1.Account Address Trigger

```
trigger AccountAddressTrigger on Account (before insert,before update ) {

    for(Account account:Trigger.New){
        if(account.Match_Billing_Address__c == True){
            account.ShippingPostalCode = account.BillingPostalCode;

        }
    }
}
```

2.closed opportunity Trigger

```
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;
    }
}
```

3.Get started with apex unit test

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;
	}

}
```

TestVerifyDate.apxc

```
@isTest
private class TestVerifyDate {

   @isTest static void Test_CheckDates_case1(){
      Date D = VerifyDate.CheckDates(date.parse('01/01/2020'),date.parse('01/05/2020'));
      System.assertEquals(date.parse('01/05/2020'),D);
   }

    @isTest static void Test_CheckDates_case2(){
      Date D = VerifyDate.CheckDates(date.parse('01/01/2020'),date.parse('05/05/2020'));
      System.assertEquals(date.parse('01/31/2020'),D);
    }

    @isTest static void Test_DateWithin30Days_case1(){
      boolean flag =
VerifyDate.DateWithin30Days(date.parse('01/01/2020'),date.parse('12/30/2019'));
      System.assertEquals(false,flag);
    }

   @isTest static void Test_DateWithin30Days_case2(){
      boolean flag =
VerifyDate.DateWithin30Days(date.parse('01/01/2020'),date.parse('02/02/2020'));
      System.assertEquals(false,flag);
   }

   @isTest static void Test_DateWithin30Days_case3(){
      boolean flag =
VerifyDate.DateWithin30Days(date.parse('01/01/2020'),date.parse('01/15/2020'));
      System.assertEquals(true,flag);
   }

   @isTest static void Test_SetEndofMonthDate(){
      Date returndate = VerifyDate.SetEndofMonthDate(date.parse('01/01/2020'));
   }

}
```

4.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

```
@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());
   }

}
```

5.Create Test Data For Apex Tests

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;

   }

}
```

6.Future Methods

AccountProcessor

```
public class AccountProcessor {
            @future
   public static void countContacts(List<Id> accountIds){

       List<Account> accountsToUpadate = 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;

    }
}
```

Apex code for Futurre methods

```
List<Id> accountIds = new List<Id>();
accountIds.add('001Iw000001p5HwIAI');

AccountProcessor.countContacts(accountIds);
```

AccountProcessorTEST.apx

```
@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 test(){
        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();
    }


}
```

Contro Processes 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) {

        //Retrieving the 200 first leads where lead source is in blank.
        List<Lead> leads = [SELECT ID, LeadSource FROM Lead where
LeadSource = '' LIMIT 200];

        //Setting the LeadSource field the 'Dreamforce' value.
        for (Lead lead : leads) {
            lead.LeadSource = 'Dreamforce';
        }

        //Updating all elements in the list.
        update leads;
    }

}
```

DailyLeadProcessorTest

```
@isTest
private class DailyLeadProcessorTest {

   @isTest
```

```
    public static void testDailyLeadProcessor(){

        //Creating new 200 Leads and inserting them.
        List<Lead> leads = new List<Lead>();
        for (Integer x = 0; x < 200; x++) {
            leads.add(new Lead(lastname='lead number ' + x, company='company number ' + x));
        }
        insert leads;

        //Starting test. Putting in the schedule and running the DailyLeadProcessor execute method.
        Test.startTest();
        String jobId = System.schedule('DailyLeadProcessor', '0 0 12 * * ?', new DailyLeadProcessor());
        Test.stopTest();

        //Once the job has finished, retrieve all modified leads.
        List<Lead> listResult = [SELECT ID, LeadSource FROM Lead where LeadSource = 'Dreamforce' LIMIT
200];

        //Checking if the modified leads are the same size number that we created in the start of this
method.
        System.assertEquals(200, listResult.size());

    }
}
```

Apex REST Callouts

AnimalLOcater.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);
    }
}
```

AnimalLocaterTest.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);
    }
}
```

AnimalLocaterMockTest.apxc

```
@isTest
global class AnimalLocatorMock implements HttpCalloutMock {
    global HTTPResponse respond(HTTPRequest request) {
         HttpResponse response = new HttpResponse();
        response.setHeader('Content-Type', 'application/json');
        response.setBody('{"animal":{"id":1,"name":"chicken","eats":"chicken food","says":"cluck cluck"}}');
        response.setStatusCode(200);
        return response;
    }
}
```

Apex SOAP Callouts

ParkLocator.apxc

```
public class ParkLocator {
    public static String[] country(String country){
        ParkService.ParksImplPort parks = new ParkService.ParksImplPort();
        String[] parksname = parks.byCountry(country);
        return parksname;
    }
}
```

ParkLocatorTest.apxc

```
@isTest
private class ParkLocatorTest{
    @isTest
    static void testParkLocator() {
        Test.setMock(WebServiceMock.class, new ParkServiceMock());
        String[] arrayOfParks = ParkLocator.country('India');

        System.assertEquals('Park1', arrayOfParks[0]);
    }
}
```

ParkServiceMock.apxc

```
@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) {
        ParkService.byCountryResponse response_x = new ParkService.byCountryResponse();
        List<String> lstOfDummyParks = new List<String> {'Park1','Park2','Park3'};
        response_x.return_x = lstOfDummyParks;

        response.put('response_x', response_x);
    }
}
```

AsyncParkService

```
//Generated by wsdl2apex

public class AsyncParkService {
    public class byCountryResponseFuture extends System.WebServiceCalloutFuture {
        public String[] getValue() {
            ParkService.byCountryResponse response =
(ParkService.byCountryResponse)System.WebServiceCallout.endInvoke(this);
            return response.return_x;
        }
    }
    public class AsyncParksImplPort {
        public String endpoint_x = 'https://th-apex-soap-service.herokuapp.com/service/parks';
        public Map<String,String> inputHttpHeaders_x;
        public String clientCertName_x;
        public Integer timeout_x;
        private String[] ns_map_type_info = new String[]{'http://parks.services/', 'ParkService'};
        public AsyncParkService.byCountryResponseFuture beginByCountry(System.Continuation
continuation,String arg0) {
            ParkService.byCountry request_x = new ParkService.byCountry();
            request_x.arg0 = arg0;
            return (AsyncParkService.byCountryResponseFuture) System.WebServiceCallout.beginInvoke(
              this,
              request_x,
              AsyncParkService.byCountryResponseFuture.class,
              continuation,
              new String[]{endpoint_x,
              '',
              'http://parks.services/',
              'byCountry',
              'http://parks.services/',
              'byCountryResponse',
              'ParkService.byCountryResponse'}
            );
        }
    }
}
```