Apex Triggers

1.Build Apex Triggers -ClosedOpportunityTrigger

```
trigger ClosedOpportunityTrigger on Opportunity (after insert, after update) {
 List tasklist = new List();
for(Opportunity opp: Trigger.New){
 if(opp.StageName == 'Closed Won'){
 tasklist.add(new Task(Subject = 'Follow Up Test Task', WhatId = opp.Id));
  }    }
 if(tasklist.size()>0){
 insert tasklist;    }
```

Asynchronous Apex

Use Future Methods--AccountProcessor

```
 public class AccountProcessor {
 @Future
public static void countContacts(List accountIds){
 Map> accContacts = new Map>();
List accsForUpdate = new List();
 if(accountIds != null){
For(Account acc : [SELECT id,(SELECT id,Name FROM Contacts)FROM Account where
id in: accountIds]){
  accContacts.put(acc.Id,acc.contacts); }
for(Id key : accContacts.keySet()){
Account a = New Account(id = key);
a.Number_of_Contacts__c = accContacts.get(key).size();
accsForUpdate.add(a);           }
```

```
update accsForUpdate;
}
 }
}
```

AccountProcessor Test

```
@isTest
 public class AccountProcessorTest {
 @testSetup
static void setupAccount(){
 List accounts = RandomAccountContactFactory.generateRandomAccounts(1);
 insert accounts;
 List contacts = RandomAccountContactFactory.generateRandomContacts(3, 'TestAP' ,
accounts.get(0).id);
 insert contacts;
 }
 @isTest
  static void testAccountProcessor(){
List accIds = new List();
for(Account a : [select id from Account]){
accIds.add(a.id);        }
Test.startTest();
AccountProcessor.countContacts(accIds);
Test.stopTest();
 }
}
```

Use Batch Apex--LeadProcessor

```
 global class LeadProcessor implements
 Database.Batchable, Database.Stateful {
// instance member to retain state across transactions     global Integer
recordsProcessed = 0;
global Database.QueryLocator start(Database.BatchableContext bc) {
return Database.getQueryLocator('SELECT Id, LeadSource FROM Lead');     }
global void execute(Database.BatchableContext bc, List scope){
```

```
// process each batch of records
 List leads = new List();
 for (Lead lead : scope) {
 lead.LeadSource = 'Dreamforce';
   // increment the instance member counter
 recordsProcessed = recordsProcessed + 1;        }
update leads;     }
 global void finish(Database.BatchableContext bc){
System.debug(recordsProcessed + ' records processed. Shazam!');
 }
}
```

LeadProcessor Test

```
@isTest
 public class LeadProcessorTest {
@testSetup
static void setup() {
List leads = new List();
// insert 200 leads
for (Integer i=0;i<200;i++) {
leads.add(new Lead(LastName='Lead '+i,
 Company='Lead', Status='Open - Not Contacted'));        }
insert leads;     }
static testmethod void test() {
 Test.startTest();
 LeadProcessor lp = new LeadProcessor();
   Id batchId = Database.executeBatch(lp, 200);
 Test.stopTest();
// after the testing stops, assert records were updated properly
System.assertEquals(200, [select count() from lead where LeadSource = 'Dreamforce']);
} }
Control Process with Queueable Apex ---  AddPrimaryContact
public class AddPrimaryContact implements Queueable {
public contact c;
public String state;
```

```
 public AddPrimaryContact(Contact c, String state) {
this.c = c;
 this.state = state;    }
public void execute(QueueableContext qc) {
system.debug('this.c = '+this.c+' this.state = '+this.state);
List acc_lst = new List([select id, name, BillingState from account where
account.BillingState = :this.state limit 200]);
   List c_lst = new List();
for(account a: acc_lst) {
contact c = new contact();
c = this.c.clone(false, false, false, false);
 c.AccountId = a.Id;
 c_lst.add(c);       }
 insert c_lst;    } }
```

AddPrimaryContact Test

```
@IsTest
public class AddPrimaryContactTest {
 @IsTest
public static void testing() {
List acc_lst = new List();
for (Integer i=0; i<50;i++) {
account a = new account(name=string.valueOf(i),billingstate='NY');
system.debug('account a = '+a);
acc_lst.add(a);
  }
  for (Integer i=0; i<50;i++) {
   account a = new account(name=string.valueOf(50+i),billingstate='CA');
system.debug('account a = '+a);
     acc_lst.add(a);
   }
  insert acc_lst;
 Test.startTest();
contact c = new contact(lastname='alex');
```

```
AddPrimaryContact apc = new AddPrimaryContact(c,'CA');        s
ystem.debug('apc = '+apc);
System.enqueueJob(apc);
Test.stopTest();
List c_lst = new List([select id from contact]);
Integer size = c_lst.size();
   system.assertEquals(50, size);     } }
```

Schedule Jobs Using The Apex Scheduler Unit--DailyLeadProcessor

```
global class DailyLeadProcessor implements Schedulable {
  global void execute(SchedulableContext ctx) {
  List lList = [Select Id, LeadSource from Lead where LeadSource = null];
if(!lList.isEmpty()) {
for(Lead l: lList) { l.LeadSource = 'Dreamforce'; } update lList;
}     } }
DailyLeadProcessorTest
 @isTest
private class DailyLeadProcessorTest {
static testMethod void testDailyLeadProcessor() {
String CRON_EXP = '0 0 1 * * ?'; List lList = new List();
 for (Integer i = 0; i < 200; i++) {
 lList.add(new Lead(LastName='Dreamforce'+i, Company='Test1 Inc.', Status='Open -
Not Contacted')); }
insert lList; Test.startTest();
String jobId = System.schedule('DailyLeadProcessor', CRON_EXP, new
DailyLeadProcessor());
 }
}
```

Apex REST Callouts--AnimalLocator

```
public class AnimalLocator {
public class cls_animal {
 public Integer id;
```

```
public String name;
public String eats;
public String says; }
 public class JSONOutput{
public cls_animal animal;
//public JSONOutput parse(String json){
//return (JSONOutput) System.JSON.deserialize(json, JSONOutput.class); //}
}

public static String getAnimalNameById (Integer id) {
Http http = new Http();
HttpRequest request = new HttpRequest();
 request.setEndpoint('https://th-apex-http-callout.herokuapp.com/animals/' + id);
//request.setHeader('id', String.valueof(id)); -- cannot be used in this challenge :)
request.setMethod('GET');
HttpResponse response = http.send(request);
system.debug('response: ' + response.getBody());
//Map map_results = (Map) JSON.deserializeUntyped(response.getBody());
 jsonOutput results = (jsonOutput) JSON.deserialize(response.getBody(),
jsonOutput.class);
 //Object results = (Object) map_results.get('animal');
system.debug('results= ' + results.animal.name); return(results.animal.name);
} }
```

AnimalLocator Test

```
@IsTest
public class AnimalLocatorTest {
@isTest public static void testAnimalLocator() {
Test.setMock(HttpCalloutMock.class, new AnimalLocatorMock());
//Httpresponse response = AnimalLocator.getAnimalNameById(1);
 String s = AnimalLocator.getAnimalNameById(1);
system.debug('string returned: ' + s);
 } }
```

AnimalLocatorMock

```
@IsTest
global class AnimalLocatorMock implements HttpCalloutMock {
global HTTPresponse respond(HTTPrequest request) {
Httpresponse response = new Httpresponse();
 response.setStatusCode(200);
//-- directly output the JSON, instead of creating a logic
 //response.setHeader('key, value)
//Integer id = Integer.valueof(request.getHeader('id'));
//Integer id = 1; //List lst_body = new List {'majestic badger', 'fluffy bunny'};
//system.debug('animal return value: ' + lst_body[id]);
response.setBody('{"animal":{"id":1,"name":"chicken","eats":"chicken food","says":"cluck
cluck"}}');
return response;
 } }
```

Apex SOAP callouts--ParkService

```
public class ParkService {
public class byCountryResponse {
public String[] return_x;
private String[] return_x_type_info = new String[]{
'return','http://parks.services/',null,'0','- 1','false'};
private String[] apex_schema_type_info = new
String[]{'http://parks.services/','false','false'};
private String[] field_order_type_info = new String[]{'return_x'};
 }
public class byCountry {
 public String arg0;
private String[] arg0_type_info = new
String[]{'arg0','http://parks.services/',null,'0','1','false'};
 private String[] apex_schema_type_info = new
String[]{'http://parks.services/','false','false'};
 private String[] field_order_type_info = new String[]{'arg0'};
}
 public class ParksImplPort {
public String endpoint_x = 'https://th-apex-soap-service.herokuapp.com/service/parks';
public Map inputHttpHeaders_x;
```

```
 public Map outputHttpHeaders_x;
 public String clientCertName_x;
 public String clientCert_x;
 public String clientCertPasswd_x;
 public Integer timeout_x;
private String[] ns_map_type_info = new String[]{'http://parks.services/', 'ParkService'}

public String[] byCountry(String arg0) {
ParkService.byCountry request_x = new ParkService.byCountry();
 request_x.arg0 = arg0;
ParkService.byCountryResponse response_x;
 Map response_map_x = new Map();
response_map_x.put('response_x', response_x);
WebServiceCallout.invoke( this, request_x, response_map_x, new String[]{endpoint_x, ''
'http://parks.services/', 'byCountry', 'http://parks.services/', 'byCountryResponse',
'ParkService.byCountryResponse'} );
 response_x = response_map_x.get('response_x'); return response_x.return_x;
 } } }

ParkServiceMock
@isTest
global class ParkServiceMock implements WebServiceMock {
global void doInvoke( Object stub,
Object request,
Map response,
String endpoint,
String soapAction,
String requestName,
String responseNS,
String responseName,
String responseType) {
 ParkService.byCountryResponse response_x = new ParkService.byCountryResponse();
response_x.return_x = new List < String > {'a', 'b'}; response.put('response_x',
response_x); } }
```

ParkLocator

```
public class ParkLocator {
```

```
public static List < String > country(String Country) {
ParkService.ParksImplPort obj = new ParkService.ParksImplPort();
return obj.byCountry(Country);
 } }
```

ParkLocator Test

```
@isTest
 private class ParkLocatorTest {
 @isTest
static void testCallout() {
 Test.setMock(WebServiceMock.class, new ParkServiceMock());
List < String > result = ParkLocator.country('Test');
 } }
```

Apex Web Services--- 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;
 } }
```

AccountManager Test

```
 @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;
} }
```

Apex Testing--- Get Started with Apex Unit Tests

```
 VerifyDate-
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
 @TestVisible
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
@TestVisible
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; } }
```

Test VerifyDate

```
@isTest private
class TestVerifyDate{
 @isTest
static void Test_CheckDates_case1(){
 Date D=VerifyDate.CheckDates(date.parse('01/01/2020'),date.parse('01/05/2020'));
System.assertEquals(date.parse('01/05/2020'),D); }
 @isTest
static void Test_CheckDates_case2(){
Date D=VerifyDate.CheckDates(date.parse('01/01/2020'),date.parse('05/05/2020'));
System.assertEquals(date.parse('01/31/2020'),D); }
@isTest
static void Test_DateWithin30Days_casel(){
Boolean
flag=VerifyDate.DateWithin30Days(date.parse('01/01/2020'),date.parse('12/30/2019'));
System.assertEquals(false,flag); }
 @isTest
static void Test_DateWithin30Days_case2(){
 Boolean flag =
VerifyDate.DateWithin30Days(date.parse('01/01/2020'),date.parse('02/02/2019'));
System.assertEquals(false,flag); }
 @isTest
static void Test_DateWithin30Days_case3(){
Boolean
flag=VerifyDate.DateWithin30Days(date.parse('01/01/2020'),date.parse('01/15/2020'));
System.assertEquals(true,flag); }
 @isTest
static void Test_SetEndOfMonthDate(){
```

```
Date returndate=VerifyDate.SetEndOfMonthDate(date.parse('01/01/2020'));
 } }
Test Apex Triggers Units RestrictContactByNametrigger
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
public class TestRestrictContactByName { @isTest static void Test_insertupdateContact() {
 Contact cnt=new Contact();
cnt.LastName='INVALIDNAME';
Test.startTest();
Database.SaveResult result = Database.insert(cnt,false);
 Test.stopTest();
System.assert(!result.isSuccess());
 System.assert(result.getErrors().size()>0);
System.assertEquals('The Last Name"INVALIDNAME" is not allowed for
DML',result.getErrors()[0].getMessage());
 } }
```

Create Test Data for Apex Tests Unit

RandomAccountContactFactory

```
public class RandomAccountContactFactory {
public static List generateRandomContacts (Integer numContacts, String lastName,Id accId){
 List contacts = new List();
 for(integer i = 0; i generateRandomAccounts (Integer numAccounts){
 List accounts = new List();
for(integer i = 0; i <newContact();i++){

Contact c = new Contact();
 c.FirstName = 'Trail' + i;
 c.LastName = lastName + i;
```

```
 c.AccountId = accId;
contacts.add(c);
}
return contacts;
}
Contact c = new Contact();
c.FirstName = 'Trail' + i;
c.LastName = lastName + i;
 c.AccountId = accId;
contacts.add(c);
 } return contacts;
}
```

MaintenanceRequest

```
trigger MaintenanceRequest on Case (before update, after update) {
if(Trigger.isUpdate && Trigger.isAfter){
MaintenanceRequestHelper.updateWorkOrders(Trigger.New, Trigger.OldMap);
 } }
MaintenanceRequestHelper
public with sharing class MaintenanceRequestHelper {
 public static void updateworkOrders(List updWorkOrders, Map nonUpdCaseMap) {
 Set validIds = new Set();
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);
 } } }
 //When an existing maintenance request of type Repair or Routine Maintenance is
closed,
//create a new maintenance request for a future routine checkup.
 if (!validIds.isEmpty()){
Map closedCases = new Map([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 maintenanceCycles = new Map();
 //calculate the maintenance request due dates by using the maintenance cycle defined
on the related equipment records.
 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 newCases = new List();
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 multiple pieces of equipment are used in the maintenance request,
//define the due date by applying the shortest maintenance cycle to today's date.
//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);
}

WarehouseCalloutService public with sharing class WarehouseCalloutService implements Queueable { private static final String WAREHOUSE_URL = 'https://th-superbadgeapex.herokuapp.com/equipment'; //Write a class that makes a REST callout to an external warehouse system to get a list of equipment that needs to be updated. //The callout's JSON response returns the equipment records that you upsert in Salesforce. @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 product2List = new List(); System.debug(response.getStatusCode()); if (response.getStatusCode() == 200){ List jsonResponse = (List)JSON.deserializeUntyped(response.getBody()); System.debug(response.getBody()) //class maps the following fields: //warehouse SKU will be external ID for identifying which equipment records to update within Salesforce for (Object jR : jsonResponse){ Map mapJson = (Map)jR; Product2 product2 = new

Product2(); //replacement part (always true), product2.Replacement_Part__c = (Boolean) mapJson.get('replacement'); //cost product2.Cost__c = (Integer) mapJson.get('cost'); //current inventory product2.Current_Inventory__c = (Double) mapJson.get('quantity'); //lifespan product2.Lifespan_Months__c = (Integer) mapJson.get('lifespan'); //maintenance cycle product2.Maintenance_Cycle__c = (Integer) mapJson.get('maintenanceperiod'); //warehouse SKU 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'); } } WarehouseCalloutServiceTest @IsTest private class WarehouseCalloutServiceTest { // implement your mock callout test here @isTest static void testWarehouseCallout() { test.startTest(); test.setMock(HttpCalloutMock.class, new WarehouseCalloutServiceMock()); WarehouseCalloutService.execute(null); test.stopTest(); List product2List = new List(); product2List = [SELECT ProductCode FROM Product2]; System.assertEquals(3, product2List.size()); System.assertEquals('55d66226726b611100aaf741', product2List.get(0).ProductCode); System.assertEquals('55d66226726b611100aaf742', product2List.get(1).ProductCode); System.assertEquals('55d66226726b611100aaf743', product2List.get(2).ProductCode); } } WarehouseSyncSchedule global class WarehouseSyncSchedule implements Schedulable { global void execute(SchedulableContext ctx) { WarehouseCalloutService.runWarehouseEquipmentSync(); } } WarehouseSyncScheduleTest @isTest public class WarehouseSyncScheduleTest { @isTest static void WarehousescheduleTest(){ String scheduleTime = '00 00 01 * * ?'; Test.startTest(); Test.setMock(HttpCalloutMock.class, new WarehouseCalloutServiceMock()); String jobID=System.schedule('Warehouse Time To Schedule to Test', scheduleTime, new WarehouseSyncSchedule()); Test.stopTest(); //Contains schedule information for a scheduled job. CronTrigger is similar to a cron job on UNIX systems. // This object is available in API version 17.0 and later. CronTrigger a=[SELECT Id FROM CronTrigger where NextFireTime > today]; System.assertEquals(jobID, a.Id,'Schedule '); } } MaintenanceRequestHelperTest @isTest public with sharing class MaintenanceRequestHelperTest { // createVehicle private static Vehicle__c createVehicle(){ Vehicle__c vehicle = new Vehicle__C(name = 'Testing Vehicle'); return vehicle; } // createEquipment 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; } //
createMaintenanceRequest 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;
} // createEquipmentMaintenanceItem 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; } @isTest
private 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__c from case where status ='New'];
Equipment_Maintenance_Item__c workPart = [select id from
Equipment_Maintenance_Item__c where Maintenance_Request__c =:newCase.Id]; list
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()); } @isTest private
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 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); }
```

@isTest private static void testBulk(){ list vehicleList = new list(); list equipmentList = new list(); list equipmentMaintenanceItemList = new list(); list caseList = new list(); list oldCaseIds = new list(); 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 newCase = [select id from case where status ='New']; list workParts = [select id from Equipment_Maintenance_Item__c where Maintenance_Request__c in: oldCaseIds]; system.assert(newCase.size() == 300); list allCase = [select id from case]; system.assert(allCase.size() == 600); } } WarehouseCalloutServiceMock @isTest global class WarehouseCalloutServiceMock implements HttpCalloutMock { // implement http mock callout global static HttpResponse respond(HttpRequest request) { HttpResponse response = new HttpResponse(); response.setHeader('Content-Type', 'application/json'); response.setBody('[{"_id":"55d66226726b611100aaf741","replacement":false,"quantity":5,"name": "Generator 1000 kW","maintenanceperiod":365,"lifespan":120,"cost":5000,"sku":"100003"},{"_id":"55d66226726b611 100aaf742","replacement":true,"quantity":183,"name":"Cooling Fan","maintenanceperiod":0,"lifespan":0,"cost":300,"sku":"100004"},{"_id":"55d66226726b611100a af743","replacement":true,"quantity":143,"name":"Fuse 20A","maintenanceperiod":0,"lifespan":0,"cost":22,"sku":"100005"}]'); response.setStatusCode(200); return response; } } WarehouseCalloutServiceTest @IsTest private class WarehouseCalloutServiceTest { // implement your mock callout test here @isTest static void testWarehouseCallout() { test.startTest(); test.setMock(HttpCalloutMock.class, new WarehouseCalloutServiceMock()); WarehouseCalloutService.execute(null); test.stopTest(); List product2List = new List(); product2List = [SELECT ProductCode FROM Product2]; System.assertEquals(3, product2List.size()); System.assertEquals('55d66226726b611100aaf741', product2List.get(0).ProductCode); System.assertEquals('55d66226726b611100aaf742', product2List.get(1).ProductCode); System.assertEquals('55d66226726b611100aaf743', product2List.get(2).ProductCode); } } WarehouseSyncSchedule global class WarehouseSyncSchedule implements Schedulable { global void execute(SchedulableContext ctx) { WarehouseCalloutService.runWarehouseEquipmentSync(); } }

WarehouseSyncScheduleTest @isTest public class WarehouseSyncScheduleTest { @isTest static void WarehousescheduleTest(){ String scheduleTime = '00 00 01 * * ?'; Test.startTest(); Test.setMock(HttpCalloutMock.class, new WarehouseCalloutServiceMock()); String jobID=System.schedule('Warehouse Time To Schedule to Test', scheduleTime, new WarehouseSyncSchedule()); Test.stopTest(); //Contains schedule information for a scheduled job. CronTrigger is similar to a cron job on UNIX systems. // This object is available in API version 17.0 and later. CronTrigger a=[SELECT Id FROM CronTrigger where NextFireTime > today]; System.assertEquals(jobID, a.Id,'Schedule '); } }