Apex Triggers

Get Started with Apex Triggers

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



List<Account> acclst=new List<Account>();
for(account a:trigger.new){
if(a.Match_Billing_Address__c==true && a.BillingPostalCode!=null){
a.ShippingPostalCode=a.BillingPostalCode;
}

}
}
```

2. Bulk Apex Triggers

```
trigger ClosedOpportunityTrigger on Opportunity (after insert, after update) {

List<Task> taskList = new List<Task>();
for(Opportunity opp : Trigger.new) {
 //Only create Follow Up Task only once when Opp StageName is to 'Closed Won' on Create
if(Trigger.isInsert) {
```

```
if(Opp.StageName == 'Closed Won') {

taskList.add(new Task(Subject = 'Follow Up Test Task', WhatId = opp.Id));

}

}

 //Only create Follow Up Task only once when Opp StageName changed to 'Closed Won' on Update

if(Trigger.isUpdate) {

if(Opp.StageName == 'Closed Won'

&& Opp.StageName != Trigger.oldMap.get(opp.Id).StageName) {

taskList.add(new Task(Subject = 'Follow Up Test Task', WhatId = opp.Id));

}

}

}


if(taskList.size()>0) {

insert taskList;

}

}
```

Apex Testing

1. *Get Started with Apex Unit Tests*

VerifyDate class :

```
public class VerifyDate {
//method to handle potential checks against two dates
public static Date CheckDates(Date date1, Date date2) {
//if date2 is within the next 30 days of date1, use date2. Otherwise use the end of the month
if(DateWithin30Days(date1,date2)) {
return date2;
} else {
return SetEndOfMonthDate(date1);
}
}
//method to check if date2 is within the next 30 days of date1
private static Boolean DateWithin30Days(Date date1, Date date2) {
//check for date2 being in the past
if( date2 < date1) { return false; }
//check that date2 is within (>=) 30 days of date1
Date date30Days = date1.addDays(30); //create a date 30 days away from date1
if( date2 >= date30Days ) { return false; }
else { return true; }
}

//method to return the end of the month of a given date
```

```
private static Date SetEndOfMonthDate(Date date1) {
Integer totalDays = Date.daysInMonth(date1.year(), date1.month());
Date lastDay = Date.newInstance(date1.year(), date1.month(), totalDays);
return lastDay;
}

}
```

TestVerifyDate:

```
@isTest public class
TestVerifyDate
{
static testMethod void testMethod1()
{
Date d = VerifyDate.CheckDates(System.today(),System.today()+1);
Date d1 = VerifyDate.CheckDates(System.today(),System.today()+60);  }
}
```

2. Test Apex Triggers

RestrictContactByName :

```
trigger RestrictContactByName on Contact (before insert, before update) {

//check contacts prior to insert or update for invalid data

For (Contact c : Trigger.New) {

if(c.LastName == 'INVALIDNAME') {  //invalidname is invalid

c.AddError('The Last Name "'+c.LastName+'" is not allowed for DML');

}

}

}
```

```
TestRestrictContactByName :

        @isTest          private          class
TestRestrictContactByName {


static testMethod void  metodoTest()

{

List<Contact> listContact= new List<Contact>();

    Contact    c1    =    new    Contact(FirstName='Francesco',    LastName='Riggio'    ,
email='Test@test.com');

Contact c2 = new Contact(FirstName='Francesco1', LastName =

'INVALIDNAME',email='Test@test.com');

                              listContact.add(c1);
```

```
listContact.add(c2);

Test.startTest();

try

{

insert listContact;

}

catch(Exception ee)

{

}

Test.stopTest();

}

}
```

3.Create Test Data for Apex Tests

RandomContactFactory class :

```
//@isTest public class RandomContactFactory {

public static List<Contact> generateRandomContacts(Integer numContactsToGenerate, String FName) {

List<Contact> contactList = new List<Contact>();

for(Integer i=0;i<numContactsToGenerate;i++) {

Contact c = new Contact(FirstName=FName + ' ' + i, LastName = 'Contact '+i);
```

```
contactList.add(c);
System.debug(c);
}
//insert contactList;
System.debug(contactList.size());
return contactList;
}

}
```

Asynchronous Apex

1.Use Future Methods

```
public class AccountProcessor {
@future
public static void countContacts(List<Id> accountIds){
List<Account> accounts = [Select Id, Name from Account Where Id IN : accountIds];
List<Account> updatedAccounts = new List<Account>();
for(Account account : accounts){
  account.Number_of_Contacts__c = [Select count() from Contact Where AccountId =: account.Id];
    System.debug('No Of Contacts = ' + account.Number_of_Contacts__c);
updatedAccounts.add(account);
```

```
}
update updatedAccounts;
}

}
```

test class///

```
@isTest public class
AccountProcessorTest {

@isTest
public static void testNoOfContacts(){
Account a = new Account();
a.Name = 'Test Account';
Insert a;
Contact c = new Contact();
c.FirstName = 'Bob';
c.LastName =  'Willie';
c.AccountId = a.Id;
Contact c2 = new Contact();
c2.FirstName = 'Tom';
c2.LastName = 'Cruise';
c2.AccountId = a.Id;
```

```
List<Id> acctIds = new List<Id>();
acctIds.add(a.Id);
Test.startTest();
AccountProcessor.countContacts(acctIds);
Test.stopTest();
}

}
```

2.Use Batch Apex

```
public class LeadProcessor implements Database.Batchable<sObject> {
public Database.QueryLocator start(Database.BatchableContext bc) {
// collect the batches of records or objects to be passed to execute
return Database.getQueryLocator([Select LeadSource From Lead ]);
}
public void execute(Database.BatchableContext bc, List<Lead> leads){
// process each batch of records
for (Lead Lead : leads) {
lead.LeadSource = 'Dreamforce';
}
update leads;
}
```

```
public void finish(Database.BatchableContext bc){
}

}

test class//

@isTest
public class LeadProcessorTest {
@testSetup
static void setup() {
List<Lead> leads = new List<Lead>();
for(Integer counter=0 ;counter <200;counter++){
Lead lead = new Lead();
lead.FirstName ='FirstName';
lead.LastName ='LastName'+counter;
lead.Company ='demo'+counter;
leads.add(lead);
}
insert leads;
}
@isTest static void test() {
```

```
Test.startTest();

LeadProcessor leadProcessor = new LeadProcessor();

Id batchId = Database.executeBatch(leadProcessor);

Test.stopTest();

}


}
```

3.Control Processes with Queueable Apex

```
public

class AddPrimaryContact implements Queueable


{

private Contact c;

private String state;

public  AddPrimaryContact(Contact c, String state)

{

this.c = c;

this.state = state;

}

public void execute(QueueableContext context)

{

 List<Account> ListAccount = [SELECT ID, Name ,(Select id,FirstName,LastName from contacts )
FROM ACCOUNT WHERE BillingState = :state LIMIT 200];
```

```
List<Contact> lstContact = new List<Contact>();

for (Account acc:ListAccount)

{

Contact cont = c.clone(false,false,false,false);

cont.AccountId =  acc.id;

lstContact.add( cont );

}

if(lstContact.size() >0 )

{

insert lstContact;

}

}

test class///

        @isTest         public         class
AddPrimaryContactTest

{

 @isTest static void TestList()

{

List<Account> Teste = new List <Account>();

for(Integer i=0;i<50;i++)

{
```

```
Teste.add(new Account(BillingState = 'CA', name = 'Test'+i));        }

for(Integer j=0;j<50;j++)

{

Teste.add(new Account(BillingState = 'NY', name = 'Test'+j));

}

insert Teste;


Contact co = new Contact();

co.FirstName='demo';

co.LastName ='demo';

insert co;

String state = 'CA';

AddPrimaryContact apc = new AddPrimaryContact(co, state);

Test.startTest();

System.enqueueJob(apc);

Test.stopTest();

} }
```

4.Schedule Jobs Using the Apex Scheduler

```
public class DailyLeadProcessor implements Schedulable  {

Public void execute(SchedulableContext SC){

List<Lead> LeadObj=[SELECT Id from Lead where LeadSource=null limit 200];
```

```
for(Lead l:LeadObj){

l.LeadSource='Dreamforce';

update l;

}

}

}
```

test class ///

```
 @isTest  private  class  DailyLeadProcessorTest  {  static

testMethod void testDailyLeadProcessor() {

                    String CRON_EXP = '0 0 1 * * ?';

                    List<Lead> lList = new List<Lead>();

          for (Integer i = 0; i < 200; i++) {

                              lList.add(new   Lead(LastName='Dreamforce'+i,   Company='Test1   Inc.',
Status='Open - Not Contacted'));

                      }    insert

                   lList;

                   Test.startT

                   est();

 String jobId = System.schedule('DailyLeadProcessor', CRON_EXP, new
```

```
DailyLeadProcessor());
        }
}
```

Apex Integration Services

1.Apex REST Callouts Class

AnimalLocator//

```
public class AnimalLocator{
public static String getAnimalNameById(Integer x){
Http http = new Http();
HttpRequest req = new HttpRequest();
 req.setEndpoint('https://th-apex-http-callout.herokuapp.com/animals/' + x);
req.setMethod('GET');
Map<String, Object> animal= new Map<String, Object>();
HttpResponse res = http.send(req);
if (res.getStatusCode() == 200) {
 Map<String, Object> results = (Map<String, Object>)JSON.deserializeUntyped(res.getBody());
animal = (Map<String, Object>) results.get('animal');
```

```
}
return (String)animal.get('name');
}
}

AnimalLocatorTest//

    @isTest    private    class
AnimalLocatorTest{
@isTest static void AnimalLocatorMock1() {
Test.setMock(HttpCalloutMock.class, new AnimalLocatorMock());
string result = AnimalLocator.getAnimalNameById(3);
String expectedResult = 'chicken';
System.assertEquals(result,expectedResult );
}
}
AnimalLocatorMock//

  @isTest  global  class  AnimalLocatorMock  implements
HttpCalloutMock {
```

```
// Implement this interface method

global HTTPResponse respond(HTTPRequest request) {

// Create a fake response

HttpResponse response = new HttpResponse();

response.setHeader('Content-Type', 'application/json');

 response.setBody('{"animals": ["majestic badger", "fluffy bunny", "scary bear", "chicken", "mighty moose"]}');

response.setStatusCode(200);

return response;

}

}
```

2.Apex SOAP Callouts

ParkLocator class/////

```
public class ParkLocator {

public static string[] country(string theCountry) {

  ParkService.ParksImplPort    parkSvc  =  new  ParkService.ParksImplPort();  //  remove  space

return parkSvc.byCountry(theCountry);

}
```

```
}
```

ParkLocatorTest class//////

```
    @isTest     private     class
ParkLocatorTest {
 @isTest static void testCallout() {
 Test.setMock(WebServiceMock.class, new ParkServiceMock ());
 String country = 'United States';
 List<String> result = ParkLocator.country(country);
  List<String>  parks  =  new  List<String>{'Yellowstone',  'Mackinac  National  Park',  'Yosemite'};
System.assertEquals(parks, result);
 }
 }
```

ParkServiceMock class /////

```
   @isTest    global    class    ParkServiceMock    implements
WebServiceMock {
 global void doInvoke(
```

```
Object stub,

Object request,

Map<String, Object> response,

String endpoint,

String soapAction,

String requestName,

String responseNS,

String responseName,

String responseType) {

// start - specify the response you want to send

ParkService.byCountryResponse response_x = new ParkService.byCountryResponse();

response_x.return_x = new List<String>{'Yellowstone', 'Mackinac National Park', 'Yosemite'};

// end

response.put('response_x', response_x);

}

}
```

4.Apex Web Services

AccountManagerTest/////

```
    @isTest     private     class
```

```
AccountManagerTest {


 private static testMethod void getAccountTest1() {

 Id recordId = createTestRecord();

 // Set up a test request

 RestRequest request = new RestRequest();

   request.requestUri   =   'https://na1.salesforce.com/services/apexrest/Accounts/'+   recordId
+'/contacts' ;

 request.httpMethod = 'GET';

 RestContext.request = request;

 // Call the method to test

 Account thisAccount = AccountManager.getAccount();

 // Verify results

 System.assert(thisAccount != null);

 System.assertEquals('Test record', thisAccount.Name);


 }


 // Helper method

 static Id createTestRecord() {

 // Create test record

 Account TestAcc = new Account(

 Name='Test record');
```

```
insert TestAcc;
Contact TestCon= new Contact(
LastName='Test',
AccountId = TestAcc.id);
return TestAcc.Id;
}
}
```

AccountManager//////

```
    @RestResource(urlMapping='/Accounts/*/contacts')
global class AccountManager {
 @HttpGet
global static Account getAccount() {
RestRequest req = RestContext.request;
String accId = req.requestURI.substringBetween('Accounts/', '/contacts');
Account acc = [SELECT Id, Name, (SELECT Id, Name FROM Contacts)
FROM Account WHERE Id = :accId];
return acc;
}}
```

APEX SPECLIALIST SUPERBADGE

Challenge #1

```
MaintenanceRequest.triggertrigger MaintenanceRequest on

Case (before update, after update) { Map<Id,Case>

validCaseMap = new Map<Id,Case>();


if(Trigger.isUpdate && Trigger.isAfter){

for(Case caseHere: Trigger.new){

if (caseHere.IsClosed && (caseHere.Type.equals('Repair') || caseHere.Type.equals('Routine Maintenance'))){

validCaseMap.put(caseHere.Id, caseHere);

}

}

if(!validCaseMap.values().isEmpty()){

        MaintenanceRequestHelper.createNewRequest(validCaseMap);

}

}

}
```

MaintenanceRequestHelper.clspublic

```
class MaintenanceRequestHelper {

public static void createNewRequest(Map<Id, Case> validCaseMap){

List<Case> newCases = new List<Case>();

Map<Id, Integer> productMaintenanceCycleMap = new Map<Id, Integer>();

Map<Id, Integer> workPartMaintenanceCycleMap = new Map<Id, Integer>();

for (Product2 productHere : [select Id, Maintenance_Cycle__c from Product2]) {

if (productHere.Maintenance_Cycle__c != null) {

productMaintenanceCycleMap.put(productHere.Id,

Integer.valueOf(productHere.Maintenance_Cycle__c));

}

}

for (Work_Part__c workPart : [select Id, Equipment__c, Maintenance_Request__c from Work_Part__c where Maintenance_Request__c in :validCaseMap.keySet()]) {

if (workPart.Equipment__c != null) {

if(!workPartMaintenanceCycleMap.containsKey(workPart.Maintenance_Request__c)){
```

```
                    workPartMaintenanceCycleMap.put(workPart.Maintenance_Request__c,
productMaintenanceCycleMap.get(workPart.Equipment__c));

 }

        else    if(productMaintenanceCycleMap.get(workPart.Equipment__c)    <
workPartMaintenanceCycleMap.get(workPart.Maintenance_Request__c)){

                    workPartMaintenanceCycleMap.put(workPart.Maintenance_Request__c,
productMaintenanceCycleMap.get(workPart.Equipment__c));

 }

 }

 }

 for(Case caseHere: validCaseMap.values()){

 Case newCase = new Case();

 newCase.Vehicle__c = caseHere.Vehicle__c;

 newCase.Equipment__c = caseHere.Equipment__c;

 newCase.Type = 'Routine Maintenance';

  newCase.Subject = String.isBlank(caseHere.Subject) ? 'Routine Maintenance Request' :
caseHere.Subject + ' New';

 newCase.Date_Reported__c = Date.today();

 newCase.Date_Due__c =
```

```
workPartMaintenanceCycleMap.containsKey(caseHere.Product__c) ?
        Date.today().addDays(workPartMaintenanceCycleMap.get(caseHere.Product__c))        :
Date.today();          newCase.Status = 'New';

newCase.Product__c = caseHere.Product__c;
newCase.AccountId = caseHere.AccountId;
newCase.ContactId = caseHere.ContactId;
newCase.AssetId = caseHere.AssetId;
newCase.Origin = caseHere.Origin;
newCase.Reason = caseHere.Reason;
newCases.add(newCase);
}
if(newCases.size() > 0){
insert newCases;
}
}
}
```

Challenge #2

WarehouseCalloutService.cls

public with sharing class WarehouseCalloutService { private static final String WAREHOUSE_URL = 'https://th-superbadgeapex.herokuapp.com/equipment';

// complete this method to make the callout (using @future) to the

// REST endpoint and update equipment on hand.

@future(callout=true)

public static void runWarehouseEquipmentSync(){

Http http = new Http();

HttpRequest request = new HttpRequest();

request.setEndpoint(WAREHOUSE_URL);

request.setMethod('GET');

HttpResponse response = http.send(request);

if (response.getStatusCode() == 200) {

List<Object> results = (List<Object>) JSON.deserializeUntyped(response.getBody());

List<Product2> equipmentList = new List<Product2>();

for (Object record: results) {

```
Map<String, Object> recordMap = (Map<String, Object>)record;

Product2 equipment = new Product2();

equipment.Name = (String)recordMap.get('name');

equipment.Cost__c = (Decimal)recordMap.get('cost');

equipment.ProductCode = (String)recordMap.get('_id');

equipment.Current_Inventory__c = (Integer)recordMap.get('quantity');

equipment.Maintenance_Cycle__c = (Integer)recordMap.get('maintenanceperiod');

equipment.Replacement_Part__c = (Boolean)recordMap.get('replacement');

equipment.Lifespan_Months__c = (Integer)recordMap.get('lifespan');

equipment.Warehouse_SKU__c = (String)recordMap.get('sku');

equipmentList.add(equipment);
}

if(equipmentList.size() > 0){

upsert equipmentList;

}

}
```

}

}

challange #3*\WarehouseSyncSchedule.clspublic class*

WarehouseSyncSchedule implements Schedulable{

// implement scheduled code here

public void execute(System.SchedulableContext context){

WarehouseCalloutService.runWarehouseEquipmentSync();

}

}

Challenge #4*@isTestpublic class*

MaintenanceRequestTest {

@testSetup

static void setup(){

Product2 prod = new Product2();

prod.Cost__c = 50;

```
prod.Name = 'Ball Valve 10 cm';

prod.Lifespan_Months__c = 12;

prod.Maintenance_Cycle__c = 365;

prod.Current_Inventory__c = 50;       prod.Replacement_Part__c = true;

prod.Warehouse_SKU__c = '100009';

insert prod;

Product2 prod2 = new Product2();

prod2.Cost__c = 50;

prod2.Name = 'Ball Valve 10 cm';

prod2.Lifespan_Months__c = 12;

prod2.Maintenance_Cycle__c = 240;

prod2.Current_Inventory__c = 50;

prod2.Replacement_Part__c = true;

prod2.Warehouse_SKU__c = '100009';

insert prod2;

List<Case> caseList = new List<Case>();       for(Integer i=0; i<300; i++) {
```

```
Case caseNew = new Case();

caseNew.Subject = 'Maintenance ' + i;

caseNew.Type = 'Other';

caseNew.Status = 'New';

caseNew.Equipment__c = prod.Id;

caseNew.SuppliedName = 'Test';

caseList.add(caseNew);

if(i==10){

caseNew.Subject = 'Maintenance test 10';

}

}

insert caseList;

List<Work_Part__c> workPartList = new List<Work_Part__c>();

for(Case caseHere : [select Id, Subject from Case where SuppliedName = 'Test']) {

Work_Part__c workPart = new Work_Part__c();

workPart.Maintenance_Request__c = caseHere.Id;
```

```
workPart.Equipment__c = prod.Id;

workPartList.add(workPart);

if(caseHere.Subject == 'Maintenance test 10'){

Work_Part__c workPart2 = new Work_Part__c();

workPart2.Maintenance_Request__c = caseHere.Id;

workPart2.Equipment__c = prod2.Id;

workPartList.add(workPart2);

}

}

insert workPartList;

}

@isTest

static void testMaintenanceRequest(){

List<Case> caseList = new List<Case>();

for(Case caseHere : [select Id from Case where SuppliedName = 'Test']) {

caseHere.Type = 'Repair';
```

```
caseHere.Status = 'Closed';

caseList.add(caseHere);

}

Test.startTest();

update caseList;

System.assertEquals(300, [SELECT count() FROM Case WHERE Type = 'Routine

Maintenance' and Date_Reported__c = :Date.today()]);

Test.stopTest();

}

}
```

Challenge #5

WarehouseCalloutServiceMock.cls

```
public class WarehouseCalloutServiceMock implements HttpCalloutMock {

private String responseJson = '[' +

  '{"_id":"55d66226726b611100aaf741","replacement":false,"quantity":5,"name":"Generator
1000 kW","maintenanceperiod":365,"lifespan":120,"cost":5000,"sku":"100003"},' +

   '{"_id":"55d66226726b611100aaf742","replacement":true,"quantity":183,"name":"Cooling
Fan","maintenanceperiod":0,"lifespan":0,"cost":300,"sku":"100004"},' +
```

'{"_id":"55d66226726b611100aaf743","replacement":true,"quantity":143,"name":"Fuse

20A","maintenanceperiod":0,"lifespan":0,"cost":22,"sku":"100005"}' +

']';

// Implement this interface method

public HTTPResponse respond(HTTPRequest request) {

// Create a fake response

HttpResponse response = new HttpResponse();

response.setHeader('Content-Type', 'application/json');

response.setBody(responseJson);

response.setStatusCode(200);

return response;

}

}

WarehouseCalloutServiceTest.cls

@isTestprivate class

WarehouseCalloutServiceTest {

```
@isTest

static void testRunWarehouseEquipmentSync(){

Test.setMock(HttpCalloutMock.class, new WarehouseCalloutServiceMock());

Test.startTest();

WarehouseCalloutService.runWarehouseEquipmentSync();

Test.stopTest();

System.assertEquals(3, [select count() from Product2]);

}

}
```

Challenge #6

WarehouseSyncScheduleTest.cls

```
@isTestpublic class WarehouseSyncScheduleTest {

public static String CRON_EXP = '0 0 1 * * ?';

@isTest

static void testExecute(){
```

```
Test.setMock(HttpCalloutMock.class, new WarehouseCalloutServiceMock());

Test.startTest();

String jobId = System.schedule('WarehouseSyncScheduleTest', CRON_EXP, new

WarehouseSyncSchedule());

Test.stopTest();

 System.assertEquals(1, [SELECT count() FROM CronTrigger WHERE CronJobDetail.Name =
'WarehouseSyncScheduleTest']);

}

}
```