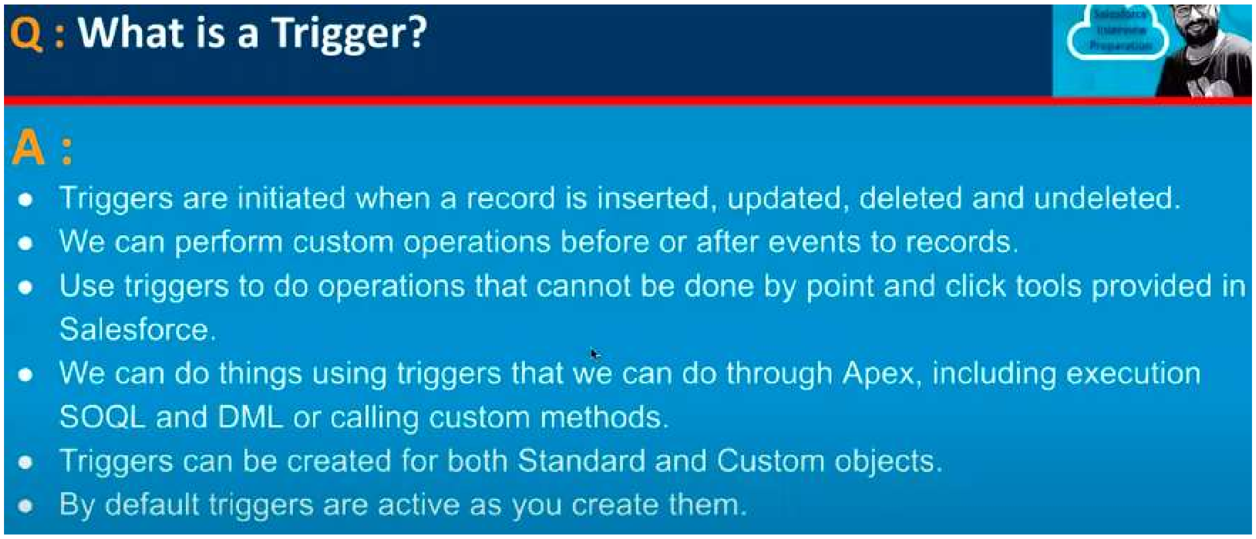
1.

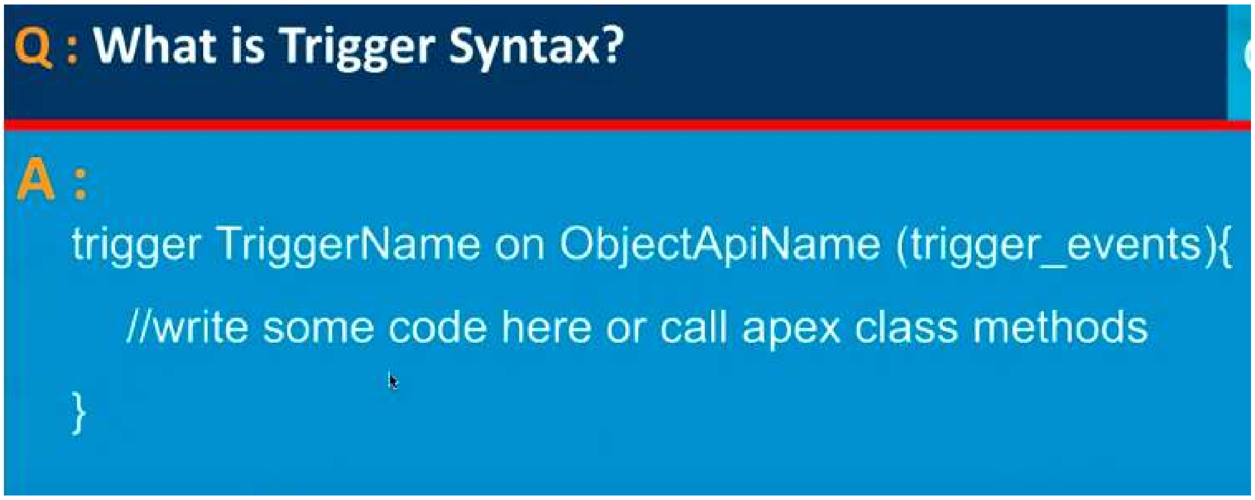


Q : What is a Trigger?

A :

- Triggers are initiated when a record is inserted, updated, deleted and undeleted.
- We can perform custom operations before or after events to records.
- Use triggers to do operations that cannot be done by point and click tools provided in Salesforce.
- We can do things using triggers that we can do through Apex, including execution SOQL and DML or calling custom methods.
- Triggers can be created for both Standard and Custom objects.
- By default triggers are active as you create them.

2.

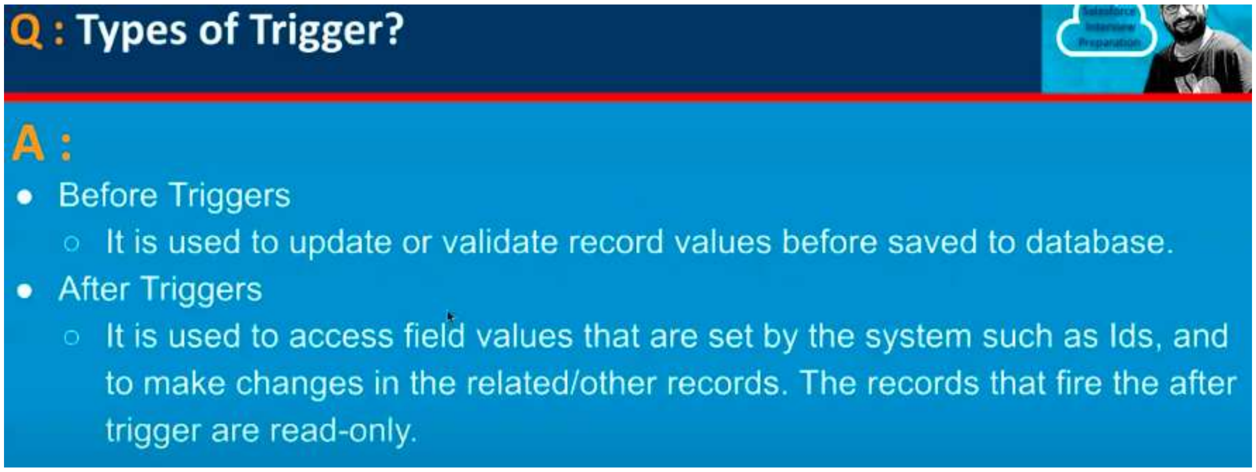


Q : What is Trigger Syntax?

A :

```
trigger TriggerName on ObjectApiName (trigger_events){
    //write some code here or call apex class methods
}
```

3.

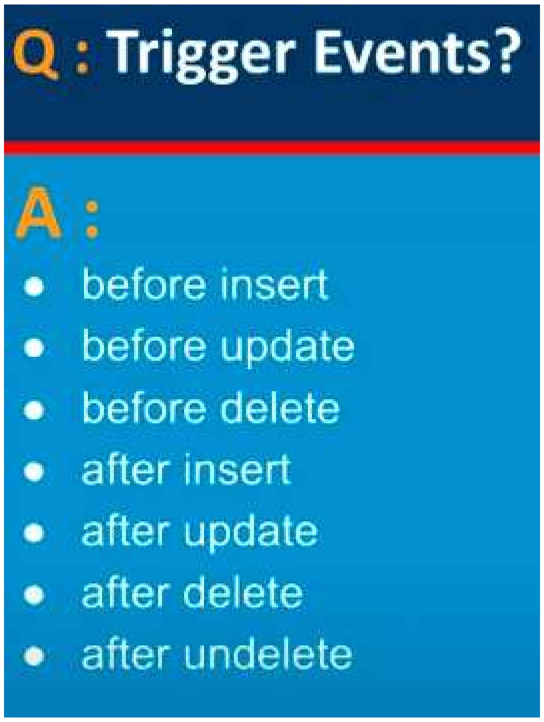


Q : Types of Trigger?

A :

- Before Triggers
  - It is used to update or validate record values before saved to database.
- After Triggers
  - It is used to access field values that are set by the system such as Ids, and to make changes in the related/other records. The records that fire the after trigger are read-only.

4.

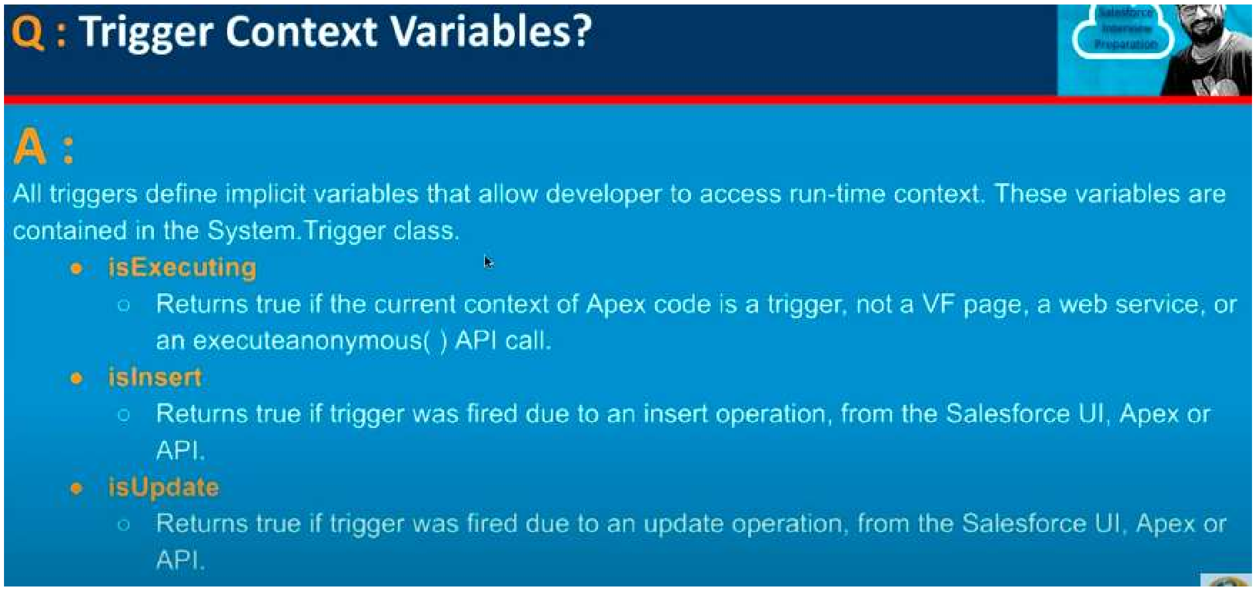


Q : Trigger Events?

A :

- before insert
- before update
- before delete
- after insert
- after update
- after delete
- after undelete

5.



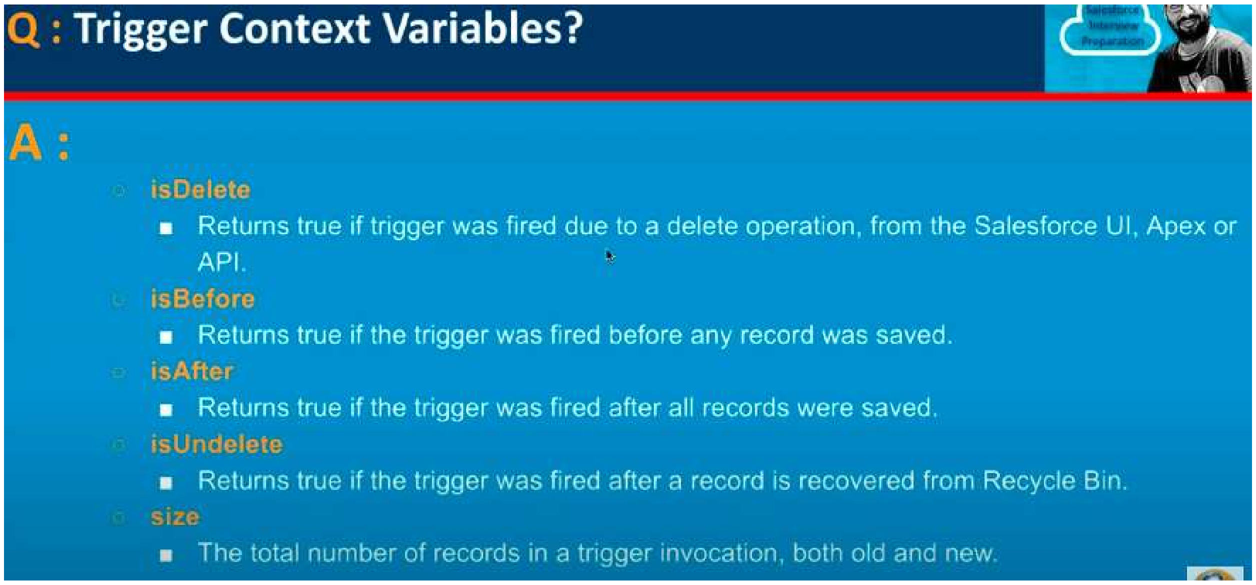
Q : Trigger Context Variables?

A :

All triggers define implicit variables that allow developer to access run-time context. These variables are contained in the System.Trigger class.

- **isExecuting**
  - Returns true if the current context of Apex code is a trigger, not a VF page, a web service, or an executeanonymous( ) API call.
- **isInsert**
  - Returns true if trigger was fired due to an insert operation, from the Salesforce UI, Apex or API.
- **isUpdate**
  - Returns true if trigger was fired due to an update operation, from the Salesforce UI, Apex or API.

6.

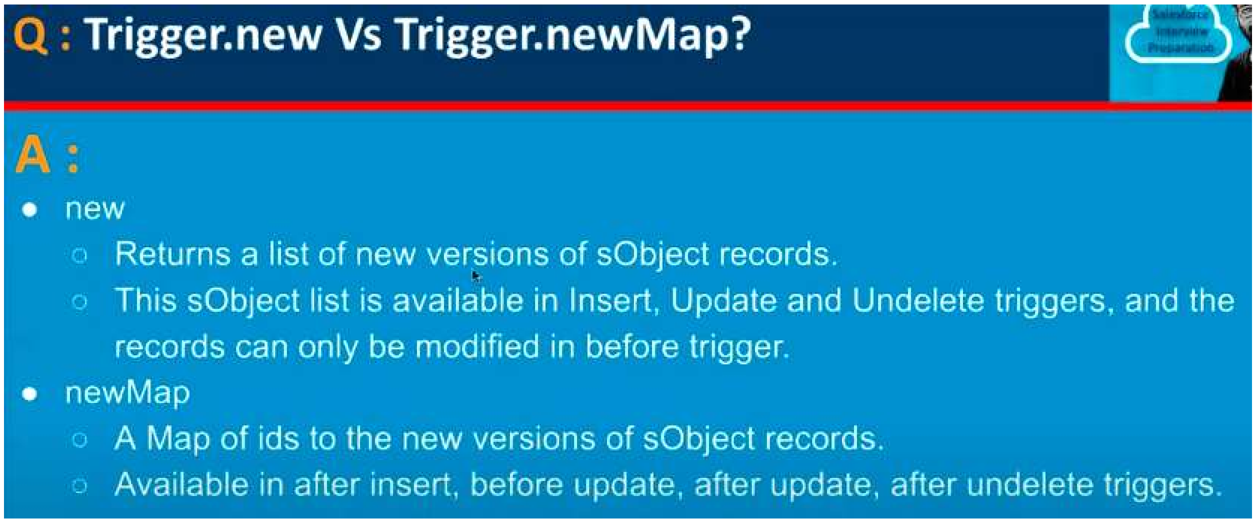


Q : Trigger Context Variables?

A :

- **isDelete**
  - Returns true if trigger was fired due to a delete operation, from the Salesforce UI, Apex or API.
- **isBefore**
  - Returns true if the trigger was fired before any record was saved.
- **isAfter**
  - Returns true if the trigger was fired after all records were saved.
- **isUndelete**
  - Returns true if the trigger was fired after a record is recovered from Recycle Bin.
- **size**
  - The total number of records in a trigger invocation, both old and new.

7.

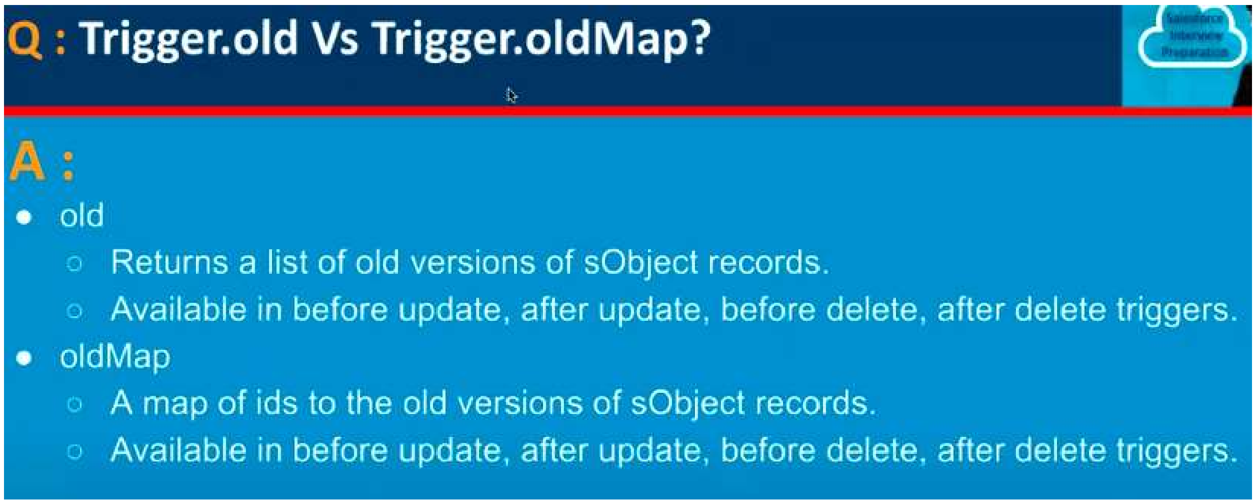


Q : Trigger.new Vs Trigger.newMap?

A :

- new
  - Returns a list of new versions of sObject records.
  - This sObject list is available in Insert, Update and Undelete triggers, and the records can only be modified in before trigger.
- newMap
  - A Map of ids to the new versions of sObject records.
  - Available in after insert, before update, after update, after undelete triggers.

8.

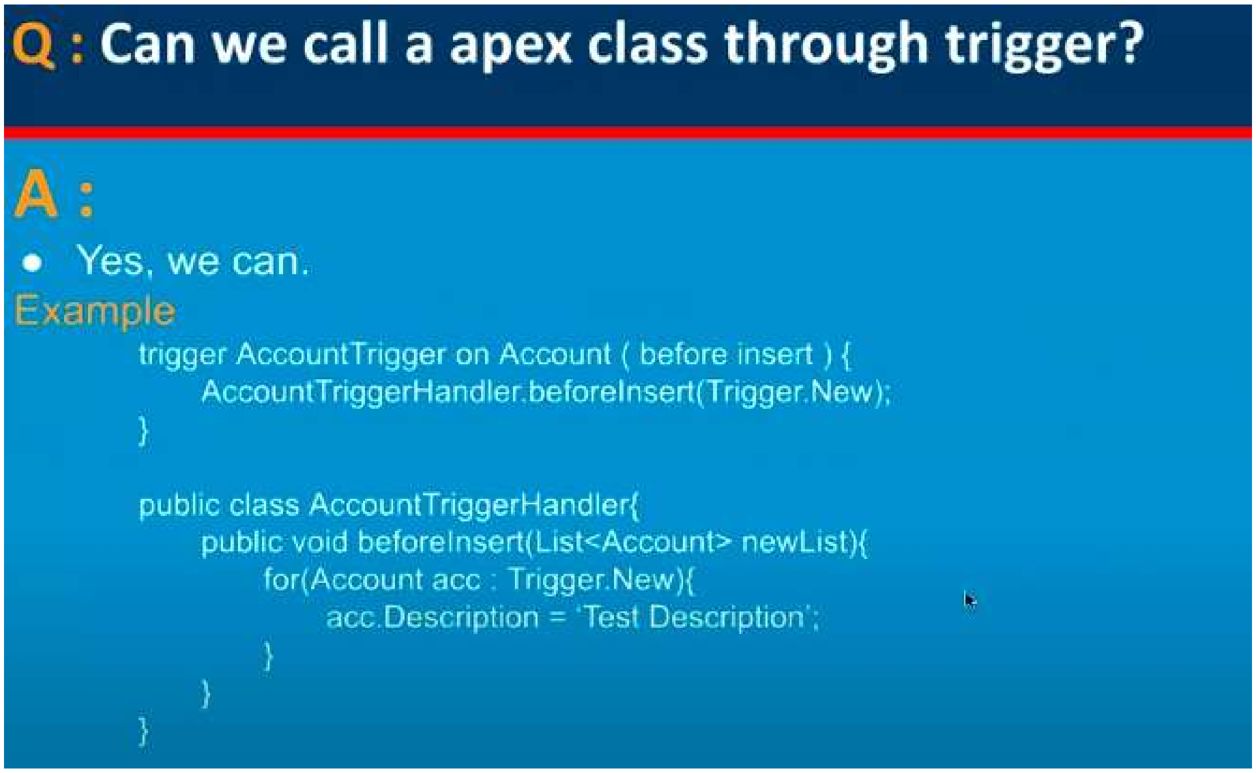


Q : Trigger.old Vs Trigger.oldMap?

A :

- old
  - Returns a list of old versions of sObject records.
  - Available in before update, after update, before delete, after delete triggers.
- oldMap
  - A map of ids to the old versions of sObject records.
  - Available in before update, after update, before delete, after delete triggers.

9.



Q : Can we call a apex class through trigger?

A :

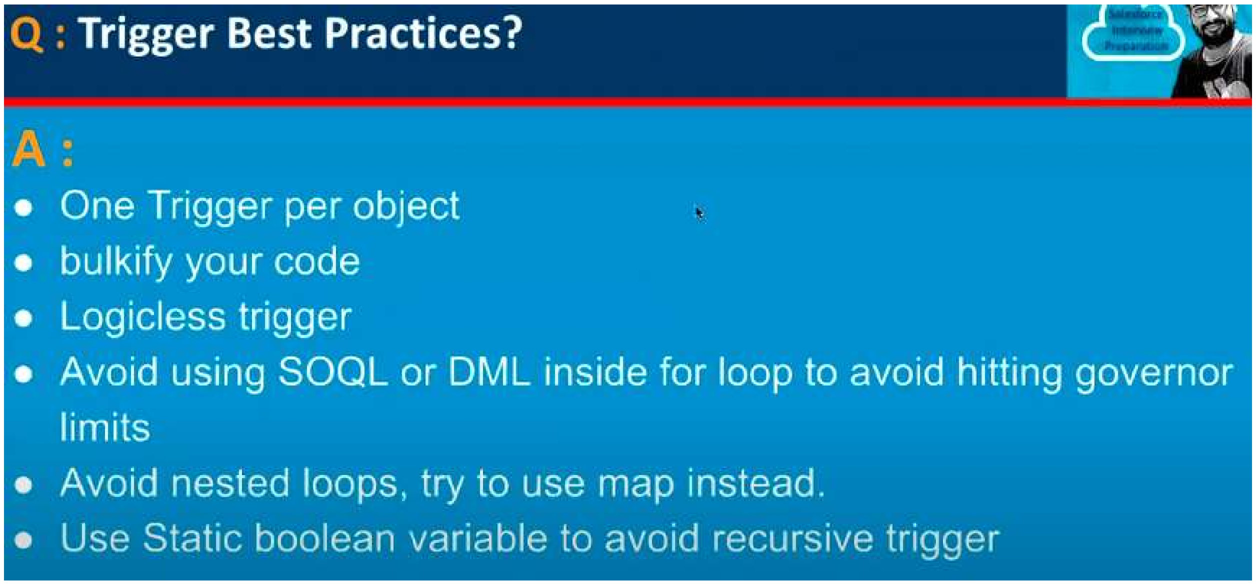
- Yes, we can.

Example

```
trigger AccountTrigger on Account ( before insert ) {
        AccountTriggerHandler.beforeInsert(Trigger.New);
}

public class AccountTriggerHandler{
        public void beforeInsert(List<Account> newList){
                for(Account acc : Trigger.New){
                        acc.Description = 'Test Description';
                }
        }
}
```

10.

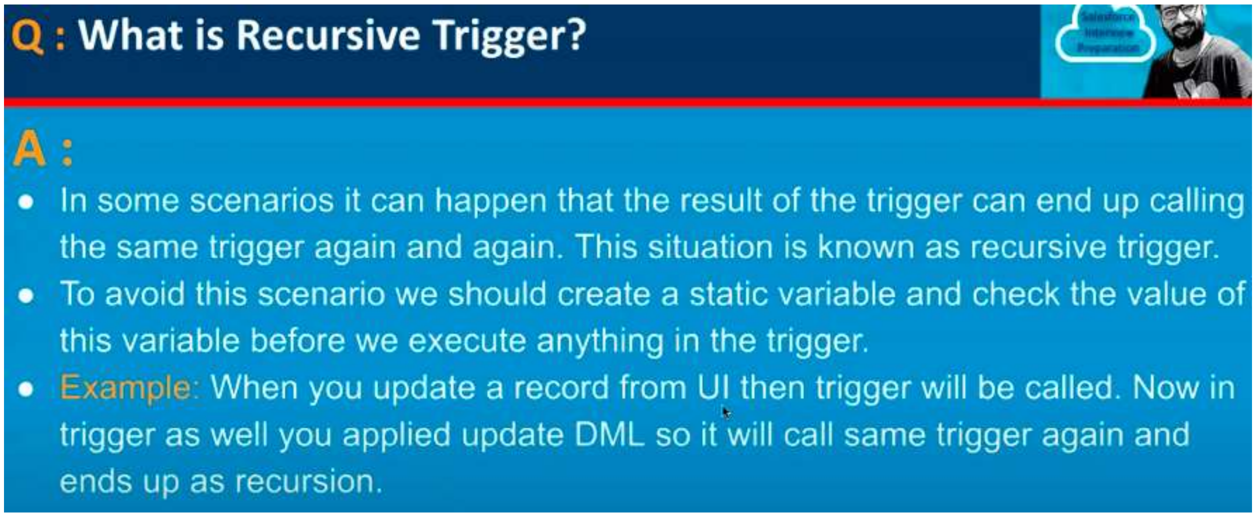


Q : Trigger Best Practices?

A :

- One Trigger per object
- bulkify your code
- Logicless trigger
- Avoid using SOQL or DML inside for loop to avoid hitting governor limits
- Avoid nested loops, try to use map instead.
- Use Static boolean variable to avoid recursive trigger

11.

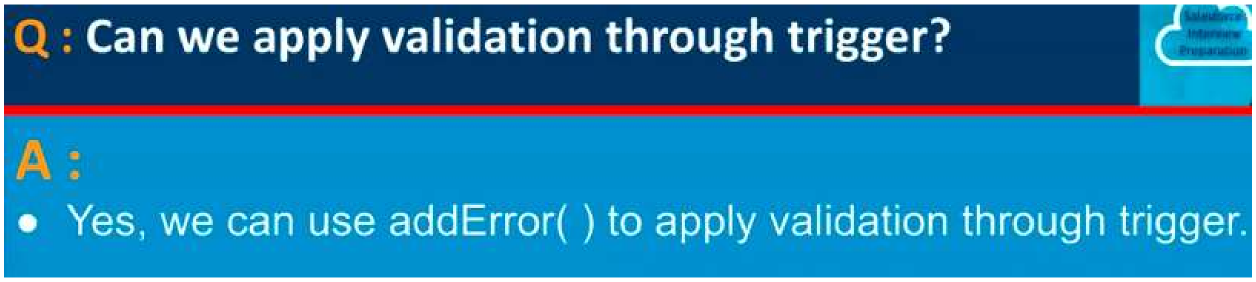


Q : What is Recursive Trigger?

A :

- In some scenarios it can happen that the result of the trigger can end up calling the same trigger again and again. This situation is known as recursive trigger.
- To avoid this scenario we should create a static variable and check the value of this variable before we execute anything in the trigger.
- Example: When you update a record from UI then trigger will be called. Now in trigger as well you applied update DML so it will call same trigger again and ends up as recursion.

12.

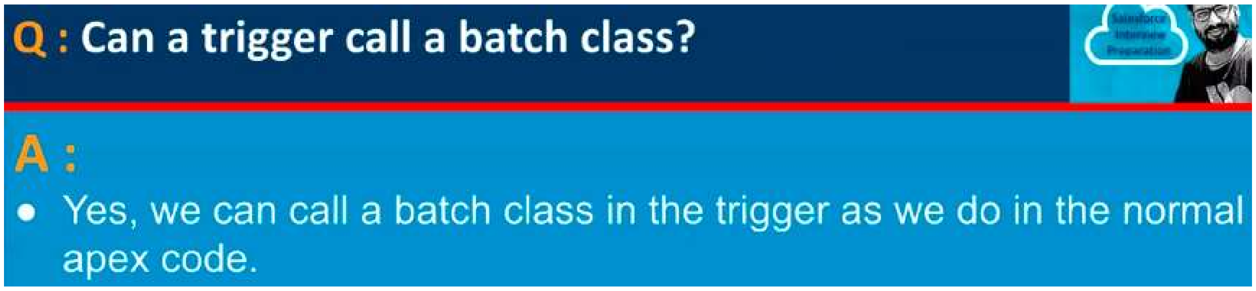


Q : Can we apply validation through trigger?

A :

- Yes, we can use addError( ) to apply validation through trigger.

13.

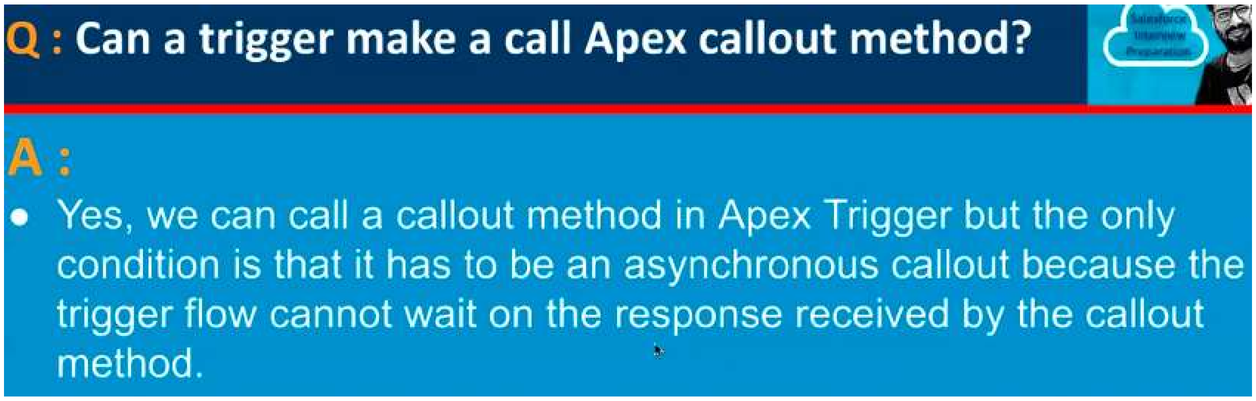


Q : Can a trigger call a batch class?

A :

- Yes, we can call a batch class in the trigger as we do in the normal apex code.

14.

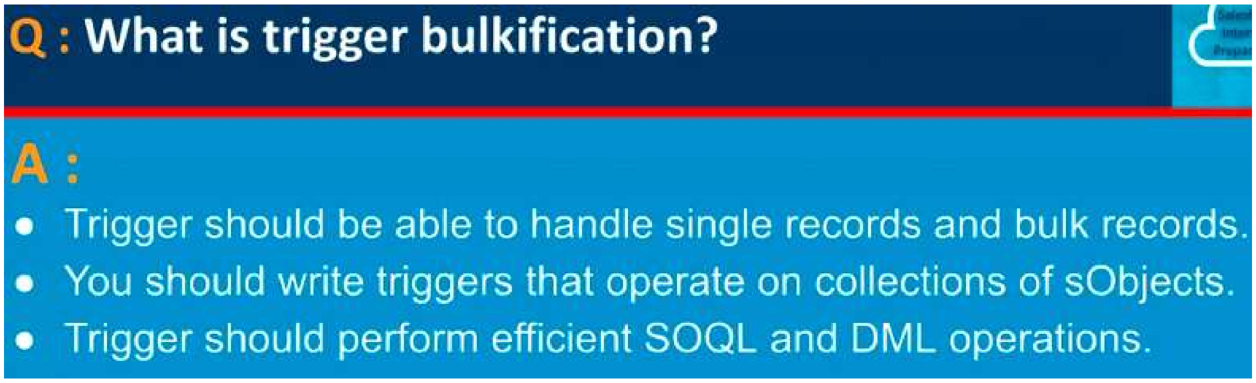


Q : Can a trigger make a call Apex callout method?

A :

- Yes, we can call a callout method in Apex Trigger but the only condition is that it has to be an asynchronous callout because the trigger flow cannot wait on the response received by the callout method.

15.

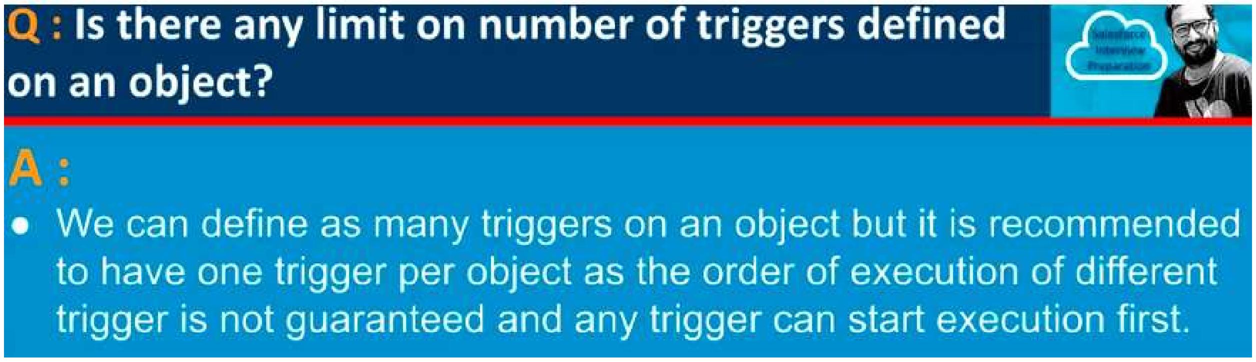


Q : What is trigger bulkification?

A :

- Trigger should be able to handle single records and bulk records.
- You should write triggers that operate on collections of sObjects.
- Trigger should perform efficient SOQL and DML operations.

16.

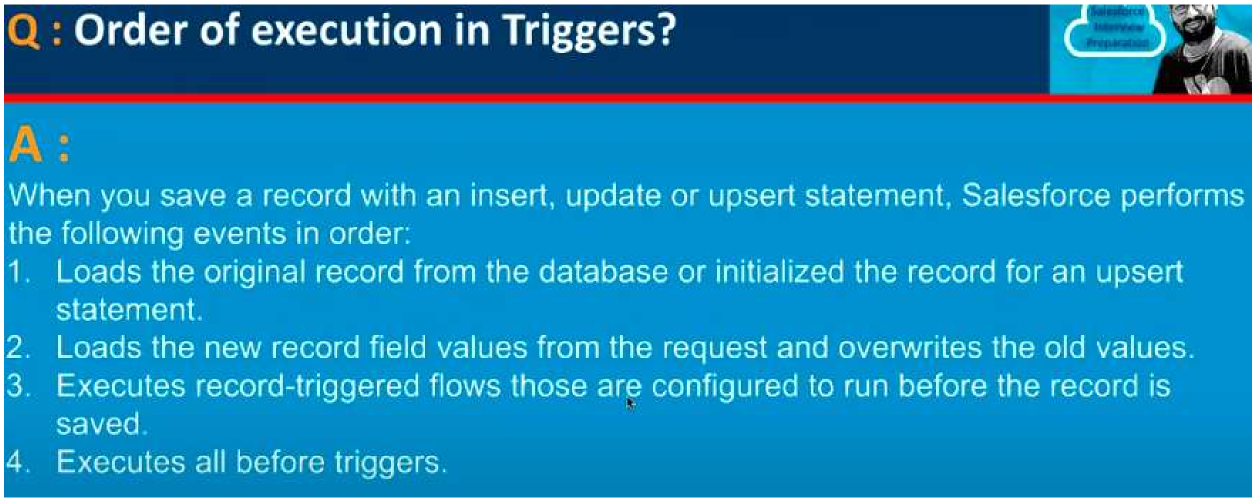


Q : Is there any limit on number of triggers defined on an object?

A :

- We can define as many triggers on an object but it is recommended to have one trigger per object as the order of execution of different trigger is not guaranteed and any trigger can start execution first.

17.

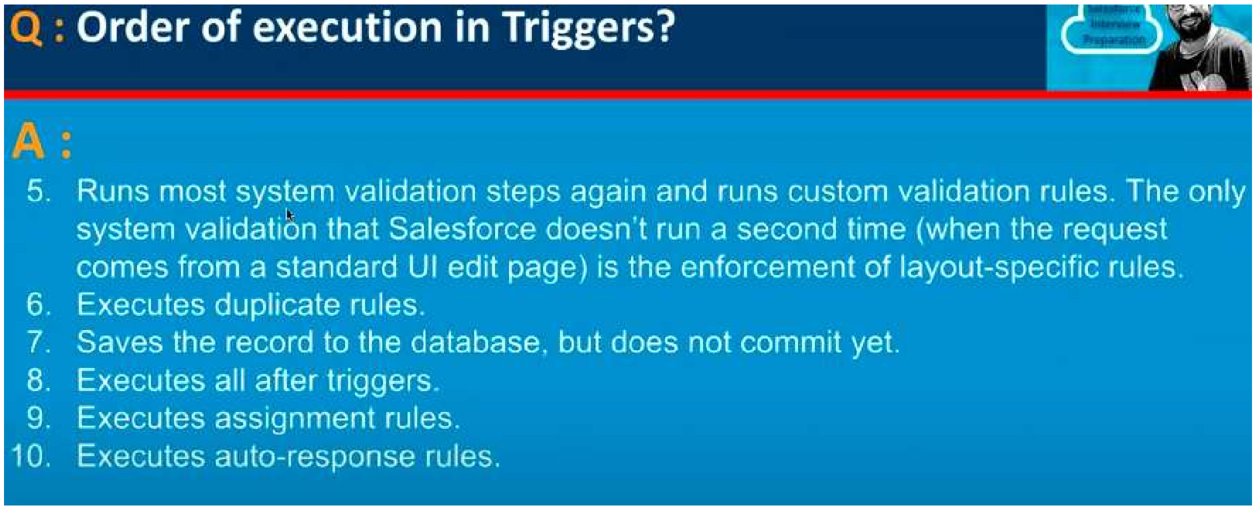


Q : Order of execution in Triggers?

A :

When you save a record with an insert, update or upsert statement, Salesforce performs the following events in order:

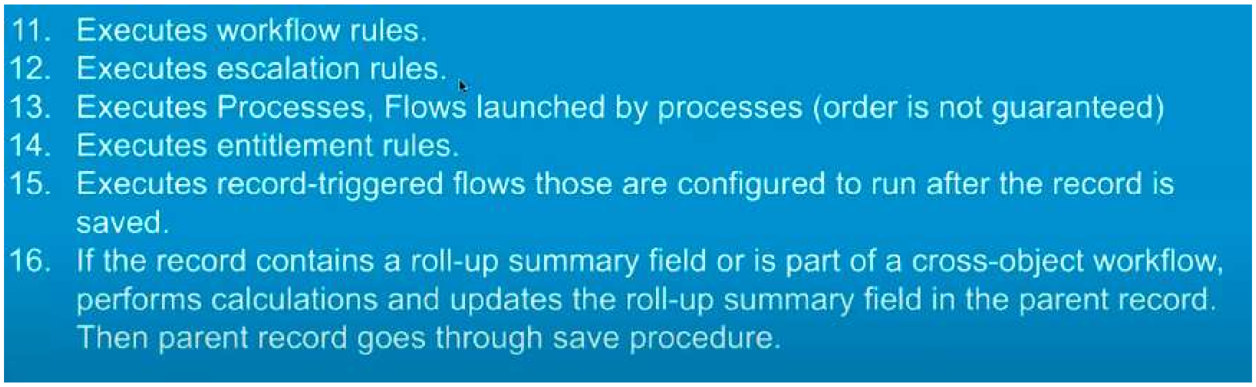
1. Loads the original record from the database or initialized the record for an upsert statement.
2. Loads the new record field values from the request and overwrites the old values.
3. Executes record-triggered flows those are configured to run before the record is saved.
4. Executes all before triggers.



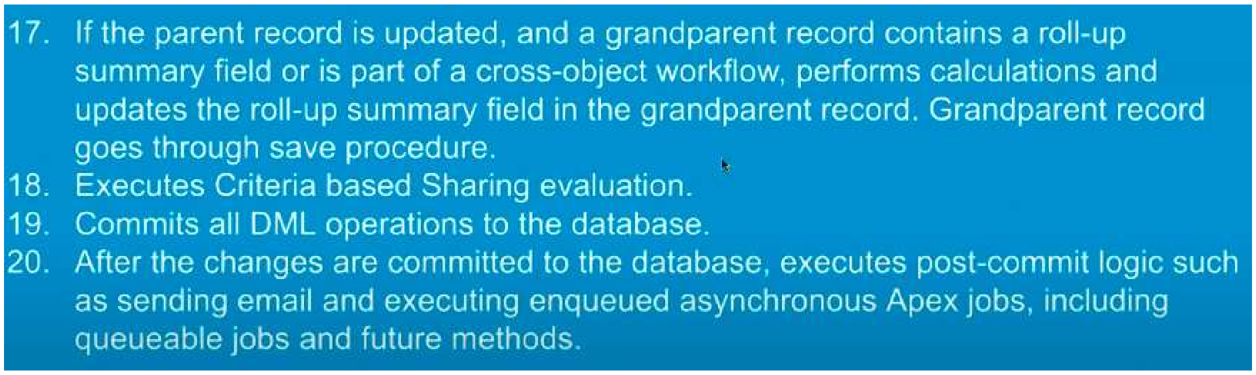
Q : Order of execution in Triggers?

A :

5. Runs most system validation steps again and runs custom validation rules. The only system validation that Salesforce doesn't run a second time (when the request comes from a standard UI edit page) is the enforcement of layout-specific rules.
6. Executes duplicate rules.
7. Saves the record to the database, but does not commit yet.
8. Executes all after triggers.
9. Executes assignment rules.
10. Executes auto-response rules.



11. Executes workflow rules.
12. Executes escalation rules.
13. Executes Processes, Flows launched by processes (order is not guaranteed)
14. Executes entitlement rules.
15. Executes record-triggered flows those are configured to run after the record is saved.
16. 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. Then parent record goes through save procedure.



17. 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.
18. Executes Criteria based Sharing evaluation.
19. Commits all DML operations to the database.
20. After the changes are committed to the database, executes post-commit logic such as sending email and executing enqueued asynchronous Apex jobs, including queueable jobs and future methods.