APEX TRIGGERS

* **Get Started with Apex Triggers**

**1.AccountAddressTrigger.apxt**

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**

**1.ClosedOpportunityTrigger.apxt**

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 TEST:**

**1.VerifyDate.apxc**

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;

               }

}

**2.TestVerifyDate.apxc**

@isTest

public class TestVerifyDate {

   @isTest static void test1(){

        Date d=VerifyDate.CheckDates(Date.parse('05/22/2022'),Date.parse('05/24/2022'));

       System.assertEquals(Date.parse('05/24/2022'), d);

     }

    @isTest static void test2(){

        Date d=VerifyDate.CheckDates(Date.parse('05/22/2022'),Date.parse('07/24/2022'));

       System.assertEquals(Date.parse('05/31/2022'), d);

     }

}

* **TEST APEX TRIGGERS:**

**1.RestrictContactByName.apxt**

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');

                              }

               }

}

* **CREATE TEST DATA FOR APEX TESTS:**

**1.RandomContactFactory.apxc**

public class RandomContactFactory {

    public static List<Contact> generateRandomContacts(Integer num,String lastName){

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

        for(Integer i=1;i<=num;i++)

        {

            Contact ct = new Contact(FirstName= 'Test '+i,LastName =lastName);

            contactList.add(ct);

        }

        return contactList;

    }

}

ASYNCHRONOUS APEX

* **USE FUTURE METHODS:**

**1.AccountProcessor.apxc**

public class AccountProcessor {

    @future

    public static void countContacts(List<Id> accountIds) {

        List<Account> accList=[Select Id, Number\_of\_Contacts\_\_c, (Select Id from Contacts) from Account where Id in :accountIds];

        For (Account acc : accList) {

            acc.Number\_of\_Contacts\_\_c = acc.Contacts.size();

        }

        update acclist;

    }

}

**2.AccountProcessorTest.apxc**

@isTest

public class AccountProcessorTest {

    public static testmethod void testAccountProcessor(){

        Account a = new Account();

        a.Name = 'Test Account';

        insert a;

        Contact con =new Contact();

        con.FirstName='Binary';

        con.LastName='Programming';

        con.AccountId=a.Id;

        insert con;

        List<Id> accListId = new List<Id>();

        accListId.add(a.Id);

        Test.startTest();

        AccountProcessor.countContacts(accListId);

        Test.stopTest();

        Account acc=[Select Number\_Of\_Contacts\_\_c from Account where Id =: a.Id];

        System.assertEquals(Integer.valueOf(acc.Number\_Of\_Contacts\_\_c),1);

    }

}

* **USE BATCH APEX:**

**1.LeadProcessor.apxc**

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);

    }

}

**2.LeadProcessorTest.apxc**

@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();

    }

}

* **CONTROL PROCESSES WITH QUEUEABLE APEX:**

**1.AddPrimaryContact.apxc**

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];

     }

}

**2.AddPrimaryContactTest.apxc**

@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');

        // startTest/stopTest block to force async processes to run

        Test.startTest();

        System.enqueueJob(addit);

        Test.stopTest();

        // Validate the job ran. Check if record have correct parentId now

        System.assertEquals(50,[select count() from Contact where accountId in (Select Id from Account where BillingState='CA')]);

    }

}

* **SCHEDULE JOBS USING APEX SCHEDULER:**

**1.DailyLeadProcessor.apxc**

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;

        }

    }

**2.DailyLeadProcessorTest.apxc**

@isTest

private class DailyLeadProcessorTest {

    private static String CORN\_EXP = '0 0 0 ? \* \* \*';

    @isTest

    public static void testSchedulableClass(){

        //Creating new 200 Leads and inserting them.

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

        for (Integer i = 0; i< 500; i++) {

            if(i<250)

            leads.add(new Lead(LastName='Connock',Company='Salesforce'));

            else

                leads.add(new Lead(LastName='Connock',Company='Salesforce',LeadSource='Other'));

        }

        insert leads;

        //Starting test. Putting in the schedule and running the DailyLeadProcessor execute method.

        Test.startTest();

        String jobId = System.schedule('Process Leads', CORN\_EXP, new DailyLeadProcessor());

        Test.stopTest();

        //Once the job has finished, retrieve all modified leads.

        List<Lead> updatedLeads = [SELECT Id, LeadSource FROM Lead where LeadSource = 'Dreamforce'];

          System.assertEquals(200,updatedLeads.size(),'ERROR: At least 1 record not updated correctly');

        //Checking if the modified leads are the same size number that we created in the start of this method.

            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:**

**1.AnimalLocator.apxc**

public class AnimalLocator{

    public static String getAnimalNameById(Integer animalId){

     String animalName;

        Http http = new Http();

       HttpRequest request = new HttpRequest();

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

request.setMethod('GET');

        HttpResponse response = http.send(request);

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

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

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

            animalName=string.valueOf(animal.get('name'));

}

    return animalName;

    }

}

**2.AnimalLocatorMock.apxc**

@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;

    }

}

**3.AnimalLocatorTest.apxc**

@isTest

private class AnimalLocatorTest {

    @isTest static void getAnimalNameByIdTest(){

        Test.setMock(HttpCalloutMock.class,new AnimalLocatorMock());

        String response=AnimalLocator.getAnimalNameById(1);

        System.assertEquals('chicken',response);

    }

}

* **APEX SOAP CALLOUTS:**

**1.ParkService.apxc**

//Generated by wsdl2apex

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;

        }

    }

}

**2.ParkServiceMock.apxc**

@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

        List<String> parks=new List<string>();

               parks.add('Yosemite');

               parks.add('Yellowstone');

               parks.add('Another Park');

           ParkService.byCountryResponse response\_x =

            new ParkService.byCountryResponse();

        response\_x.return\_x = parks;

        // end

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

   }

}

**3.ParkLocator.apxc**

public class ParkLocator {

    public static List<String> country(String country){

        ParkService.ParksImplPort parkservice=new parkService.ParksImplPort();

        return parkservice.byCountry(country);

    }

}

**4.ParkLocatorTest.apxc**

@isTest

private class ParkLocatorTest{

    @isTest static void testCallout() {

        // This causes a fake response to be generated

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

        // Call the method that invokes a callout

       String country='United States';

       List<String> result=ParkLocator.country(country);

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

         parks.add('Yosemite');

               parks.add('Yellowstone');

               parks.add('Another Park');

        // Verify that a fake result is returned

        System.assertEquals(parks, result);

    }

}

* **APEX WEB SERVICES:**

**1.AccountManager.apxc**

@RestResource(urlMapping = '/Account/\*/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 Limit 1];

        return result;

    }

}

**2.AccountManagerTest.apxc**

@IsTest

private class AccountManagerTest {

@isTest static void testGetContactsByAccountId()

{

    Id recordId =  createTestRecord();

    RestRequest request = new RestRequest();

    request.requestUri = 'https://yourInstance.my.salesforce.com/sercices/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

* **AUTOMATE RECORD CREATION:**

**1)MaintenanceRequest.apxt**

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

    if(Trigger.isUpdate && Trigger.isAfter){

        MaintenanceRequestHelper.updateWorkOrders(Trigger.New, Trigger.OldMap);

    }

}

**2)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);

                }

            }

        }

        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;

        }

    }

}

* **SYNCHRONIZATION SALESFORCE DATA WITH AN EXTERNAL SYSTEM:**

**1)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);

            }

        }

    }

}

* **SCHEDULE SYNCHRONIZATION USING APEX CODE:**

**1)WarehouseSyncSchedule.apxc**

global class WarehouseSyncSchedule implements Schedulable {

    global void execute(SchedulableContext ctx) {

        WarehouseCalloutService.runWarehouseEquipmentSync();

    }

}

* **TEST AUTOMATION LOGIC:**

**1)MaintenanceRequestHelperTest.apxc**

@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);

    }

}

**2)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);

                }

            }

        }

        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;

        }

    }

}

**3)MaintenanceRequest.apxt**

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

    if(Trigger.isUpdate && Trigger.isAfter){

        MaintenanceRequestHelper.updateWorkOrders(Trigger.New, Trigger.OldMap);

    }

}

* **TEST CALLOUT LOGIC:**

**1)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);

            }

        }

    }

}

**2)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]);

    }

}

**3)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.setStatusCode(200);

        return response;

    }

}

* **TEST SCHEDULING LOGIC:**

**1)WarehouseSyncSchedule.apxc**

global class WarehouseSyncSchedule implements Schedulable {

    global void execute(SchedulableContext ctx) {

        WarehouseCalloutService.runWarehouseEquipmentSync();

    }

}

**2)WarehouseSyncScheduleTest.apxc**

@isTest

public class WarehouseSyncScheduleTest {

    @isTest static void WarehousescheduleTest(){

        String scheduleTime = '00 00 01 \* \* ?';

        Test.startTest();

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

        String jobID=System.schedule('Warehouse Time To Schedule to Test', scheduleTime, new WarehouseSyncSchedule());

        Test.stopTest();

        //Contains schedule information for a scheduled job. CronTrigger is similar to a cron job on UNIX systems.

        // This object is available in API version 17.0 and later.

        CronTrigger a=[SELECT Id FROM CronTrigger where NextFireTime > today];

        System.assertEquals(jobID, a.Id,'Schedule ');

    }

}