

Kushal Gaikwad

Apex Modules Code:

APEX TRIGGERS

Get Started with Apex Triggers

“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;
    }
```

```
}
```

Bulk Apex Triggers

“ClosedOpportunityTrigger.apxt”

```
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

“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
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
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;
}
}

```

"TestVerifyDate.apxc"

```

@Test
public class TestVerifyDate
{
    static testMethod void testMethod1()

```

```

    {
        Date d = VerifyDate.CheckDates(System.today(),System.today()+1);
        Date d1 = VerifyDate.CheckDates(System.today(),System.today()+60);
    }
}
Test Apex Triggers
"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");
        }

    }

}

}

```

"TestRestrictContactByName.apxc"

```

@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 Test
"RandomContactFactory.apxc"

//@isTest
public class RandomContactFactory {
    public static List<Contact> generateRandomContacts(Integer
numContactsToGenerate, String FName) {
        List<Contact> contactList = new List<Contact>();

        for(Integer i=0;i<numContactsToGenerate;i++) {
            Contact c = new Contact(FirstName=FName + ' ' + i, LastName = 'Contact ' +i);
            contactList.add(c);
            System.debug(c);
        }
        //insert contactList;
        System.debug(contactList.size());
        return contactList;
    }
}

ASYNCHRONOUS APEX
Use Future Method
"AccountProcessor.apxc"

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;
}

}
"AccountProcessorTest"

```

```

@Test
public class AccountProcessorTest {

    @Test
    public static void testFunc() {
        account acc = new account();
        acc.name = 'MATW INC';
        insert acc;

        contact con = new contact();
        con.lastname = 'Mann1';
    }
}

```

```

con.AccountId = acc.Id;
insert con;
contact con1 = new contact();
con1.lastname = 'Mann2';
con1.AccountId = acc.Id;
insert con1;

```

```

List<Id> acc_list = new List<Id>();
acc_list.add(acc.Id);
Test.startTest();
    AccountProcessor.countContacts(acc_list);
Test.stopTest();
List<account> acc1 = new List<account>([select Number_of_Contacts__c from
account where id = :acc.id]);
    system.assertEquals(2,acc1[0].Number_of_Contacts__c);
}

```

```

}

```

Use Batch Apex

“LeadProcessor.apxc”

global class LeadProcessor implements

Database.Batchable<sObject>, Database.Stateful {

```

// instance member to retain state across transactions

```

```

global Integer recordsProcessed = 0;

```

```

global Database.QueryLocator start(Database.BatchableContext bc) {

```

```

    return Database.getQueryLocator('SELECT Id, LeadSource FROM Lead');

```

```

}

```

```

global void execute(Database.BatchableContext bc, List<Lead> scope){

```

```

    // process each batch of records

```

```

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

```

```

    for (Lead lead : scope) {

```

```

        lead.LeadSource = 'Dreamforce';

```

```

        // increment the instance member counter
        recordsProcessed = recordsProcessed + 1;

    }
    update leads;
}

global void finish(Database.BatchableContext bc){
    System.debug(recordsProcessed + ' records processed. Shazam!');

}
}
"LeadProcessorTest.apxc"

@isTest
public class LeadProcessorTest {
    @testSetup
    static void setup() {
        List<Lead> leads = new List<Lead>();
        // insert 200 leads
        for (Integer i=0;i<200;i++) {
            leads.add(new Lead(LastName='Lead '+i,
                Company='Lead', Status='Open - Not Contacted'));
        }
        insert leads;
    }

    static testmethod void test() {
        Test.startTest();
        LeadProcessor lp = new LeadProcessor();
        Id batchId = Database.executeBatch(lp, 200);
        Test.stopTest();

        // after the testing stops, assert records were updated properly
        System.assertEquals(200, [select count() from lead where LeadSource =
'Dreamforce']);
    }
}

```

“AddPrimaryContact.apxc”

```
public class AddPrimaryContact implements Queueable{
    Contact con;
    String state;

    public AddPrimaryContact(Contact con, String state){
        this.con = con;
        this.state = state;
    }
    public void execute(QueueableContext qc){
        List<Account> lstOfAccs = [SELECT Id FROM Account WHERE BillingState = :state
LIMIT 200];

        List<Contact> lstOfConts = new List<Contact>();
        for(Account acc : lstOfAccs){
            Contact conInst = con.clone(false,false,false,false);
            conInst.AccountId = acc.Id;

            lstOfConts.add(conInst);
        }

        INSERT lstOfConts;
    }
}
```

“AddPrimaryContactTest.apxc”

```
@isTest
public class AddPrimaryContactTest{
    @testSetup
    static void setup(){
        List<Account> lstOfAcc = new List<Account>();
        for(Integer i = 1; i <= 100; i++){
            if(i <= 50)
                lstOfAcc.add(new Account(name='AC'+i, BillingState = 'NY'));
            else
                lstOfAcc.add(new Account(name='AC'+i, BillingState = 'CA'));
        }
    }
}
```



```
    INSERT IstOfAcc;  
}
```

```
static testmethod void testAddPrimaryContact(){  
    Contact con = new Contact(LastName = 'TestCont');  
    AddPrimaryContact addPCIns = new AddPrimaryContact(CON,'CA');  
  
    Test.startTest();  
    System.enqueueJob(addPCIns);  
    Test.stopTest();  
  
    System.assertEquals(50, [select count() from Contact]);  
}
```

```
}
```

Schedule Jobs Using Apex Scheduler

“DailyLeadProcessor.apxc”

```
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;  
        }  
    }  
}
```

“DailyLeadProcessorTest.apxc”

```
@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();
}
}

```