Apex Trigger

AccountAddressTrigger.apxt

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

ClosedOpportunityTrigger.apxt

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

   List<Task> task = new List<Task>();

        for(Opportunity op : Trigger.New)
   {

      if (op.StageName == 'Closed Won')
      {


         task.add(new Task(whatid = op.Id, subject = 'Follow Up Test Task'));

      }

   }
   insert task;
}
```

Apex Testing

VerifyDate.apxt

```
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.apxt

```
@isTest
private class TestVerifyDate {
        @isTest static void testCheckDates1() {
      date myDate1 = date.newInstance(1990, 10, 21);
      date myDate2 = date.newInstance(1990, 11, 21);

      System.debug(VerifyDate.CheckDates(myDate1, myDate2));

   }

   @isTest static void testCheckDates2() {
      date myDate1 = date.newInstance(1990, 10, 21);
      date myDate2 = date.newInstance(1990, 9, 21);
      date checkdate = date.newInstance(1990, 10, 31);

      System.assertEquals(checkDate,VerifyDate.CheckDates(myDate1, myDate2));
   }

   @isTest static void testCheckDates3() {
      date myDate1 = date.newInstance(1990, 10, 21);
      date myDate2 = date.newInstance(1990, 10, 21);
      date checkdate = date.newInstance(1990, 10, 31);

      System.debug(VerifyDate.CheckDates(myDate1, myDate2));
   }

}
```

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.apxt

```
@isTest
public class TestRestrictContactByName
{
   @isTest static void testContactname1()
   {
      Contact c = new Contact(LastName = 'INVALIDNAME');

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

     // Database.InsertResult result = Database.insert(c, false);

         //Insert c;
      Test.stopTest();

       System.assert(!result.isSuccess());
      System.assert(result.getErrors().size() > 0);
   }
}
```

RandomContactFactory.apxt

```
public class RandomContactFactory
{
        public static List<Contact> generateRandomContacts(Integer n, String
clastName)
   {
      List<Contact> c = new List<Contact>();

      for(Integer i=0;i<n;i++) {
         Contact name = new Contact(FirstName='Test ' + i + ' '+ clastName);
         //String output = name + ' '+ clastName;
         c.add(name);
```

```
        }

        return c;
    }
}
```

Asynchronous Apex

AccountProcessor.apxt

```
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;
}
}
```

AccountProcessorTest.apxt

```
@isTest
private class AccountProcessorTest {
        @IsTest
  private static void AccountProcessorTest()
  {
     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();

    List<Account> accs = [Select Id, Number_Of_Contacts__c from account];
    for (Account acc : accs)
    {
       System.assertEquals(1, acc.Number_Of_Contacts__c, 'error');
    }
 }
}
```

LeadProcessor.apxt

```
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');
    }
}
```

LeadProcessorTest.apxt

```
@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= 'Connock', company = 'Salesforce'));
        }
        insert leads;

        Test.startTest();

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

```
        List<Lead> updateLeads = [select Id from Lead where LeadSource = 'Dreamforce'];
        System.assertEquals(200, updateLeads.size(), 'Error');
    }
}
```

AddPrimaryContact.apxt

```
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)
    {
        List<Account> accounts = [Select Id from account where BillingState = :state Limit
200];

        List<Contact> contacts = new List<Contact>();


        for (Account acc : accounts)
        {
            Contact contactClone = contact.clone();
            contactClone.AccountId = acc.Id;
            contacts.add(contactClone);
        }

        insert contacts;
    }

}
```

AddPrimaryContactTest.apxt

```
@isTest
public class AddPrimaryContactTest {
        @isTest
   private static void testQueueableClass()
   {
      List<Account> accounts = new List<Account>();
      for(Integer i=0 ; i<500; i++)
      {
         Account acc = new Account (Name = 'Test 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;

      Test.startTest();

      Id jobId = System.enqueueJob(new AddPrimaryContact(contact, 'CA'));

      Test.stopTest();

      List<Contact> contacts = [Select Id from contact where
contact.account.Billingstate = 'CA'];
      System.assertEquals(200, contacts.size(), 'Error');
```

```
    }
}
```

DailyLeadProcessor.apxt

```
public class DailyLeadProcessor implements Schedulable {
    public void execute(SchedulableContext ctx) {

        List<Lead> leads = [Select Id, leadsource from lead where leadsource = null limit
200];
        for(lead l : leads)
        {
            l.leadsource = 'Dreamforce';
        }

        update leads;
    }

}
```

DailyLeadProcessorTest.apxt

```
@isTest
public class DailyLeadProcessorTest {
    private static String CRON_EXP = '0 0 0 ? * * *';

@isTest
    private static void testSchedulableClass()
    {
        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;

        Test.startTest();

        String jobId = System.schedule('process leads', CRON_EXP, new
DailyLeadProcessor());

        Test.stopTest();

        List<lead> updatedLeads = [select Id, leadsource from lead where leadsource =
'Dreamforce'];
        System.assertEquals(200, updatedLeads.size(), 'error');

        List<CronTrigger> cts = [Select id, TimesTriggered, nextFiretime from crontrigger
where id = : jobid];
        System.debug('Next Fire time ' + cts[0].NextFireTime);
    }
}
```

Apex Integration Services