Apex Triggers

Get Started with Apex Triggers

```
trigger AccountAddressTrigger on Account (before insert,before update)
{
   for (Account account:Trigger.New)
   {
      if((account.Match_Billing_Address__c==True)&&(account.BillingPostalCode !=
NULL))
      {
         account.ShippingPostalCode = account.BillingPostalCode;
      }
   }
}
```

Bulk Apex Triggers

```
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

apex class

```
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;
  }

}
```

Test Apex Triggers

apex class

```
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');
                }
        }
```

}
test class

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

Create Test Data for Apex Tests

apex class

```
public class RandomContactFactory {
   public static List<Contact> generateRandomContacts(Integer numcnt,String l){
      List<Contact> contacts=new List<Contact>();
      for(Integer i=0;i<numcnt;i++){
         Contact cnt=new Contact(FirstName='Test '+i,LastName=l);
         contacts.add(cnt);
      }
      return contacts;
   }
}
```

Asynchronous Apex

Use Future Methods

apex class

```
public class AccountProcessor {
   @future
   public static void countContacts(List<Id> accountIds){
      List<Account> accounts=[SELECT Id,(SELECT Id FROM Contacts) FROM Account
WHERE Id IN:accountIds];
         for(Account acc:accounts){
           acc.Number_Of_Contacts__c=acc.Contacts.size();
         }
      update accounts;
   }
}
```

test class

```
@isTest
private class AccountProcessorTest {

   @isTest
   private static void countContactsTest() {

      List<Account> accounts=new List<Account>();
      for(Integer i=0;i<300;i++){
         accounts.add(new Account(Name='TestContact'+i));
      }
      insert accounts;

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

```
        for(Account acc:accounts){
            contacts.add(new
Contact(FirstName=acc.Name,LastName='TestContact',AccountId=acc.Id));
            accountids.add(acc.Id);
        }
        insert contacts;

        Test.startTest();
        AccountProcessor.countContacts(accountids);
        Test.stopTest();
    }
}
```

Use Batch Apex

apex class

```
public class LeadProcessor implements Database.Batchable<sObject> {
    public Database.QueryLocator start(Database.BatchableContext dbc){
        return Database.getQueryLocator([SELECT Id,Name FROM Lead]);
    }
    public void execute(Database.BatchableContext dbc,List<Lead> leads){
        for(Lead l:leads){
            l.LeadSource='Dreamforce';
        }
        update leads;
    }
    public void finish(Database.BatchableContext dbc){
        System.debug('Done');
    }
}
```

test class

```
@isTest
private class LeadProcessorTest {
    @isTest
```

```
    private static void testBatchClass(){
        List<Lead> leads=new List<Lead>();
        for(Integer i=0;i<200;i++){
            leads.add(new Lead(LastName='Parichha',Company='Salesforce'));

        }
        insert leads;

        Test.startTest();
        LeadProcessor lp=new LeadProcessor();
        Id batchid=Database.executeBatch(lp,200);
        test.stopTest();

        List<Lead> updatedleads=[SELECT Id FROM Lead WHERE
Leadsource='Dreamforce'];
        System.assertEquals(200, updatedleads.size());
    }
}
```

Control Process With Queueable Apex

apex class

```
public without sharing class AddPrimaryContact implements Queueable {
    private Contact contact;
    private String state;

    public AddPrimaryContact(Contact inputcontact,String inputstate){
        this.contact=inputcontact;
        this.state=inputstate;
    }
    public void execute(QueueableContext context){
        List<Contact> contacts=new List<Contact>();

        List<Account> accounts=[SELECT Id FROM Account WHERE BillingState= :state
LIMIT 200];
```

```
        for (Account acc: accounts){
            Contact clonecontact=contact.clone();
            clonecontact.AccountId=acc.Id;
            contacts.add(clonecontact);
        }
        insert contacts;
    }

}
```

test class

```
@isTest
private class AddPrimaryContactTest {
    @isTest
    private static void testQueueableClass(){
        List<Account> accounts=new List<Account>();
        for(Integer i=0;i<500;i++){
            Account acc=new Account(Name='Test Account');
            if(i<250){
                acc.BillingState='NY';
            }
            else{
                acc.BillingState='CA';
            }
            accounts.add(acc);
        }
        insert accounts;

        Contact contact=new Contact(FirstName='Deependra',LastName='Parichha');
        insert contact;

        Test.startTest();
        Id jobId=System.enqueueJob(new AddPrimaryContact(contact,'CA'));
        Test.stopTest();
```

```
    List<Contact> contacts=[SELECT Id FROM Contact WHERE
Contact.Account.BillingState='CA' ];
    System.assertEquals(200,contacts.size());
  }
}
```

Schedule Jobs Using the Apex Scheduler

apex class

```
public without sharing class DailyLeadProcessor implements Schedulable {

  public void execute(SchedulableContext ctx){
    List<Lead> leads=[SELECT id,LeadSource FROM Lead WHERE LeadSource=null
LIMIT 200];
    for(Lead l:leads){
      l.LeadSource='Dreamforce';
    }
    update leads;
  }
}
```

test class

```
@isTest
private class DailyLeadProcessorTest {
  private static String CRON_EXP='0 0 0 ? * * *';

  @isTest
  private static void testSchedulabelClass(){
    List <Lead> leads=new List<Lead>();
    for(Integer i=0;i<500;i++){
      if(i<250){
        leads.add(new Lead(LastName='Parichha',Company='Salesforce'));
      }
      else{
        leads.add(new
```

```
Lead(LastName='Parichha',Company='Salesforce',LeadSource='Other'));
          }

        }
        insert leads;

        Test.startTest();
        String jobid=System.schedule('Process Leads',CRON_EXP,new
DailyLeadProcessor());
        Test.stopTest();

        List<Lead> updatedLeads=[SELECT ID,LeadSource FROM Lead WHERE
LeadSource='Dreamforce'];
        System.assertEquals(200,updatedLeads.size());

        List<CronTrigger> cts=[SELECT Id,TimesTriggered,NextFireTime FROM CronTrigger
WHERE Id=:jobid ];
        System.debug('Next Fire Time'+cts[0].NextFireTime);
    }
}
```

Apex Integration Services

Apex Rest Callouts

apex class

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

test class

```
@isTest
private class AnimalLocatorTest {
   @isTest static void AnimalLocatorMock1(){
   try{
      Test.setMock(HttpCalloutMock.class,new AnimalLocatorMock());
      String result=AnimalLocator.getAnimalNameById(3);
      String expectedRes='chicken';
      System.assertEquals(result,expectedRes);}
      catch(exception e){
         System.debug('The following exception has occuried: '+e.getMessage());
      }
```

```
    }
}
```

unit tests

```
@isTest
global class AnimalLocatorMock implements HttpCalloutMock {
    global HttpResponse respond(HttpRequest request){
        HttpResponse response=new HttpResponse();
        response.setHeader('Content-Type','application/json');
        response.setBody('{"animals":["majestic badger","fluffy bunny","scary
bear","chicken"]}');
        response.setStatusCode(200);
        return response;
    }
}
```

Apex Soap Callouts

apex class

```
public class ParkLocator {
    public static string[] country(String country){
        parkService.ParksImplPort park= new parkService.ParksImplPort();
        return park.byCountry(country);
    }
}
```

test class

```
@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> expectedres=new List<String>{'Yellowstone', 'Mackinac National Park',
```

```
'Yosemite'};
        System.assertEquals(result,expectedres);
    }
}
```

unit tests

```
@isTest
global class ParkServiceMock implements WebServiceMock {
   global void doInvoke(
           Object stub,
           Object request,
           Map<String, Object> response,
           String endpoint,
           String soapAction,
           String requestName,
           String responseNS,
           String responseName,
           String responseType) {
       // start - specify the response you want to send
       ParkService.byCountryResponse response_x=new
ParkService.byCountryResponse();
       response_x.return_x = new List<String>{'Yellowstone', 'Mackinac National Park',
'Yosemite'};
       // end
       response.put('response_x', response_x);
   }
}
```

Apex Web Services

apex class

```
@RestResource(urlMapping='/Account/*/contacts')
global with sharing class AccountManager {
    @HttpGet
    global static Account getAccount() {
        RestRequest request = RestContext.request;
        // grab the caseId from the end of the URL
```

```
        String accountId = request.requestURI.substringBetween('Accounts/','/contacts');
        Account result =  [SELECT Id,Name,(SELECT Id,Name FROM Contacts)
                    FROM Account
                    WHERE Id = :accountId];
        return result;

    }
}
```

test class

```
@IsTest
private class AccountManagerTest {
    @isTest static void testGetContactsByAccountId() {
        Id recordId = createTestRecord();
        // Set up a test request
        RestRequest request = new RestRequest();
        request.requestUri =

'https://yourInstance.my.salesforce.com/services/apexrest/Accounts/'+recordId+'/cont
acts';
        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 caseTest = new Account(
            Name='Test record');
        insert caseTest;
        Contact contactcase=new
Contact(FirstName='Deependra',LastName='Parichha',AccountId=casetest.Id);
        insert contactcase;
```

```
        return caseTest.Id;
    }
}
```

APEX SPECIALIST SUPERBADGE

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

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

WarehouseCalloutService

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

    //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(){
        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());

        //class maps the following fields: replacement part (always true), cost, current
inventory, lifespan, maintenance cycle, and warehouse SKU
        //warehouse SKU will be external ID for identifying which equipment records to
update within Salesforce
        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();
    }

}
```

WarehouseSyncSchedule

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

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<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 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<Vehicle__C> vehicleList = new list<Vehicle__C>();
        list<Product2> equipmentList = new list<Product2>();
        list<Equipment_Maintenance_Item__c> equipmentMaintenanceItemList = new
list<Equipment_Maintenance_Item__c>();
        list<case> caseList = new list<case>();
        list<id> oldCaseIds = new list<id>();

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

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

        for(integer i = 0; i < 300; i++){

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

```
        }
        insert equipmentMaintenanceItemList;

        test.startTest();
        for(case cs : caseList){
            cs.Status = 'Closed';
            oldCaseIds.add(cs.Id);
        }
        update caseList;
        test.stopTest();

        list<case> newCase = [select 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

```
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

```
trigger MaintenanceRequest on Case (before update, after update) {
```

```
    if(Trigger.isUpdate && Trigger.isAfter){
        MaintenanceRequestHelper.updateWorkOrders(Trigger.New, Trigger.OldMap);
    }
}
```

WarehouseCalloutService

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

    //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(){
        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());

            //class maps the following fields: replacement part (always true), cost, current inventory, lifespan, maintenance cycle, and warehouse SKU
            //warehouse SKU will be external ID for identifying which equipment records to update within Salesforce
            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();
  }

}
```

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<Product2> product2List = new List<Product2>();
```

```
        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);
    }
}
```

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":"55d66226
726b611100aaf742","replacement":true,"quantity":183,"name":"Cooling
Fan","maintenanceperiod":0,"lifespan":0,"cost":300,"sku":"100004"},{"_id":"55d66226726b6
11100aaf743","replacement":true,"quantity":143,"name":"Fuse
20A","maintenanceperiod":0,"lifespan":0,"cost":22,"sku":"100005"}]');
        response.setStatusCode(200);

        return response;
    }
}
```

WarehouseSyncScheduleTest

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