is only available in update and delete triggers.
- **Trigger.size**: The total number of records in a trigger invocation, both old and new.

Trigger Context Variables Considerations

- trigger.new and trigger.old cannot be used in Apex DML operations.
- You can use an object to change its own field values using trigger.new, but only in before triggers. In all after triggers, trigger.new is not saved, so a runtime exception is thrown.
- trigger.old is always read-only.
- You cannot delete trigger.new.

[6] Difference between trigger.new and trigger.old
Ans: Triger.new is a command which returns the list of records that have been added recently to the sObjects. To be more precise, those records will be returned which are yet to be saved to the database. Note that this sObject list is only available in insert and update triggers, and the records can only be modified in before triggers. But, Trigger.old returns a list of the old versions of the sObject records. Note that this sObject list is only available in update and delete triggers.

[7] Difference between trigger.new and trigger.newMap
Ans: **Trigger.new**
Returns a list of the new versions of the sObject records. Note that this sObject list is only available in insert and update triggers, and the records can only be modified in before triggers.
Trigger.NewMap
A map of IDs to the new versions of the sObject records, this map is only available in before update, after insert, and after update triggers.

suppose you have a custom object Custom_obj__c,

Trigger.new means it is a List<Custom_obj__c> = trigger.new
And,
Trigger.newMap means it is a map<Id, Custom_obj__c> = trigger.newMap

NOTE : In before insert context your Trigger.NewMap will always be null because in before context records are not submitted to the database, so the **Id** is not generated yet. That's why in before insert we don't use Trigger.NewMap But in After insert, **Id** is generated so we can use Trigger.NewMap
In case of before and after update, the **Id** has already been generated in the insert event. So we can use Trigger.NewMap in before and after update.

[8] Difference between tigger.old and trigger.oldMap
Ans: Trigger.old: Returns a list of the old versions of the sObject records.
Note that this sObject list is only available in the update and delete triggers.

Trigger.oldMap: A map of IDs to the old versions of the sObject records.Note that this map is only available in the update and delete triggers.

suppose you have a custom object Custom_obj__c = trigger.old,

Trigger.old means it is a List<Custom_obj__c> = trigger.oldMap
And,

Trigger.oldMap means it is a map<Id, Custom_obj__c>

[9] Difference between trigger.newMap and trigger.oldMap
Ans: Trigger.NewMap: Trigger.newMap returns map of new records which are trying to insert into Database. This is available in Before Insert, Before Update, After Insert, After Update Triggers and undelete Triggers. This list of records can only modified in Before triggers.

Trigger.OldMap: Trigger.oldMap returns map of old records which are updated with new values. These List of records already there in Database. Trigger.oldmap available in Before update, after update, Before Delete and After Delete triggers.

[10] What is handler and helper in trigger
Ans: To make a trigger logic less, we use handler and helper apex class. This also helps to better code readability and code re-usability.

[11] Best practice of triggers
Ans:
1) One Trigger Per Object
A single Apex Trigger is all you need for one particular object. If you develop multiple Triggers for a single object, you have no way of controlling the order of execution if those Triggers can run in the same contexts.

2) Logic-less Triggers
If you write methods in your Triggers, those can't be exposed for test purposes. You also can't expose logic to be re-used anywhere else in your org.

3) Context-Specific Handler Methods
Create context-specific handler methods in Trigger handlers.

4) Bulkify your Code
Bulkifying Apex code refers to the concept of making sure the code properly handles more than one record at a time.

5) Avoid SOQL Queries or DML statements inside FOR Loops
An individual Apex request gets a maximum of 100 SOQL queries before exceeding that governor limit. So if this trigger is invoked by a batch of more than 100 Account records, the governor limit will throw a runtime exception

6) Using Collections, Streamlining Queries, and Efficient For Loops
It is important to use Apex Collections to efficiently query data and store the data in memory. A combination of using collections and streamlining SOQL queries can substantially help writing efficient Apex code and avoid governor limits

7) Querying Large Data Sets
The total number of records that can be returned by SOQL queries in a request is 50,000. If returning a large set of queries causes you to exceed your heap limit, then a SOQL query for loop must be used instead. It can process multiple batches of records through the use of internal calls to query and queryMore.

8) Use @future Appropriately
It is critical to write your Apex code to efficiently handle bulk or many records at a time. This is also true for asynchronous Apex methods (those annotated with the @future keyword). The differences between synchronous and asynchronous Apex can be found.

9) Avoid Hardcoding IDs
When deploying Apex code between sandbox and production environments, or installing Force.com AppExchange packages, it is essential to avoid hardcoding IDs in the Apex code. By doing so, if the record IDs change between environments, the logic can dynamically identify the proper data to operate against and not fail.

Few more Best Practices for Triggers
There should only be one trigger for each object.
Avoid complex logic in triggers. To simplify testing and resuse, triggers should delegate to Apex classes which contain the actual execution logic. See Mike Leach's excellent trigger template for more info.
Bulkify any "helper" classes and/or methods.
Trigers should be "bulkified" and be able to process up to 200 records for each call.
Execute DML statements using collections instead of individual records per DML statement.
Use Collections in SOQL "WHERE" clauses to retrieve all records back in single query
Use a consistent naming convention including the object name (e.g., AccountTrigger)

[12] What is recursive trigger ? How to avoid it.
Ans: Recursion occurs when same code is executed again and again. It can lead to infinite loop and which can result to governor limit sometime. Sometime it can also result in unexpected output.
OR
When a trigger calls to itself again and again, then this concept is known as trigger (In this case: trigger calls itself at 16 Times and then throw recursion exception).

To avoid recursion, use a static Boolean flag variable.

Example :

```
public class RecursiveTriggerHandler{
    public static Boolean isFirstTime = true;
}

trigger SampleTrigger on Contact (after update){

   Set<String> accIdSet = new Set<String>();

   if(RecursiveTriggerHandler.isFirstTime){
      RecursiveTriggerHandler.isFirstTime = false;

      Contact c = new Contact();
c.LastName = 'Disney';
insert c;
      }

      // Use accIdSet in some way
   }
}
```

[13] What is order of execution in salesforce
Ans: When you save a record with an insert, update, or upsert statement, Salesforce performs the following events in order.

- The original record is loaded from the database.
- System Validation Rules.
- Executes all before triggers.
- Custom Validation rules.
- Executes duplicate rules.
- Saves the record to the database, but doesn't commit yet.
- Executes all after triggers.
- Executes assignment rules.
- Executes auto-response rules.
- Executes workflow rules.
- If there are workflow field updates, updates the record again.
- If the record was updated with workflow field updates, fires before and after triggers one more time. Custom validation rules, duplicate rules, and escalation rules are not run again.
- Executes processes and flows launched via processes and flow trigger workflow actions.
- Executes escalation rules.
- Executes entitlement rules.
- If the record contains a roll-up summary field or is part of a cross-object workflow, performs calculations and updates the roll-up summary field in the parent record. Parent record goes through save procedure.
- If the parent record is updated, and a grandparent record contains a roll-up summary field or is part of a cross-object workflow, performs calculations and updates the roll-up summary field in the grandparent record. Grandparent record goes through save procedure.

- Executes Criteria Based Sharing evaluation.
- Commits all DML operations to the database.
- Executes post-commit logic, such as sending email.

Work on All following triggers for better practice :

Scenario 1 : When we are trying to insert new record into object. If there is any record existing with same account name it should prevent duplicate record.

Scenario 2: Write a trigger to prefix Account Name with 'Mr' when new record is inserted.

Scenario 3: Whenever a new record is created into account object . Before this new record is inserted into Account, delete all the contacts records with this account name.

Scenario 4: Whenever a new transaction is performed successfully then update the customer object balance field based on
If Transaction Type=Deposit, Balance= balance +amount ; withdraw balance= balance-amount;
Note: Customers and Transaction has lookup Detail Relation.

Scenario 5: Whenever a new contact is created for account update the corresponding account phone with the new contact phone field.

Scenario 6: Whenever customer record is updated, before updating the record create new record in test object with old values of customer record.

Scenario 7: To update the owner of a case based on the values selected within a picklist and populate the owner field with the created by field data. When we have selected any Field name=Status
Picklist Values=
priced-(Initial)
Priced-(Re-priced)
Price(File Loaded)

Scenario 8: Write a trigger that will prevent a user from creating a lead that already exists as a contact. We will use the lead /contact email address to detect duplicates.
Lead is created or updated.

1. Lead has an email address.
2. 2. Try to find a matching contact based on email address.(Using SOQL)
3. If a match is found give an error
4. If a match is not found do nothing.

Scenario 9: When we are trying to delete customer record delete all the corresponding child records from test object, where customer is lookup field in the object.

Scenario 10: When we create the opportunity with probability =50% then the opportunity owner will be automatically added to Account Team of the associated account for the opportunity.

Scenario 11: Invoking the apex class from the trigger. Whenevr new customers record is created /updated then income tax should be calculated based on salary and should be updated to field income tax (Currency field).

Scenario 12: If we delete any of the existing customer record then first create new test object record with customer record values then delete customer record.

Scenario 13: Whenever we try to update the phone of account record then update the related contact phone number with the new Account phone number before account record is updated.
When we delete the account record then delete the corresponding contact records.

Scenario 14: Suppose there is scenario where one trigger perform update operation, which results in invocation of second trigger and the update operation in second trigger acts as triggering criteria for trigger one.

Scenario 15: Write a trigger which shows recursive trigger error.

Scenario 16: Create 'Sales Rep' field with datatype (Text) on the account object. When we create account record, the account owner will be automatically added to Sales Rep field. When we update the Account owner of the record, then also the Sales Rep Will be automatically updated.

Scenario 17: Create the field Called 'Contact Relationship' checkbox on the contact object and create the object called "Contact Relationship" which is related list to the contact.(Lookup Relationship).
Now logic is when we create Contact by checking Contact Relationship checkbox then contact relationship will be created automatically for that contact.

Scenario 18: When we change the owner of the Contact Relationship, then the owner name will be automatically populated in the Contact Relationship name field.

Scenario 19: Create the field Called 'Contact Relationship' checkbox on the contact object and create the object called "Contact Relationship" which is related list to the contact.(Lookup Relationship).
When we delete the contact then contact Relationship will be deleted automatically.

Scenario 20: Create the field Called 'Contact Relationship' checkbox on the contact object and create the object called "Contact Relationship" which is related list to the contact.(Lookup Relationship).
When we undelete the contact then contact Relationship will be Undeleted automatically.

Scenario 21: Create field Called 'Count of Contacts' on Account object. When we add the contacts for that Account then count will populate in the field on Account details page. When we delete the contacts for that account then count will update automatically.
Note: The above logic will be applicable when we have lookup relationship. But when we have the master detail relationship, then we can create Rollup Summary field to get the count of child records using "Count" function.

Scenario 22: Trigger Scenario (Global Logic Software Private Limited)
There are two following object and given fields having lookup relationship,

Object : CAR
Make
Model
Car Number

Object Name : Reservation
Select CAR
From Date
To Date

Writer a trigger that should prevent duplication reservation of a car.
(Example : If a car is reserved for the duration 15 May 2020 to 20 May 2020, then following reservation should **not** be allowed)
[1] 14 May to 2020 to 16 May 2020
[2] 14 May to 21 May
[3] 15 May to 20 May
[4] 15 May to 21 May
[5] 16 May to 18 May
[6] 17 May to 22 May

[14] Can you edit an apex trigger/ apex class in production environment? Can you edit a Visualforce page in production environment?
Ans: No, it is not possible to edit apex classes and triggers directly in production environment. It needs to be done first in Developer edition or testing org or in Sandbox org. Then, to deploy it in production, a user with Author Apex permission must deploy the triggers and classes using deployment tools. However, Visualforce pages can be created and edited in both sandbox and in production. Only if the page has to do something unique (different values), it would have to be developed via Sandbox.

[15] Where you used a trigger in your project?
Ans: Tell appropriate trigger you have written in your project.