APEX TRIGGERS

Get Started with Apex Triggers

AccountAddressTrigger

```
trigger AccountAddressTrigger on Account (before insert, before update) {
for(Account account:Trigger.New){
if(account.Match_Billing_Address__c == True){
account.ShippingPostalCode = account.BillingPostalCode;
}
}
}
```

Bulk Apex Triggers

Apex trigger

ClosedOpportunityTrigger

```
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;
}
}
```

Salesforce Developer Catalyst

Apex Testing

Get Started with Apex Unit Tests

VerifyDate

```
public class VerifyDate {
public static Date CheckDates(Date date1, Date date2) {
if(DateWithin30Days(date1,date2)) {
return date2;
} else {
return SetEndOfMonthDate(date1);
}
}
@TestVisible private static Boolean DateWithin30Days(Date date1, Date date2) {
if( date2 < date1) { return false; }
Date date30Days = date1.addDays(30);
if( date2 >= date30Days ) { return false; }
else { return true; }
}
@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;
}
}
```

Salesforce Developer Catalyst

TestVerifyDate

```
@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('05/05/2030'));
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'));
}
}
```

Salesforce Developer Catalyst

Test Apex Triggers

RestrictContactByName

```
trigger RestrictContactByName on Contact (before insert, before update) {
//check contacts prior to insert or update for invalid data
For (Contact c : Trigger.New) {
if(c.LastName == 'INVALIDNAME') { //invalidname is invalid
c.AddError('The Last Name "'+c.LastName+'" is not allowed for DML
');
}
}
}
```

TestRestrictContactByName

```
@isTest
public class TestRestrictContactByName {
@isTest 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

RandomContactFactory

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

Salesforce Developer Catalyst

```
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

AccountProcessor

```
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;
}
}
```

AccountProcessorTest

```
@IsTest
public class AccountProcessorTest {
@IsTest
private static void testCountContacts(){
Account newAccount = new Account(Name='Test Account');
```

Salesforce Developer Catalyst

```
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

LeadProcessor

```
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;
```

Salesforce Developer Catalyst

```
}
global void finish(Database.BatchableContext bc){
system.debug('count = '+ count);
}
}
```

LeadProcessorTest

```
@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

AddPrimaryContact

```
public class AddPrimaryContact implements Queueable {
public contact c;
public String state;
```

Salesforce Developer Catalyst

```
public AddPrimaryContact(Contact c, String state) {
this.c = c;
this.state = state;
}
public void execute(QueueableContext qc) {
system.debug('this.c = '+this.c+' this.state = '+this.state);
List<Account> acc_lst = new List<account>([select id, name, BillingState from account
where account.BillingState = :this.state limit 200]);
List<contact> c_lst = new List<contact>();
for(account a: acc_lst) {
contact c = new contact();
c = this.c.clone(false, false, false, false);
c.AccountId = a.Id;
c_lst.add(c);
}
insert c_lst;
}
}
```

AddPrimaryContactTest

```
@IsTest
public class AddPrimaryContactTest {
@IsTest
public static void testing() {
List<account> acc_lst = new List<account>();
for (Integer i=0; i<50;i++) {
account a = new account(name=string.valueOf(i),billingstate='NY');
system.debug('account a = '+a);
acc_lst.add(a);
}
for (Integer i=0; i<50;i++) {
account a = new account(name=string.valueOf(50+i),billingstate='CA');
system.debug('account a = '+a);
acc_lst.add(a);
}
insert acc_lst;
```

Salesforce Developer Catalyst

```
Test.startTest();
contact c = new contact(lastname='alex');
AddPrimaryContact apc = new AddPrimaryContact(c,'CA');
system.debug('apc = '+apc);
System.enqueueJob(apc);
Test.stopTest();
List<contact> c_lst = new List<contact>([select id from contact]);
Integer size = c_lst.size();
system.assertEquals(50, size);
}
}
```

Schedule Jobs Using the Apex Scheduler

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;
}
}
}
```

DailyLeadProcessorTest

```
@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;
```

Salesforce Developer Catalyst

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

Apex Integration Services

Apex REST Callouts

AnimalLocator

```
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

```
@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 );
}
```

Salesforce Developer Catalyst

```
}
```

AnimalLocatorMock

```
@isTest
global class AnimalLocatorMock implements HttpCalloutMock {
global HTTPResponse respond(HTTPRequest request) {
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 SOAP Callouts

ParkService

```
//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'};
```

Salesforce Developer Catalyst

```
}
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;
}
}
}
```

ParkLocator

```
public class ParkLocator { public static string[] country(string theCountry) {
```

Salesforce Developer Catalyst

```
ParkService.ParksImplPort parkSvc = new ParkService.ParksImplPort(); // remove space
return parkSvc.byCountry(theCountry);
}
}
```

ParkLocatorTest

```
@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

AccountManager

```
@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

```
@isTest
private class AccountManagerTest {
```

Salesforce Developer Catalyst

```
private static testMethod void getAccountTest1() {
Id recordId = createTestRecord();
RestRequest request = new RestRequest();
request.requestUri = 'https://na1.salesforce.com/services/apexrest/Accounts/'+ recordId
+'/contacts' ;
request.httpMethod = 'GET';
RestContext.request = request;
Account thisAccount = AccountManager.getAccount();
// Verify results
System.assert(thisAccount != null);
System.assertEquals('Test record', thisAccount.Name);
}
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

Automated Record Creation (Step 2)

MaintenanceRequestHelper

```
public with sharing class MaintenanceRequestHelper {
public static void updateworkOrders(List<Case> updWorkOrders, Map<Id,Case>
nonUpdCaseMap) {
Set<Id> validIds = new Set<Id>();
```

Salesforce Developer Catalyst

```
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)
```

Salesforce Developer Catalyst

```
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;
}
}
}
```

MaitenanceRequest

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

Synchronize Salesforce data with an external system (Step 3)

WarehouseCalloutService

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

Salesforce Developer Catalyst

```
@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 = (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){
```

Salesforce Developer Catalyst

```
runWarehouseEquipmentSync();
}
}
```

Schedule synchronization using Apex code (Step 4)

WarehouseSyncShedule

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

Test automation logic (Step 5)

MaintenanceRequestHelperTest

```
@istest
public with sharing class MaintenanceRequestHelperTest {
private static final string STATUS_NEW = 'New';
private static final string WORKING = 'Working';
private static final string CLOSED = 'Closed';
private static final string REPAIR = 'Repair';
private static final string REQUEST_ORIGIN = 'Web';
private static final string REQUEST_TYPE = 'Routine Maintenance';
private static final string REQUEST_SUBJECT = 'Testing subject';
PRIVATE STATIC Vehicle__c createVehicle(){
Vehicle__c Vehicle = new Vehicle__C(name = 'SuperTruck');
return Vehicle;
}
PRIVATE STATIC Product2 createEq(){
product2 equipment = new product2(name = 'SuperEquipment',
lifespan_months__C = 10,
maintenance_cycle__C = 10
```

Salesforce Developer Catalyst

```
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;
```

Salesforce Developer Catalyst

```
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;
```

Salesforce Developer Catalyst

```
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++){
```

Salesforce Developer Catalyst

```
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

```
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);
}
}
}
```

Salesforce Developer Catalyst

```
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>();
```

Salesforce Developer Catalyst

```
for (Case nc : newCases){
for (Equipment_Maintenance_Item__c wp :
closedCasesM.get(nc.ParentId).Equipment_Maintenance_Items__r){
Equipment_Maintenance_Item__c wpClone = wp.clone();
wpClone.Maintenance_Request__c = nc.Id;
ClonedWPs.add(wpClone);
}
}
insert ClonedWPs;
}
}
}
```

MaintenanceRequest

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

Test callout logic (Step 6)

WarehouseCalloutService

```
public with sharing class WarehouseCalloutService {
private static final String WAREHOUSE_URL = 'https://th-superbadge-
apex.herokuapp.com/equipment';
//@future(callout=true)
public static void runWarehouseEquipmentSync(){
Http http = new Http();
HttpRequest request = new HttpRequest();
request.setEndpoint(WAREHOUSE_URL);
request.setMethod('GET');
HttpResponse response = http.send(request);
```

Salesforce Developer Catalyst

```
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);
}
}
}
}
```

WarehouseCalloutServiceTest

```
@isTest
private class WarehouseCalloutServiceTest {
@isTest
static void testWareHouseCallout(){
Test.startTest();
Test.setMock(HTTPCalloutMock.class, new WarehouseCalloutServiceMock());
WarehouseCalloutService.runWarehouseEquipmentSync();
Test.stopTest()
```

Salesforce Developer Catalyst

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

WarehouseCalloutServiceMock

```
@isTest
global class WarehouseCalloutServiceMock implements HttpCalloutMock {
// implement http mock callout
global static HttpResponse respond(HttpRequest request){
System.assertEquals('https://th-superbadge-apex.herokuapp.com/equipment',
request.getEndpoint());
System.assertEquals('GET', request.getMethod());
// Create a fake response
HttpResponse response = new HttpResponse();
response.setHeader('Content-Type', 'application/json');
response.setBody('[{"_id":"55d66226726b611100aaf741","replacement":false,"quantity":5,"nam
e":"Generator 1000 kW","maintenanceperiod":365,"lifespan":120,"cost":5000,"sku":"100003"}]');
response.setStatusCode(200);
return response;
}
}
```

Test scheduling logic (Step 7)

WarehouseSyncSchedule

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

WarehouseSyncScheduleTest

```
@isTest
public with sharing class WarehouseSyncScheduleTest {
@isTest static void test() {
```

Salesforce Developer Catalyst

```
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();
}
}
```

WarehouseCalloutServiceMock

```
@isTest
global class WarehouseCalloutServiceMock implements HttpCalloutMock {
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;
}
}
```

Trailhead URL : https://trailblazer.me/id/ujwam2

sincerely,

UJWAL . M