

## Apex Triggers : Bulk Apex Triggers

### ClosedOpportunityTrigger

```
trigger ClosedOpportunityTrigger on Opportunity (after insert,after update) {  
    List<Task> tasklist=new List<Task>();  
    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;  
    }  
}
```

## Apex Testing : 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
@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;
}
}
```

## TestVerifyDate Class

```
@isTest
public 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_case1(){

        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'));
    }

}
```

## Apex Testing : Test Apex Triggers

### Apex trigger 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

```
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());

    }

}

}
```

## Apex Testing : Create Test Data for Apex Tests

### RandomContactFactory Class

```
public class RandomContactFactory {

    public static List<Contact> generateRandomContacts(Integer numcnt,string lastname){

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

```
for(Integer i=0;i<numcnt;i++){  
    Contact cnt=new Contact(FirstName='Test'+i,LastName= lastname);  
    contacts.add(cnt);  
}  
return contacts;  
  
}  
}
```

## Asynchronous Apex : Use Future Method

### AccountProcessor class

```
public class AccountProcessor {  
    @future  
    public static void countContacts(List<Id> accountIds){  
        List<Account> accountsToUpdate = new List<Account>();  
        List<Account> accounts=[Select Id,Name,(Select Id from Contacts) from Account Where Id in  
            :accountIds];  
        For(Account acc: accounts){  
            List<Contact> contactList = acc.Contacts;  
            acc.Number_Of_Contacts__c=contactList.size();  
            accountsToUpdate.add(acc);  
        }  
        update accountsToUpdate;  
    }  
}
```

### AccountProcessorTest class

```
@IsTest
```

```
private class AccountProcessorTest {  
    @IsTest  
    private static void testCountContacts(){  
        Account newAccount = new Account(Name='Test Account');  
        insert newAccount;  
        Contact newContact1= new Contact(FirstName  
            ='John',LastName='Doe',AccountId=newAccount.Id);  
        insert newContact1;  
        Contact newContact2= new Contact(FirstName  
            ='Jane',LastName='Doe',AccountId=newAccount.Id);  
        insert newContact2;  
        List<Id> accountIds=new List<Id>();  
        accountIds.add(newAccount.Id);  
  
        Test.startTest();  
        AccountProcessor.countContacts(accountIds);  
        Test.stopTest();  
    }  
}
```

## Asynchronous Apex : Use Batch Apex

### LeadProcessor class

```
global class LeadProcessor implements Database.Batchable<sObject>{  
    global Integer count =0;  
    global Database.QueryLocator start(Database.BatchableContext bc){  
        return Database.getQueryLocator('SELECT ID, LeadSource FROM Lead');  
    }  
    global void execute (Database.BatchableContext bc,List<Lead> L_list){  
        List<lead> L_list_new = new List<lead>();
```

```
for(lead L:L_list){
    L.leadsource='DreamForce';
    L_list_new.add(L);
    count+=1;
}
update L_list_new;
}
global void finish(Database.BatchableContext bc){
    system.debug('count = '+ count);
}
}
```

### LeadProcessorTest class

```
@isTest
public class LeadProcessorTest {
    @isTest
    public static void testit(){
        List<lead> L_list=new List<lead>();
        for(Integer i=0;i<200;i++){
            Lead L=new lead();
            L.LastName='name' + i;
            L.Company='Company';
            L.Status='Random Status';
            L_list.add(L);
        }
        insert L_list;
        Test.startTest();
        LeadProcessor lp=new LeadProcessor();
        Id batchId=Database.executeBatch(lp);
        Test.stopTest();
    }
}
```

```
}  
}
```

## Asynchronous Apex : Control Processes with Queueable Apex

### AddPrimaryContact class

```
public class AddPrimaryContact implements Queueable {  
    private Contact con;  
    private String state;  
    public AddPrimaryContact(Contact con,String state){  
        this.con=con;  
        this.state=state;  
    }  
    public void execute(QueueableContext context){  
        List<Account> accounts=[Select Id, Name,(Select FirstName,LastName,Id from contacts) from  
            Account where BillingState=:state Limit 200];  
        List<Contact> primaryContacts=new List<Contact>();  
        for(Account acc:accounts){  
            Contact c=con.clone();  
            c.AccountId=acc.Id;  
            primaryContacts.add(c);  
        }  
        if(primaryContacts.size()>0){  
            insert primaryContacts;  
        }  
    }  
}
```



## AddPrimaryContactTest

@isTest

```
public class AddPrimaryContactTest {  
    static testmethod void testQueueable(){  
        List<Account> testAccounts=new List<Account>();  
        for(Integer i=0;i<50;i++)testAccounts.add(new Account(Name='Account'+i,BillingState='CA'));  
        for(Integer j=0;j<50;j++)testAccounts.add(new Account(Name='Account'+j,BillingState='NY'));  
        insert testAccounts;  
        Contact testContact = new Contact(FirstName='John',LastName='Doe');  
        insert testContact;  
        AddPrimaryContact addit=new addPrimaryContact(testContact,'CA');  
        Test.startTest();  
        system.enqueueJob(addit);  
        Test.stopTest();  
        System.assertEquals(50,[Select count() from Contact where accountId in(Select Id from Account  
            where BillingState='CA')]);  
    }  
}
```

## Asynchronous Apex : Schedule Jobs Using the Apex Scheduler

### DailyLeadProcessor

```
public class DailyLeadProcessor implements Schedulable {  
    Public void execute(SchedulableContext SC){
```

```
List<Lead> LeadObj=[SELECT Id from Lead where LeadSource=null limit 20];
for(Lead l:LeadObj){
    l.LeadSource='Dreamforce';
    update l;
}
}
```

## DailyLeadProcessorTest

@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.startTest();

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

```
}
```

## Apex Integration Services -Apex REST Callouts

### 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');

        String strResp="";

        HttpResponse res = http.send(req);

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

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

            Map<string,object> animals=(map<string,object>)results.get('animal');

            strResp=string.valueOf(animals.get('name'));

        }

        return strResp;

    }

}
```

### 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('{"animal":{"id":1,"name":"chicken","eats":"chicken food","says":"cluck cluck"}}');  
        response.setStatusCode(200);  
        return response;  
    }  
}
```

## Apex Integration Services -Apex SOAP Callouts

### ParkLocator

```
public class ParkLocator {  
    public static String[] country(String theCountry) {  
        ParkService.ParksImplPort parkSvc = new ParkService.ParksImplPort(); // remove space  
        return parkSvc.byCountry(theCountry);  
    }  
}
```

### ParkLocatorTest

@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

@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) {

ParkService.byCountryResponse response\_x = new ParkService.byCountryResponse();

response\_x.return\_x = new List<String>{'Yellowstone', 'Mackinac National Park', 'Yosemite'};

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

}

}

## Apex Integration Services-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;

    }

}
```

### AccountManagerTest

```
@isTest

private class AccountManagerTest {

    private static testMethod void getAccountTest1() {

        Id recordId = createTestRecord();

        RestRequest request = new RestRequest();

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

            +'/contacts' ;

        request.httpMethod = 'GET';

        RestContext.request = request;

    }

}
```

```
Account thisAccount = AccountManager.getAccount();

System.assert(thisAccount != null);

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

}

static Id createTestRecord() {
    Account TestAcc = new Account(
        Name='Test record');
    insert TestAcc;
    Contact TestCon= new Contact(
        LastName='Test',
        AccountId = TestAcc.id);
    return TestAcc.Id;
}
}
```

# Apex Superbadge

## 1-Automate record creation

### MaintenanceRequestHelper.apxc :-

```
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);
        }
    }
}

//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<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>();

    //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<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 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);
}

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;  
  }  
}  
}
```

### **MaintenanceRequest.apxt :-**

```
trigger MaintenanceRequest on Case (before update, after update) {  
    if(Trigger.isUpdate && Trigger.isAfter){  
        MaintenanceRequestHelper.updateWorkOrders(Trigger.New, Trigger.OldMap);  
    }  
}
```

### **Challenge 2-Synchronize Salesforce data with an external system**

#### **WarehouseCalloutService.apxc :-**

```
public with sharing class WarehouseCalloutService {  
  
    private static final String WAREHOUSE_URL = 'https://th-superbadge-  
apex.herokuapp.com/equipment';  
  
    //@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);
```

```
List<Product2> warehouseEq = new List<Product2>();
```

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

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

```
    System.debug(response.getBody());
```

```
    for (Object eq : jsonResponse){
```

```
        Map<String,Object> mapJson = (Map<String,Object>)eq;
```

```
        Product2 myEq = new Product2();
```

```
        myEq.Replacement_Part__c = (Boolean) mapJson.get('replacement');
```

```
        myEq.Name = (String) mapJson.get('name');
```

```
        myEq.Maintenance_Cycle__c = (Integer) mapJson.get('maintenanceperiod');
```

```
        myEq.Lifespan_Months__c = (Integer) mapJson.get('lifespan');
```

```
        myEq.Cost__c = (Decimal) mapJson.get('lifespan');
```

```
        myEq.Warehouse_SKU__c = (String) mapJson.get('sku');
```

```
        myEq.Current_Inventory__c = (Double) mapJson.get('quantity');
```

```
        warehouseEq.add(myEq);
```

```
    }
```

```
if (warehouseEq.size() > 0){
```

```
    upsert warehouseEq;
```

```
    System.debug('Your equipment was synced with the warehouse one');
```

```
    System.debug(warehouseEq);
```

```
}
```

```
}
```

```
}
```

```
}
```

## Challenge 3-Schedule synchronization using Apex code

```
global with sharing class WarehouseSyncSchedule implements Schedulable {  
    // implement scheduled code here  
    global void execute (SchedulableContext ctx){  
        System.enqueueJob(new WarehouseCalloutService());  
    }  
}
```

## Challenge 4 Test automation logic

### MaintenanceRequestHelperTest.apxc :-

```
@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<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());  
}
```

@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<case> allCase = [select id from case];
```

```
Equipment_Maintenance_Item__c equipmentMaintenanceltem = [select id
                    from Equipment_Maintenance_Item__c
                    where Maintenance_Request__c = :createdCase.Id];
```

```
system.assert(equipmentMaintenanceltem != null);
system.assert(allCase.size() == 1);
}
```

@isTest

```
private static void testBulk(){
    list<Vehicle__C> vehicleList = new list<Vehicle__C>();
    list<Product2> equipmentList = new list<Product2>();
    list<Equipment_Maintenance_Item__c> equipmentMaintenanceltemList = 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++){
```



```
equipmentMaintenanceItem.add(createEquipmentMaintenanceItem(equipmentList.get(i).id,  
caseList.get(i).id));
```

```
}
```

```
insert equipmentMaintenanceItem;
```

```
test.startTest();
```

```
for(case cs : caseList){
```

```
    cs.Status = 'Closed';
```

```
    oldCaseIds.add(cs.Id);
```

```
}
```

```
update caseList;
```

```
test.stopTest();
```

```
list<case> newCase = [select id
```

```
    from case
```

```
    where status ='New'];
```

```
list<Equipment_Maintenance_Item__c> workParts = [select id
```

```
    from Equipment_Maintenance_Item__c
```

```
    where Maintenance_Request__c in: oldCaseIds];
```

```
system.assert(newCase.size() == 300);
```

```
list<case> allCase = [select id from case];
```

```
system.assert(allCase.size() == 600);
```

```
}
```

```
}
```

**MaintenanceRequestHelper.apxc :**

```
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);
                }
            }
        }
    }

    //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<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>();

        //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<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 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);  
}  
  
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;
}
}
```

### **MaintenanceRequest.apxt :-**

```
trigger MaintenanceRequest on Case (before update, after update) {
    if(Trigger.isUpdate && Trigger.isAfter){
        MaintenanceRequestHelper.updateWorkOrders(Trigger.New, Trigger.OldMap);
    }
}
```

## **Challenge 5-Test callout logic**

### **WarehouseCalloutService.apxc :-**

```
public with sharing class WarehouseCalloutService {

    private static final String WAREHOUSE_URL = 'https://th-superbadge-
apex.herokuapp.com/equipment';
```

```
//@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);

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

    if (response.getStatusCode() == 200){
        List<Object> jsonResponse = (List<Object>)JSON.deserializeUntyped(response.getBody());
        System.debug(response.getBody());

        for (Object eq : jsonResponse){
            Map<String, Object> mapJson = (Map<String, Object>)eq;
            Product2 myEq = new Product2();
            myEq.Replacement_Part__c = (Boolean) mapJson.get('replacement');
            myEq.Name = (String) mapJson.get('name');
            myEq.Maintenance_Cycle__c = (Integer) mapJson.get('maintenanceperiod');
            myEq.Lifespan_Months__c = (Integer) mapJson.get('lifespan');
            myEq.Cost__c = (Decimal) mapJson.get('lifespan');
            myEq.Warehouse_SKU__c = (String) mapJson.get('sku');
            myEq.Current_Inventory__c = (Double) mapJson.get('quantity');
            warehouseEq.add(myEq);
        }

        if (warehouseEq.size() > 0){
```

```
        upsert warehouseEq;

        System.debug('Your equipment was synced with the warehouse one');

        System.debug(warehouseEq);
    }

}

}
```

### **WarehouseCalloutServiceTest.apxc :-**

@isTest

```
private class WarehouseCalloutServiceTest {

    @isTest

    static void testWareHouseCallout(){

        Test.startTest();

        // implement mock callout test here

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

        WarehouseCalloutService.runWarehouseEquipmentSync();

        Test.stopTest();

        System.assertEquals(1, [SELECT count() FROM Product2]);

    }

}
```

### **WarehouseCalloutServiceMock.apxc :-**

@isTest

```
global class WarehouseCalloutServiceMock implements HttpCalloutMock {

    // implement http mock callout

    global static HttpResponse respond(HttpRequest request){
```

```
System.assertEquals('https://th-superbadge-apex.herokuapp.com/equipment',
request.getEndpoint());

System.assertEquals('GET', request.getMethod());

// Create a fake response

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"}');

response.getStatusCode(200);

return response;
}
}
```

## Challenge 6-Test scheduling logic

### WarehouseSyncSchedule.apxc :-

```
global with sharing class WarehouseSyncSchedule implements Schedulable {

    // implement scheduled code here

    global void execute (SchedulableContext ctx){

        System.enqueueJob(new WarehouseCalloutService());

    }

}
```

### **WarehouseSyncScheduleTest.apxc :-**

@isTest

```
public with sharing class WarehouseSyncScheduleTest {
```

```
    // implement scheduled code here
```

```
    //
```

```
    @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();
```

```
    }
```

```
}
```