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 {

         public static Date CheckDates(Date date1, Date date2) {

                  if(DateWithin30Days(date1,date2)) {
                           return date2;
                  } else {
```

```
                return SetEndOfMonthDate(date1);
            }
        }

        @TestVisible private static Boolean DateWithin30Days(Date date1, Date date2) {

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

        @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) {

 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());
            }
}
```

APEX REST CALLOUTS

AnimalLocator:

```
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');
        }
    }
```
AnimalLocatorTest:
```
@isTest
public class AnimalLocatorTest {
    @isTest static void AnimalLocatorMock1(){
        Test.setMock(HttpCalloutMock.class,new AnimalLocatorMock());
        string result=AnimalLocator.getAnimalNameById(3);
        String expectedResult='chicken';
        System.assertEquals(result, expectedResult);
    }

}
```
AnimalLocatorMock:
```
@isTest
global class AnimalLocatorMock implements HttpCalloutMock{
    global HTTPResponse respond(HTTPRequest request){
        HttpResponse response=new HttpResponse();
        response.setHeader('Content-Type','application/json');
        response.setBody('{"animals":["bird","bunny","bear","chicken"]}');
        response.setStatusCode(200);
        return response;
    }
}
```

APEX SOAP CALLOUTS

ParkLocator:
```
public class ParkLocator {
    public static string[] country(string theCountry) {
        ParkService.ParksImplPort  parkSvc = new  ParkService.ParksImplPort(); // remove space
        return parkSvc.byCountry(theCountry);
    }
}
```

ParkLocatorTest:
```
@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);
  }
}
```

ParkServiceMock :

```
@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();
      response_x.return_x = new List<String>{'Yellowstone', 'Mackinac National Park', 'Yosemite'};
      response.put('response_x', response_x);
  }
}
```

APEX WEBSERVICES

AccountManager:

```
@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;
```

```
    }
}
```

AccountManagerTest:

```
@isTest
private class AccountManagerTest {

    private static testMethod void getAccountTest1() {
        Id recordId = createTestRecord();
        RestRequest request = new RestRequest();
        request.requestUri = 'https://na1.salesforce.com/services/apexrest/Accounts/'+ recordId
+'/contacts' ;
        request.httpMethod = 'GET';
        RestContext.request = request;
        Account thisAccount = AccountManager.getAccount();
        System.assert(thisAccount != null);
        System.assertEquals('Test record', thisAccount.Name);

    }
        static Id createTestRecord() {
        Account TestAcc = new Account(
          Name='Test record');
        insert TestAcc;
        Contact TestCon= new Contact(
        LastName='Test',
        AccountId = TestAcc.id);
        return TestAcc.Id;
    }
}
```

APEX SPECIALIST SUPERBADGE

Step2-Automate Record Creation:
1.MaintenanceRequest.apxt

```
trigger MaintenanceRequest on Case (before update, after update) {
    if(Trigger.isUpdate && Trigger.isAfter){
        MaintenanceRequestHelper.updateworkOrders(Trigger.New,Trigger.OldMap);
    }
}
```

2.MaintenanceRequestHelper.apxc

```
public with sharing class MaintenanceRequestHelper {
    public static void updateworkOrders(List<Case> updWorkOrders, Map<Id,Case>
nonUpdCaseMap) {
        Set<Id> validIds = new Set<Id>();
        For (Case c : updWorkOrders){
            if (nonUpdCaseMap.get(c.Id).Status != 'Closed' && c.Status == 'Closed'){
                if (c.Type == 'Repair' || c.Type == 'Routine Maintenance'){
                    validIds.add(c.Id);
                }
            }
        }

        if (!validIds.isEmpty()){
            Map<Id,Case> closedCases = new Map<Id,Case>([SELECT Id, Vehicle__c, Equipment__c,
Equipment__r.Maintenance_Cycle__c,
                                                          (SELECT Id,Equipment__c,Quantity__c FROM
Equipment_Maintenance_Items__r)
                                                          FROM Case WHERE Id IN :validIds]);
            Map<Id,Decimal> maintenanceCycles = new Map<ID,Decimal>();
            AggregateResult[] results = [SELECT Maintenance_Request__c,
                                   MIN(Equipment__r.Maintenance_Cycle__c)cycle
                                   FROM Equipment_Maintenance_Item__c
                                   WHERE Maintenance_Request__c IN :ValidIds GROUP BY
Maintenance_Request__c];

            for (AggregateResult ar : results){
                maintenanceCycles.put((Id) ar.get('Maintenance_Request__c'), (Decimal)ar.get('cycle'));
            }

            List<Case> newCases = new List<Case>();
            for(Case cc : closedCases.values()){
                Case nc = new Case (
                    ParentId = cc.Id,
                    Status = 'New',
                    Subject = 'Routine Maintenance',
                    Type = 'Routine Maintenance',
                    Vehicle__c = cc.Vehicle__c,
                    Equipment__c =cc.Equipment__c,
                    Origin = 'Web',
                    Date_Reported__c = Date.Today()
                );
                If (maintenanceCycles.containskey(cc.Id)){
                    nc.Date_Due__c = Date.today().addDays((Integer) maintenanceCycles.get(cc.Id));
                } else {
                  // nc.Date_Due__c = Date.today().addDays((Integer)
cc.Equipment__r.maintenance_Cycle__c);
                }

                newCases.add(nc);
            }

            insert newCases;

            List<Equipment_Maintenance_Item__c> clonedList = new
List<Equipment_Maintenance_Item__c>();
            for (Case nc : newCases){
                for (Equipment_Maintenance_Item__c clonedListItem :
closedCases.get(nc.ParentId).Equipment_Maintenance_Items__r){
                    Equipment_Maintenance_Item__c item = clonedListItem.clone();
                    item.Maintenance_Request__c = nc.Id;
                    clonedList.add(item);
                }
            }
            insert clonedList;
        }
    }
}
```

Step 3: Synchronize Salesforce Data with an External System

1.WarehouseCalloutService.apxc

```
public with sharing class WarehouseCalloutService implements Queueable {
    private static final String WAREHOUSE_URL=
'https://thsuperbadgeapex.herokuapp.com/equipment';
  @future(callout=true)
   public static void runWarehouseEquipmentSync(){
       System.debug('go into runWarehouseEquipmentSync');
       Http http = new Http();
       HttpRequest request = new HttpRequest();
```

```
        request.setEndpoint(WAREHOUSE_URL);
        request.setMethod('GET');
        HttpResponse response = http.send(request);

        List<Product2> product2List = new List<Product2>();
        System.debug(response.getStatusCode());
        if (response.getStatusCode() == 200){
            List<Object> jsonResponse =
(List<Object>)JSON.deserializeUntyped(response.getBody());
            System.debug(response.getBody());
              for (Object jR : jsonResponse){
              Map<String,Object> mapJson = (Map<String,Object>)jR;
              Product2 product2 = new Product2();
              product2.Replacement_Part__c = (Boolean) mapJson.get('replacement');
              product2.Cost__c = (Integer) mapJson.get('cost');
              product2.Current_Inventory__c = (Double) mapJson.get('quantity');
              product2.Lifespan_Months__c = (Integer) mapJson.get('lifespan');
              product2.Maintenance_Cycle__c = (Integer) mapJson.get('maintenanceperiod');
              product2.Warehouse_SKU__c = (String) mapJson.get('sku');
              product2.Name = (String) mapJson.get('name');
              product2.ProductCode = (String) mapJson.get('_id');
              product2List.add(product2);
            }

            if (product2List.size() > 0){
                upsert product2List;
                System.debug('Your equipment was synced with the warehouse one');
            }
        }
    }

    public static void execute (QueueableContext context){
        System.debug('start runWarehouseEquipmentSync');
        runWarehouseEquipmentSync();
        System.debug('end runWarehouseEquipmentSync');
    }

}
```

Step 4-Schedule Synchronization:

1.WarehouseSyncSchedule.apxc

```
global with sharing class WarehouseSyncSchedule implements Schedulable{
   global void execute(SchedulableContext ctx){
      System.enqueueJob(new WarehouseCalloutService());
   }
}
```

Step 5-Test Automation Logic:

1.MaintenanceRequestHelper.apxc

```
public with sharing class MaintenanceRequestHelper {
   public static void updateworkOrders(List<Case> updWorkOrders, Map<Id,Case>
nonUpdCaseMap) {
      Set<Id> validIds = new Set<Id>();
      For (Case c : updWorkOrders){
         if (nonUpdCaseMap.get(c.Id).Status != 'Closed' && c.Status == 'Closed'){
            if (c.Type == 'Repair' || c.Type == 'Routine Maintenance'){
               validIds.add(c.Id);
            }
         }
      }


      if (!validIds.isEmpty()){
         Map<Id,Case> closedCases = new Map<Id,Case>([SELECT Id, Vehicle__c, Equipment__c,
Equipment__r.Maintenance_Cycle__c, (SELECT Id,Equipment__c,Quantity__c FROM
Equipment_Maintenance_Items__r) FROM Case WHERE Id IN :validIds]);
         Map<Id,Decimal> maintenanceCycles = new Map<ID,Decimal>();
         AggregateResult[] results = [SELECT Maintenance_Request__c,
                              MIN(Equipment__r.Maintenance_Cycle__c)cycle
                              FROM Equipment_Maintenance_Item__c
                              WHERE Maintenance_Request__c IN :ValidIds GROUP BY
Maintenance_Request__c];

         for (AggregateResult ar : results){
            maintenanceCycles.put((Id) ar.get('Maintenance_Request__c'), (Decimal)
ar.get('cycle'));
         }
```

```
            List<Case> newCases = new List<Case>();
            for(Case cc : closedCases.values()){
                Case nc = new Case (
                    ParentId = cc.Id,
                    Status = 'New',
                    Subject = 'Routine Maintenance',
                    Type = 'Routine Maintenance',
                    Vehicle__c = cc.Vehicle__c,
                    Equipment__c =cc.Equipment__c,
                    Origin = 'Web',
                    Date_Reported__c = Date.Today()
                );

                                If (maintenanceCycles.containskey(cc.Id)){
                    nc.Date_Due__c = Date.today().addDays((Integer) maintenanceCycles.get(cc.Id));
                } else {
                   // nc.Date_Due__c = Date.today().addDays((Integer)
cc.Equipment__r.maintenance_Cycle__c);
                }

                newCases.add(nc);
            }

            insert newCases;

            List<Equipment_Maintenance_Item__c> clonedList = new
List<Equipment_Maintenance_Item__c>();
            for (Case nc : newCases){
                for (Equipment_Maintenance_Item__c clonedListItem :
closedCases.get(nc.ParentId).Equipment_Maintenance_Items__r){
                    Equipment_Maintenance_Item__c item = clonedListItem.clone();
                    item.Maintenance_Request__c = nc.Id;
                    clonedList.add(item);
                }
            }
            insert clonedList;
        }
    }
}
```

Step 6-Test Callout Logic1.

1.WarehouseCalloutService.apxc

```
public with sharing class WarehouseCalloutService implements Queueable {
   private static final String WAREHOUSE_URL = 'https://th-superbadge-
apex.herokuapp.com/equipment';
@future(callout=true)
 public static void runWarehouseEquipmentSync(){
        System.debug('go into runWarehouseEquipmentSync');
        Http http = new Http();
        HttpRequest request = new HttpRequest();
        request.setEndpoint(WAREHOUSE_URL);
        request.setMethod('GET');
        HttpResponse response = http.send(request);
        List<Product2> product2List = new List<Product2>();
        System.debug(response.getStatusCode());
        if (response.getStatusCode() == 200){
            List<Object> jsonResponse =
(List<Object>)JSON.deserializeUntyped(response.getBody());
            System.debug(response.getBody());
               for (Object jR : jsonResponse){
                Map<String,Object> mapJson = (Map<String,Object>)jR;
                Product2 product2 = new Product2();
                product2.Replacement_Part__c = (Boolean) mapJson.get('replacement');
                product2.Cost__c = (Integer) mapJson.get('cost');
                product2.Current_Inventory__c = (Double) mapJson.get('quantity');
                product2.Lifespan_Months__c = (Integer) mapJson.get('lifespan');
                product2.Maintenance_Cycle__c = (Integer) mapJson.get('maintenanceperiod');
                product2.Warehouse_SKU__c = (String) mapJson.get('sku');
                product2.Name = (String) mapJson.get('name');
                product2.ProductCode = (String) mapJson.get('_id');
                product2List.add(product2);
            }

            if (product2List.size() > 0){
                upsert product2List;
                System.debug('Your equipment was synced with the warehouse one');
            }
        }
   }
```

```
    public static void execute (QueueableContext context){
        System.debug('start runWarehouseEquipmentSync');
        runWarehouseEquipmentSync();
        System.debug('end runWarehouseEquipmentSync');
    }

}
```

2.WarehouseCalloutServiceTest.apxc

```
@IsTest
private class WarehouseCalloutServiceTest {
    @isTest
    static void testWarehouseCallout() {
        test.startTest();
        test.setMock(HttpCalloutMock.class, new WarehouseCalloutServiceMock());
        WarehouseCalloutService.execute(null);
        test.stopTest();

        List<Product2> product2List = new List<Product2>();
        product2List = [SELECT ProductCode FROM Product2];

        System.assertEquals(3, product2List.size());
        System.assertEquals('55d66226726b611100aaf741', product2List.get(0).ProductCode);
        System.assertEquals('55d66226726b611100aaf742', product2List.get(1).ProductCode);
        System.assertEquals('55d66226726b611100aaf743', product2List.get(2).ProductCode);
    }
}
```

3.WarehouseCalloutServiceMock.apxc

```
@isTest
global class WarehouseCalloutServiceMock implements HttpCalloutMock {
    global static HttpResponse respond(HttpRequest request) {

        HttpResponse response = new HttpResponse();
        response.setHeader('Content-Type', 'application/json');

response.setBody('[{"_id":"55d66226726b611100aaf741","replacement":false,"quantity":5,"name":
"Generator 1000
kW","maintenanceperiod":365,"lifespan":120,"cost":5000,"sku":"100003"},{"_id":"55d66226726b611
100aaf742","replacement":true,"quantity":183,"name":"Cooling
Fan","maintenanceperiod":0,"lifespan":0,"cost":300,"sku":"100004"},{"_id":"55d66226726b611100a
af743","replacement":true,"quantity":143,"name":"Fuse
20A","maintenanceperiod":0,"lifespan":0,"cost":22,"sku":"100005"}]');
        response.setStatusCode(200);

        return response;
    }
}
```

Step 7-Test Scheduling Logic:
1.WarehouseSyncSchedule.apxc

```
global with sharing class WarehouseSyncSchedule implements Schedulable{
    global void execute(SchedulableContext ctx){
        System.enqueueJob(new WarehouseCalloutService());
    }
}
```

2. WarehouseSyncScheduleTest.apxc

```
@isTest
public with sharing class WarehouseSyncScheduleTest {
    @isTest static void test() {
        String scheduleTime = '00 00 00 * * ? *';
        Test.startTest();
        Test.setMock(HttpCalloutMock.class, new WarehouseCalloutServiceMock());
        String jobId = System.schedule('Warehouse Time to Schedule to test', scheduleTime, new
WarehouseSyncSchedule());
        CronTrigger c = [SELECT State FROM CronTrigger WHERE Id =: jobId];
        System.assertEquals('WAITING', String.valueOf(c.State), 'JobId does not match');
        Test.stopTest();
    }
}
```