

APEX TRIGGERS:

Getting Started with Apex Triggers;

```
trigger AccountAddressTrigger on Account (before insert, before update) {
    For(Account accountAddress: Trigger.new){
        if(accountAddress.BillingPostalCode !=null && accountAddress.Match_Billing_Address__c
==true){
            accountAddress.ShippingPostalCode=accountAddress.BillingPostalCode;
        }
    }
}
```

Bulk Apex Triggers;

```
trigger ClosedOpportunityTrigger on Opportunity (before insert, before update) {
    List<Task> newTask = new List <Task>();
```

```
    //Grab the Opportunity Id's from Opps that are Closed Won from the Context Variable and
store them in opp
```

```
    for(Opportunity opp : [SELECT Id FROM Opportunity
        WHERE StageName = 'Closed Won' IN :Trigger.New]){
```

```
        //Create a Follow Up Task against Id's that are stored in the variable opp
        newTask.add(new Task(Subject = 'Follow Up Test Task',
            Priority = 'High',
            WhatId = opp.Id));
```

```
        //Insert new Tasks
        {insert newTask;
```

```
    }
}
}
```

APEX TESTING:

Get Started with Apex Testing;

@isTest

```
private class TestVerifyDate {
    static testMethod void TestVerifyDate() {
        VerifyDate.CheckDates(System.today(),System.today().addDays(10));
        VerifyDate.CheckDates(System.today(),System.today().addDays(78));
```

```
}  
}
```

Test Apex Triggers;

@isTest

```
private class TestRestrictContactByName {
```

```
    @isTest static void testInvalidName() {
```

```
        //try inserting a Contact with INVALIDNAME
```

```
        Contact myConact = new Contact(LastName='INVALIDNAME');
```

```
        insert myConact;
```

```
        // Perform test
```

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

```
        Database.SaveResult result = Database.insert(myConact, false);
```

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

```
        // Verify
```

```
        // In this case the creation should have been stopped by the trigger,
```

```
        // so verify that we got back an error.
```

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

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

```
        System.assertEquals('Cannot create contact with invalid last name.',  
                             result.getErrors()[0].getMessage());
```

```
    }
```

```
}
```

Create Test data for Apex Tests;

```
public class RandomContactFactory {
```

```
    public static List<Contact> generateRandomContacts(Integer count, String name) {
```

```
        List<Contact> contactList = new List<Contact>();
```

```
        for(Integer index = 1; index <= count; index++) {
```

```
            Contact c = new Contact();
```

```
            c.FirstName = name + index;
```

```
            contactList.add(c);
```

```
        }
```

```
        return contactList;
```

```
    }
```

```
}
```

ASYNCHRONOUS APEX:

Asynchronous apex using future methods;

```
public class AccountProcessor {

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

        Map<Id,Integer> account_cno = new Map<Id,Integer>();
        List<account> account_Lst_all = new List<account>([select id, (select id from contacts) from
account]);
        for(account a:account_Lst_all) {
            account_cno.put(a.id,a.contacts.size()); //populate the map

        }

        List<account> account_Lst = new List<account>(); // list of account that we will upsert

        for(Id accountId : accountId_Lst) {
            if(account_cno.containsKey(accountId)) {
                account acc = new account();
                acc.Id = accountId;
                acc.Number_of_Contacts__c = account_cno.get(accountId);
                account_Lst.add(acc);
            }

        }
        upsert account_Lst;
    }
}
```

Asynchronous Apex using Batch Apex;

```
global class LeadProcessor implements Database.Batchable<sObject> {
    global Integer count = 0;

    global Database.QueryLocator start (Database.BatchableContext bc) {
        return Database.getQueryLocator('Select Id, LeadSource from lead');
    }
}
```

```

global void execute (Database.BatchableContext bc,List<Lead> l_lst) {
    List<lead> l_lst_new = new List<lead>();
    for(lead l : l_lst) {
        l.leadsource = 'Dreamforce';
        l_lst_new.add(l);
        count+=1;
    }
    update l_lst_new;
}

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

```

Control Processes with Queueable Apex;

Apex Class;

```

global class LeadProcessor implements Database.Batchable<Subject>
{
    global Database.QueryLocator start(Database.BatchableContext bc)
    {
        return Database.getQueryLocator([Select LastName From Lead ]);
    }

    global void execute(Database.BatchableContext bc, List<Lead> scope)
    {
        for (Lead Leads : scope)
        {
            Leads.LeadSource = 'Dreamforce';
        }
        update scope;
    }

    global void finish(Database.BatchableContext bc){ }
}

```

Test Class;

```

@isTest
public class LeadProcessorTest
{
    static testMethod void testMethod1()

```

```

{
    List<Lead> lstLead = new List<Lead>();
    for(Integer i=0 ;i <200;i++)
    {
        Lead led = new Lead();
        led.FirstName ='FirstName';
        led.LastName ='LastName'+i;
        led.Company ='demo'+i;
        lstLead.add(led);
    }

    insert lstLead;

    Test.startTest();

    LeadProcessor obj = new LeadProcessor();
    DataBase.executeBatch(obj);

    Test.stopTest();
}
}

```

Schedule jobs using apex scheduler;

Apex Class;

```

global class DailyLeadProcessor implements Schedulable{
    global void execute(SchedulableContext ctx){
        List<Lead> leads = [SELECT Id, LeadSource FROM Lead WHERE LeadSource = "];

        if(leads.size() > 0){
            List<Lead> newLeads = new List<Lead>();

            for(Lead lead : leads){
                lead.LeadSource = 'DreamForce';
                newLeads.add(lead);
            }

            update newLeads;
        }
    }
}
}

```

Test Class;

@isTest

private class DailyLeadProcessorTest{

 //Seconds Minutes Hours Day_of_month Month Day_of_week optional_year

 public static String CRON_EXP = '0 0 0 2 6 ? 2022';

 static testmethod void testScheduledJob(){

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

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

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

 leads.add(lead);

 }

 insert leads;

 Test.startTest();

 // Schedule the test job

 String jobId = System.schedule('Update LeadSource to DreamForce', CRON_EXP, new DailyLeadProcessor());

 // Stopping the test will run the job synchronously

 Test.stopTest();

 }

}

APEX INTEGRATION SERVICES;

Apex Rest Callouts;

public class AnimalLocator {

 public class cls_animal {

 public Integer id;

 public String name;

 public String eats;

 public String says;

 }

public class JSONOutput{

```

    public cls_animal animal;

    //public JSONOutput parse(String json){
    //return (JSONOutput) System.JSON.deserialize(json, JSONOutput.class);
    //}
}

public static String getAnimalNameById (Integer id) {
    Http http = new Http();
    HttpRequest request = new HttpRequest();
    request.setEndpoint('https://th-apex-http-callout.herokuapp.com/animals/' + id);
    //request.setHeader('id', String.valueOf(id)); -- cannot be used in this challenge :)
    request.setMethod('GET');
    HttpResponse response = http.send(request);
    system.debug('response: ' + response.getBody());
    //Map<String,Object> map_results = (Map<String,Object>)
JSON.deserializeUntyped(response.getBody());
    jsonOutput results = (jsonOutput) JSON.deserialize(response.getBody(), jsonOutput.class);
    //Object results = (Object) map_results.get('animal');
        system.debug('results= ' + results.animal.name);
    return(results.animal.name);
}

}

Mock Class;
@Test
global class AnimalLocatorMock implements HttpCalloutMock {

    global HTTPResponse respond(HTTPRequest request) {
        Httpresponse response = new Httpresponse();
        response.setStatusCode(200);
        //-- directly output the JSON, instead of creating a logic
        //response.setHeader('key, value)
        //Integer id = Integer.valueOf(request.getHeader('id'));
        //Integer id = 1;
        //List<String> lst_body = new List<String> {'majestic badger', 'fluffy bunny'};
        //system.debug('animal return value: ' + lst_body[id]);
        response.setBody('{"animal":{"id":1,"name":"chicken","eats":"chicken food","says":"cluck
cluck"}}');
        return response;
    }
}

```

```

}
Test Class;
@IsTest
public class AnimalLocatorTest {
    @isTest
    public static void testAnimalLocator() {
        Test.setMock(HttpCalloutMock.class, new AnimalLocatorMock());
        //HttpResponse response = AnimalLocator.getAnimalNameById(1);
        String s = AnimalLocator.getAnimalNameById(1);
        system.debug('string returned: ' + s);
    }
}

```

Apex Web Services;

Apex Class;

```

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

```

```

    @HttpGet
    global static account getAccount() {

        RestRequest request = RestContext.request;

        String accountId = request.requestURI.substring(request.requestURI.lastIndexOf('/')-18,
            request.requestURI.lastIndexOf('/'));
        List<Account> a = [select id, name, (select id, name from contacts) from account where id =
:accountId];
        List<contact> co = [select id, name from contact where account.id = :accountId];
        system.debug('** a[0]= ' + a[0]);
        return a[0];

    }

}

```

Apex Test Class;

```

@Istest(SeeAllData=true)
public class AccountManagerTest {

```



```

@IsTest
public static void testaccountmanager() {
    RestRequest request = new RestRequest();
    request.requestUri = 'https://mannharleen-dev-
ed.my.salesforce.com/services/apexrest/Accounts/00190000016cw4tAAA/contacts';
    request.httpMethod = 'GET';
    RestContext.request = request;

    system.debug('test account result = ' + AccountManager.getAccount());

```

```

}
Apex SOAP Callouts;
Class;

```

```

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

```

```

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

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

```

```

@isTest

```

```

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 SPECIALIST SUPERBADGE;

Challenge 1;

Maintenance Request Trigger;

```

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

```

```

    Map<Id,Case> validCaseMap = new Map<Id,Case>();

```

```

    if(Trigger.isUpdate && Trigger.isAfter){
        for(Case caseHere: Trigger.new){
            if (caseHere.IsClosed && (caseHere.Type.equals('Repair') ||
caseHere.Type.equals('Routine Maintenance'))){
                validCaseMap.put(caseHere.Id, caseHere);
            }
        }
    }

```

```

    if(!validCaseMap.values().isEmpty()){
        MaintenanceRequestHelper.createNewRequest(validCaseMap);
    }
}

```

```

}

```

Maintenance Request Helper;

```

public class MaintenanceRequestHelper {

```

```

public static void createNewRequest(Map<Id, Case> validCaseMap){
    List<Case> newCases = new List<Case>();
    Map<Id, Integer> productMaintenanceCycleMap = new Map<Id, Integer>();
    Map<Id, Integer> workPartMaintenanceCycleMap = new Map<Id, Integer>();

    for (Product2 productHere : [select Id, Maintenance_Cycle__c from Product2]) {
        if (productHere.Maintenance_Cycle__c != null) {
            productMaintenanceCycleMap.put(productHere.Id,
Integer.valueOf(productHere.Maintenance_Cycle__c));
        }
    }

    for (Work_Part__c workPart : [select Id, Equipment__c, Maintenance_Request__c from
Work_Part__c where Maintenance_Request__c in :validCaseMap.keySet()]) {
        if (workPart.Equipment__c != null) {
            if(!workPartMaintenanceCycleMap.containsKey(workPart.Maintenance_Request__c)){
                workPartMaintenanceCycleMap.put(workPart.Maintenance_Request__c,
productMaintenanceCycleMap.get(workPart.Equipment__c));
            }
            else if(productMaintenanceCycleMap.get(workPart.Equipment__c) <
workPartMaintenanceCycleMap.get(workPart.Maintenance_Request__c)){
                workPartMaintenanceCycleMap.put(workPart.Maintenance_Request__c,
productMaintenanceCycleMap.get(workPart.Equipment__c));
            }
        }
    }

    for(Case caseHere: validCaseMap.values()){
        Case newCase = new Case();
        newCase.Vehicle__c = caseHere.Vehicle__c;
        newCase.Equipment__c = caseHere.Equipment__c;
        newCase.Type = 'Routine Maintenance';
        newCase.Subject = String.isBlank(caseHere.Subject) ? 'Routine Maintenance Request' :
caseHere.Subject + ' New';
        newCase.Date_Reported__c = Date.today();
        newCase.Date_Due__c =
workPartMaintenanceCycleMap.containsKey(caseHere.Product__c) ?
Date.today().addDays(workPartMaintenanceCycleMap.get(caseHere.Product__c)) :
Date.today();
        newCase.Status = 'New';
    }
}

```

```

        newCase.Product__c = caseHere.Product__c;
        newCase.AccountId = caseHere.AccountId;
        newCase.ContactId = caseHere.ContactId;
        newCase.AssetId = caseHere.AssetId;
        newCase.Origin = caseHere.Origin;
        newCase.Reason = caseHere.Reason;

        newCases.add(newCase);
    }

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

}

```

Challenge 2;

Warehouse Callout Service;

public with sharing class WarehouseCalloutService {

```

    private static final String WAREHOUSE_URL = 'https://th-superbadge-
    apex.herokuapp.com/equipment';

```

```

    // complete this method to make the callout (using @future) to the
    // REST endpoint and update equipment on hand.

```

```

    @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);

```

```

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

```

```

            List<Object> results = (List<Object>) JSON.deserializeUntyped(response.getBody());

```

```

            List<Product2> equipmentList = new List<Product2>();

```

```

            for (Object record: results) {

```

```
Map<String, Object> recordMap = (Map<String, Object>)record;
Product2 equipment = new Product2();
```

```
equipment.Name = (String)recordMap.get('name');
equipment.Cost__c = (Decimal)recordMap.get('cost');
equipment.ProductCode = (String)recordMap.get('_id');
equipment.Current_Inventory__c = (Integer)recordMap.get('quantity');
equipment.Maintenance_Cycle__c = (Integer)recordMap.get('maintenanceperiod');
equipment.Replacement_Part__c = (Boolean)recordMap.get('replacement');
equipment.Lifespan_Months__c = (Integer)recordMap.get('lifespan');
equipment.Warehouse_SKU__c = (String)recordMap.get('sku');
```

```
equipmentList.add(equipment);
```

```
}
```

```
if(equipmentList.size() > 0){
```

```
    upsert equipmentList;
```

```
}
```

```
}
```

```
}
```

```
}
```

```
execute Anonymous;
```

```
WarehouseCalloutService.runWarehouseEquipmentSync();
```

Challenge 3;

WarehouseSyncScheduler;

```
public class WarehouseSyncSchedule implements Schedulable{
```

```
    // implement scheduled code here
```

```
    public void execute(System.SchedulableContext context){
```

```
        WarehouseCalloutService.runWarehouseEquipmentSync();
```

```
    }
```

```
}
```

```
execute anonymous;
```

```
System.schedule('WarehouseSyncScheduleTest', '0 0 1 * * ?', new WarehouseSyncSchedule());
```

Challenge 4;

Maintenance Request Test;

```

@Test
public class MaintenanceRequestTest {

    @testSetup
    static void setup(){
        Product2 prod = new Product2();
        prod.Cost__c = 50;
        prod.Name = 'Ball Valve 10 cm';
        prod.Lifespan_Months__c = 12;
        prod.Maintenance_Cycle__c = 365;
        prod.Current_Inventory__c = 50;
        prod.Replacement_Part__c = true;
        prod.Warehouse_SKU__c = '100009';
        insert prod;

        Product2 prod2 = new Product2();
        prod2.Cost__c = 50;
        prod2.Name = 'Ball Valve 10 cm';
        prod2.Lifespan_Months__c = 12;
        prod2.Maintenance_Cycle__c = 240;
        prod2.Current_Inventory__c = 50;
        prod2.Replacement_Part__c = true;
        prod2.Warehouse_SKU__c = '100009';
        insert prod2;

        List<Case> caseList = new List<Case>();
        for(Integer i=0; i<300; i++) {
            Case caseNew = new Case();
            caseNew.Subject = 'Maintenance ' + i;
            caseNew.Type = 'Other';
            caseNew.Status = 'New';
            caseNew.Equipment__c = prod.Id;
            caseNew.SuppliedName = 'Test';
            caseList.add(caseNew);

            if(i==10){
                caseNew.Subject = 'Maintenance test 10';
            }
        }

        insert caseList;
    }
}

```

```

List<Work_Part__c> workPartList = new List<Work_Part__c>();
for(Case caseHere : [select Id, Subject from Case where SuppliedName = 'Test']) {
    Work_Part__c workPart = new Work_Part__c();
    workPart.Maintenance_Request__c = caseHere.Id;
    workPart.Equipment__c = prod.Id;
    workPartList.add(workPart);

    if(caseHere.Subject == 'Maintenance test 10'){
        Work_Part__c workPart2 = new Work_Part__c();
        workPart2.Maintenance_Request__c = caseHere.Id;
        workPart2.Equipment__c = prod2.Id;
        workPartList.add(workPart2);
    }
}

insert workPartList;
}

@isTest
static void testMaintenanceRequest(){
    List<Case> caseList = new List<Case>();
    for(Case caseHere : [select Id from Case where SuppliedName = 'Test']) {
        caseHere.Type = 'Repair';
        caseHere.Status = 'Closed';
        caseList.add(caseHere);
    }

    Test.startTest();
    update caseList;
    System.assertEquals(300, [SELECT count() FROM Case WHERE Type = 'Routine
Maintenance' and Date_Reported__c = :Date.today()]);
    Test.stopTest();

}
}

Challenge 5;
Warehouse Callout Service Mock;
public class WarehouseCalloutServiceMock implements HttpCalloutMock {
    private String responseJson = '[' +

```

```
'{"_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"},"' +  
    '];
```

```
    // Implement this interface method  
    public HTTPResponse respond(HTTPRequest request) {  
        // Create a fake response  
        HttpResponse response = new HttpResponse();  
        response.setHeader('Content-Type', 'application/json');  
        response.setBody(responseJson);  
        response.setStatusCode(200);  
        return response;  
    }  
}  
Warehouse Callout Service Test;  
@isTest  
private class WarehouseCalloutServiceTest {  
  
    @isTest  
    static void testRunWarehouseEquipmentSync(){  
        Test.setMock(HttpCalloutMock.class, new WarehouseCalloutServiceMock());  
  
        Test.startTest();  
        WarehouseCalloutService.runWarehouseEquipmentSync();  
        Test.stopTest();  
  
        System.assertEquals(3, [select count() from Product2]);  
    }  
  
}
```

```
Challenge 6;  
Warehouse Sync Schedule Test;  
@isTest
```



```
public class WarehouseSyncScheduleTest {
    public static String CRON_EXP = '0 0 1 * * ?';

    @isTest
    static void testExecute(){
        Test.setMock(HttpCalloutMock.class, new WarehouseCalloutServiceMock());

        Test.startTest();
        String jobId = System.schedule('WarehouseSyncScheduleTest', CRON_EXP, new
WarehouseSyncSchedule());
        Test.stopTest();

        System.assertEquals(1, [SELECT count() FROM CronTrigger WHERE CronJobDetail.Name =
'WarehouseSyncScheduleTest']);
    }
}
```