

Modules

Apex Trigger 1st sub-module, created a trigger

```
trigger AccountAddressTrigger on Account (before insert,before update) {    List<Account> acclist =
new List<Account>();  for(Account a : trigger.new){    if((a.Match_Billing_Address__c
==true)&&(account.BillingPostalCode != NULL))      a.ShippingPostalCode = a.BillingPostalCode;    }
}
```

Apex Trigger 2nd sub-module, created a trigger

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

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

```
    for(Opportunity opp : Trigger.new) {
```

```
        if(Trigger.isInsert) {
```

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

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

```
            }
```

```
        }
```

```
        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 1st sub-module, created a class

@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);

}

}

Apex Testing 2nd sub-module, created a class

@isTest

public class TestRestrictContactByName {

@isTest

public static void testContact(){

Contact ct = new Contact();

ct.LastName = 'INVALIDNAME';

Database.SaveResult res = Database.insert(ct,false);

System.assertEquals('The Last Name "INVALIDNAME" is not allowed for

```

DML',res.getErrors()[0].getMessage());
    }
}

```

Apex Testing 3rd sub-module, created a class

```

public class RandomContactFactory {

    public static List<Contact> generateRandomContacts(Integer num, String lastName){
        List<Contact> contactList = new List<Contact>();
        for(Integer i = 1; i<=num; i++){
            Contact ct = new Contact(FirstName = 'Test '+i, LastName =lastName);
            contactList.add(ct);
        }
        return contactList;
    }
}

```

Asynchronous Apex 1st sub-module

Accountprocessor(created a class) :-

```

public class AccountProcessor {

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

        List<Account> accList = [Select Id, Number_Of_Contacts__c, (Select Id from Contacts) from Account
        where Id in :accountIds];

        For(Account acc : accList){

            acc.Number_Of_Contacts__c = acc.Contacts.size();
        }
    }
}

```

```

    }
    update accList;
}
}

```

Accountprocessor test(created a class) :-

@isTest

```
public class AccountProcessorTest {
```

```
    public static testmethod void testAccountProcessor(){
```

```
        Account a = new Account();
```

```
        a.Name = 'Test Account';
```

```
        insert a;
```

```
        Contact con = new Contact();
```

```
        con.FirstName = 'Lakhan';
```

```
        con.LastName = 'Mukhija';
```

```
        con.AccountId = a.Id;
```

```
        insert con;
```

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

```
        accListId.add(a.Id);
```

```
        Test.startTest();
```

```
        AccountProcessor.countContacts(accListId);
```

```
        Test.stopTest();
```

```
        Account acc = [Select Number_of_Contacts__c from Account where Id =: a.Id];
```

```

        System.assertEquals(Integer.valueOf(acc.Number_of_Contacts__c),1);
    }
}

```

Asynchronous Apex 2nd sub-module

LeadProcessor(created a class) :-

```

public class LeadProcessor implements Database.Batchable<sObject> {

    public Database.QueryLocator start(Database.BatchableContext bc) {
        // collect the batches of records or objects to be passed to execute
        return Database.getQueryLocator([Select LeadSource From Lead ]);
    }

    public void execute(Database.BatchableContext bc, List<Lead> leads){
        // process each batch of records
        for (Lead Lead : leads) {
            lead.LeadSource = 'Dreamforce';
        }
        update leads;
    }

    public void finish(Database.BatchableContext bc){
    }

}

```

LeadProcessorTest(created a class) :-

```

@isTest

public class LeadProcessorTest {

    @testSetup
    static void setup() {

```

```

List<Lead> leads = new List<Lead>();
for(Integer counter=0 ;counter <200;counter++){
    Lead lead = new Lead();
    lead.FirstName ='FirstName';
    lead.LastName ='LastName'+counter;
    lead.Company ='demo'+counter;
    leads.add(lead);
}
insert leads;
}

```

```

@isTest static void test() {
    Test.startTest();
    LeadProcessor leadProcessor = new LeadProcessor();
    Id batchId = Database.executeBatch(leadProcessor);
    Test.stopTest();
}
}

```

Asynchronous Apex 3rd sub-module

AddPrimaryContact (created a class):-

```

public class AddPrimaryContact implements Queueable
{
    private Contact c;
    private String state;
    public AddPrimaryContact(Contact c, String state)
    {
        this.c = c;
        this.state = state;
    }
}

```

```

    }

    public void execute(QueueableContext context)
    {
        List<Account> ListAccount = [SELECT ID, Name ,(Select id,FirstName,LastName from contacts )
FROM ACCOUNT WHERE BillingState = :state LIMIT 200];

        List<Contact> lstContact = new List<Contact>();
        for (Account acc:ListAccount)
        {
            Contact cont = c.clone(false,false,false,false);
            cont.AccountId = acc.id;
            lstContact.add( cont );
        }

        if(lstContact.size() >0 )
        {
            insert lstContact;
        }

    }

}

```

AddPrimaryContactTest(created a class) :-

```

@Test
public class AddPrimaryContactTest
{
    @Test static void TestList()
    {
        List<Account> Teste = new List <Account>();
        for(Integer i=0;i<50;i++)
    }
}

```

```

{
    Teste.add(new Account(BillingState = 'CA', name = 'Test'+i));
}

for(Integer j=0;j<50;j++)
{
    Teste.add(new Account(BillingState = 'NY', name = 'Test'+j));
}

insert Teste;

Contact co = new Contact();
co.FirstName='demo';
co.LastName ='demo';
insert co;
String state = 'CA';

AddPrimaryContact apc = new AddPrimaryContact(co, state);
Test.startTest();
    System.enqueueJob(apc);
Test.stopTest();
}
}

```

Asynchronous Apex 4th sub-module

DailyLeadProcessor(created a class) :-

```

public class DailyLeadProcessor implements Schedulable {

    Public void execute(SchedulableContext SC){

        List<Lead> LeadObj=[SELECT Id from Lead where LeadSource=null limit 200];

        for(Lead l:LeadObj){

            l.LeadSource='Dreamforce';

```



```

        update l;
    }
}
}

```

DailyLeadProcessorTest (created a class):-

@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());

    }

}

```

Apex Integration Services 2nd sub-module

AnimalLocator (created a class):-

```

public class AnimalLocator{

    public static String getAnimalNameById(Integer x){

        Http http = new Http();

        HttpRequest req = new HttpRequest();

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

        req.setMethod('GET');
    }
}

```

```

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

    HttpResponse res = http.send(req);

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

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

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

    }

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

}
}

```

AnimalLocatorTest(created a class) :-

```

@Test
private class AnimalLocatorTest{

    @Test static void AnimalLocatorMock1() {

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

        String result = AnimalLocator.getAnimalNameById(3);

        String expectedResult = 'chicken';

        System.assertEquals(result,expectedResult );

    }

}

```

AnimalLocatorMock (created a class):-

```

@Test
global class AnimalLocatorMock implements HttpCalloutMock {

    // Implement this interface method

    global HttpResponse respond(HTTPRequest request) {

        // Create a fake response

        HttpResponse response = new HttpResponse();

        response.setHeader('Content-Type', 'application/json');

        response.setBody('{"animals": ["majestic badger", "fluffy bunny", "scary bear", "chicken", "mighty moose"]}');

    }

}

```

```

        response.setStatusCode(200);

        return response;
    }
}

```

Apex Integration Services 3rd sub-module

ParkLocator (created a class):-

```

public class ParkLocator {

    public static string[] country(string theCountry) {

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

        return parkSvc.byCountry(theCountry);

    }

}

```

ParkLocatorTest (created a class):-

```

@Test
private class ParkLocatorTest {

    @Test 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);

    }

}

```

ParkLocatorMock (created a class):-

```

@Test
global class ParkServiceMock implements WebServiceMock {

    global void doInvoke(

        Object stub,

```

```

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

```

Apex Integration Services 4th sub-module

AccountManager (created a class):-

```

@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;
    }
}

```

AccountManagerTest(created a class) :-

@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 Superbadge

Challenge 2

MaintenanceRequestHelper(Edited class):-

```

public with sharing class MaintenanceRequestHelper {

    public static void updateWorkOrders(List<Case> updWorkOrders, Map<Id,Case> nonUpdCaseMap) {

        Set<Id> validIds = new Set<Id>();

        For (Case c : updWorkOrders){

            if (nonUpdCaseMap.get(c.Id).Status != 'Closed' && c.Status == 'Closed'){

                if (c.Type == 'Repair' || c.Type == 'Routine Maintenance'){

                    validIds.add(c.Id);

                }

            }

        }

    }

}

```

```

//When an existing maintenance request of type Repair or Routine Maintenance is closed,
//create a new maintenance request for a future routine checkup.

if (!validIds.isEmpty()){

    Map<Id,Case> closedCases = new Map<Id,Case>([SELECT Id, Vehicle__c, Equipment__c,
Equipment__r.Maintenance_Cycle__c,

                                (SELECT Id,Equipment__c,Quantity__c FROM
Equipment_Maintenance_Items__r)

                                FROM Case WHERE Id IN :validIds]);

    Map<Id,Decimal> maintenanceCycles = new Map<ID,Decimal>();

    //calculate the maintenance request due dates by using the maintenance cycle defined on the
related equipment records.

```

```

AggregateResult[] results = [SELECT Maintenance_Request__c,
                               MIN(Equipment__r.Maintenance_Cycle__c)cycle
                               FROM Equipment_Maintenance_Item__c
                               WHERE Maintenance_Request__c IN :ValidIds GROUP BY
Maintenance_Request__c];

for (AggregateResult ar : results){
    maintenanceCycles.put((Id) ar.get('Maintenance_Request__c'), (Decimal) ar.get('cycle'));
}

List<Case> newCases = new List<Case>();
for(Case cc : closedCases.values()){
    Case nc = new Case (
        ParentId = cc.Id,
        Status = 'New',
        Subject = 'Routine Maintenance',
        Type = 'Routine Maintenance',
        Vehicle__c = cc.Vehicle__c,
        Equipment__c =cc.Equipment__c,
        Origin = 'Web',
        Date_Reported__c = Date.Today()
    );

    //If multiple pieces of equipment are used in the maintenance request,
    //define the due date by applying the shortest maintenance cycle to today's date.
    If (maintenanceCycles.containsKey(cc.Id)){
        nc.Date_Due__c = Date.today().addDays((Integer) maintenanceCycles.get(cc.Id));
    } else {
        nc.Date_Due__c = Date.today().addDays((Integer) cc.Equipment__r.maintenance_Cycle__c);
    }
}

```

```

    }

    newCases.add(nc);
}

insert newCases;

List<Equipment_Maintenance_Item__c> clonedList = new
List<Equipment_Maintenance_Item__c>();
for (Case nc : newCases){
    for (Equipment_Maintenance_Item__c clonedListItem :
closedCases.get(nc.ParentId).Equipment_Maintenance_Items__r){
        Equipment_Maintenance_Item__c item = clonedListItem.clone();
        item.Maintenance_Request__c = nc.Id;
        clonedList.add(item);
    }
}
insert clonedList;
}
}
}

```

MaintenanceRequest(Edited trigger):-

```

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

```

Challenge 3

WarehouseCalloutService (Edited class):-


```

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 4

WarehouseSyncSchedule (Edited class):-

```

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

```

```
}  
}
```

Challenge 5

MaintenanceRequestHelperTest (Edited class):-

@istest

```
public with sharing class MaintenanceRequestHelperTest {
```

```
    private static final string STATUS_NEW = 'New';  
    private static final string WORKING = 'Working';  
    private static final string CLOSED = 'Closed';  
    private static final string REPAIR = 'Repair';  
    private static final string REQUEST_ORIGIN = 'Web';  
    private static final string REQUEST_TYPE = 'Routine Maintenance';  
    private static final string REQUEST_SUBJECT = 'Testing subject';
```

```
    PRIVATE STATIC Vehicle__c createVehicle(){  
        Vehicle__c Vehicle = new Vehicle__C(name = 'SuperTruck');  
        return Vehicle;  
    }
```

```
    PRIVATE STATIC Product2 createEq(){  
        product2 equipment = new product2(name = 'SuperEquipment',  
            lifespan_months__C = 10,  
            maintenance_cycle__C = 10,  
            replacement_part__c = true);  
        return equipment;  
    }
```

```
PRIVATE STATIC Case createMaintenanceRequest(id vehicleId, id equipmentId){
```

```
    case cs = new case(Type=REPAIR,  
        Status=STATUS_NEW,  
        Origin=REQUEST_ORIGIN,  
        Subject=REQUEST_SUBJECT,  
        Equipment__c=equipmentId,  
        Vehicle__c=vehicleId);
```

```
    return cs;
```

```
}
```

```
PRIVATE STATIC Equipment_Maintenance_Item__c createWorkPart(id equipmentId,id requestId){
```

```
    Equipment_Maintenance_Item__c wp = new Equipment_Maintenance_Item__c(Equipment__c =  
equipmentId,
```

```
        Maintenance_Request__c = requestId);
```

```
    return wp;
```

```
}
```

```
@istest
```

```
private static void testMaintenanceRequestPositive(){
```

```
    Vehicle__c vehicle = createVehicle();
```

```
    insert vehicle;
```

```
    id vehicleId = vehicle.Id;
```

```
    Product2 equipment = createEq();
```

```
    insert equipment;
```

```
    id equipmentId = equipment.Id;
```

```
    case somethingToUpdate = createMaintenanceRequest(vehicleId,equipmentId);
```

```
insert somethingToUpdate;
```

```
Equipment_Maintenance_Item__c workP = createWorkPart(equipmentId,somethingToUpdate.id);
```

```
insert workP;
```

```
test.startTest();
```

```
somethingToUpdate.status = CLOSED;
```

```
update somethingToUpdate;
```

```
test.stopTest();
```

```
Case newReq = [Select id, subject, type, Equipment__c, Date_Reported__c, Vehicle__c,  
Date_Due__c
```

```
from case
```

```
where status =:STATUS_NEW];
```

```
Equipment_Maintenance_Item__c workPart = [select id
```

```
from Equipment_Maintenance_Item__c
```

```
where Maintenance_Request__c =:newReq.Id];
```

```
system.assert(workPart != null);
```

```
system.assert(newReq.Subject != null);
```

```
system.assertEquals(newReq.Type, REQUEST_TYPE);
```

```
SYSTEM.assertEquals(newReq.Equipment__c, equipmentId);
```

```
SYSTEM.assertEquals(newReq.Vehicle__c, vehicleId);
```

```
SYSTEM.assertEquals(newReq.Date_Reported__c, system.today());
```

```
}
```

```
@istest
```

```
private static void testMaintenanceRequestNegative(){
```

```

Vehicle__C vehicle = createVehicle();

insert vehicle;

id vehicleId = vehicle.Id;


product2 equipment = createEq();

insert equipment;

id equipmentId = equipment.Id;


case emptyReq = createMaintenanceRequest(vehicleId,equipmentId);

insert emptyReq;


Equipment_Maintenance_Item__c workP = createWorkPart(equipmentId, emptyReq.Id);

insert workP;


test.startTest();

emptyReq.Status = WORKING;

update emptyReq;

test.stopTest();


list<case> allRequest = [select id
                        from case];


Equipment_Maintenance_Item__c workPart = [select id
                                           from Equipment_Maintenance_Item__c
                                           where Maintenance_Request__c = :emptyReq.Id];


system.assert(workPart != null);

system.assert(allRequest.size() == 1);
}

```

```

@istest
private static void testMaintenanceRequestBulk(){
    list<Vehicle__C> vehicleList = new list<Vehicle__C>();
    list<Product2> equipmentList = new list<Product2>();
    list<Equipment_Maintenance_Item__c> workPartList = new
list<Equipment_Maintenance_Item__c>();
    list<case> requestList = new list<case>();
    list<id> oldRequestIds = new list<id>();

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

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

    for(integer i = 0; i < 300; i++){
        workPartList.add(createWorkPart(equipmentList.get(i).id, requestList.get(i).id));
    }
    insert workPartList;

    test.startTest();
    for(case req : requestList){

```

```

    req.Status = CLOSED;

    oldRequestIds.add(req.Id);
}

```

```

update requestList;

```

```

test.stopTest();

```

```

list<case> allRequests = [select id
                        from case
                        where status =: STATUS_NEW];

```

```

list<Equipment_Maintenance_Item__c> workParts = [select id
                                                from Equipment_Maintenance_Item__c
                                                where Maintenance_Request__c in: oldRequestIds];

```

```

system.assert(allRequests.size() == 300);

```

```

}
}

```

MaintenanceRequestHelper (Edited class):-

```

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;
}
}
}

```

MaintenanceRequest (Edited trigger):-

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

```

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

```

Challenge 6

WarehouseCalloutService (Edited class):-

```

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');
}

}

```

WarehouseCalloutServiceTest (Edited class):-

```

@Test
private class WarehouseCalloutServiceTest {
    // implement your mock callout test here
    @isTest
    static void testWarehouseCallout() {
        test.startTest();
        test.setMock(HttpCalloutMock.class, new WarehouseCalloutServiceMock());
        WarehouseCalloutService.execute(null);
        test.stopTest();

        List<Product2> product2List = new List<Product2>();
        product2List = [SELECT ProductCode FROM Product2];

        System.assertEquals(3, product2List.size());
        System.assertEquals('55d66226726b611100aaf741', product2List.get(0).ProductCode);
        System.assertEquals('55d66226726b611100aaf742', product2List.get(1).ProductCode);
        System.assertEquals('55d66226726b611100aaf743', product2List.get(2).ProductCode);
    }
}

```

```
}  
}
```

WarehouseCalloutServiceMock (Edited class):-

@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;

```
    }  
}
```

Challenge 7

WarehouseSyncSchedule (Edited class):-

global class WarehouseSyncSchedule implements Schedulable {

 global void execute(SchedulableContext ctx) {

 WarehouseCalloutService.runWarehouseEquipmentSync();

```
    }  
}
```

WarehouseSyncScheduleTest (Edited class):-

@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 ');
```

```
    }
```

```
}
```