APEX TRIGGERS

<u>*GET STARTED WITH APEX TRIGGERS</u>**:**

AccountAddressTrigger.apxt

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

<u>*BULK APEX TRIGGERS:</u>

ClosedOpportunityTrigger.apxt

```
trigger ClosedOpportunityTrigger on Opportunity (before insert,after insert) {
          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

<u>*GET STARTED WITH APEX UNIT TEST:</u>

1.VerifyDate.apxc

```
public class VerifyDate {

          //method to handle potential checks against two dates
          public static Date CheckDates(Date date1, Date date2) {
```

```
			//if date2 is within the next 30 days of date1, use date2.  Otherwise use the end
of the month
			if(DateWithin30Days(date1,date2)) {
				return date2;
			} else {
				return SetEndOfMonthDate(date1);
			}
		}

		//method to check if date2 is within the next 30 days of date1
		@TestVisible private static Boolean DateWithin30Days(Date date1, Date date2) {
			//check for date2 being in the past
		if( date2 < date1) { return false; }

		//check that date2 is within (>=) 30 days of date1
		Date date30Days = date1.addDays(30); //create a date 30 days away from date1
			if( date2 >= date30Days ) { return false; }
			else { return true; }
		}

		//method to return the end of the month of a given date
		@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;
		}

}
```

2.TestVerifyDate.apxc

```
isTest
public class TestVerifyDate {

   @isTest static void Test_CheckDates_case1(){
      Date D = VerifyDate.CheckDates(date.parse('01/01/2022'),date.parse('01/05/2022'));
      System.assertEquals(date.parse('01/05/2022'), D);
   }

   @isTest static void Test_CheckDates_case2(){
      Date D = VerifyDate.CheckDates(date.parse('01/01/2022'),date.parse('05/05/2022'));
      System.assertEquals(date.parse('01/31/2022'), D);
   }

   @isTest static void Test_DateWithin30Days_case1(){
      Boolean flag = VerifyDate.DateWithin30Days(date.parse('01/01/2022'),
date.parse('12/30/2021'));
      System.assertEquals(false, flag);

   }
   @isTest static void Test_DateWithin30Days_case2(){
      Boolean flag = VerifyDate.DateWithin30Days(date.parse('01/01/2022'),
date.parse('02/02/2021'));
      System.assertEquals(false, flag);

   }
   @isTest static void Test_DateWithin30Days_case3(){
      Boolean flag = VerifyDate.DateWithin30Days(date.parse('01/01/2022'),
date.parse('01/15/2022'));
      System.assertEquals(true, flag);
   }
   @isTest static void Test_SetEndOfMonth(){
      Date returndate = VerifyDate.SetEndOfMonthDate(date.parse('01/01/2022'));
   }
}
```

<u>*TEST APEX TRIGGERS:</u>

1.RestrictContactByName.apxt

```
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');
                    }

          }
}
```

CREATE TEST DATA FOR APEX TESTS:

1.RandomContactFactory.apxc

```
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:

1.AccountProcessor.apxc

```
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;
   }
}
```

2.AccountProcessorTest.apxc

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

1.LeadProcessor.apxc

```
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);
    }
}
```

2.LeadProcessorTest.apxc

```
@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:

1.AddPrimaryContact.apxc

```
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;
        }
    }
}
```

2.AddPrimaryContactTest.apxc

```
@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 APEX SCHEDULER:

1.DailyLeadProcessor.apxc

```
global class DailyLeadProcessor implements Schedulable {
 global void execute(SchedulableContext ctx) {
        List<Lead> lList = [Select Id, LeadSource from Lead where LeadSource = null];

        if(!lList.isEmpty()) {
   for(Lead l: lList) {
    l.LeadSource = 'Dreamforce';
   }
   update lList;
  }
    }

}
```

2.DailyLeadProcessorTest.apxc

```
@isTest
public class DailyLeadProcessorTest {
//Seconds Minutes Hours Day_of_month Month Day_of_week optional_year
    public static String CRON_EXP = '0 0 0 2 4 ? 2023';

    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

<u>*APEX REST CALLOUTS:</u>

1.AnimalLocator.apxc

```
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');
    }
}
```

2.AnimalLocatorMock.apxc

```
@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;
    }
}
```

3.AnimalLocatorTest.apxc

```
@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 );
    }
}
```

APEX SOAP CALLOUTS:

1.ParkService.apxc

```
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;
        }
    }
}
```

2.ParkLocatorTest.apxc

```
@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:

1.AccountManager.apxc

```
@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;
   }
}
```

2.AccountManagerTest.apxc

```
@isTest
private class AccountManagerTest {

   private static testMethod void getAccountTest1() {
      Id recordId = createTestRecord();
      // Set up a test request
      RestRequest request = new RestRequest();
      request.requestUri = 'https://na1.salesforce.com/services/apexrest/Accounts/'+
 recordId +'/contacts' ;
      request.httpMethod = 'GET';
```

```
        RestContext.request = request;
        // Call the method to test
        Account thisAccount = AccountManager.getAccount();
        // Verify results
        System.assert(thisAccount != null);
        System.assertEquals('Test record', thisAccount.Name);

    }

    // Helper method
        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;
    }
}
```