**Triggers:**

Triggers are Apex code that execute before or after an insert, update, delete or undelete event occurs on an sObject. These are like classes with a particular syntax that lets you specify when they should run, depending on how a database record is modified

Trigger Syntax:

trigger TriggerName on ObjectName (trigger\_events) {

code\_block

}

**Trigger Events:** You can specify multiple trigger events in a comma-separated list if you want the trigger to execute before or after insert, update, delete, and undelete operations. The events you can specify are:

before insert

before update

before delete

after insert

after update

after delete

after undelete

Trigger example:

trigger RestrictInvoiceDeletion on Invoice\_Statement\_\_c (before delete) {

for (Invoice\_Statement\_\_c invoice : [SELECT Id FROM Invoice\_Statement\_\_c WHERE Id IN (SELECT Invoice\_Statement\_\_c FROM Line\_Item\_\_c) AND Id IN :Trigger.old])

{

Trigger.oldMap.get(invoice.Id).addError( 'Cannot delete invoice statement with line items');

}

}

The above example shows that The trigger fires before one or more Invoice\_Statement\_\_c sObjects are deleted. This behavior is specified by the before delete parameter. The trigger contains a SOQL for loop that iterates over the invoice statements with line items. The query checks whether the Id values of the invoice statements are part of the set of parent invoice statements that are returned by the nested query. The second condition restricts the set of invoice statements queried to the ones that this trigger targets. This is done by checking whether each Id value is contained in Trigger.old. Trigger.old contains the set of old records that are to be deleted but haven’t been deleted yet. For each invoice statement that meets the conditions, the trigger adds a custom error message by using the addError method, which prevents the deletion of the record. This custom error message appears in the user interface when you attempt to delete an invoice statement with line items.

**Invoking the Trigger**:

The trigger fires before the deletion of invoice statements, so you can cause it to fire by deleting an invoice statement either through the user interface or programmatically.

**Apex Unit tests:**

Writing unit tests for your code is fundamental to developing Apex code. You must have 75% test coverage to be able to deploy your Apex code to your production organization. In addition, the tests counted as part of the test coverage must pass. Testing is key to ensuring the quality of your application.

By default, Apex test methods don’t have access to pre-existing data in the organization. You must create your own test data for each test method. In this way, tests won’t depend on organization data and won’t fail because of missing data when the data it depends on no longer exists. Test data isn’t committed to the database and is rolled back when the test execution completes. This means that you don’t have to delete the data that is created in the tests. When the test finishes execution, the data created during test execution won’t be persisted in the organization and won’t be available. You can create test data either in your test method or you can write utility test classes containing methods for test data creation that can be called by other tests. There are some objects that tests can still access in the organization. They’re metadata objects and objects used to manage your organization, such as User or Profile.

Example of Test class:

@isTest

public class TestDataFactory {

public static Invoice\_Statement\_\_c createOneInvoiceStatement( Boolean withLineItem) {

Invoice\_Statement\_\_c testInvoice = createInvoiceStatement();

if (withLineItem == true) {

Merchandise\_\_c m = createMerchandiseItem('Orange juice');

AddLineItem(testInvoice, m);

}

return testInvoice;

}

//Helper methods

private static Merchandise\_\_c createMerchandiseItem(String merchName) {

Merchandise\_\_c m = new Merchandise\_\_c(

Name=merchName,

Description\_\_c='Fresh juice',

Price\_\_c=2,

Total\_Inventory\_\_c=1000);

insert m; return m;

}

private static Invoice\_Statement\_\_c createInvoiceStatement() {

Invoice\_Statement\_\_c inv = new Invoice\_Statement\_\_c( Description\_\_c='Test Invoice');

insert inv; return inv;

}

private static Line\_Item\_\_c AddLineItem(Invoice\_Statement\_\_c inv, Merchandise\_\_c m) {

Line\_Item\_\_c lineItem = new Line\_Item\_\_c(

Invoice\_Statement\_\_c = inv.Id,

Merchandise\_\_c = m.Id,

Unit\_Price\_\_c = m.Price\_\_c,

Units\_Sold\_\_c = (Double)(10\*Math.random()+1));

insert lineItem;

return lineItem;

}

}