GET STARTED WITH APEX TRIGGERS

```
trigger AccountAddressTrigger on Account (before insert) {
   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;
   }


}
```

GET STARTED WITH APEX UNIT TESTS

VerifyDate:

```
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;
        }

}
```

TestVerifyDate:

```
@isTest
public class TestVerifyDate {
  @isTest static void Test_CheckDates_case1(){
    Date d = VerifyDate.CheckDates(Date.parse('01/01/2020'), Date.parse('01/03/2020'));
    System.assertEquals(Date.parse('01/03/2020'),d);
  }
  @isTest static void Test_CheckDates_case2(){
    Date d = VerifyDate.CheckDates(Date.parse('01/01/2020'), Date.parse('03/03/2020'));
    System.assertEquals(Date.parse('01/31/2020'),d);
  }
}
```

TEST APEX TRIGGERS

RestrictContactByName:

```
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:

```
@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

RandomContactFactory:

```
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;
    }

}
```

USE FUTURE METHODS

AccountProcessor:

```
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;

    }

}
```

AccountProcessorTest:

```
@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

LeadProcessor:

```
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);
   }

}
```

LeadProcessorTest:

```
@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 PROCESSOR WITH QUEUEABLE APEX

AddPrimaryContact:

```
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;

      }
    }
}
```

AddPrimaryContactTest:

```
@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

DailyLeadProcessor:

```
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;
}
}
}
```

DailyLeadProcessorTest:

```
@isTest
private class DailyLeadProcessorTest {
            static testMethod void testDailyLeadProcessor() {
            String CRON_EXP = '0 0 1 * * ?';
            List<Lead> lList = new List<Lead>();
               for (Integer i = 0; i < 200; i++) {
                      lList.add(new Lead(LastName='Dreamforce'+i, Company='Test1 Inc.',
Status='Open - Not Contacted'));
            }
            insert lList;

            Test.startTest();
            String jobId = System.schedule('DailyLeadProcessor', CRON_EXP, new
DailyLeadProcessor());
            }
}
```