**Apex Basics and Database**

Get Started with Apex

public class StringArrayTest {

public static List<String> generateStringArray(Integer n){

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

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

myArray.add('Test '+i);

}

return myArray;

}

}

Manipulate records with DML

public class AccountHandler

{

public static Account insertNewAccount(String Name)

{

Account a = new Account();

a.Name = name;

try

{

insert a;

}

catch(Exception e)

{

return null;

}

{

return a ;

}

}

}

Write SOQL Queries

public class ContactSearch

{

public static List<Contact> searchForContacts(String lastName, String mailingPostalCode)

{

List<Contact> retList = [SELECT Id, Name, LastName, MailingPostalCode

FROM Contact

WHERE LastName = :lastName and MailingPostalCode = :mailingPostalCode];

system.debug('Found'+ retList);

return retList;

}

}

Write SOSL Queries

public class ContactAndLeadSearch

{

public static List<List<sObject>> searchContactsAndLeads(string challenge\_word)

{

List<List<sObject>> searchList = [FIND :challenge\_word IN ALL FIELDS

RETURNING Contact(Name), Lead(Name)];

return searchList;

}

}

**Build Apex Coding Skills**

Object oriented Programming for Admins

Use sObjects and DML

public class AccountHandler

{

public static void insertAccount(Integer Num)

{

List<Account> addAccounts = new List<Account>();

Integer i=1;

While(i <= Num)

{

Account objAcc = new Account();

objAcc.name = 'Acme Inc n'+i;

objAcc.AccountNumber = 'A000n'+i;

addAccounts.add(objAcc);

i++;

}

insert addAccounts;

}

}

SOQL for Admins

Create SOQL queries in Apex classes

public class AccountUtility {

public static void viewAnnualRevenue(){

List<Account> accountsList= [SELECT Name, AnnualRevenue FROM Account];

for(Account acc : accountsList){

String acctRev = 'account name: ' + acc.Name + ', Annual Revenue: ' + acc.AnnualRevenue;

system.debug(acctRev);

}

}

}

Create relationship queries with custom objects

public class PropertyUtility {

public static void newListedProperties(){

List<Property\_\_c> newPropList = new List<Property\_\_c>();

newPropList = [SELECT Id, Name, Broker\_\_r.Email\_\_c, Days\_On\_Market\_\_c FROM Property\_\_c where Days\_On\_Market\_\_c <= 30];

for(Property\_\_c p: newPropList){

String propEmail = p.Name + ':' + p.Broker\_\_r.Email\_\_c;

System.debug(propEmail);

}

}

}

Use Apex to Automate business process

Create the automation to run the free bouquet promotion

//Create the class

public class OrderItemUtility {

//Create the method that will add free bonus bouquet when order is activated

public static void addBonusBouquet(List<Order> ordersFromTrigger) {

//TO DO 3.1: Determine if we have a bonus product and get its ID to add to the order

//TO DO 2.1: Create a list to store any new bouquets we'll insert later

List<OrderItem> newBouquets = new List<OrderItem>();

//TO DO 2.2: Loop over orders in ordersFromTrigger, for each order (called currentOrder) do something

for(Order currentOrder : ordersFromTrigger) {

//TO DO 2.3: Verify the order status is 'Activated'

if(currentOrder.Status == 'Activated') {

//TO DO 2.4: Create a new bouquet and set values

OrderItem freeBouquet = new OrderItem(

OrderId = currentOrder.id, //this is the order we're linking the bouquet to

//PricebookEntryId = entry.id,

numberOfFlowers\_\_c = 3,

description = 'FREE Bouquet',

Quantity = 1,

colorTheme\_\_c = 'Spectacular Sunset',

percentOfOpening\_\_c = 0,

UnitPrice = 0.00

);

//TO DO 2.5: Add the freeBouquet sObject to your list

newBouquets.add(freeBouquet);

//TO DO 2.6: Close the "if" and "for loop" sections

} //end if

} //end for loop

//TO DO 3.2: Use DML to add the new bouquet to the Order

//TO DO 3.3: Close the if section

} //end method

} //end class

Use salesforce data in Apex code

//Create the class

public class OrderItemUtility {

//Create the method that will add free bonus bouquet when order is activated

public static void addBonusBouquet(List<Order> ordersFromTrigger) {

//TO DO 3.1: Determine if we have a bonus product and get its ID to add to the order

// Use SOQL to get the ID of the bonus bouquet and store it in an sObject variable called bonusProduct

List<Product2> bonusProductList = [SELECT Id, ProductCode FROM Product2 WHERE ProductCode = 'BOT-BB-12'];

Product2 bonusProduct = new Product2();

if(bonusProductList.size() > 0) {

bonusProduct = bonusProductList[0];

// Use SOQL to get the price book entry ID associated with the bonusProduct and store it in an sObject variable called entry

// Every Product has an assosiated PricebookEntry

List<PricebookEntry> entryList = [SELECT Id, Product2Id FROM PricebookEntry WHERE Product2Id = :bonusProduct.Id];

PricebookEntry entry = new PricebookEntry();

if(entryList.size() > 0) {

entry = entryList[0];

}

//TO DO 2.1: Create a list to store any new bouquets we'll insert later

List<OrderItem> newBouquets = new List<OrderItem>();

//TO DO 2.2: Loop over orders in ordersFromTrigger, for each order (called currentOrder) do something

for(Order currentOrder : ordersFromTrigger) {

//TO DO 2.3: Verify the order status is 'Activated'

if(currentOrder.Status == 'Activated') {

//TO DO 2.4: Create a new bouquet and set values

OrderItem freeBouquet = new OrderItem(

OrderId = currentOrder.id, //this is the order we're linking the bouquet to

PricebookEntryId = entry.id,

numberOfFlowers\_\_c = 3,

description = 'FREE Bouquet',

Quantity = 1,

colorTheme\_\_c = 'Spectacular Sunset',

percentOfOpening\_\_c = 0,

UnitPrice = 0.00

);

//TO DO 2.5: Add the freeBouquet sObject to your list

newBouquets.add(freeBouquet);

//TO DO 2.6: Close the "if" and "for loop" sections

} //end if

} //end for loop

//TO DO 3.2: Use DML to add the new bouquet to the Order

insert newBouquets;

//TO DO 3.3: Close the if section

} //end if

} //end method

} //end class

Create and test a trigger

trigger orderTrigger on Order(before update) {

OrderItemUtility.addBonusBouquet(Trigger.new);

}

**Apex Triggers**

Get started with apex triggers

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

for(Account account:Trigger.New){

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

account.ShippingPostalCode = account.BillingPostalCode;

}

}

}

Bulk Apex Triggers

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

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

for(Opportunity opp: Trigger.New){

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

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

}

}

If(tasklist.size()>0){

insert tasklist;

}

}

**Apex Testing**

Create a unit test for a simple apex class

public class VerifyDate {

//method to handle potential checks against two dates

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

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

if(DateWithin30Days(date1,date2)) {

return date2;

} else {

return SetEndOfMonthDate(date1);

}

}

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

@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;

}

}

@isTest

public 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('01/05/2020'));

System.assertEquals(true, flag);

}

@isTest static void Test\_SetEndOfMonthDate(){

Date returndate = VerifyDate.SetEndOfMonthDate(date.parse('01/01/2020'));

}

}

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 Test\_insertupdateContact(){

Contact cnt = new Contact();

cnt.LastName = 'INVALIDNAME';

Test.startTest();

Database.SaveResult result = Database.insert(cnt, false);

Test.stopTest();

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

System.assert(result.getErrors().size() > 0);

System.assertEquals('The Last Name "INVALIDNAME" is not allowed for DML', result.getErrors()[0].getMessage());

}

}

Create test data for apex tests

public class RandomContactFactory {

public static List<Contact> generateRandomContacts(Integer numcnt, string lastname){

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

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

Contact cnt = new Contact(FirstName = 'Test'+i,LastName = lastname);

contacts.add(cnt);

}

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];

for(Account acc:accounts){

List<Contact> contactList = acc.Contacts;

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

accountsToUpdate.add(acc);

}

update accountsToUpdate;

}

}

@isTest

public 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 Apex scheduler

global class DailyLeadProcessor implements Schedulable{

global void execute(SchedulableContext ctx){

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

if(leads.size() > 0){

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

for(Lead lead : leads){

lead.LeadSource = 'DreamForce';

newLeads.add(lead);

}

update newLeads;

}

}

}

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

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

Lead lead = new Lead(LastName = 'Test ' + i, LeadSource = '', Company = 'Test Company ' + i, Status = 'Open - Not Contacted');

leads.add(lead);

}

insert leads;

Test.startTest();

// 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();

}

}

**Apex Integration services**

Apex REST callouts

public class AnimalLocator{

public static String getAnimalNameById(Integer x){

Http http = new Http();

HttpRequest req = new HttpRequest();

req.setEndpoint('https://th-apex-http-callout.herokuapp.com/animals/'

+ x);

req.setMethod('GET');

Map<String, Object> animal= new Map<String, Object>();

HttpResponse res = http.send(req);

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

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

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

}

return (String)animal.get('name');

}

}

@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 );

}

}

Apex SOAP Callouts

public class ParkLocator {

public static string[] country(string theCountry) {

ParkService.ParksImplPort parkSvc = new ParkService.ParksImplPort(); // remove space

return parkSvc.byCountry(theCountry);

}

}

@isTest

private class ParkLocatorTest {

@isTest static void testCallout() {

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

String country = 'United States';

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

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

System.assertEquals(parks, result);

}

}

Apex Web services

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

global 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 {

private static testMethod void getAccountTest1() {

Id recordId = createTestRecord();

// Set up a test request

RestRequest request = new RestRequest();

request.requestUri = 'https://na1.salesforce.com/services/apexrest/Accounts/'+

recordId +'/contacts' ;

request.httpMethod = 'GET';

RestContext.request = request;

// Call the method to test

Account thisAccount = AccountManager.getAccount();

// Verify results

System.assert(thisAccount != null);

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

}

// Helper method

static Id createTestRecord() {

// Create test record

Account TestAcc = new Account(

Name='Test record');

insert TestAcc;

Contact TestCon= new Contact(

LastName='Test',

AccountId = TestAcc.id);

return TestAcc.Id

;

}

}

**Apex specialist super badge**

Challenge 1

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));  
                } 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> 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;  
        }  
    }  
}

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

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

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

    }

}

Challenge 2

public with sharing class WarehouseCalloutService implements Queueable {  
    private static final String WAREHOUSE\_URL = 'https://th-superbadge-apex.herokuapp.com/equipment';  
      
    //class that makes a REST callout to an external warehouse system to get a list of equipment that needs to be updated.  
    //The callout’s JSON response returns the equipment records that you upsert in Salesforce.   
      
    @future(callout=true)  
    public static void runWarehouseEquipmentSync(){  
        Http http = new Http();  
        HttpRequest request = new HttpRequest();  
          
        request.setEndpoint(WAREHOUSE\_URL);  
        request.setMethod('GET');  
        HttpResponse response = http.send(request);  
          
        List<Product2> warehouseEq = new List<Product2>();  
          
        if (response.getStatusCode() == 200){  
            List<Object> jsonResponse = (List<Object>)JSON.deserializeUntyped(response.getBody());  
            System.debug(response.getBody());  
              
            //class maps the following fields: replacement part (always true), cost, current inventory, lifespan, maintenance cycle, and warehouse SKU  
            //warehouse SKU will be external ID for identifying which equipment records to update within Salesforce  
            for (Object eq : jsonResponse){  
                Map<String,Object> mapJson = (Map<String,Object>)eq;  
                Product2 myEq = new Product2();  
                myEq.Replacement\_Part\_\_c = (Boolean) mapJson.get('replacement');  
                myEq.Name = (String) mapJson.get('name');  
                myEq.Maintenance\_Cycle\_\_c = (Integer) mapJson.get('maintenanceperiod');  
                myEq.Lifespan\_Months\_\_c = (Integer) mapJson.get('lifespan');  
                myEq.Cost\_\_c = (Integer) mapJson.get('cost');  
                myEq.Warehouse\_SKU\_\_c = (String) mapJson.get('sku');  
                myEq.Current\_Inventory\_\_c = (Double) mapJson.get('quantity');  
                myEq.ProductCode = (String) mapJson.get('\_id');  
                warehouseEq.add(myEq);  
            }  
              
            if (warehouseEq.size() > 0){  
                upsert warehouseEq;  
                System.debug('Your equipment was synced with the warehouse one');  
            }  
        }  
    }  
      
    public static void execute (QueueableContext context){  
        runWarehouseEquipmentSync();  
    }  
      
}

Challenge 3

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

Challenge 4

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;

        }

    }

}

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);

                }

            }

        }

        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;

        }

    }

}

Challenge 5

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);  
            }  
              
        }  
    }  
}

@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]);  
}   
}

@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

global class WarehouseSyncSchedule implements Schedulable { global void execute(SchedulableContext ctx) { WarehouseCalloutService.runWarehouseEquipmentSync(); } }

@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 ');  
  
  
}  
}

