Apex Triggers

Get Started with Apex Triggers

Account Address Trigger

```
trigger AccountAddressTrigger on Account (before insert,before update) {
    for(Account acc:Trigger.new){
        if(acc.Match_Billing_Address__c==true){
            acc.ShippingPostalCode =acc.BillingPostalCode;
        }
    }
}
```

Bulk Apex Triggers

Closed Opportunity Trigger

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

Get Started with Apex Unit Tests Code

Verify Date 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;
        }

}

@isTest
public class TestVerifyDate {

   @isTest static void testCheckDates1(){
      Date d1= VerifyDate.CheckDates(Date.parse('01/01/2020'),Date.parse('01/19/2020'));
      System.assertEquals(Date.parse('01/19/2020'), d1);
   }
   @isTest static void testCheckDates2(){
      Date d1= VerifyDate.CheckDates(Date.parse('01/01/2020'),Date.parse('06/20/2020'));
      System.assertEquals(Date.parse('01/31/2020'), d1);
   }
}
```

Test Apex Triggers

Restrict ContactBy Name trigger

```
trigger RestrictContactByName on Contact (before insert, before update) {
        For (Contact c : Trigger.New) {
                if(c.LastName == 'INVALIDNAME')
    {
                        c.AddError('The Last Name "'+c.LastName+'" is not allowed for DML');
                }
        }
}
```

Test Restrict Contact By Name

```
@isTest
public class TestRestrictContactByName {
   @isTest static  void TestInsert(){
      Contact coen= new Contact(LastName='INVALIDNAME');
      Database.SaveResult res= Database.insert(coen,false);
      System.assertEquals('The Last Name "INVALIDNAME" is not allowed for DML',
res.getErrors()[0].getMessage());

   }
   @isTest static void  TestUpdate(){
      Contact coen = new Contact(FirstName='Joe');
      coen.LastName='INVALIDNAME';
      Database.SaveResult res= Database.update(coen,false);
      System.assertEquals('The Last Name "INVALIDNAME" is not allowed for DML',
res.getErrors()[0].getMessage());
   }
}
```

Create Test Data for Apex Tests

Random Contact Factory

```
public class RandomContactFactory
{
   public static List<Contact> generateRandomContacts(Integer num,String lname)
   {
      List<Contact> cent=new List<Contact>();
      for(Integer i=0;i<num;i++)
      {
```

```
            Contact c= new Contact(FirstName='Test'+i);
             cent.add(c);
        }
        return cent;
    }
}
```

Asynchronous Apex

Account Processor Test

```
@isTest
public class AccountProcessorTest {
    public static  Testmethod void Testaccprosstest(){
        Account a= new Account();
        a.Name='Test account';
        Insert a;
        Contact cont= new Contact();
        cont.FirstName='JJJ';
        cont.LastName='hase';
        cont.AccountId= a.Id;
        insert cont;
```

```
        List<Id> setaccd=new List<Id>();
        setaccd.add(a.Id);
        Test.startTest();
        AccountProcessor.countContacts(setaccd);
        Test.stopTest();
        Account accd=[SELECT Number_of_contacts__c    FROM Account WHERE Id= :a.Id LIMIT 1];
        System.assertEquals(Integer.valueOf(accd.Number_Of_Contacts__c), 1);


    }

}
```

Use Batch Apex

Lead Processor

```
global class LeadProcessor implements Database.Batchable <SObject> {

    global Database.QueryLocator start(Database.BatchableContext bc){
        String Query='Select id,LeadSource from Lead';
        return Database.getQueryLocator(Query);
            }
    global void execute(Database.BatchableContext bc, List<Lead> scope){
        for(Lead l: scope){
            l.LeadSource='DreamForce';
        }
        update scope;
    }

    global void finish(Database.BatchableContext bc){
        Id job= bc.getJobId();
        System.debug(job);
    }
}
```

Lead Processor Test

```
@istest
private class LeadProcessorTest {
    @istest
    static void tetslead(){
        List<Lead> l= new List<Lead>();
        lead l1= new Lead();
```

```
        l1.LastName='surya';
        l1.Company='Company';
        l1.Status='Closed-Converted';
        l1.LeadSource='Dreamforce';
        l.add(l1);
        insert l;

    Test.startTest();
    LeadProcessor lp= new LeadProcessor();
    Id jobid= Database.executeBatch(lp);
    Test.stopTest();
    }
}
```

Control Processes with Queueable Apex

Add Primary Contact

```
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;
        }
    }
```

```
}
```

Add PrimaryContact Test

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

Schedule Jobs Using the Apex Scheduler

Daily Lead Processor

```
global class DailyLeadProcessor implements Schedulable{
   global void execute(SchedulableContext ctx){
      List<Lead> leads = [SELECT Id, LeadSource FROM Lead WHERE LeadSource = ''];

      if(leads.size() > 0){
         List<Lead> newLeads = new List<Lead>();
```

```
        for(Lead lead : leads){
            lead.LeadSource = 'DreamForce';
            newLeads.add(lead);
        }

        update newLeads;
    }
  }
}
```

DailyLeadProcessorTest

```
@isTest
private class DailyLeadProcessorTest{
    public static String CRON_EXP = '0 0 0 2 6 ? 2022';
   static testmethod void testScheduledJob(){
      List<Lead> leads = new List<Lead>();
      for(Integer i = 0; i < 200; i++){
         Lead lead = new Lead(LastName = 'Test ' + i, LeadSource = '', Company = 'Test Company ' + i,
Status = 'Open - Not Contacted');
         leads.add(lead);
      }

      insert leads;
      Test.startTest();
      String jobId = System.schedule('Update LeadSource to DreamForce', CRON_EXP, new
DailyLeadProcessor());
      Test.stopTest();
   }
}
```

Apex Integration Services

Apex REST Callouts

Animal Locator

```
public class AnimalLocator {
   public  static String  getAnimalNameById(Integer nu){
      Http http = new Http();
      String stRand;
      HttpRequest request = new HttpRequest();
      request.setEndpoint('https://th-apex-http-callout.herokuapp.com/animals/'+nu);
      request.setMethod('GET');
```

```
        HttpResponse response = http.send(request);
        if(response.getStatusCode() == 200) {
            Map<String, Object> results = (Map<String, Object>)
JSON.deserializeUntyped(response.getBody());
            Map<String, Object>  animals = (Map<String, Object> ) results.get('animal');
            stRand= string.valueOf(animals.get('name'));
        }

        return stRand;
    }
}
```

Animal Locator Mock

```
@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;
    }
}
```

Animal Locator Test

```
@isTest
public class AnimalLocatorTest{
    @isTest static void AnimalLocatorTest(){
        Test.setMock(HttpCallOutMock.class, new AnimalLocatorMock());
              string result=AnimalLocator.getAnimalNameById(1);
        string expectedResult='chicken';
        System.assertEquals(result, expectedResult);
    }
}
```

Apex SOAP Callouts

Park Service

```
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<String,String> inputHttpHeaders_x;
        public Map<String,String> 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<String, ParkService.byCountryResponse> response_map_x = new Map<String,
ParkService.byCountryResponse>();
            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;
```

```
    }
  }
}
```

ParkLocator

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

Park Service Mock

```
@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) {
      List<String> prk=new List<String> ();
       prk.add('Yosemite');
       prk.add('Yellowstone');
       prk.add('Another Park');
      ParkService.byCountryResponse response_x =
        new ParkService.byCountryResponse();
      response_x.return_x = prk;
      response.put('response_x', response_x);
    }
}
```

ParkLocatorTest

```
@isTest
public class ParkLocatorTest {
  @isTest static void testCallout() {
```

```
        Test.setMock(WebServiceMock.class, new ParkServiceMock());
        String country= 'United States';
        List<String> result = ParkLocator.country(country);
        List<String> prk=new List<String> ();
            prk.add('Yosemite');
            prk.add('Yellowstone');
            prk.add('Another Park');
        System.assertEquals(prk, result);
    }
}
```

Apex Web Services

Account Manager

```
@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');
        Account result =  [SELECT Id,Name ,(Select Id, Name from Contacts)FROM Account WHERE Id =
:accountId];
        return result;
    }
}
```

Account Manager Test

```
@isTest
private class AccountManagerTest {
    @isTest static void testGetContactsByAccountsId() {
        Id recordId = createTestRecord();
        RestRequest request = new RestRequest();
        request.requestUri = 'https://yourInstance.my.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 accountTest = new Account(
            Name='Test record');
        insert accountTest;
        Contact contactTest=new Contact(
            FirstName='John',
            LastName='Doe',
            AccountId=accountTest.Id);
        insert contactTest;
        return accountTest.Id;
    }
}
```

Apex Specialist Superbadge

Automated Record Creation

```
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()){
            List<Case> newCases = new List<Case>();
            Map<Id,Case> closedCasesM = 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'));
    }

      for(Case cc : closedCasesM.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));
        }

        newCases.add(nc);
      }

      insert newCases;

      List<Equipment_Maintenance_Item__c> clonedWPs = new
List<Equipment_Maintenance_Item__c>();
      for (Case nc : newCases){
        for (Equipment_Maintenance_Item__c wp :
closedCasesM.get(nc.ParentId).Equipment_Maintenance_Items__r){
          Equipment_Maintenance_Item__c wpClone = wp.clone();
          wpClone.Maintenance_Request__c = nc.Id;
          ClonedWPs.add(wpClone);

        }
      }
      insert ClonedWPs;
```

```
        }
    }
}

@istest
public with sharing class MaintenanceRequestHelperTest {

    private static final string STATUS_NEW = 'New';
    private static final string WORKING = 'Working';
    private static final string CLOSED = 'Closed';
    private static final string REPAIR = 'Repair';
    private static final string REQUEST_ORIGIN = 'Web';
    private static final string REQUEST_TYPE = 'Routine Maintenance';
    private static final string REQUEST_SUBJECT = 'Testing subject';

    PRIVATE STATIC Vehicle__c createVehicle(){
        Vehicle__c Vehicle = new Vehicle__C(name = 'SuperTruck');
        return Vehicle;
    }

    PRIVATE STATIC Product2 createEq(){
        product2 equipment = new product2(name = 'SuperEquipment',
                                          lifespan_months__C = 10,
                                          maintenance_cycle__C = 10,
                                          replacement_part__c = true);
        return equipment;
    }

    PRIVATE STATIC Case createMaintenanceRequest(id vehicleId, id equipmentId){
        case cs = new case(Type=REPAIR,
                           Status=STATUS_NEW,
                           Origin=REQUEST_ORIGIN,
                           Subject=REQUEST_SUBJECT,
                           Equipment__c=equipmentId,
                           Vehicle__c=vehicleId);
        return cs;
    }

    PRIVATE STATIC Equipment_Maintenance_Item__c createWorkPart(id equipmentId,id
requestId){
        Equipment_Maintenance_Item__c wp = new
```

```
Equipment_Maintenance_Item__c(Equipment__c = equipmentId,
                                              Maintenance_Request__c = requestId);
    return wp;
  }


  @istest
  private static void testMaintenanceRequestPositive(){
    Vehicle__c vehicle = createVehicle();
    insert vehicle;
    id vehicleId = vehicle.Id;

    Product2 equipment = createEq();
    insert equipment;
    id equipmentId = equipment.Id;

    case somethingToUpdate = createMaintenanceRequest(vehicleId,equipmentId);
    insert somethingToUpdate;

    Equipment_Maintenance_Item__c workP =
createWorkPart(equipmentId,somethingToUpdate.id);
    insert workP;

    test.startTest();
    somethingToUpdate.status = CLOSED;
    update somethingToUpdate;
    test.stopTest();

    Case newReq = [Select id, subject, type, Equipment__c, Date_Reported__c, Vehicle__c,
Date_Due__c
            from case
            where status =:STATUS_NEW];

    Equipment_Maintenance_Item__c workPart = [select id
                                from Equipment_Maintenance_Item__c
                                where Maintenance_Request__c =:newReq.Id];

    system.assert(workPart != null);
    system.assert(newReq.Subject != null);
    system.assertEquals(newReq.Type, REQUEST_TYPE);
    SYSTEM.assertEquals(newReq.Equipment__c, equipmentId);
```

```
    SYSTEM.assertEquals(newReq.Vehicle__c, vehicleId);
    SYSTEM.assertEquals(newReq.Date_Reported__c, system.today());
}

@istest
private static void testMaintenanceRequestNegative(){
    Vehicle__C vehicle = createVehicle();
    insert vehicle;
    id vehicleId = vehicle.Id;

    product2 equipment = createEq();
    insert equipment;
    id equipmentId = equipment.Id;

    case emptyReq = createMaintenanceRequest(vehicleId,equipmentId);
    insert emptyReq;

    Equipment_Maintenance_Item__c workP = createWorkPart(equipmentId, emptyReq.Id);
    insert workP;

    test.startTest();
    emptyReq.Status = WORKING;
    update emptyReq;
    test.stopTest();

    list<case> allRequest = [select id
                             from case];

    Equipment_Maintenance_Item__c workPart = [select id
                                              from Equipment_Maintenance_Item__c
                                              where Maintenance_Request__c = :emptyReq.Id];

    system.assert(workPart != null);
    system.assert(allRequest.size() == 1);
}

@istest
private static void testMaintenanceRequestBulk(){
    list<Vehicle__C> vehicleList = new list<Vehicle__C>();
    list<Product2> equipmentList = new list<Product2>();
    list<Equipment_Maintenance_Item__c> workPartList = new
```

```
list<Equipment_Maintenance_Item__c>();
        list<case> requestList = new list<case>();
        list<id> oldRequestIds = new list<id>();

        for(integer i = 0; i < 300; i++){
            vehicleList.add(createVehicle());
            equipmentList.add(createEq());
        }
        insert vehicleList;
        insert equipmentList;

        for(integer i = 0; i < 300; i++){
            requestList.add(createMaintenanceRequest(vehicleList.get(i).id, equipmentList.get(i).id));
        }
        insert requestList;

        for(integer i = 0; i < 300; i++){
            workPartList.add(createWorkPart(equipmentList.get(i).id, requestList.get(i).id));
        }
        insert workPartList;

        test.startTest();
        for(case req : requestList){
            req.Status = CLOSED;
            oldRequestIds.add(req.Id);
        }
        update requestList;
        test.stopTest();

        list<case> allRequests = [select id
                                  from case
                                  where status =: STATUS_NEW];

        list<Equipment_Maintenance_Item__c> workParts = [select id
                                                         from Equipment_Maintenance_Item__c
                                                         where Maintenance_Request__c in: oldRequestIds];

        system.assert(allRequests.size() == 300);
    }
}
```

Synchronize Salesforce data with an external system

```
public with sharing class WarehouseCalloutService implements Queueable {
    private static final String WAREHOUSE_URL = 'https://th-
    superbadgeapex.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 = (Integer) mapJson.get('cost');
        myEq.Warehouse_SKU__c = (String) mapJson.get('sku');
        myEq.Current_Inventory__c = (Double) mapJson.get('quantity');
        myEq.ProductCode = (String) mapJson.get('_id');
        warehouseEq.add(myEq);
    }
    if (warehouseEq.size() > 0){
        upsert warehouseEq;
        System.debug('Your equipment was synced with the warehouse one');
        }
      }
    }
    public static void execute (QueueableContext context){
        runWarehouseEquipmentSync();
       }
    }
```

Schedule synchronization using Apex code (Step 4)

WarehouseSyncShedule

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

Maintenance Request trigger

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

Maintenance Request Helper 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()){
      List<Case> newCases = new List<Case>();
      Map<Id,Case> closedCasesM = 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'));
        }

            for(Case cc : closedCasesM.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));
                }

                newCases.add(nc);
            }

            insert newCases;

            List<Equipment_Maintenance_Item__c> clonedWPs = new
List<Equipment_Maintenance_Item__c>();
            for (Case nc : newCases){
                for (Equipment_Maintenance_Item__c wp :
closedCasesM.get(nc.ParentId).Equipment_Maintenance_Items__r){
                    Equipment_Maintenance_Item__c wpClone = wp.clone();
                    wpClone.Maintenance_Request__c = nc.Id;
                    ClonedWPs.add(wpClone);

                }
            }
            insert ClonedWPs;
        }
    }
}
```

Maintenance Request Helper Test

```
@istest
public with sharing class MaintenanceRequestHelperTest {

    private static final string STATUS_NEW = 'New';
    private static final string WORKING = 'Working';
    private static final string CLOSED = 'Closed';
    private static final string REPAIR = 'Repair';
    private static final string REQUEST_ORIGIN = 'Web';
    private static final string REQUEST_TYPE = 'Routine Maintenance';
    private static final string REQUEST_SUBJECT = 'Testing subject';

    PRIVATE STATIC Vehicle__c createVehicle(){
        Vehicle__c Vehicle = new Vehicle__C(name = 'SuperTruck');
        return Vehicle;
    }

    PRIVATE STATIC Product2 createEq(){
        product2 equipment = new product2(name = 'SuperEquipment',
                                          lifespan_months__C = 10,
                                          maintenance_cycle__C = 10,
                                          replacement_part__c = true);
        return equipment;
    }

    PRIVATE STATIC Case createMaintenanceRequest(id vehicleId, id equipmentId){
        case cs = new case(Type=REPAIR,
                           Status=STATUS_NEW,
                           Origin=REQUEST_ORIGIN,
                           Subject=REQUEST_SUBJECT,
                           Equipment__c=equipmentId,
                           Vehicle__c=vehicleId);
        return cs;
    }

    PRIVATE STATIC Equipment_Maintenance_Item__c createWorkPart(id equipmentId,id
requestId){
        Equipment_Maintenance_Item__c wp = new
Equipment_Maintenance_Item__c(Equipment__c = equipmentId,
                                                    Maintenance_Request__c = requestId);
        return wp;
```

```
    }


    @istest
    private static void testMaintenanceRequestPositive(){
        Vehicle__c vehicle = createVehicle();
        insert vehicle;
        id vehicleId = vehicle.Id;

        Product2 equipment = createEq();
        insert equipment;
        id equipmentId = equipment.Id;

        case somethingToUpdate = createMaintenanceRequest(vehicleId,equipmentId);
        insert somethingToUpdate;

        Equipment_Maintenance_Item__c workP =
createWorkPart(equipmentId,somethingToUpdate.id);
        insert workP;

        test.startTest();
        somethingToUpdate.status = CLOSED;
        update somethingToUpdate;
        test.stopTest();

        Case newReq = [Select id, subject, type, Equipment__c, Date_Reported__c, Vehicle__c,
Date_Due__c
                    from case
                    where status =:STATUS_NEW];

        Equipment_Maintenance_Item__c workPart = [select id
                                    from Equipment_Maintenance_Item__c
                                    where Maintenance_Request__c =:newReq.Id];

        system.assert(workPart != null);
        system.assert(newReq.Subject != null);
        system.assertEquals(newReq.Type, REQUEST_TYPE);
        SYSTEM.assertEquals(newReq.Equipment__c, equipmentId);
        SYSTEM.assertEquals(newReq.Vehicle__c, vehicleId);
        SYSTEM.assertEquals(newReq.Date_Reported__c, system.today());
    }
```

```
    @istest
    private static void testMaintenanceRequestNegative(){
        Vehicle__C vehicle = createVehicle();
        insert vehicle;
        id vehicleId = vehicle.Id;

        product2 equipment = createEq();
        insert equipment;
        id equipmentId = equipment.Id;

        case emptyReq = createMaintenanceRequest(vehicleId,equipmentId);
        insert emptyReq;

        Equipment_Maintenance_Item__c workP = createWorkPart(equipmentId, emptyReq.Id);
        insert workP;

        test.startTest();
        emptyReq.Status = WORKING;
        update emptyReq;
        test.stopTest();

        list<case> allRequest = [select id
                                 from case];

        Equipment_Maintenance_Item__c workPart = [select id
                                                  from Equipment_Maintenance_Item__c
                                                  where Maintenance_Request__c = :emptyReq.Id];

        system.assert(workPart != null);
        system.assert(allRequest.size() == 1);
    }


    @istest
    private static void testMaintenanceRequestBulk(){
        list<Vehicle__C> vehicleList = new list<Vehicle__C>();
        list<Product2> equipmentList = new list<Product2>();
        list<Equipment_Maintenance_Item__c> workPartList = new
list<Equipment_Maintenance_Item__c>();
        list<case> requestList = new list<case>();
```

```
        list<id> oldRequestIds = new list<id>();

        for(integer i = 0; i < 300; i++){
            vehicleList.add(createVehicle());
            equipmentList.add(createEq());
        }
        insert vehicleList;
        insert equipmentList;

        for(integer i = 0; i < 300; i++){
            requestList.add(createMaintenanceRequest(vehicleList.get(i).id, equipmentList.get(i).id));
        }
        insert requestList;

        for(integer i = 0; i < 300; i++){
            workPartList.add(createWorkPart(equipmentList.get(i).id, requestList.get(i).id));
        }
        insert workPartList;

        test.startTest();
        for(case req : requestList){
            req.Status = CLOSED;
            oldRequestIds.add(req.Id);
        }
        update requestList;
        test.stopTest();

        list<case> allRequests = [select id
                                  from case
                                  where status =: STATUS_NEW];

        list<Equipment_Maintenance_Item__c> workParts = [select id
                                                         from Equipment_Maintenance_Item__c
                                                         where Maintenance_Request__c in: oldRequestIds];

        system.assert(allRequests.size() == 300);
    }
}
```

Test callout logic (Step 6)

Warehouse Callout Service

```
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);
```

```
        }
      }
    }
}
```

Warehouse Callout Service Test

```
@isTest
private class WarehouseCalloutServiceTest {
  @isTest
  static void testWareHouseCallout(){
    Test.startTest();
    Test.setMock(HTTPCalloutMock.class, new WarehouseCalloutServiceMock());
    WarehouseCalloutService.runWarehouseEquipmentSync();
    Test.stopTest();
    System.assertEquals(1, [SELECT count() FROM Product2]);
  }
}
```

Warehouse Callout Service Mock

```
@isTest
global class WarehouseCalloutServiceMock implements HttpCalloutMock {
        global static HttpResponse respond(HttpRequest request){
        System.assertEquals('https://th-superbadge-apex.herokuapp.com/equipment',
        request.getEndpoint());
        System.assertEquals('GET', request.getMethod());
        HttpResponse response = new HttpResponse();
        response.setHeader('Content-Type', 'application/json');
        response.setBody('[{"_id":"55d66226726b611100aaf741","replacement":false,"quantity":5,"na
me":"Generator 1000 kW","maintenanceperiod":365,"lifespan":120,"cost":5000,"sku":"100003"}]');
        response.setStatusCode(200);
        return response;
        }
}
```

Test scheduling logic (Step 7)

Warehouse Sync Schedule

```
global class WarehouseSyncSchedule implements Schedulable {
  global void execute(SchedulableContext ctx) {

    WarehouseCalloutService.runWarehouseEquipmentSync();
  }
}
```

Warehouse Sync Schedule Test

```
@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();
      CronTrigger a=[SELECT Id FROM CronTrigger where NextFireTime > today];
      System.assertEquals(jobID, a.Id,'Schedule ')
   }
}
```

Warehouse Callout Service Mock

```
@isTest
global class WarehouseCalloutServiceMock implements HttpCalloutMock {
   global static HttpResponse respond(HttpRequest request){

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

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

response.setBody('[{"_id":"55d66226726b611100aaf741","replacement":false,"quantity":5,"name":"Gen
erator 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;
   }

}
```

Yours faithfully,
Aditya.
Trailhead URL: https://trailblazer.me/id/aditr13