

### 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 Appex triggers-->

```
trigger ClosedOpportunityTrigger on Opportunity (before 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;
    }
}
```

### Get started with Apex unit tests-->

```
@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'));
    }
}

```

```

@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'));
    }
}

```

Test apex Trigger-->

```

@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" in not allowed for DML',
result.getErrors()[0].getMessage());
    }
}

```

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

```

    for(contact c: Trigger.New){
        if(c.LastName == 'INVALIDNAME'){
            c.AddError('The Last Name "' +c.LastName+ '" is not allowed for DML');
        }
    }
}

```

Create Test Data for Apex Tests-->

```

//@isTest
public class RandomContactFactory{
    public static List<Contact> generateRandomContacts(Integer numContactsToGenerate, String
FName){
        List<Contact> contactList = new List<Contact>();

        for(Integer i=0;i<numContactsToGenerate;i++){
            Contact c = new Contact(FirstName=FName + ' ' + i, LastName = 'Contact ' + i);
            contactList.add(c);
            System.debug(c);
        }
        //insert contactList;
        System.debug(contactList.size());
        return contactList;
    }
}

```

Use Future Methods-->

```

@Test
public class AccountProcessorTest{
    @isTest
    public static void testNoOfContacts(){
        Account a = new Account();
        a.Name = 'Test Account';
        Insert a;

        Contact c = new Contact();
        c.FirstName = 'Bob';
        c.LastName = 'Willie';
        c.AccountId = a.Id;

        Contact c2 = new Contact();
        c2.FirstName = 'Tom';
        c2.LastName = 'Cruise';
        c2.AccountId = a.Id;

        List<Id> acctIds = new List<Id>();
        acctIds.add(a.Id);

        Test.startTest();
    }
}

```

```

        AccountProcessor.countContacts(acctIds);
        Test.stopTest();
    }
}

```

```

public class AccountProcessor{
    @future
    public static void countContacts(List<Id> accountIds){
        List<Account> accounts = [Select Id, Name from Account Where Id IN : accountIds];
        List<Account> updatedAccounts = new List<Account>();
        for(Account account : accounts){
            account.Number_of_Contacts__c = [Select count() from Contact Where AccountId =:
account.Id];
            System.debug('No of Contacts =' + account.Number_of_Contacts__c);
            updatedAccounts.add(account);
        }
        update updatedAccounts;
    }
}

```

Use Batch Apex-->

```

public class LeadProcessor implements Database.Batchable<sObject>{
    public Database.QueryLocator start(Database.BatchableContext bh)
    {
        return Database.getQueryLocator([Select LeadSource From Lead]);
    }
    public void execute(Database.BatchableContext bh, List<Lead> leads)
    {
        for(Lead Lead : leads)
        {
            lead.LeadSource = 'Dreamforce';
        }
        update leads;
    }
    public void finish(Database.BatchableContext bh)
    {
    }
}

```

```
}
```

```
@isTest
public class LeadProcessorTest
{
    @testSetup

    static void setup()
    {
        List<Lead> leads = new List<Lead>();

        for (Integer i=0;i<200;i++)
        {
            leads.add(new Lead(LastName='Lead '+i,
                               Company='Lead',Status='Open - Not Contacted'));
        }
        insert leads;
    }
    static testmethod void test()
    {
        Test.startTest();
        LeadProcessor bh = new LeadProcessor();
        Id batchId = Database.executeBatch(bh, 200);
        Test.stopTest();

        System.assertEquals(200, [select count() from lead where LeadSource ='Dreamforce']);
    }
}
```

Control Processes with Queueable Apex-->

```
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 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;
    }
}
}

```

```

@Test
public class AddPrimaryContactTest{

    @Test static void TestList()
    {
        List<Account> family = new List <Account>();

        for(Integer b=0; b<50 ; b++)
        {
            family.add(new Account(BillingState = 'CA' ,name = 'Swadhin' +b));
        }
        for(Integer h=0; h<50; h++)
        {
            family.add(new Account(BillingState = 'NY', name = 'Shrivastav' +h));
        }
        insert family;

        Contact con = new Contact();
    }
}

```

```

        con.FirstName = 'Kamini';
con.LastName = 'Shrivastav';
insert con;
    String state = 'CA';

AddPrimaryContact apc = new AddPrimaryContact(con, state);
        Test.startTest();
        System.EnqueueJob(apc);
        Test.stopTest();
    }}

```

### Schedule Jobs Using the Apex Scheduler-->

```

    @isTest
private class DailyLeadProcessorTest{
    static testMethod void testDailyLeadProcessor()
    {
        String CRON_EXP = '0 0 1 * * ?';
        List<Lead> lList = new List<Lead>();
        for(Integer i =0 ; i<200 ; i++ ){
            lList.add(new Lead(LastName = 'Dreamforce'+i, Company='Test1 Inc.',Status='Open - Not
Contacted'));
        }
        insert lList;

        Test.startTest();
        String jobId = System.schedule('DailyLeadProcessor', CRON_EXP, new
DailyLeadProcessor());
    }
}

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;
        }
    }
}

```



## Apex REST Callouts-->

```
public class AnimalCallouts {

    public static HttpResponse makeGetCallout() {
        Http http = new Http();
        HttpRequest request = new HttpRequest();
        request.setEndpoint("");
        request.setMethod('GET');
        HttpResponse response = http.send(request);
        // if the request is successful, parse the JSON response.
        if (response.getStatusCode() == 200) {
            // Deserializes the JSON string into collections of primitive data types.
            Map<String, Object> results = (Map<String, Object>)
JSON.deserializeUntyped(response.getBody());
            // Cast the values in the 'animals' key as a list
            List<Object> animals = (List<Object>) results.get('animals');
            System.debug('Received the following animals:');
            for (Object animal: animals) {
                System.debug(animal);
            }
        }
        return response;
    }

    public static HttpResponse makePostCallout() {
        Http http = new Http();
        HttpRequest request = new HttpRequest();
        request.setEndpoint("");
        request.setMethod('POST');
        request.setHeader('Content-Type', 'application/json;charset=UTF-8');
        request.setBody('{"name":"mighty moose"}');
        HttpResponse response = http.send(request);
        // Parse the JSON response
        if (response.getStatusCode() != 201) {
            System.debug('The status code returned was not expected: ' +
                response.getStatusCode() + ' ' + response.getStatus());
        } else {
            System.debug(response.getBody());
        }
    }
}
```

```

        return response;
    }
}

```

@isTest

```
private class AnimalsCalloutsTest {
```

```

    @isTest static void testGetCalloutsTest() {
        // Create the mock response based on a static resource
        StaticResourceCalloutMock mock = new StaticResourceCalloutMock();
        mock.setStaticResource('GetAnimalResource');
        mock.setStatusCode(200);
        mock.setHeader('Content-Type', 'application/json;charset=UTF-8');
        // Associate the callout with a mock response
        Test.setMock(HttpCalloutMock.class, mock);
        // Call method to test
        HttpResponse result = AnimalsCallouts.makeGetCallout();
        // Verify mock response is not null
        System.assertNotEquals(null, result, 'The callout returned a null response.');
```

// Verify status code

```

        System.assertEquals(200,result.getStatusCode(), 'The status code is not 200.');
```

// Verify content type

```

        System.assertEquals('application/json;charset=UTF-8',
            result.getHeader('Content-Type'),
            'The content value is not expected.');
```

// Verify the array contains 3 items

```

        Map<String, Object> results = (Map<String, Object>)
            JSON.deserializeUntyped(result.getBody());
        List<Object> animals = (List<Object>) results.get('animals');
        System.assertEquals(3, animals.size(),
            'The array should only contain 3 items.');
```

```

    }
}

```

Apex SOAP Callouts--->

//Generated by wsdl2apex

```

public class ParkService {
    public class byCountryResponse {
        public String[] return_x;
        private String[] return_x_type_info = new String[]{return,'http://parks.services/',null,'0','1','false'};
        private String[] apex_schema_type_info = new String[]{"http://parks.services/","false","false"};
        private String[] field_order_type_info = new String[]{"return_x"};
    }
    public class byCountry {
        public String arg0;
        private String[] arg0_type_info = new String[]{"arg0","http://parks.services/',null,'0','1','false'};
        private String[] apex_schema_type_info = new String[]{"http://parks.services/","false","false"};
        private String[] field_order_type_info = new String[]{"arg0"};
    }
    public class ParksImplPort {
        public String endpoint_x = 'https://th-apex-soap-service.herokuapp.com/service/parks';
        public Map<String,String> inputHttpHeaders_x;
        public Map<String,String> outputHttpHeaders_x;
        public String clientCertName_x;
        public String clientCert_x;
        public String clientCertPasswd_x;
        public Integer timeout_x;
        private String[] ns_map_type_info = new String[]{"http://parks.services/", 'ParkService'};
        public String[] byCountry(String arg0) {
            ParkService.byCountry request_x = new ParkService.byCountry();
            request_x.arg0 = arg0;
            ParkService.byCountryResponse response_x;
            Map<String, ParkService.byCountryResponse> response_map_x = new Map<String,
ParkService.byCountryResponse>();
            response_map_x.put('response_x', response_x);
            WebServiceCallout.invoke(
                this,
                request_x,
                response_map_x,
                new String[]{endpoint_x,
                ",
                'http://parks.services/',
                'byCountry',
                'http://parks.services/',
                'byCountryResponse',
                'ParkService.byCountryResponse'}

```

```

        );
        response_x = response_map_x.get('response_x');
        return response_x.return_x;
    }
}
}

```

```

public class ParkLocator {
    public static String[] country(String country){
        ParkService.ParksImplPort parks = new ParkService.ParksImplPort();
        String[] parksname = parks.byCountry(country);
        return parksname;
    }
}

```

```

@Test
private class ParkLocatorTest{
    @Test
    static void testParkLocator() {
        Test.setMock(WebServiceMock.class, new ParkServiceMock());
        String[] arrayOfParks = ParkLocator.country('India');

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

```

```

@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) {
        ParkService.byCountryResponse response_x = new ParkService.byCountryResponse();
        List<String> lstOfDummyParks = new List<String> {'Park1','Park2','Park3'};
        response_x.return_x = lstOfDummyParks;
    }
}

```

```

        response.put('response_x', response_x);
    }
}

```

Apex Web Services---->

```

@IsTest
private class AccountManagerTest{
    @isTest static void testAccountManager(){
        Id recordId = getTestAccountId();
        // Set up a test request
        RestRequest request = new RestRequest();
        request.requestUri =
            'https://ap5.salesforce.com/services/apexrest/Accounts/'+ recordId +'/contacts';
        request.httpMethod = 'GET';
        RestContext.request = request;

        // Call the method to test
        Account acc = AccountManager.getAccount();

        // Verify results
        System.assert(acc != null);
    }

    private static Id getTestAccountId(){
        Account acc = new Account(Name = 'TestAcc2');
        Insert acc;

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

        return acc.Id;
    }
}

```

```

@RestResource(urlMapping='/Accounts/*/contacts')
global with sharing class AccountManager{

```

```

@HttpGet
global static Account getAccount(){
    RestRequest req = RestContext.request;
    String accId = req.requestURI.substringBetween('Accounts/', '/contacts');
    Account acc = [SELECT Id, Name, (SELECT Id, Name FROM Contacts)
                  FROM Account WHERE Id = :accId];

    return acc;
}
}

```

step2 Automate record creation----->

```

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

```

```

public with sharing class MaintenanceRequestHelper {
    public static void updateWorkOrders(List<Case> updWorkOrders, Map<Id,Case>
nonUpdCaseMap) {
        Set<Id> validIds = new Set<Id>();
        For (Case c : updWorkOrders){
            if (nonUpdCaseMap.get(c.Id).Status != 'Closed' && c.Status == 'Closed'){
                if (c.Type == 'Repair' || c.Type == 'Routine Maintenance'){
                    validIds.add(c.Id);
                }
            }
        }
    }
}

```

```

//When an existing maintenance request of type Repair or Routine Maintenance is closed,
//create a new maintenance request for a future routine checkup.
if (!validIds.isEmpty()){
    Map<Id,Case> closedCases = new Map<Id,Case>([SELECT Id, Vehicle__c, Equipment__c,
Equipment__r.Maintenance_Cycle__c,
                (SELECT Id,Equipment__c,Quantity__c FROM
Equipment_Maintenance_Items__r)
                FROM Case WHERE Id IN :validIds]);
    Map<Id,Decimal> maintenanceCycles = new Map<ID,Decimal>();
}

```

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

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

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

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

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

```
    newCases.add(nc);  
}
```

```
insert newCases;
```

```

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

```

step3 Synchronize Salesforce data with an external system-->

```

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');
}

}

```

step4 Schedule synchronization--->

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

step5 Test automation logic---->

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

```
public with sharing class MaintenanceRequestHelper {
    public static void updateworkOrders(List<Case> updWorkOrders, Map<Id,Case>
nonUpdCaseMap) {
        Set<Id> validIds = new Set<Id>();
        For (Case c : updWorkOrders){
            if (nonUpdCaseMap.get(c.Id).Status != 'Closed' && c.Status == 'Closed'){
                if (c.Type == 'Repair' || c.Type == 'Routine Maintenance'){
                    validIds.add(c.Id);
                }
            }
        }
    }
}
```

```
//When an existing maintenance request of type Repair or Routine Maintenance is closed,
//create a new maintenance request for a future routine checkup.
if (!validIds.isEmpty()){
    Map<Id,Case> closedCases = new Map<Id,Case>([SELECT Id, Vehicle__c, Equipment__c,
Equipment__r.Maintenance_Cycle__c,
(SELECT Id,Equipment__c,Quantity__c FROM
Equipment_Maintenance_Items__r)
FROM Case WHERE Id IN :validIds]);
```

```

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

//calculate the maintenance request due dates by using the maintenance cycle defined
on the related equipment records.
AggregateResult[] results = [SELECT Maintenance_Request__c,
                               MIN(Equipment__r.Maintenance_Cycle__c)cycle
                               FROM Equipment_Maintenance_Item__c
                               WHERE Maintenance_Request__c IN :ValidIds GROUP BY
Maintenance_Request__c];

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

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

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

    newCases.add(nc);
}

insert newCases;

```

```

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

```

@isTest

public with sharing class MaintenanceRequestHelperTest {

// createVehicle

```

private static Vehicle__c createVehicle(){
    Vehicle__c vehicle = new Vehicle__C(name = 'Testing Vehicle');
    return vehicle;
}

```

// createEquipment

```

private static Product2 createEquipment(){
    product2 equipment = new product2(name = 'Testing equipment',
        lifespan_months__c = 10,
        maintenance_cycle__c = 10,
        replacement_part__c = true);
    return equipment;
}

```

// createMaintenanceRequest

```

private static Case createMaintenanceRequest(id vehicleId, id equipmentId){
    case cse = new case(Type='Repair',
        Status='New',
        Origin='Web',
        Subject='Testing subject',

```

```

        Equipment__c=equipmentId,
        Vehicle__c=vehicleId);
    return cse;
}

// createEquipmentMaintenanceItem
private static Equipment_Maintenance_Item__c createEquipmentMaintenanceItem(id
equipmentId,id requestId){
    Equipment_Maintenance_Item__c equipmentMaintenanceItem = new
Equipment_Maintenance_Item__c(
    Equipment__c = equipmentId,
    Maintenance_Request__c = requestId);
    return equipmentMaintenanceItem;
}

@Test
private static void testPositive(){
    Vehicle__c vehicle = createVehicle();
    insert vehicle;
    id vehicleId = vehicle.Id;

    Product2 equipment = createEquipment();
    insert equipment;
    id equipmentId = equipment.Id;

    case createdCase = createMaintenanceRequest(vehicleId,equipmentId);
    insert createdCase;

    Equipment_Maintenance_Item__c equipmentMaintenanceItem =
createEquipmentMaintenanceItem(equipmentId,createdCase.id);
    insert equipmentMaintenanceItem;

    test.startTest();
    createdCase.status = 'Closed';
    update createdCase;
    test.stopTest();

    Case newCase = [Select id,
        subject,
        type,
        Equipment__c,

```

```
    Date_Reported__c,  
    Vehicle__c,  
    Date_Due__c  
from case  
where status ='New'];
```

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

```
list<case> allCase = [select id from case];  
system.assert(allCase.size() == 2);
```

```
system.assert(newCase != null);  
system.assert(newCase.Subject != null);  
system.assertEquals(newCase.Type, 'Routine Maintenance');  
SYSTEM.assertEquals(newCase.Equipment__c, equipmentId);  
SYSTEM.assertEquals(newCase.Vehicle__c, vehicleId);  
SYSTEM.assertEquals(newCase.Date_Reported__c, system.today());  
}
```

@isTest

```
private static void testNegative(){  
    Vehicle__C vehicle = createVehicle();  
    insert vehicle;  
    id vehicleId = vehicle.Id;
```

```
    product2 equipment = createEquipment();  
    insert equipment;  
    id equipmentId = equipment.Id;
```

```
    case createdCase = createMaintenanceRequest(vehicleId,equipmentId);  
    insert createdCase;
```

```
    Equipment_Maintenance_Item__c workP = createEquipmentMaintenanceItem(equipmentId,  
createdCase.Id);  
    insert workP;
```

```
    test.startTest();  
    createdCase.Status = 'Working';  
    update createdCase;  
    test.stopTest();
```

```

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

Equipment_Maintenance_Item__c equipmentMaintenanceltem = [select id
                    from Equipment_Maintenance_Item__c
                    where Maintenance_Request__c = :createdCase.Id];

system.assert(equipmentMaintenanceltem != null);
system.assert(allCase.size() == 1);
}

@isTest
private static void testBulk(){
    list<Vehicle__C> vehicleList = new list<Vehicle__C>();
    list<Product2> equipmentList = new list<Product2>();
    list<Equipment_Maintenance_Item__c> equipmentMaintenanceltemList = new
list<Equipment_Maintenance_Item__c>();
    list<case> caseList = new list<case>();
    list<id> oldCaseIds = new list<id>();

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

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

    for(integer i = 0; i < 300; i++){

equipmentMaintenanceltemList.add(createEquipmentMaintenanceltem(equipmentList.get(i).id,
caseList.get(i).id));
    }
    insert equipmentMaintenanceltemList;

    test.startTest();
    for(case cs : caseList){

```

```

        cs.Status = 'Closed';
        oldCaseIds.add(cs.Id);
    }
    update caseList;
    test.stopTest();

    list<case> newCase = [select id
                        from case
                        where status ='New'];

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

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

    list<case> allCase = [select id from case];
    system.assert(allCase.size() == 600);
}
}

```

```

@isTest
public with sharing class MaintenanceRequestHelperTest {

    // createVehicle
    private static Vehicle__c createVehicle(){
        Vehicle__c vehicle = new Vehicle__C(name = 'Testing Vehicle');
        return vehicle;
    }

    // createEquipment
    private static Product2 createEquipment(){
        product2 equipment = new product2(name = 'Testing equipment',
                                           lifespan_months__c = 10,
                                           maintenance_cycle__c = 10,
                                           replacement_part__c = true);
        return equipment;
    }
}

```



```

// createMaintenanceRequest
private static Case createMaintenanceRequest(id vehicleId, id equipmentId){
    case cse = new case(Type='Repair',
        Status='New',
        Origin='Web',
        Subject='Testing subject',
        Equipment__c=equipmentId,
        Vehicle__c=vehicleId);
    return cse;
}

// createEquipmentMaintenanceItem
private static Equipment_Maintenance_Item__c createEquipmentMaintenanceItem(id
equipmentId,id requestId){
    Equipment_Maintenance_Item__c equipmentMaintenanceItem = new
Equipment_Maintenance_Item__c(
    Equipment__c = equipmentId,
    Maintenance_Request__c = requestId);
    return equipmentMaintenanceItem;
}

@isTest
private static void testPositive(){
    Vehicle__c vehicle = createVehicle();
    insert vehicle;
    id vehicleId = vehicle.Id;

    Product2 equipment = createEquipment();
    insert equipment;
    id equipmentId = equipment.Id;

    case createdCase = createMaintenanceRequest(vehicleId,equipmentId);
    insert createdCase;

    Equipment_Maintenance_Item__c equipmentMaintenanceItem =
createEquipmentMaintenanceItem(equipmentId,createdCase.id);
    insert equipmentMaintenanceItem;

    test.startTest();
    createdCase.status = 'Closed';
}

```

```
update createdCase;  
test.stopTest();
```

```
Case newCase = [Select id,  
                subject,  
                type,  
                Equipment__c,  
                Date_Reported__c,  
                Vehicle__c,  
                Date_Due__c  
from case  
where status ='New'];
```

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

```
list<case> allCase = [select id from case];  
system.assert(allCase.size() == 2);
```

```
system.assert(newCase != null);  
system.assert(newCase.Subject != null);  
system.assertEquals(newCase.Type, 'Routine Maintenance');  
SYSTEM.assertEquals(newCase.Equipment__c, equipmentId);  
SYSTEM.assertEquals(newCase.Vehicle__c, vehicleId);  
SYSTEM.assertEquals(newCase.Date_Reported__c, system.today());  
}
```

```
@isTest
```

```
private static void testNegative(){  
    Vehicle__C vehicle = createVehicle();  
    insert vehicle;  
    id vehicleId = vehicle.Id;
```

```
product2 equipment = createEquipment();  
insert equipment;  
id equipmentId = equipment.Id;
```

```
case createdCase = createMaintenanceRequest(vehicleId,equipmentId);  
insert createdCase;
```

```
Equipment_Maintenance_Item__c workP = createEquipmentMaintenanceItem(equipmentId,
```

```

createdCase.Id);
    insert workP;

    test.startTest();
    createdCase.Status = 'Working';
    update createdCase;
    test.stopTest();

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

    Equipment_Maintenance_Item__c equipmentMaintenanceltem = [select id
                                                                from Equipment_Maintenance_Item__c
                                                                where Maintenance_Request__c = :createdCase.Id];

    system.assert(equipmentMaintenanceltem != null);
    system.assert(allCase.size() == 1);
}

@isTest
private static void testBulk(){
    list<Vehicle__C> vehicleList = new list<Vehicle__C>();
    list<Product2> equipmentList = new list<Product2>();
    list<Equipment_Maintenance_Item__c> equipmentMaintenanceltemList = new
list<Equipment_Maintenance_Item__c>();
    list<case> caseList = new list<case>();
    list<id> oldCaseIds = new list<id>();

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

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

    for(integer i = 0; i < 300; i++){

```

```

equipmentMaintenanceItem.add(createEquipmentMaintenanceItem(equipmentList.get(i).id,
caseList.get(i).id));
    }
    insert equipmentMaintenanceItem;

    test.startTest();
    for(case cs : caseList){
        cs.Status = 'Closed';
        oldCaseIds.add(cs.Id);
    }
    update caseList;
    test.stopTest();

    list<case> newCase = [select id
                        from case
                        where status ='New'];

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

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

    list<case> allCase = [select id from case];
    system.assert(allCase.size() == 600);
}
}

```

step6 Test callout logic--->

```

public with sharing class WarehouseCalloutService implements Queueable {
    private static final String WAREHOUSE_URL = 'https://th-superbadge-
apex.herokuapp.com/equipment';

```

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

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

```

@future(callout=true)
public static void runWarehouseEquipmentSync(){
    System.debug('go into runWarehouseEquipmentSync');
    Http http = new Http();
    HttpRequest request = new HttpRequest();

    request.setEndpoint(WAREHOUSE_URL);
    request.setMethod('GET');
    HttpResponse response = http.send(request);

    List<Product2> product2List = new List<Product2>();
    System.debug(response.getStatusCode());
    if (response.getStatusCode() == 200){
        List<Object> jsonResponse =
(List<Object>)JSON.deserializeUntyped(response.getBody());
        System.debug(response.getBody());

        //class maps the following fields:
        //warehouse SKU will be external ID for identifying which equipment records to update
        within Salesforce
        for (Object jR : jsonResponse){
            Map<String,Object> mapJson = (Map<String,Object>)jR;
            Product2 product2 = new Product2();
            //replacement part (always true),
            product2.Replacement_Part__c = (Boolean) mapJson.get('replacement');
            //cost
            product2.Cost__c = (Integer) mapJson.get('cost');
            //current inventory
            product2.Current_Inventory__c = (Double) mapJson.get('quantity');
            //lifespan
            product2.Lifespan_Months__c = (Integer) mapJson.get('lifespan');
            //maintenance cycle
            product2.Maintenance_Cycle__c = (Integer) mapJson.get('maintenanceperiod');
            //warehouse SKU
            product2.Warehouse_SKU__c = (String) mapJson.get('sku');

            product2.Name = (String) mapJson.get('name');
            product2.ProductCode = (String) mapJson.get('_id');
            product2List.add(product2);
        }
    }
}

```

```

        if (product2List.size() > 0){
            upsert product2List;
            System.debug('Your equipment was synced with the warehouse one');
        }
    }
}

public static void execute (QueueableContext context){
    System.debug('start runWarehouseEquipmentSync');
    runWarehouseEquipmentSync();
    System.debug('end runWarehouseEquipmentSync');
}
}

```

```

@Test
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":"55d66226726b611
100aaf742","replacement":true,"quantity":183,"name":"Cooling
Fan","maintenanceperiod":0,"lifespan":0,"cost":300,"sku":"100004"},{"_id":"55d66226726b611100a
af743","replacement":true,"quantity":143,"name":"Fuse
20A","maintenanceperiod":0,"lifespan":0,"cost":22,"sku":"100005"}]');
        response.setStatusCode(200);

        return response;
    }
}

```

```

@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);
    }
}

```

step7 test scheduling logic-->

```

@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":"55d66226726b611
100aaf742","replacement":true,"quantity":183,"name":"Cooling
Fan","maintenanceperiod":0,"lifespan":0,"cost":300,"sku":"100004"},{"_id":"55d66226726b611100a
af743","replacement":true,"quantity":143,"name":"Fuse
20A","maintenanceperiod":0,"lifespan":0,"cost":22,"sku":"100005"}');
        response.setStatusCode(200);

        return response;
    }
}

```

```

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

```

```

@isTest
public with sharing class WarehouseSyncScheduleTest {
    // implement scheduled code here
    //
    @isTest static void test() {
        String scheduleTime = '00 00 00 * * ? *';
        Test.startTest();
        Test.setMock(HttpCalloutMock.class, new WarehouseCalloutServiceMock());
        String jobId = System.schedule('Warehouse Time to Schedule to test', scheduleTime, new
WarehouseSyncSchedule());
        CronTrigger c = [SELECT State FROM CronTrigger WHERE Id =: jobId];
        System.assertEquals('WAITING', String.valueOf(c.State), 'JobId does not match');

        Test.stopTest();
    }
}

```