Apex Triggers

Get Started with Apex Triggers

trigger AccountAddressTrigger on Account (before insert, before update) {

for(Account a : Trigger.new){

If (a.Match\_Billing\_Address\_\_c == true) {

a.ShippingPostalCode = a.BillingPostalCode;

}

}

}

# Bulk Apex Triggers

trigger ClosedOpportunityTrigger on Opportunity (after insert, after update) {

List<Task> taskList = new List<Task>();

for(Opportunity opp : Trigger.new) {

//Only create Follow Up Task only once when Opp StageName is to 'Closed Won' on Create

if(Trigger.isInsert) {

if(Opp.StageName == 'Closed Won') {

taskList.add(new Task(Subject = 'Follow Up Test Task', WhatId = opp.Id));

}

}

//Only create Follow Up Task only once when Opp StageName changed to 'Closed Won' on Update

if(Trigger.isUpdate) {

if(Opp.StageName == 'Closed Won'

&& Opp.StageName != Trigger.oldMap.get(opp.Id).StageName) {

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

# 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 test1(){

Date d=VerifyDate.CheckDates(Date.parse('01/01/2020'),Date.parse('01/03/2020'));

System.assertEquals(Date.parse('01/03/2020'),d);

}

@isTest static void test2(){

Date d=VerifyDate.CheckDates(Date.parse('01/01/2020'),Date.parse('03/03/2020'));

System.assertEquals(Date.parse('01/31/2020'),d);

}

}

# Test Apex Triggers

# 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');

# }

# }

# }

# @IsTest

# public class TestRestrictContactByName {

# @IsTest static void createBadContact(){

# 

# Contact c=new Contact(Firstname='John',LastName='INVALIDNAME');

# 

# Test.startTest();

# Database.SaveResult result = Database.insert(c, false);

# Test.stopTest();

# 

# System.assert(!result.isSuccess());

# }

# }

# Create Test Data for Apex Tests

# public class RandomContactFactory {

# 

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

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

# 

# for(Integer i=0;i<numofContacts;i++) {

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

# contacts.add(c);

# }

# System.debug(contacts);

# return contacts;

# }

# }

# Asynchronous Apex

# Use future Methods

# public class AccountProcessor {

# @future

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

# List<Account> accountsToUpdate = new List<Account>();

# List<Account> accounts = [Select Id, Name, (Select Id from Contacts) from Account Where Id IN :accountIds];

# // process account records to do awesome stuff

# For(Account acc:accounts){

# List<Contact> contactList = acc.Contacts;

# acc.Number\_Of\_Contacts\_\_c = contactList.size();

# accountsToUpdate.add(acc);

# }

# update accountsToUpdate;

# }

# }

# @IsTest

# private class AccountProcessorTest {

# @IsTest

# private static void testCountContacts(){

# Account newAccount = new Account (Name='Test Account');

# insert newAccount;

# Contact newContact1 = new Contact(FirstName='John',LastName='Doe',AccountId = newAccount.Id);

# insert newContact1;

# Contact newContact2 = new Contact(FirstName='Jane',LastName='Doe',AccountId = newAccount.Id);

# insert newContact2;

# List<Id> accountIds = new List<Id>();

# accountIds.add(newAccount.Id);

# Test.startTest();

# AccountProcessor.countContacts(accountIds);

# Test.stopTest();

# }

# }

# Use Batch Apex

# 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);

# }

# }

@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

# public class AddPrimaryContact implements Queueable{

# private Contact con;

# private String state;

# public AddPrimaryContact(Contact con, String state) {

# this.con = con;

# this.state = state;

# }

# public void execute(QueueableContext context){

# List<Account>

# accounts = [Select Id, Name, (Select FirstName, LastName, Id from contacts)

# from Account where BillingState = :state Limit 200];

# List<Contact> primaryContacts = new List<Contact>();

# for(Account acc:accounts) {

# Contact c = con.clone();

# C.AccountId = acc.Id;

# primaryContacts.add(c);

# }

# if(primaryContacts.size() > 0){

# insert primaryContacts;

# }

# }

# }

# @IsTest

# public class AddPrimaryContactTest {

# static testmethod void testQueueable(){

# List<Account> testAccounts = new List<Account>();

# for(Integer i=0;i<50; i++){

# testAccounts.add(new Account (Name='Account '+i,BillingState='CA'));

# }

# for(Integer j=0; j<50; j++){

# testAccounts.add(new Account (Name='Account '+j,Billingstate='NY'));

# }

# insert testAccounts;

# Contact testContact = new Contact(FirstName = 'John', LastName ='Doe');

# insert testContact;

# AddPrimaryContact addit = new addPrimaryContact(testContact, 'CA');

# Test.startTest();

# system.enqueueJob(addit);

# Test.stopTest();

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

# 

# }

# }

# Schedule Jobs Using the Apex Scheduler

# @isTest

# private class DailyLeadProcessorTest {

# static testMethod void testDailyLeadProcessor() {

# String CRON\_EXP = '0 0 1 \* \* ?';

# List<Lead> lList = new List<Lead>();

# for (Integer i = 0; i < 200; i++) {

# lList.add(new Lead(LastName='Dreamforce'+i, Company='Test1 Inc.', Status='Open - Not Contacted'));

# }

# insert lList;

# 

# Test.startTest();

# String jobId = System.schedule('DailyLeadProcessor', CRON\_EXP, new DailyLeadProcessor());

# }

# }

# global class DailyLeadProcessor implements Schedulable{

# global void execute(SchedulableContext ctx) {

# List<lead> leadstoupdate = new List<lead>();

# List<Lead> leads = [Select id From Lead Where LeadSource = NULL Limit 200];

# for(Lead l:leads){

# l.LeadSource = 'Dreamforce';

# leadstoupdate.add(l);

# }

# update leadstoupdate;

# }

# }

# Apex Integration Services

# Apex REST Callouts

# public class AnimalLocator

# {

# public static String getAnimalNameById(Integer id)

# {

# Http http = new Http();

# HttpRequest request = new HttpRequest();

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

# request.setMethod('GET');

# HttpResponse response = http.send(request);

# String strResp = '';

# system.debug('\*\*\*\*\*\*response '+response.getStatusCode());

# system.debug('\*\*\*\*\*\*response '+response.getBody());

# // If the request is successful, parse the JSON response.

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

# {

# // Deserializes the JSON string into collections of primitive data types.

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

# // Cast the values in the 'animals' key as a list

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

# System.debug('Received the following animals:' + animals );

# strResp = string.valueof(animals.get('name'));

# System.debug('strResp >>>>>>' + strResp );

# }

# return strResp ;

# }

# 

# }

# @isTest

# private class AnimalLocatorTest{

# @isTest static void AnimalLocatorMock1() {

# Test.SetMock(HttpCallOutMock.class, new AnimalLocatorMock());

# string result=AnimalLocator.getAnimalNameById(3);

# string expectedResult='chicken';

# System.assertEquals(result, expectedResult);

# }

# }

# @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;

# }

# }

# Apex SOAP Callouts

# //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;

# }

# }

# }

# @isTest

# global class ParkServiceMock implements WebServiceMock {

# global void doInvoke(

# Object stub,

# Object request,

# Map<String, Object> response,

# String endpoint,

# String soapAction,

# String requestName,

# String responseNS,

# String responseName,

# String responseType) {

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

# List<String> lstOfDummyParks = new List<String> {'Park1','Park2','Park3'};

# response\_x.return\_x = lstOfDummyParks;

# 

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

# }

# }

# public class ParkLocator {

# public static String[] country(String country){

# ParkService.ParksImplPort parks = new ParkService.ParksImplPort();

# String[] parksname = parks.byCountry(country);

# return parksname;

# }

# }

# @isTest

# private class ParkLocatorTest{

# @isTest

# static void testParkLocator() {

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

# String[] arrayOfParks = ParkLocator.country('India');

# 

# System.assertEquals('Park1', arrayOfParks[0]);

# }

# }

# Apex web Services

# @RestResource(urlMapping='/Accounts/\*/contacts')

# global with sharing class AccountManager{

# @HttpGet

# global static Account getAccount(){

# RestRequest req = RestContext.request;

# String accId = req.requestURI.substringBetween('Accounts/', '/contacts');

# Account acc = [SELECT Id, Name, (SELECT Id, Name FROM Contacts)

# FROM Account WHERE Id = :accId];

# 

# return acc;

# }

# }

# @IsTest

# private class AccountManagerTest{

# @isTest static void testAccountManager(){

# Id recordId = getTestAccountId();

# // Set up a test request

# RestRequest request = new RestRequest();

# request.requestUri =

# 'https://ap5.salesforce.com/services/apexrest/Accounts/'+ recordId +'/contacts';

# request.httpMethod = 'GET';

# RestContext.request = request;

# 

# // Call the method to test

# Account acc = AccountManager.getAccount();

# 

# // Verify results

# System.assert(acc != null);

# }

# 

# private static Id getTestAccountId(){

# Account acc = new Account(Name = 'TestAcc2');

# Insert acc;

# 

# Contact con = new Contact(LastName = 'TestCont2', AccountId = acc.Id);

# Insert con;

# 

# return acc.Id;

# }

# }

# Apex Specialist

# Automate 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);

# }

# }

# }

# 

# //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;

# }

# }

# }

# @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);

# }

# }

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

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

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

# }

# }

# Synchronize salesforce data with an external ystem

# public with sharing class WarehouseCalloutService implements Queueable {

# private static final String WAREHOUSE\_URL = 'https://th-superbadge-apex.herokuapp.com/equipment';

# 

# //Write a class that makes a REST callout to an external warehouse system to get a list of equipment that needs to be updated.

# //The callout’s JSON response returns the equipment records that you upsert in Salesforce.

# 

# @future(callout=true)

# public static void runWarehouseEquipmentSync(){

# System.debug('go into runWarehouseEquipmentSync');

# Http http = new Http();

# HttpRequest request = new HttpRequest();

# 

# request.setEndpoint(WAREHOUSE\_URL);

# request.setMethod('GET');

# HttpResponse response = http.send(request);

# 

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

# System.debug(response.getStatusCode());

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

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

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

# 

# //class maps the following fields:

# //warehouse SKU will be external ID for identifying which equipment records to update within Salesforce

# for (Object jR : jsonResponse){

# Map<String,Object> mapJson = (Map<String,Object>)jR;

# Product2 product2 = new Product2();

# //replacement part (always true),

# product2.Replacement\_Part\_\_c = (Boolean) mapJson.get('replacement');

# //cost

# product2.Cost\_\_c = (Integer) mapJson.get('cost');

# //current inventory

# product2.Current\_Inventory\_\_c = (Double) mapJson.get('quantity');

# //lifespan

# product2.Lifespan\_Months\_\_c = (Integer) mapJson.get('lifespan');

# //maintenance cycle

# product2.Maintenance\_Cycle\_\_c = (Integer) mapJson.get('maintenanceperiod');

# //warehouse SKU

# product2.Warehouse\_SKU\_\_c = (String) mapJson.get('sku');

# 

# product2.Name = (String) mapJson.get('name');

# product2.ProductCode = (String) mapJson.get('\_id');

# product2List.add(product2);

# }

# 

# if (product2List.size() > 0){

# upsert product2List;

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

# }

# }

# }

# 

# public static void execute (QueueableContext context){

# System.debug('start runWarehouseEquipmentSync');

# runWarehouseEquipmentSync();

# System.debug('end runWarehouseEquipmentSync');

# }

# 

# }

# Execute anonym windows :

# WarehouseCalloutService.runWarehouseEquipmentSync();

# @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":"55d66226726b611100aaf742","replacement":true,"quantity":183,"name":"Cooling Fan","maintenanceperiod":0,"lifespan":0,"cost":300,"sku":"100004"},{"\_id":"55d66226726b611100aaf743","replacement":true,"quantity":143,"name":"Fuse 20A","maintenanceperiod":0,"lifespan":0,"cost":22,"sku":"100005"}]');

# response.setStatusCode(200);

# 

# return response;

# }

# }

# @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);

# }

# }

# Schedule Synchronization:

# global with sharing class WarehouseSyncSchedule implements Schedulable {

# // implement scheduled code here

# global void execute (SchedulableContext ctx){

# System.enqueueJob(new WarehouseCalloutService());

# }

# }

# Test automation logic:

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

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

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

# }

# }

# 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;

# }

# }

# }

# @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);

# }

# }

# Test callout logic

# public with sharing class WarehouseCalloutService implements Queueable {

# private static final String WAREHOUSE\_URL = 'https://th-superbadge-apex.herokuapp.com/equipment';

# 

# //Write a class that makes a REST callout to an external warehouse system to get a list of equipment that needs to be updated.

# //The callout’s JSON response returns the equipment records that you upsert in Salesforce.

# 

# @future(callout=true)

# public static void runWarehouseEquipmentSync(){

# System.debug('go into runWarehouseEquipmentSync');

# Http http = new Http();

# HttpRequest request = new HttpRequest();

# 

# request.setEndpoint(WAREHOUSE\_URL);

# request.setMethod('GET');

# HttpResponse response = http.send(request);

# 

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

# System.debug(response.getStatusCode());

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

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

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

# 

# //class maps the following fields:

# //warehouse SKU will be external ID for identifying which equipment records to update within Salesforce

# for (Object jR : jsonResponse){

# Map<String,Object> mapJson = (Map<String,Object>)jR;

# Product2 product2 = new Product2();

# //replacement part (always true),

# product2.Replacement\_Part\_\_c = (Boolean) mapJson.get('replacement');

# //cost

# product2.Cost\_\_c = (Integer) mapJson.get('cost');

# //current inventory

# product2.Current\_Inventory\_\_c = (Double) mapJson.get('quantity');

# //lifespan

# product2.Lifespan\_Months\_\_c = (Integer) mapJson.get('lifespan');

# //maintenance cycle

# product2.Maintenance\_Cycle\_\_c = (Integer) mapJson.get('maintenanceperiod');

# //warehouse SKU

# product2.Warehouse\_SKU\_\_c = (String) mapJson.get('sku');

# 

# product2.Name = (String) mapJson.get('name');

# product2.ProductCode = (String) mapJson.get('\_id');

# product2List.add(product2);

# }

# 

# if (product2List.size() > 0){

# upsert product2List;

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

# }

# }

# }

# 

# public static void execute (QueueableContext context){

# System.debug('start runWarehouseEquipmentSync');

# runWarehouseEquipmentSync();

# System.debug('end runWarehouseEquipmentSync');

# }

# 

# }

# @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);

# }

# }

# @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":"55d66226726b611100aaf742","replacement":true,"quantity":183,"name":"Cooling Fan","maintenanceperiod":0,"lifespan":0,"cost":300,"sku":"100004"},{"\_id":"55d66226726b611100aaf743","replacement":true,"quantity":143,"name":"Fuse 20A","maintenanceperiod":0,"lifespan":0,"cost":22,"sku":"100005"}]');

# response.setStatusCode(200);

# 

# return response;

# }

# }

# Test Scheduling logic

# @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();

# }

# }