Apex Triggers

Get Started with Apex Triggers

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

    for(Account account:Trigger.New){
        if(account.Match_Billing_Address__c == True){
            account.ShippingPostalCode = account.BillingPostalCode;

        }
    }
}
```

Bulk Apex Triggers

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

    for(Opportunity opp: Trigger.New){
        if(opp.StageName == 'Closed won'){
            tasklist.add(new Task(Subject = 'Follow Up Test Task',whatId = opp.Id));
        }
    }

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

}
```

Apex Testing

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('02/02/2019'));
        System.assertEquals(false, flag);
    }
     @isTest static void Test_DateWithin30Days_case3(){
        Boolean flag = VerifyDate.DateWithin30Days(date.parse('01/01/2020'),
date.parse('01/15/2019'));
        System.assertEquals(true, flag);
    }
    @isTest static void Test_SetEndOfMonthDate(){
        Date returndate = VerifyDate.SetEndOfMonthDate(date.parse('01/01/2020'));
    }
 }
```

Test Apex Triggers

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

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

		}



}

@isTest
public class TestRestrictContactByName {
   @isTest static void Test_insertupdateContact(){
      Contact cnt = new Contact();
      cnt.LastName = 'INVALIDNAME';

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

      System.assert(!result.isSuccess());
      System.assert(result.getErrors().size()>0);
      System.assertEquals('The Last Name "INVALIDNAME" is not allowed for DML',
result.getErrors()[0].getMessage());
   }
}
```

Create Test Data for Apex Tests

```
public class RandomContactFactory {

    public static List<Contact> generateRandomContacts(Integer numcnt , string
lastname){
        List<Contact> contacts = new List<Contact>();
        for(Integer i=0;i<numcnt;i++){
            Contact cnt = new Contact(FirstName = 'Test '+i, LastName = lastname);
            contacts.add(cnt);
        }
        return contacts;
    }

}
```

Asynchronous Apex

Use Future Methods

```
public class AccountProcessor {
    @future
    public static void countContacts(List<Id>accountIds){
        List<Account> accountsToUpdate = new List<Account>();

        List<Account> accounts = [Select Id, Name , (Select Id from Contacts) from
Account Where Id in :accountIds];

        For(Account acc:accounts){
            List<Contact> contactList = acc.Contacts;
            acc.Number_Of_Contacts__c = contactList.size();
            accountsToUpdate.add(acc);

        }
        update accountsToUpdate;
    }

}

@isTest
```

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

        Contact newContact1 = new Contact(FirstName='Jhon',LastName =
'Deo',AccountId=newAccount.Id);
        insert newContact1;

        Contact newContact2 = new Contact(FirstName='Jane',LastName =
'Deo',AccountId=newAccount.Id);
        insert newContact2;

        List<Id>accountIds = new List<Id>();
        accountids.add(newAccount.Id);

        Test.startTest();
        Accountprocessor.countContacts(accountIds);
        Test.stopTest();

    }

}
```

Use Batch Apex

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

    global Database.QueryLocator start(Database.BatchableContext bc){
        return Database.getQueryLocator('SELECT ID ,LeadSource FROM Lead');

    }
    global void execute (Database.BatchableContext bc,List<Lead> L_list){
        List<lead> L_list_new= new List<lead>();
        for(lead L:L_list){
            L.leadsource='Dreamforce';
            L_list_new.add(L);
            count+=1;
        }
        update L_list_new;
    }
    global void finish(Database.BatchableContext bc){
        system.debug('count='+count);
    }

}

@isTest
public class LeadProcessorTest {

    @isTest
    public static void testit(){
        List<lead> L_list = new List<lead>();

        for(Integer i=0;i<200;i++){
            Lead L = new lead();
            L.LastName = 'name' +1;
            L.Company = 'Company';
            L.Status = 'Random Status';
            L_list.add(L);

        }
        insert L_list;

        Test.startTest();
        LeadProcessor lp = new LeadProcessor();
        Id batchId = Database.executeBatch(lp);
        Test.stopTest();

    }

}
```

Control Processes with Queueable Apex

```
public class AddPrimaryContact implements Queueable{

    private Contact con;
    private String state;

    public  AddPrimaryContact(Contact con,String state){
        this.con=con;
        this.state=state;
    }

    public void execute(QueueableContext context){
        List<Account> accounts=[Select Id , Name , (Select FirstName , LastName ,Id from
contacts)
                                from Account where BillingState = :state Limit 200];
        List<Contact> primaryContacts = new List<Contact>();

        for(Account acc:accounts){
            Contact c = con.clone();
            c.AccountId = acc.Id;
            primaryContacts.add(c);
        }

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

    }

}


@isTest
public class AddPrimaryContactTest {
    static testmethod void testQueueable(){
        List<Account> testAccounts = new List<Account>();
        for(Integer i=0;i<50;i++){
            testAccounts.add(new Account(Name='Account '+i,BillingState='CA'));
        }
        for(Integer j=0;j<50;j++){
            testAccounts.add(new Account(Name ='Account '+j,BillingState='NY'));
        }
        insert testAccounts;

        Contact testContact = new Contact(FirstName ='John',LastName='Doe');
        insert testContact;

        AddPrimaryContact addit = new addPrimaryContact(testContact, 'CA');

        Test.startTest();
        system.enqueueJob(addit);
        Test.stopTest();

        System.assertEquals(50,[Select count() from Contact where accountId in (Select Id
from Account where BillingState='CA')]);
    }
}
```

Schedule Jobs Using the Apex Scheduler

```
global class DailyLeadProcessor implements Schedulable {
    global void execute(SchedulableContext ctx){
        List<lead> leadstoupdate =  new List<lead>();
        List<Lead> leads=[Select id From Lead where LeadSource=NULL Limit 200];

        for(Lead l:leads){
            l.LeadSource= 'Dreamforce';
            leadstoupdate.add(l);
        }
        update leadstoupdate;
    }

}

@isTest
private class DailyLeadProcessorTest {

    public static String CRON_EXP = '0 0 0 15 3 7  2022';
    static testmethod void testScheduledJob(){
        List<Lead> leads = new List<lead>();
        for(Integer i=0;i<200;i++){
            Lead l =new Lead(
                FirstName = 'First ' +i,
                LastName ='LastName',
                Company ='The Inc'
            );
            leads.add(l);
        }
        insert leads;

        Test.startTest();

        String jobId = System.schedule('ScheduledApexTest',CRON_EXP,new
DailyLeadProcessor());
        Test.stopTest();

        List<Lead> checkleads = new List<Lead>();
        checkleads = [Select Id From Lead Where LeadSource = 'Dreamforce' and Company
= 'The Inc'];

        System.assertEquals(200,checkleads.size(), 'Leads were not created');
    }

}
```

Apex Integration Services

Apex REST Callouts

```
public class AnimalLocator{
   public static String getAnimalNameById(Integer x){
       Http http = new Http();
       HttpRequest req = new HttpRequest();
       req.setEndpoint('https://th-apex-http-callout.herokuapp.com/animals/' + x);
       req.setMethod('GET');
       Map<String, Object> animal= new Map<String, Object>();
       HttpResponse res = http.send(req);
           if (res.getStatusCode() == 200) {
       Map<String, Object> results = (Map<String,
Object>)JSON.deserializeUntyped(res.getBody());
     animal = (Map<String, Object>) results.get('animal');
       }
return (String)animal.get('name');
   }
}

@isTest
private class AnimalLocatorTest{
   @isTest static void AnimalLocatorMock1() {
       Test.setMock(HttpCalloutMock.class, new AnimalLocatorMock());
       string result = AnimalLocator.getAnimalNameById(3);
       String expectedResult = 'chicken';
       System.assertEquals(result,expectedResult );
   }
}

@isTest
global class AnimalLocatorMock implements HttpCalloutMock {
    // Implement this interface method
   global HTTPResponse respond(HTTPRequest request) {
       // Create a fake response
       HttpResponse response = new HttpResponse();
       response.setHeader('Content-Type', 'application/json');
```

```
        response.setBody('{"animals": ["majestic badger", "fluffy bunny", "scary bear",
"chicken", "mighty moose"]}');
        response.setStatusCode(200);
        return response;
    }
}
```

Apex SOAP Callouts

```
public class ParkLocator {
    public static string[] country(string theCountry) {
        ParkService.ParksImplPort  parkSvc = new  ParkService.ParksImplPort(); // remove
space
        return parkSvc.byCountry(theCountry);
    }
}

@isTest
private class ParkLocatorTest {
    @isTest static void testCallout() {
        Test.setMock(WebServiceMock.class, new ParkServiceMock ());
        String country = 'United States';
        List<String> result = ParkLocator.country(country);
        List<String> parks = new List<String>{'Yellowstone', 'Mackinac National Park',
'Yosemite'};
         System.assertEquals(parks, result);
    }
}

@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) {
        // start - specify the response you want to send
        ParkService.byCountryResponse response_x = new
ParkService.byCountryResponse();
        response_x.return_x = new List<String>{'Yellowstone', 'Mackinac National Park',
'Yosemite'};
        // end
        response.put('response_x', response_x);
   }
}
```

Apex Web Services

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

    @HttpGet
    global static Account getAccount(){
        RestRequest request = RestContext.request;
        string accountId = request.requestURI.substringBetween('Accounts/','/contacts');
        Account result = [SELECT Id, Name, (Select Id, Name from Contacts) from Account
where Id=:accountId Limit 1];
        return result;
    }
}

@IsTest
private class AccountManagerTest {
    @isTest static void testGetContactsByAccountId(){
        Id recordId = createTestRecord();
        RestRequest request = new RestRequest();
        request.requestUri =
'https://yourInstance.my.salesforce.com/services/apexrest/Accounts/'
                                                + recordId+'/contacts';
        request.httpMethod = 'GET';
        RestContext.request = request;
        Account thisAccount = AccountManager.getAccount();
        System.assert(thisAccount != null);
        System.assertEquals('Test record', thisAccount.Name);
    }

    static Id createTestRecord(){
        Account accountTest = new Account(
                        Name ='Test record');
        insert accountTest;

        Contact contactTest = new Contact(
                        FirstName='John',
                        LastName = 'Doe',
                        AccountId = accountTest.Id
        );
        insert contactTest;

        return accountTest.Id;
    }
}
```

Apex Specialist

```
trigger MaintenanceRequest on Case (before update, after update) {
    //ToDo: Call MaintenanceRequestHelper.updateWorkOrders
    if(trigger.isAfter){
        MaintenanceRequestHelper.updateWorkOrders();
    }
}
```

```
public with sharing class MaintenanceRequestHelper {
    public static void updateWorkOrders() {
        List<case> newCaseList = new List<case>();
        Integer avgAmount=10000;

        List<Equipment_Maintenance_Item__c> newEMI = new
List<Equipment_Maintenance_Item__c>();
        List<case> caseList = [SELECT id,Vehicle__c,Subject,ProductID,Product__c, (SELECT
id from Equipment_Maintenance_Items__r) from case where status='closed' and Type
IN ('Repair', 'Routine Maintenance') and ID IN :Trigger.new LIMIT 200];
        Map<id,Equipment_Maintenance_Item__c> equip = new
map<id,Equipment_Maintenance_Item__c>([Select ID, Equipment__c,
Quantity__c,Equipment__r.id,Equipment__r.Maintenance_Cycle__c from
Equipment_Maintenance_Item__c ]);
        for(case c: caseList){
            case newCase = new Case();
            newCase.Type = 'Routine Maintenance';
            newCase.Status = 'New';
            newCase.Vehicle__c = c.Vehicle__c;
            newCase.Subject =  String.isBlank(c.Subject) ? 'Routine Maintenance Request' :
c.Subject;
            newCase.Date_Reported__c = Date.today();
            newCase.ProductId = c.ProductId;
            newCase.Product__c = c.Product__c;
            newCase.parentID = c.Id;


            for(Equipment_Maintenance_Item__c emi : c.Equipment_Maintenance_Items__r ){
                avgAmount =
Math.min(avgAmount,Integer.valueOf(equip.get(emi.id).Equipment__r.Maintenance_Cyc
le__c));
                newEMI.add(new Equipment_Maintenance_Item__c(
                    Equipment__c = equip.get(emi.id).Equipment__c,
                    Maintenance_Request__c = c.id,
                    Quantity__c = equip.get(emi.id).Quantity__c));
            }
            Date dueDate = date.TODAY().adddays(avgAmount);
            newCase.Date_Due__c =dueDate;
            newCaseList.add(newCase);

        }
        if(newCaseList.size()>0){
            Database.insert(newCaseList);
        }

        for(Case c2: newCaseList){
            for(Equipment_Maintenance_Item__c emi2 : newEmi){
                if(c2.parentID == emi2.Maintenance_Request__c){
                    emi2.Maintenance_Request__c = c2.id;
                }
            }
        }

        if(newEmi.size()>0){
            Database.insert(newEmi);
        }
    }
}
```