Apex Triggers

1)Get Started with Apex Triggers

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

2)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

1)Get Started with Apex Unit Tests

```
@isTest
public class TestVerifyDate {
```

```
    //date within 30 days
    @isTest static void case1(){
        Date D1 = VerifyDate.CheckDates(date.parse('03-18-2022'),date.parse('03-22-
2022'));
        // comparing the dates
        System.assertEquals(date.parse('03-22-2022'), D1);  // (expected, actual)
    }

    //date not within 30 days
    @isTest static void case2(){
        Date D2 = VerifyDate.CheckDates(date.parse('03-18-2022'),date.parse('06-22-
2022'));
        // comparing the dates
        System.assertEquals(date.parse('06-22-2022'), D2);  // (expected, actual)
    }
}
```

2)Test Apex Triggers

```
@isTest
public class TestRestrictContactByName {

    @isTest
    public static void testContact(){
        Contact ct = new Contact();
        ct.LastName = 'INVALIDNAME';
        Database.SaveResult res = Database.insert(ct,false);
        System.assertEquals('The Last Name "INVALIDNAME" is not allowed for DML',
res.getErrors()[0].getMessage());
    }

}
```

3)Create Test Data for Apex Tests

```
public class RandomContactFactory {
    public static List<Contact> generateRandomContacts(Integer num,String lastName){
      List<Contact>  contactList = new List<Contact>();
```

```
        for (Integer i=1;i<=num ; i++){
            contact ct = new contact(FirstName = 'Test'+i, LastName = lastName );
            contactList.add(ct);
        }
        return contactList;
    }

}
```

Asynchronous Apex

1)Use Future Methods

```
public without sharing class AccountProcessor {
    @future
    public static void countContacts(List<Id> accountIds){
        List<Account> accounts =[SELECT Id, (SELECT Id FROM Contacts) FROM Account
WHERE Id IN :accountIds];

        for(Account acc: accounts){
            acc.Number_Of_Contacts__c = acc.Contacts.size();
        }
        update accounts;
    }

}


@isTest
private class AccountProcessorTest {
    @isTest
    private static void countContactsTest(){

        List<Account> accounts= new List<Account>();
        for(Integer i=0; i<300 ; i++){
            accounts.add(new Account(Name ='Test Account'+i));
        }
```

```
        insert accounts;

        List<Contact> contacts = new List<Contact>();
        List<Id> accountIds = new List<Id>();
        for(Account acc:accounts){
            contacts.add(new
Contact(FirstName=acc.Name,LastName='TestContact',AccountId=acc.Id));
            accountIds.add(acc.id);
        }
        insert contacts;

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

    }

}
```

2)Use Batch Apex

```
public without sharing class LeadProcessor implements Database.Batchable<sObject>{

    public Database.QueryLocator start(Database.BatchableContext dbc){
        return Database.getQueryLocator([SELECT Id,Name FROM Lead]);

    }

    public void execute(Database.BatchableContext dbc , List<Lead> leads){
        for(Lead l : leads){
            l.LeadSource = 'Dreamforce';
        }
        update leads;
    }

    public void finish(Database.BatchableContext dbc){
        System.debug('Done');
    }
```

```
}

@isTest
private class LeadProcessorTest {

   @isTest
   private static void testBatchClass(){

      List<Lead> leads = new List<Lead>();
      for(Integer i=0; i<200 ;i++){
         leads.add(new Lead(LastName ='Connak',Company ='Salesfrce'));
      }
     insert leads;

      Test.startTest();
      LeadProcessor lp = new LeadProcessor();
      Id batchId = Database.executeBatch(lp, 200);
      Test.stopTest();

      List<Lead> updatedLeads =[SELECT Id FROM Lead WHERE LeadSource
='Dreamforce'];
      System.assertEquals(200, updatedLeads.size(),'ERROR: At least 1 lead record not
updated correctly');

   }


}
```

3)Control Processes with Queueable Apex

```
public without sharing class AddPrimaryContact implements Queueable {

   private Contact contact;
   private String state;

   public AddPrimaryContact (Contact inputContact , String inputState){
```

```
        this.contact = inputContact;
        this.state = inputState;
    }

    public void execute (QueueableContext context){
        //retrive 200 Account records
        List<Account> accounts = [SELECT Id FROM Account WHERE BillingState = :state
LIMIT 200];

        //create empty list of contact records
        List<Contact> contacts = new List<Contact>();

        //Iterate through acc record
        for( Account acc : accounts){

            //copy con record , make thet copy a child of specific acc rec
            //& add to list of contacts
            Contact contactClone = contact.clone();
            contactClone.AccountId=acc.Id;
            contacts.add(contactClone);
        }
        insert contacts;
    }

}

@isTest
private class AddPrimaryContactTest {

    @isTest
    private static void testQueueableClass(){

        //load test data
        List<Account> accounts = new List<Account>();
        for(Integer i =0; i<500; i++){
            Account acc =new Account(Name  ='Tect account');
            if (i<250){
                acc.BillingState = 'NY';
```

```
        }else{
            acc.BillingState = 'CA';
        }
        accounts.add(acc);
    }
    insert accounts;

    Contact contact  = new Contact(FirstName='Simon',LastName='Connock');
    insert contact;

    //Perform the test
    Test.startTest();
    Id jobId =System.enqueueJob(new AddPrimaryContact(contact,'CA'));
    Test.stopTest();

    //check result
    List<Contact> contacts =[SELECT Id FROM Contact WHERE
Contact.Account.BillingState ='CA'];
    System.assertEquals(200,contacts.size(),'ERROR: Incorrect no of contact records
found');
  }

}
```

4) Schedule Jobs Using the Apex Scheduler

```
public without sharing class DailyLeadProcessor implements Schedulable {

  public void execute(SchedulableContext  ctx){
    //Get 200 Lead records & modify the leadsource field
    List<Lead> leads =[SELECT Id,LeadSource FROM Lead WHERE LeadSource = null
LIMIT 200];
    for(Lead l : leads){
      l.LeadSource = 'DreamForce';
    }

    //update modified rec
```

```
        update leads;
    }

}

@isTest
public class DailyLeadProcessorTest {

    private static String CRON_EXP ='0 0 0 ? * * *';//midnight every day

    @isTest
    private static void testSchedulableClass(){

        //Load test data
        List<Lead> leads = new List<Lead>();
        for(Integer i=0;i<500;i++){
            if(i<250){
                leads.add(new Lead(LastName='Connock',Company='Salesforce'));
            }else{
                 leads.add(new
Lead(LastName='Connock',Company='Salesforce',LeadSource='Other'));
            }
        }
        insert leads;

        //perform test
        Test.startTest();
        String jobId = System.schedule('Process Leads',CRON_EXP,new
DailyLeadProcessor());
        Test.stopTest();

        //check result
        List<Lead> updatedLeads =[SELECT Id,LeadSource FROM Lead WHERE
LeadSource ='Dreamforce'];
        System.assertEquals(200,updatedLeads.size(),'ERROR: At least 1 record not
updated correctly');

        //check the sheduled time
```

```
      List<CronTrigger> cts=[SELECT Id,TimesTriggered, NextFireTime FROM CronTrigger
WHERE Id = :jobId];
      System.debug('Next fire time'+ cts[0].NextFireTime);
   }

}
```

Apex Integration Services

1)Apex REST Callouts

```
public class AnimalLocator {

   public static String getAnimalNameById (Integer i) {
      Http http = new Http();
      HttpRequest request = new HttpRequest();
      request.setEndpoint('https://th-apex-http-callout.herokuapp.com/animals/'+i);
      request.setMethod('GET');
      HttpResponse response = http.send(request);

      //if the request is successful, parse the 350N response.


       Map<String, Object> result =(Map<String,
Object>)JSON.deserializeUntyped(response.getBody());
       Map<String, Object> animal =(Map<String, Object>)result.get('animal');
       System.debug('name: '+String.valueOf(animal.get('name')));
       return String.valueOf(animal.get('name'));


   }

}

@isTest
private class AnimalLocatorTest {

   @isTest
```

```
    static void animalLocatorTest1(){
        Test.setMock(HttpCalloutMock.class,new AnimalLocatorMock());
        String actual = AnimalLocator.getAnimalNameById(1);
        String expected = 'moose';
        System.assertEquals(actual, expected);
    }

}

@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('ContentType', 'application/json');
        response.setBody('{"animal": {"id":1, "name":"moose",
"eats":"plants","says":"bellows"}}');
        response.setStatusCode(200);
        return response;
    }
}
```

2)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/',
'ParksServices'};
      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 List<String> country(String country){
        ParkService.ParksImplPort prkSvc = new ParkService.ParksImplPort();
        return prkSvc.byCountry(country);
    }

}

@isTest
private class ParkLocatorTest {

    @isTest
    static void testCallout(){
        Test.setMock(WebServiceMock.class, new ParkServiceMock());
        String country = 'United States';
        System.assertEquals(new List<String>{'Yosemita','Sequioa','Crater
Lake'},ParkLocator.country(country));
    }

}

@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>{'Yosemita','Sequioa','Crater Lake'};
        // end
        response.put('response_x', response_x);
    }

}
```

3)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];
        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+'/co
```

```
ntacts';
        request.httpMethod = 'GET';
        RestContext.request = request;

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

        // Verify results
        System.assert(thisAcc != null);
        System.assertEquals('Test record', thisAcc.Name);
    }

    //Helper class
    static Id createTestRecord(){
        //creating record
        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;

    }
}
```