\*Salesforce Developer Catalyst Self-Learning & Super Badges\*

APEX TRIGGERS -->

1. Get Started With Apex Triggers

///AccountAddressTrigger////

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

for(Account account:Trigger.New){

if(account.Match\_Billing\_Address\_\_c == True){

account.ShippingPostalCode = account.BillingPostalCode;

}

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

2. Bulk Apex Triggers

///ClosedOpportunityTrigger///

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

List tasklist = new List();

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--->

1. Get Started with Apex Unit Tests

///VerifyDate///

public class VerifyDate {

//method to handle potential checks against two dates

public static Date CheckDates(Date date1, Date date2) {

//if date2 is within the next 30 days of date1, use date2. Otherwise use the end of the month

if(DateWithin30Days(date1,date2)) {

return date2;

}

else {

return SetEndOfMonthDate(date1);

}

}

//method to check if date2 is within the next 30 days of date1

@TestVisible private static Boolean DateWithin30Days(Date date1, Date date2) {

//check for date2 being in the past

if( date2 < date1) {

return false;

}

//check that date2 is within (>=) 30 days of date1

Date date30Days = date1.addDays(30);

//create a date 30 days away from date1

if( date2 >= date30Days ) {

return false;

}

else {

return true;

}

}

//method to return the end of the month of a given date

@TestVisible private static Date SetEndOfMonthDate(Date date1) {

Integer totalDays = Date.daysInMonth(date1.year(), date1.month());

Date lastDay = Date.newInstance(date1.year(), date1.month(), totalDays);

return lastDay;

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

///TestVerifyDate///

@isTest private class TestVerifyDate{

@istest static void Test\_CheckDates\_case1(){

Date D = VerifyDate.CheckDates(date.parse('01/01/2020'), date.parse('01/05/2020'));

System.assertEquals(date.parse('01/05/2020'), D);

}

@isTest static void Test\_CheckDates\_case2(){

Date D = VerifyDate.CheckDates (date.parse('01/01/2020'), date.parse('05/05/2020')); System.assertEquals(date.parse('01/31/2020'), D);

}

@isTest static void Test\_DateWithin30Days\_case1() {

Boolean flag = VerifyDate.DateWithin30Days (date.parse('01/01/2020'), date.parse( '12/30/2019'));

System.assertEquals(false, flag);

}

@istest static void Test\_DateWithin30Days\_case2(){

Boolean flag = VerifyDate.DateWithin30Days (date.parse('01/01/2020'), date.parse('02/02/2020'));

System.assertEquals(false, flag);

}

@isTest static void Test\_DateWithin30Days\_case3() {

Boolean flag = VerifyDate.DateWithin30Days (date.parse('01/01/2020'), date.parse('01/15/2020'));

System.assertEquals(true, flag);

}

@isTest static void Test\_SetEndOfMonthDate(){ Date returndate = VerifyDate.SetEndOfMonthDate(date.parse('01/01/2020'));

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

2. Test Apex Triggers

///RestrictContactByName///

trigger RestrictContactByName on Contact (before insert, before update) {

//check contacts prior to insert or update for invalid data

For (Contact c : Trigger.New) {

if(c.LastName == 'INVALIDNAME') {

//invalidname is invalid

c.AddError('The Last Name "'+c.LastName+'" is not allowed for DML');

}

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ///TestRestrictContactByName///

@istest public class TestRestrictcontactByName {

@isTest 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());

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

3. Create Test Data for Apex Tests

///RandomContactFactory///

public class RandomContactFactory {

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

List contactList = new List();

for(Integer i = 1; i<=num; i++){

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

contactList.add(ct);

}

return contactList;

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Asynchronous Apex--->

1. Use Future Methods

///AccountProcessor///

public class AccountProcessor{

@future

public static void countContacts(List accountIds){

List accountsToUpdate = new List();

List accounts = [Select Id, Name, (Select Id from Contacts) from Account Where

Id in :accountIds];

For(Account acc:accounts){

List contactList = acc.Contacts;

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

accountsToUpdate.add(acc);

}

update accountsToUpdate;

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ///AccountProcessorTest///

@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 accountIds = new List(); accountIds.add(newAccount.Id);

Test.startTest();

AccountProcessor.countContacts(accountIds);

Test.stopTest();

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

2. Use Batch Apex

///LeadProcessor///

global class LeadProcessor implements Database.Batchable {

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 L\_list){

List L\_list\_new = new List();

for(lead L:L\_list){

L.leadsource = 'Dreamforce';

L\_list\_new.add(L);

count += 1;

}

update L\_list\_new;

}

global void finish(Database.BatchableContext bc){

system.debug('count = ' + count);

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ///LeadProcessorTest///

@isTest public class LeadProcessorTest {

@isTest public static void testit(){

List L\_list = new List();

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();

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

3. Control Processes with Queueable Apex

///AddPrimaryContact///

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

from Account where BillingState = :state Limit 200];

List primaryContacts = new List();

for (Account acc:accounts){

Contact c = con.clone();

c.AccountId = acc.Id;

primaryContacts.add(c);

}

if(primaryContacts.size() > 0){

insert primaryContacts;

}

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ///AddPrimaryContactTest///

@isTest public class AddPrimaryContactTest{

static testmethod void testQueueable(){

List testAccounts=new List();

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

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

for(Integer j=0;j<5;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')]);

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

4. Schedule Jobs Using the Apex Scheduler

///DailyLeadProcessor///

global class DailyLeadProcessor implements Schedulable{

global void execute(SchedulableContext ctx){

List leads = [SELECT Id, LeadSource FROM Lead WHERE LeadSource = ''];

if(leads.size() > 0){

List newLeads = new List();

for(Lead lead : leads){

lead.LeadSource = 'DreamForce';

newLeads.add(lead);

}

update newLeads;

}

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ///DailyLeadProcessorTest///

@isTest private class DailyLeadProcessorTest{

//Seconds Minutes Hours Day\_of\_month Month Day\_of\_week optional\_year

public static String CRON\_EXP = '0 0 0 2 6 ? 2022';

static testmethod void testScheduledJob(){

List leads = new List();

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

Lead lead = new Lead(LastName = 'Test ' + i, LeadSource = '', Company =

'Test Company ' + i, Status = 'Open - Not Contacted');

leads.add(lead);

}

insert leads;

Test.startTest();

// Schedule the test job

String jobId = System.schedule('Update LeadSource to DreamForce',

CRON\_EXP, new DailyLeadProcessor());

// Stopping the test will run the job synchronously

Test.stopTest();

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

4. Schedule Jobs Using the Apex Scheduler

///DailyLeadProcessor///

global class DailyLeadProcessor implements Schedulable{

global void execute(SchedulableContext ctx){

List leads = [SELECT Id, LeadSource FROM Lead WHERE LeadSource = ''];

if(leads.size() > 0){

List newLeads = new List();

for(Lead lead : leads){

lead.LeadSource = 'DreamForce';

newLeads.add(lead);

}

update newLeads;

}

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ///DailyLeadProcessorTest///

@isTest private class DailyLeadProcessorTest{

//Seconds Minutes Hours Day\_of\_month Month Day\_of\_week optional\_year

public static String CRON\_EXP = '0 0 0 2 6 ? 2022';

static testmethod void testScheduledJob(){

List leads = new List();

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

Lead lead = new Lead(LastName = 'Test ' + i, LeadSource = '', Company =

'Test Company ' + i, Status = 'Open - Not Contacted'); leads.add(lead);

}

insert leads;

Test.startTest();

// Schedule the test job

String jobId = System.schedule('Update LeadSource to DreamForce', CRON\_EXP,

new DailyLeadProcessor());

// Stopping the test will run the job synchronously

Test.stopTest();

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

3. Apex SOAP Callouts

///ParkLocator///

public class ParkLocator {

public static string[] country(string theCountry){

ParkService.ParksImplPort parkSvc = new ParkService.ParksImplPort();

return parkSvc.byCountry(theCountry);

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

///ParkLocatorTest///

@isTest private class ParkLocatorTest {

@isTest static void testCallout() {

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

String country = 'United States';

List result = ParkLocator.country(country);

List parks = new List{'Yellowstone', 'Mackinac National Park', 'Yosemite'};

System.assertEquals(parks, result);

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

///ParkServiceMock///

@isTest global class ParkServiceMock implements WebServiceMock {

global void doInvoke(Object stub,Object request,Map 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{'Yellowstone', 'Mackinac National Park',

'Yosemite'};

// end

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

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

4. Apex Web Services

///AccountManager///

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

global with sharing class AccountManager {

@HttpGet

global static Account getAccount(){

RestRequest request = RestContext.request;

string accountId = request.requestURI.substringBetween('Accounts/','/contacts');

Account result = [SELECT Id, Name, (Select Id, Name from Contacts) from

Account where Id=:accountId Limit 1];

return result;

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ///AccountManagerTest///

@IsTest private class AccountManagerTest {

@isTest static void testGetContactsByAccountId(){

Id recordId = createTestRecord();

RestRequest request = new RestRequest();

request.requestUri =

'https://yourInstance.my.salesforce.com/services/apexrest/Accounts/' + recordId+'/contacts';

request.httpMethod = 'GET';

RestContext.request = request;

Account thisAccount = AccountManager.getAccount();

System.assert(thisAccount != null);

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

}

static Id createTestRecord(){

Account accountTest = new Account( Name ='Test record');

insert accountTest;

Contact contactTest = new Contact( FirstName='John', LastName = 'Doe',

AccountId = accountTest.Id);

insert contactTest;

return accountTest.Id;

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Apex Specialist SuperBadges--->

Challenge 1-Automated Record Creation

///MaitenanceRequest///

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

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

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

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

///MaintenanceRequestHelper///

public with sharing class MaintenanceRequestHelper {

public static void updateworkOrders(List updWorkOrders, Map nonUpdCaseMap) {

Set validIds = new Set();

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 newCases = new List();

Map closedCasesM = new Map([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 maintenanceCycles = new Map();

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 clonedWPs = new List();

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;

}

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Challenge 2-Synchronize Salesforce data with an external system

///WarehouseCalloutService///

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 warehouseEq = new List();

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

List jsonResponse = (List)JSON.deserializeUntyped(response.getBody());

System.debug(response.getBody());

for (Object eq : jsonResponse){

Map mapJson = (Map)eq;

Product2 myEq = new Product2(); myEq.Replacement\_Part\_\_c = (Boolean) mapJson.get('replacement'); myEq.Name = (String) mapJson.get('name'); myEq.Maintenance\_Cycle\_\_c = (Integer) mapJson.get('maintenanceperiod');

myEq.Lifespan\_Months\_\_c = (Integer) mapJson.get('lifespan'); myEq.Cost\_\_c = (Decimal) mapJson.get('lifespan'); myEq.Warehouse\_SKU\_\_c = (String) mapJson.get('sku'); myEq.Current\_Inventory\_\_c = (Double) mapJson.get('quantity'); warehouseEq.add(myEq);

}

if (warehouseEq.size() > 0){

upsert warehouseEq;

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

System.debug(warehouseEq);

}

}

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Challenge 3-Schedule synchronization using Apex code

///WarehouseSyncShedule///

global class WarehouseSyncSchedule implements Schedulable {

global void execute(SchedulableContext ctx) {

WarehouseCalloutService.runWarehouseEquipmentSync();

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Challenge 4-Test automation logic

///MaintenanceRequestHelperTest///

@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 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 vehicleList = new list();

list equipmentList = new list();

list workPartList = new list();

list requestList = new list();

list oldRequestIds = new list();

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 allRequests = [select id from case where status =: STATUS\_NEW];

list workParts = [select id from Equipment\_Maintenance\_Item\_\_c

where Maintenance\_Request\_\_c in: oldRequestIds];

system.assert(allRequests.size() == 300); } } \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

///MaintenanceRequestHelper///

public with sharing class MaintenanceRequestHelper {

public static void updateworkOrders(List updWorkOrders, Map nonUpdCaseMap) {

Set validIds = new Set();

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 newCases = new List();

Map closedCasesM = new Map([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 maintenanceCycles = new Map();

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 clonedWPs = new List();

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;

}

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

///MaintenanceRequest///

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

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

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

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Challenge 5-Test callout logic

///WarehouseCalloutService///

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 warehouseEq = new List();

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

List jsonResponse = (List)JSON.deserializeUntyped(response.getBody());

System.debug(response.getBody());

for (Object eq : jsonResponse){

Map mapJson = (Map)eq;

Product2 myEq = new Product2();

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

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

myEq.Maintenance\_Cycle\_\_c = (Integer)

mapJson.get('maintenanceperiod');

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

myEq.Cost\_\_c = (Decimal) mapJson.get('lifespan');

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

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

warehouseEq.add(myEq);

}

if (warehouseEq.size() > 0){

upsert warehouseEq;

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

System.debug(warehouseEq);

}

}

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ///WarehouseCalloutServiceTest///

@isTest private class WarehouseCalloutServiceTest {

@isTest static void testWareHouseCallout(){

Test.startTest();

// implement mock callout test here

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

WarehouseCalloutService.runWarehouseEquipmentSync(); Test.stopTest();

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

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

///WarehouseCalloutServiceMock///

@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;

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Challenge 6-Test scheduling logic

///WarehouseSyncSchedule///

global class WarehouseSyncSchedule implements Schedulable {

global void execute(SchedulableContext ctx) {

WarehouseCalloutService.runWarehouseEquipmentSync();

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

///WarehouseSyncScheduleTest///

@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 ');

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*