## APEX TRIGGERS

## **1. 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 Test**

**CLASS-**

|  |
| --- |
| publicclassVerifyDate { |

|  |
| --- |
|  |

|  |
| --- |
| //method to handle potential checks against two dates |

|  |
| --- |
| publicstaticDateCheckDates(Datedate1, Datedate2) { |

|  |
| --- |
| //if date2 is within the next 30 days of date1, use date2. Otherwise use the end of the month |

|  |
| --- |
| if(DateWithin30Days(date1,date2)) { |

|  |
| --- |
| returndate2; |

|  |
| --- |
| } else { |

|  |
| --- |
| returnSetEndOfMonthDate(date1); |

|  |
| --- |
| } |

|  |
| --- |
| } |

|  |
| --- |
|  |

|  |
| --- |
| //method to check if date2 is within the next 30 days of date1 |

|  |
| --- |
| privatestaticBooleanDateWithin30Days(Datedate1, Datedate2) { |

|  |
| --- |
| //check for date2 being in the past |

|  |
| --- |
| if( date2 < date1) { returnfalse; } |

|  |
| --- |
|  |

|  |
| --- |
| //check that date2 is within (>=) 30 days of date1 |

|  |
| --- |
| Datedate30Days = date1.addDays(30); //create a date 30 days away from date1 |

|  |
| --- |
| if( date2 >= date30Days ) { returnfalse; } |

|  |
| --- |
| else { returntrue; } |

|  |
| --- |
| } |

|  |
| --- |
|  |

|  |
| --- |
| //method to return the end of the month of a given date |

|  |
| --- |
| privatestaticDateSetEndOfMonthDate(Datedate1) { |

|  |
| --- |
| IntegertotalDays = Date.daysInMonth(date1.year(), date1.month()); |

|  |
| --- |
| DatelastDay = Date.newInstance(date1.year(), date1.month(), totalDays); |

|  |
| --- |
| returnlastDay; |

|  |
| --- |
| } |

|  |
| --- |
|  |

}

**TEST CLASS-**

@isTest

public class TestVerifyDate {

@isTest static void testOldDate(){

Date dateTest = VerifyDate.CheckDates(date.today(), date.today().addDays(-1));

System.assertEquals(date.newInstance(2016, 4, 30), dateTest);

}

@isTest static void testLessThan30Days(){

Date dateTest = VerifyDate.CheckDates(date.today(), date.today().addDays(20));

System.assertEquals(date.today().addDays(20), dateTest);

}

@isTest static void testMoreThan30Days(){

Date dateTest = VerifyDate.CheckDates(date.today(), date.today().addDays(31));

System.assertEquals(date.newInstance(2016, 4, 30), dateTest);

}

}

**2. Test Apex Triggers**

## **TRIGGER-**

|  |
| --- |
| triggerRestrictContactByNameonContact (beforeinsert, beforeupdate) { |

|  |
| --- |
|  |

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

|  |
| --- |
| For (Contactc : Trigger.New) { |

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

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

|  |
| --- |
| } |

|  |
| --- |
|  |

|  |
| --- |
| } |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

}

**TEST CLASS-**

@istest

private class TestRestrictContactByName {

@isTest Static void TestRestictedname(){

List<contact> testcontacts = New List<contact>();

contact c0 = new contact(LastName = 'Testcontact');

testcontacts.add(c0);

Contact c1 = new contact(LastName = 'INVALIDNAME');

testcontacts.add(c1);

Test.startTest();

Database.SaveResult[] srList = Database.insert(testcontacts, false);

Test.stopTest();

Database.SaveResult R0 = srlist[0];

Database.SaveResult R1 = srlist[1];

System.assertequals(r1.getErrors()[0].getmessage(),'The Last Name "INVALIDNAME" is not allowed for DML');

System.assert(r0.isSuccess());

System.assert(!r1.isSuccess());

}

}

**3. Create Test Data for Apex Tests**

**CLASS-**

public class RandomContactFactory {

public Static List<Contact> generateRandomContacts(Integer NumberOfContact, String LnameOfContact){

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

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

Contact c=new Contact(FirstName='Test '+i,LastName=LnameOfContact);

con.add(c);

}

return con;

}

}

**TEST CLASS-**

@isTest

public class RandomContactFactoryTest {

static testMethod void testfunction(){

List<Contact> Con = RandomContactFactory.generateRandomContacts(1,'Abhishek');

}}

**ASYNCHRONOUS APEX**

**2. Use Future Methods**

**CLASS-**

public class AccountProcessor {

@future

public static void countContacts(set<ID> Accnts){

List<Account> acc = [select Name from Account where Id IN :accnts];

List<Account> acc1 = new List<Account>();

for(Account a:acc){

a.Number\_of\_Contacts\_\_c = [select Count() from Contact where AccountId =:a.Id];

acc1.add(a);

}

update acc1;

}

**TEST CLASS-**

@isTest

public class AccountProcessorTest {

@isTest public static void accnt(){

Account a = new Account();

a.Name = 'Test Account';

Insert a;

Contact cont = New Contact();

cont.FirstName ='Bob';

cont.LastName ='Masters';

cont.AccountId = a.Id;

Insert cont;

Set<Id> setAccId = new Set<ID>();

setAccId.add(a.id);

Test.startTest();

AccountProcessor.countContacts(setAccId);

Test.stopTest();

}

}

**3. Use Batch Apex**

**CLASS-**

global class LeadProcessor implements Database.Batchable<Sobject>

{

global Database.QueryLocator start(Database.BatchableContext bc)

{

return Database.getQueryLocator([Select LeadSource From Lead ]);

}

global void execute(Database.BatchableContext bc, List<Lead> scope)

{

for (Lead Leads : scope)

{

Leads.LeadSource = 'Dreamforce';

}

update scope;

}

global void finish(Database.BatchableContext bc){ }

}

**TEST CLASS-**

@isTest

public class LeadProcessorTest

{

static testMethod void testMethod1()

{

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

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

{

Lead led = new Lead();

led.FirstName ='FirstName';

led.LastName ='LastName'+i;

led.Company ='demo'+i;

lstLead.add(led);

}

insert lstLead;

Test.startTest();

LeadProcessor obj = new LeadProcessor();

DataBase.executeBatch(obj);

Test.stopTest();

}

}

**4. Control Processes with Queueable Apex**

**CLASS-**

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();

}

}

**5. Schedule Jobs Using the Apex Scheduler**

**CLASS-**

global class DailyLeadProcessor implements Schedulable {

global void execute(SchedulableContext ctx) {

List<Lead> lList = [Select Id, LeadSource from Lead where LeadSource = null];

if(!lList.isEmpty()) {

for(Lead l: lList) {

l.LeadSource = 'Dreamforce';

}

update lList;

}

}

**TEST CLASS-**

@isTest

private class DailyLeadProcessorTest {

public static String CRON\_EXP = '0 0 0 15 3 ? 2022';

static testmethod void testScheduledJob() {

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

Leads.add(new lead(

name='Dream force'+i

));

}

insert Leads;

}

Test.startTest();

String jobId = System.schedule('ScheduledApexTest',

CRON\_EXP,

new DailyLeadProcessorTest());

}

## **APEX INTEGRATION SERVICES**

**2. Apex REST Callouts**

## **CLASS-**

## public class AnimalLocator {

## public static string getAnimalNameById(Integer x){

## Http http = new Http();

## HttpRequest request = new HttpRequest();

## request.setEndpoint('https://th-apex-http-callout.herokuapp.com/animals');

## request.setMethod('GET');

## Map<String, Object>animal= new Map<String,Object>();

## HttpResponse response=http.send(request);

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

## // Deserializes the JSON string into collections of primitive data types.

## Map<String, Object> results = (Map<String, Object>) JSON.deserializeUntyped(response.getBody());

## animal = (Map<String, Object>) results.get('animal');

## }

## return(String)animal.get('name');

## }

## }

**TEST CLASS-**

1. 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);

}

}

2. AnimalLocatorMock

@isTest

global class AnimalLocatorMock implements HttpCalloutMock {

global HTTPResponse respond(HTTPRequest request) {

HttpResponse response = new HttpResponse();

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

response.setBody('{"animal":{"id":1,"name":"chicken","eats":"chicken food","says":"cluck cluck"}}');

response.setStatusCode(200);

return response;

}

}

**3. Apex SOAP Callouts**

## **CLASS-**

## public class ParkLocator {

## public static String[] country(String country){

## ParkService.ParksImplPort parks = new ParkService.ParksImplPort();

## String[] parksname = parks.byCountry(country);

## return parksname;

## }

## }

## **TEST CLASS-**

1. ParkServiceMock

@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();

List<String> myStrings = new List<String> {'Park1','Park2','Park3'};

response\_x.return\_x = myStrings;

// end

response.put('response\_x', response\_x);

}

}

2. ParkLocatorTest

@isTest

private class ParkLocatorTest {

@isTest static void testCallout() {

// This causes a fake response to be generated

Test.setMock(WebServiceMock.class, new ParkServiceMock());

// Call the method that invokes a callout

List<String> result = new List<String>();

List<String> expectedvalue = new List<String>{'Park1','Park2','Park3'};

result = ParkLocator.country('India');

// Verify that a fake result is returned

System.assertEquals(expectedvalue, result);

}

}

**3. Apex Web Service-**

**CLASS-**

@RestResource(urlMapping='/Accounts/\*/contacts')

global with sharing class AccountManager{

@HttpGet

global static Account getAccount(){

RestRequest request = RestContext.request;

String accountId = request.requestURI.substringBetween('Accounts/','/contacts');

system.debug(accountId);

Account objAccount = [SELECT Id,Name,(SELECT Id,Name FROM Contacts) FROM Account WHERE Id = :accountId LIMIT 1];

return objAccount;

}

}

**TEST CLASS-**

@isTest

private class AccountManagerTest{

static testMethod void testMethod1(){

Account objAccount = new Account(Name = 'test Account');

insert objAccount;

Contact objContact = new Contact(LastName = 'test Contact',

AccountId = objAccount.Id);

insert objContact;

Id recordId = objAccount.Id;

RestRequest request = new RestRequest();

request.requestUri =

'https://sandeepidentity-dev-ed.my.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 Account', thisAccount.Name);

}

}

## APEX SPECIALIST SUPERBADGE

|  |  |  |
| --- | --- | --- |
| ***STEP 2*** AUTOMATE RECORD CREATION     * **MaintenanceRequest Trigger**   trigger MaintenanceRequest on Case (before update, after update) {  if(Trigger.isUpdate && Trigger.isAfter){  MaintenanceRequestHelper.updateWorkOrders(Trigger.New, Trigger.OldMap);  }  }   * **MaintenanceRequestHelper Class**     public with sharing class MaintenanceRequestHelper {  public static void updateworkOrders(List<Case> updWorkOrders, Map<Id,Case> nonUpdCaseMap) {  Set<Id> validIds = new Set<Id>();  For (Case c : updWorkOrders){  if (nonUpdCaseMap.get(c.Id).Status != 'Closed' && c.Status == 'Closed'){  if (c.Type == 'Repair' || c.Type == 'Routine Maintenance'){  validIds.add(c.Id);  }  }  }    if (!validIds.isEmpty()){  Map<Id,Case> closedCases = new Map<Id,Case>([SELECT Id, Vehicle\_\_c, Equipment\_\_c, Equipment\_\_r.Maintenance\_Cycle\_\_c,  (SELECT Id,Equipment\_\_c,Quantity\_\_c FROM Equipment\_Maintenance\_Items\_\_r)  FROM Case WHERE Id IN :validIds]);  Map<Id,Decimal> maintenanceCycles = new Map<ID,Decimal>();    AggregateResult[] results = [SELECT Maintenance\_Request\_\_c,  MIN(Equipment\_\_r.Maintenance\_Cycle\_\_c)cycle  FROM Equipment\_Maintenance\_Item\_\_c  WHERE Maintenance\_Request\_\_c IN :ValidIds GROUP BY Maintenance\_Request\_\_c];    for (AggregateResult ar : results){  maintenanceCycles.put((Id) ar.get('Maintenance\_Request\_\_c'), (Decimal) ar.get('cycle'));  }    List<Case> newCases = new List<Case>();  for(Case cc : closedCases.values()){  Case nc = new Case (  ParentId = cc.Id,  Status = 'New',  Subject = 'Routine Maintenance',  Type = 'Routine Maintenance',  Vehicle\_\_c = cc.Vehicle\_\_c,  Equipment\_\_c =cc.Equipment\_\_c,  Origin = 'Web',  Date\_Reported\_\_c = Date.Today()  );    If (maintenanceCycles.containskey(cc.Id)){  nc.Date\_Due\_\_c = Date.today().addDays((Integer) maintenanceCycles.get(cc.Id));  } else {  nc.Date\_Due\_\_c = Date.today().addDays((Integer) cc.Equipment\_\_r.maintenance\_Cycle\_\_c);  }    newCases.add(nc);  }    insert newCases;    List<Equipment\_Maintenance\_Item\_\_c> clonedList = new List<Equipment\_Maintenance\_Item\_\_c>();  for (Case nc : newCases){  for (Equipment\_Maintenance\_Item\_\_c clonedListItem : closedCases.get(nc.ParentId).Equipment\_Maintenance\_Items\_\_r){  Equipment\_Maintenance\_Item\_\_c item = clonedListItem.clone();  item.Maintenance\_Request\_\_c = nc.Id;  clonedList.add(item);  }  }  insert clonedList;  }  }  }    ***STEP 3*** SYNCHRONIZE SALESFORCE DATA WITH AN EXTERNAL SYSTEM   WarehouseCalloutService Classpublic with sharing class WarehouseCalloutService implements Queueable {private static final String WAREHOUSE\_URL = 'https://th-superbadge-apex.herokuapp.com/equipment';@future(callout=true)public static void runWarehouseEquipmentSync(){System.debug('go into runWarehouseEquipmentSync');Http http = new Http();HttpRequest request = new HttpRequest();request.setEndpoint(WAREHOUSE\_URL);request.setMethod('GET');HttpResponse response = http.send(request);List<Product2> product2List = new List<Product2>();System.debug(response.getStatusCode());if (response.getStatusCode() == 200){List<Object> jsonResponse = (List<Object>)JSON.deserializeUntyped(response.getBody());System.debug(response.getBody());for (Object jR : jsonResponse){Map<String,Object> mapJson = (Map<String,Object>)jR;Product2 product2 = new Product2();product2.Replacement\_Part\_\_c = (Boolean) mapJson.get('replacement');product2.Cost\_\_c = (Integer) mapJson.get('cost');product2.Current\_Inventory\_\_c = (Double) mapJson.get('quantity');product2.Lifespan\_Months\_\_c = (Integer) mapJson.get('lifespan');product2.Maintenance\_Cycle\_\_c = (Integer) mapJson.get('maintenanceperiod');product2.Warehouse\_SKU\_\_c = (String) mapJson.get('sku');product2.Name = (String) mapJson.get('name');product2.ProductCode = (String) mapJson.get('\_id');product2List.add(product2);}if (product2List.size() > 0){upsert product2List;System.debug('Your equipment was synced with the warehouse one');}}}public static void execute (QueueableContext context){System.debug('start runWarehouseEquipmentSync');runWarehouseEquipmentSync();System.debug('end runWarehouseEquipmentSync');}} ***STEP 4*** SCHEDULE SYNCHRONIZATION   WarehouseSyncSchedule Classglobal with sharing class WarehouseSyncSchedule implements Schedulable{global void execute(SchedulableContext ctx){System.enqueueJob(new WarehouseCalloutService());}} ***STEP 5*** TEST AUTOMATION LOGIC   MaintenanceRequest Triggertrigger MaintenanceRequest on Case (before update, after update) {if(Trigger.isUpdate && Trigger.isAfter){MaintenanceRequestHelper.updateWorkOrders(Trigger.New, Trigger.OldMap);}}MaintenanceRequestHelper Triggerpublic with sharing class MaintenanceRequestHelper {public static void updateworkOrders(List<Case> updWorkOrders, Map<Id,Case> nonUpdCaseMap) {Set<Id> validIds = new Set<Id>();For (Case c : updWorkOrders){if (nonUpdCaseMap.get(c.Id).Status != 'Closed' && c.Status == 'Closed'){if (c.Type == 'Repair' || c.Type == 'Routine Maintenance'){validIds.add(c.Id);}}}if (!validIds.isEmpty()){Map<Id,Case> closedCases = new Map<Id,Case>([SELECT Id, Vehicle\_\_c, Equipment\_\_c, Equipment\_\_r.Maintenance\_Cycle\_\_c,(SELECT Id,Equipment\_\_c,Quantity\_\_c FROM Equipment\_Maintenance\_Items\_\_r) FROM Case WHERE Id IN :validIds]);Map<Id,Decimal> maintenanceCycles = new Map<ID,Decimal>();AggregateResult[] results = [SELECT Maintenance\_Request\_\_c, MIN(Equipment\_\_r.Maintenance\_Cycle\_\_c)cycle FROM Equipment\_Maintenance\_Item\_\_c WHERE Maintenance\_Request\_\_c IN :ValidIds GROUP BY Maintenance\_Request\_\_c];for (AggregateResult ar : results){ maintenanceCycles.put((Id) ar.get('Maintenance\_Request\_\_c'), (Decimal) ar.get('cycle'));}List<Case> newCases = new List<Case>();for(Case cc : closedCases.values()){Case nc = new Case (ParentId = cc.Id,Status = 'New',Subject = 'Routine Maintenance',Type = 'Routine Maintenance',Vehicle\_\_c = cc.Vehicle\_\_c,Equipment\_\_c =cc.Equipment\_\_c,Origin = 'Web',Date\_Reported\_\_c = Date.Today() );nc.Date\_Due\_\_c = Date.today().addDays((Integer) maintenanceCycles.get(cc.Id));newCases.add(nc);}insert newCases;List<Equipment\_Maintenance\_Item\_\_c> clonedList = new List<Equipment\_Maintenance\_Item\_\_c>();for (Case nc : newCases){for (Equipment\_Maintenance\_Item\_\_c clonedListItem : closedCases.get(nc.ParentId).Equipment\_Maintenance\_Items\_\_r){Equipment\_Maintenance\_Item\_\_c item = clonedListItem.clone();item.Maintenance\_Request\_\_c = nc.Id;clonedList.add(item);}}insert clonedList;}}}MaintenanceRequestHelperTest@isTestpublic with sharing class MaintenanceRequestHelperTest {private static Vehicle\_\_c createVehicle(){Vehicle\_\_c vehicle = new Vehicle\_\_C(name = 'Testing Vehicle');return vehicle;}private static Product2 createEquipment(){product2 equipment = new product2(name = 'Testing equipment',lifespan\_months\_\_c = 10,maintenance\_cycle\_\_c = 10,replacement\_part\_\_c = true);return equipment;}private static Case createMaintenanceRequest(id vehicleId, id equipmentId){case cse = new case(Type='Repair',Status='New',Origin='Web',Subject='Testing subject',Equipment\_\_c=equipmentId,Vehicle\_\_c=vehicleId);return cse;}private static Equipment\_Maintenance\_Item\_\_c createEquipmentMaintenanceItem(id equipmentId,id requestId){Equipment\_Maintenance\_Item\_\_c equipmentMaintenanceItem = new Equipment\_Maintenance\_Item\_\_c(Equipment\_\_c = equipmentId,Maintenance\_Request\_\_c = requestId);return equipmentMaintenanceItem;}@isTestprivate static void testPositive(){Vehicle\_\_c vehicle = createVehicle();insert vehicle;id vehicleId = vehicle.Id;Product2 equipment = createEquipment();insert equipment;id equipmentId = equipment.Id;case createdCase = createMaintenanceRequest(vehicleId,equipmentId);insert createdCase;Equipment\_Maintenance\_Item\_\_c equipmentMaintenanceItem = createEquipmentMaintenanceItem(equipmentId,createdCase.id);insert equipmentMaintenanceItem;test.startTest();createdCase.status = 'Closed';update createdCase;test.stopTest();Case newCase = [Select id, subject, type, Equipment\_\_c, Date\_Reported\_\_c, Vehicle\_\_c, Date\_Due\_\_cfrom casewhere status ='New'];Equipment\_Maintenance\_Item\_\_c workPart = [select idfrom Equipment\_Maintenance\_Item\_\_cwhere Maintenance\_Request\_\_c =:newCase.Id];list<case> allCase = [select id from case];system.assert(allCase.size() == 2);system.assert(newCase != null);system.assert(newCase.Subject != null);system.assertEquals(newCase.Type, 'Routine Maintenance');SYSTEM.assertEquals(newCase.Equipment\_\_c, equipmentId);SYSTEM.assertEquals(newCase.Vehicle\_\_c, vehicleId);SYSTEM.assertEquals(newCase.Date\_Reported\_\_c, system.today());}@isTestprivate static void testNegative(){Vehicle\_\_C vehicle = createVehicle();insert vehicle;id vehicleId = vehicle.Id;product2 equipment = createEquipment();insert equipment;id equipmentId = equipment.Id;case createdCase = createMaintenanceRequest(vehicleId,equipmentId);insert createdCase;Equipment\_Maintenance\_Item\_\_c workP = createEquipmentMaintenanceItem(equipmentId, createdCase.Id);insert workP;test.startTest();createdCase.Status = 'Working';update createdCase;test.stopTest();list<case> allCase = [select id from case];Equipment\_Maintenance\_Item\_\_c equipmentMaintenanceItem = [select id from Equipment\_Maintenance\_Item\_\_c where Maintenance\_Request\_\_c = :createdCase.Id];system.assert(equipmentMaintenanceItem != null);system.assert(allCase.size() == 1);}@isTestprivate static void testBulk(){list<Vehicle\_\_C> vehicleList = new list<Vehicle\_\_C>();list<Product2> equipmentList = new list<Product2>();list<Equipment\_Maintenance\_Item\_\_c> equipmentMaintenanceItemList = new list<Equipment\_Maintenance\_Item\_\_c>();list<case> caseList = new list<case>();list<id> oldCaseIds = new list<id>();for(integer i = 0; i < 300; i++){vehicleList.add(createVehicle());equipmentList.add(createEquipment());}insert vehicleList;insert equipmentList;for(integer i = 0; i < 300; i++){caseList.add(createMaintenanceRequest(vehicleList.get(i).id, equipmentList.get(i).id));}insert caseList;for(integer i = 0; i < 300; i++){equipmentMaintenanceItemList.add(createEquipmentMaintenanceItem(equipmentList.get(i).id, caseList.get(i).id));}insert equipmentMaintenanceItemList;test.startTest();for(case cs : caseList){cs.Status = 'Closed';oldCaseIds.add(cs.Id);}update caseList;test.stopTest();list<case> newCase = [select idfrom casewhere status ='New'];list<Equipment\_Maintenance\_Item\_\_c> workParts = [select idfrom Equipment\_Maintenance\_Item\_\_cwhere Maintenance\_Request\_\_c in: oldCaseIds];system.assert(newCase.size() == 300);list<case> allCase = [select id from case];system.assert(allCase.size() == 600);}} ***STEP 6*** TEST CALLOUT LOGIC   [WarehouseCalloutService](https://github.com/minicruiser/Apex-Specialist-Superbadge/blob/main/step6%20Test%20callout%20logic/WarehouseCalloutService.cls)public with sharing class WarehouseCalloutService implements Queueable {private static final String WAREHOUSE\_URL = 'https://th-superbadge-apex.herokuapp.com/equipment';@future(callout=true)public static void runWarehouseEquipmentSync(){System.debug('go into runWarehouseEquipmentSync');Http http = new Http();HttpRequest request = new HttpRequest();request.setEndpoint(WAREHOUSE\_URL);request.setMethod('GET');HttpResponse response = http.send(request);List<Product2> product2List = new List<Product2>();System.debug(response.getStatusCode());if (response.getStatusCode() == 200){List<Object> jsonResponse = (List<Object>)JSON.deserializeUntyped(response.getBody());System.debug(response.getBody());for (Object jR : jsonResponse){Map<String,Object> mapJson = (Map<String,Object>)jR;Product2 product2 = new Product2();product2.Replacement\_Part\_\_c = (Boolean) mapJson.get('replacement');product2.Cost\_\_c = (Integer) mapJson.get('cost');product2.Current\_Inventory\_\_c = (Double) mapJson.get('quantity');product2.Lifespan\_Months\_\_c = (Integer) mapJson.get('lifespan');product2.Maintenance\_Cycle\_\_c = (Integer) mapJson.get('maintenanceperiod');product2.Warehouse\_SKU\_\_c = (String) mapJson.get('sku');product2.Name = (String) mapJson.get('name');product2.ProductCode = (String) mapJson.get('\_id');product2List.add(product2);}if (product2List.size() > 0){upsert product2List;System.debug('Your equipment was synced with the warehouse one');}}}public static void execute (QueueableContext context){System.debug('start runWarehouseEquipmentSync');runWarehouseEquipmentSync();System.debug('end runWarehouseEquipmentSync');}}WarehouseCalloutServiceMock@isTestglobal class WarehouseCalloutServiceMock implements HttpCalloutMock {global static HttpResponse respond(HttpRequest request) {HttpResponse response = new HttpResponse();response.setHeader('Content-Type', 'application/json');response.setBody('[{"\_id":"54d63225726c345122abf221","replacement":false,"quantity":5,"name":"Generator 1000 kW","maintenanceperiod":365,"lifespan":120,"cost":5000,"sku":"100003"},{"\_id":"54d63225726c345122abf222","replacement":true,"quantity":183,"name":"Cooling Fan","maintenanceperiod":0,"lifespan":0,"cost":300,"sku":"100004"},{"\_id":"54d63225726c345122abf223","replacement":true,"quantity":143,"name":"Fuse 20A","maintenanceperiod":0,"lifespan":0,"cost":22,"sku":"100005"}]');response.setStatusCode(200);return response;}}WarehouseCalloutServiceTest@IsTestprivate class WarehouseCalloutServiceTest {@isTeststatic void testWarehouseCallout() {test.startTest();test.setMock(HttpCalloutMock.class, new WarehouseCalloutServiceMock());WarehouseCalloutService.execute(null);test.stopTest();List<Product2> product2List = new List<Product2>();product2List = [SELECT ProductCode FROM Product2];System.assertEquals(3, product2List.size());System.assertEquals('54d63225726c345122abf221', product2List.get(0).ProductCode);System.assertEquals('54d63225726c345122abf222', product2List.get(1).ProductCode);System.assertEquals('54d63225726c345122abf223', product2List.get(2).ProductCode);}} ***STEP 7*** TEST SCHEDULING LOGIC   WarehouseCalloutServiceMock@isTestglobal class WarehouseCalloutServiceMock implements HttpCalloutMock {global static HttpResponse respond(HttpRequest request) {HttpResponse response = new HttpResponse();response.setHeader('Content-Type', 'application/json');response.setBody('[{"\_id":"54d63225726c345122abf221","replacement":false,"quantity":5,"name":"Generator 1000 kW","maintenanceperiod":365,"lifespan":120,"cost":5000,"sku":"100003"},{"\_id":"54d63225726c345122abf222","replacement":true,"quantity":183,"name":"Cooling Fan","maintenanceperiod":0,"lifespan":0,"cost":300,"sku":"100004"},{"\_id":"54d63225726c345122abf223","replacement":true,"quantity":143,"name":"Fuse 20A","maintenanceperiod":0,"lifespan":0,"cost":22,"sku":"100005"}]');response.setStatusCode(200);return response;}}WarehouseSyncScheduleglobal with sharing class WarehouseSyncSchedule implements Schedulable {global void execute (SchedulableContext ctx){System.enqueueJob(new WarehouseCalloutService());}}WarehouseSyncScheduleTest@isTestpublic with sharing class WarehouseSyncScheduleTest {@isTest static void test() {String scheduleTime = '00 00 00 \* \* ? \*';Test.startTest();Test.setMock(HttpCalloutMock.class, new WarehouseCalloutServiceMock());String jobId = System.schedule('Warehouse Time to Schedule to test', scheduleTime, new WarehouseSyncSchedule());CronTrigger c = [SELECT State FROM CronTrigger WHERE Id =: jobId];System.assertEquals('WAITING', String.valueOf(c.State), 'JobId does not match');Test.stopTest();}}          |  | | --- | |  |  |  | | --- | |  | |

## 