APEX SPECIALIST SUPERBADGE CHALLENGE

Automate record creation

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()){
         List<Case> newCases = new List<Case>();
         Map<Id,Case> closedCasesM = 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'));
      }

         for(Case cc : closedCasesM.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> clonedWPs = new
List<Equipment_Maintenance_Item__c>();
         for (Case nc : newCases){
            for (Equipment_Maintenance_Item__c wp :
closedCasesM.get(nc.ParentId).Equipment_Maintenance_Items__r){
               Equipment_Maintenance_Item__c wpClone = wp.clone();
               wpClone.Maintenance_Request__c = nc.Id;
               ClonedWPs.add(wpClone);

            }
         }
         insert ClonedWPs;
      }
   }
}
```

MaitenanceRequest.apxt :-

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

Synchronize Salesforce data with an external system

WarehouseCalloutService.apxc :-

```
public with sharing class WarehouseCalloutService implements Queueable {
   private static final String WAREHOUSE_URL = 'https://th-superbadge-
apex.herokuapp.com/equipment';

   //class that makes a REST callout to an external warehouse system to get a list of equipment
that needs to be updated.
   //The callout's JSON response returns the equipment records that you upsert in Salesforce.

   @future(callout=true)
   public static void runWarehouseEquipmentSync(){
      Http http = new Http();
      HttpRequest request = new HttpRequest();

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

      List<Product2> warehouseEq = new List<Product2>();

      if (response.getStatusCode() == 200){
         List<Object> jsonResponse =
(List<Object>)JSON.deserializeUntyped(response.getBody());
         System.debug(response.getBody());

         //class maps the following fields: replacement part (always true), cost, current inventory,
```

```
lifespan, maintenance cycle, and warehouse SKU
        //warehouse SKU will be external ID for identifying which equipment records to update
within Salesforce
        for (Object eq : jsonResponse){
          Map<String,Object> mapJson = (Map<String,Object>)eq;
          Product2 myEq = new Product2();
          myEq.Replacement_Part__c = (Boolean) mapJson.get('replacement');
          myEq.Name = (String) mapJson.get('name');
          myEq.Maintenance_Cycle__c = (Integer) mapJson.get('maintenanceperiod');
          myEq.Lifespan_Months__c = (Integer) mapJson.get('lifespan');
          myEq.Cost__c = (Integer) mapJson.get('cost');
          myEq.Warehouse_SKU__c = (String) mapJson.get('sku');
          myEq.Current_Inventory__c = (Double) mapJson.get('quantity');
          myEq.ProductCode = (String) mapJson.get('_id');
          warehouseEq.add(myEq);
        }

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

    public static void execute (QueueableContext context){
      runWarehouseEquipmentSync();
    }

}
```

execute anonymous window

```
System.enqueueJob(new WarehouseCalloutService());
```

Schedule synchronization

WarehouseSyncShedule.apxc :-

```
global with sharing class WarehouseSyncSchedule implements Schedulable{
```

```
    global void execute(SchedulableContext ctx){
        System.enqueueJob(new WarehouseCalloutService());
    }
}
```

Test automation logic

MaintenanceRequestHelperTest.apxc :-

```
@istest
public with sharing class MaintenanceRequestHelperTest {

    private static final string STATUS_NEW = 'New';
    private static final string WORKING = 'Working';
    private static final string CLOSED = 'Closed';
    private static final string REPAIR = 'Repair';
    private static final string REQUEST_ORIGIN = 'Web';
    private static final string REQUEST_TYPE = 'Routine Maintenance';
    private static final string REQUEST_SUBJECT = 'Testing subject';

    PRIVATE STATIC Vehicle__c createVehicle(){
        Vehicle__c Vehicle = new Vehicle__C(name = 'SuperTruck');
        return Vehicle;
    }

    PRIVATE STATIC Product2 createEq(){
        product2 equipment = new product2(name = 'SuperEquipment',
                                          lifespan_months__C = 10,
                                          maintenance_cycle__C = 10,
                                          replacement_part__c = true);
        return equipment;
    }

    PRIVATE STATIC Case createMaintenanceRequest(id vehicleId, id equipmentId){
        case cs = new case(Type=REPAIR,
                           Status=STATUS_NEW,
                           Origin=REQUEST_ORIGIN,
                           Subject=REQUEST_SUBJECT,
                           Equipment__c=equipmentId,
```

```
                Vehicle__c=vehicleId);
    return cs;
  }

  PRIVATE STATIC Equipment_Maintenance_Item__c createWorkPart(id equipmentId,id
requestId){
    Equipment_Maintenance_Item__c wp = new
Equipment_Maintenance_Item__c(Equipment__c = equipmentId,
                                        Maintenance_Request__c = requestId);
    return wp;
  }


  @istest
  private static void testMaintenanceRequestPositive(){
    Vehicle__c vehicle = createVehicle();
    insert vehicle;
    id vehicleId = vehicle.Id;

    Product2 equipment = createEq();
    insert equipment;
    id equipmentId = equipment.Id;

    case somethingToUpdate = createMaintenanceRequest(vehicleId,equipmentId);
    insert somethingToUpdate;

    Equipment_Maintenance_Item__c workP =
createWorkPart(equipmentId,somethingToUpdate.id);
    insert workP;

    test.startTest();
    somethingToUpdate.status = CLOSED;
    update somethingToUpdate;
    test.stopTest();

    Case newReq = [Select id, subject, type, Equipment__c, Date_Reported__c, Vehicle__c,
Date_Due__c
            from case
            where status =:STATUS_NEW];

    Equipment_Maintenance_Item__c workPart = [select id
```

```
                                from Equipment_Maintenance_Item__c
                                where Maintenance_Request__c =:newReq.Id];

    system.assert(workPart != null);
    system.assert(newReq.Subject != null);
    system.assertEquals(newReq.Type, REQUEST_TYPE);
    SYSTEM.assertEquals(newReq.Equipment__c, equipmentId);
    SYSTEM.assertEquals(newReq.Vehicle__c, vehicleId);
    SYSTEM.assertEquals(newReq.Date_Reported__c, system.today());
}

@istest
private static void testMaintenanceRequestNegative(){
    Vehicle__C vehicle = createVehicle();
    insert vehicle;
    id vehicleId = vehicle.Id;

    product2 equipment = createEq();
    insert equipment;
    id equipmentId = equipment.Id;

    case emptyReq = createMaintenanceRequest(vehicleId,equipmentId);
    insert emptyReq;

    Equipment_Maintenance_Item__c workP = createWorkPart(equipmentId, emptyReq.Id);
    insert workP;

    test.startTest();
    emptyReq.Status = WORKING;
    update emptyReq;
    test.stopTest();

    list<case> allRequest = [select id
                             from case];

    Equipment_Maintenance_Item__c workPart = [select id
                                from Equipment_Maintenance_Item__c
                                where Maintenance_Request__c = :emptyReq.Id];

    system.assert(workPart != null);
    system.assert(allRequest.size() == 1);
```

```
    }

    @istest
    private static void testMaintenanceRequestBulk(){
        list<Vehicle__C> vehicleList = new list<Vehicle__C>();
        list<Product2> equipmentList = new list<Product2>();
        list<Equipment_Maintenance_Item__c> workPartList = new
list<Equipment_Maintenance_Item__c>();
        list<case> requestList = new list<case>();
        list<id> oldRequestIds = new list<id>();

        for(integer i = 0; i < 300; i++){
          vehicleList.add(createVehicle());
           equipmentList.add(createEq());
        }
        insert vehicleList;
        insert equipmentList;

        for(integer i = 0; i < 300; i++){
           requestList.add(createMaintenanceRequest(vehicleList.get(i).id, equipmentList.get(i).id));
        }
        insert requestList;

        for(integer i = 0; i < 300; i++){
           workPartList.add(createWorkPart(equipmentList.get(i).id, requestList.get(i).id));
        }
        insert workPartList;

        test.startTest();
        for(case req : requestList){
           req.Status = CLOSED;
           oldRequestIds.add(req.Id);
        }
        update requestList;
        test.stopTest();

        list<case> allRequests = [select id
                                  from case
                                  where status =: STATUS_NEW];

        list<Equipment_Maintenance_Item__c> workParts = [select id
```

```
                                    from Equipment_Maintenance_Item__c
                                    where Maintenance_Request__c in: oldRequestIds];

        system.assert(allRequests.size() == 300);
    }
}
```

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()){
            List<Case> newCases = new List<Case>();
            Map<Id,Case> closedCasesM = 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'));
        }

            for(Case cc : closedCasesM.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));
            }

            newCases.add(nc);
         }

         insert newCases;

         List<Equipment_Maintenance_Item__c> clonedWPs = new
List<Equipment_Maintenance_Item__c>();
         for (Case nc : newCases){
            for (Equipment_Maintenance_Item__c wp :
closedCasesM.get(nc.ParentId).Equipment_Maintenance_Items__r){
               Equipment_Maintenance_Item__c wpClone = wp.clone();
               wpClone.Maintenance_Request__c = nc.Id;
               ClonedWPs.add(wpClone);

            }
         }
          insert ClonedWPs;
      }
   }
}
```

MaintenanceRequest.apxt :-

```
trigger MaintenanceRequest on Case (before update, after update) {
   if(Trigger.isUpdate && Trigger.isAfter){
```

```
        MaintenanceRequestHelper.updateWorkOrders(Trigger.New, Trigger.OldMap);
    }
}
```

Test callout logic

WarehouseCalloutService.apxc :-

```
public with sharing class WarehouseCalloutService {
    private static final String WAREHOUSE_URL = 'https://th-superbadge-
apex.herokuapp.com/equipment';

    //@future(callout=true)
    public static void runWarehouseEquipmentSync(){

        Http http = new Http();
        HttpRequest request = new HttpRequest();

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


        List<Product2> warehouseEq = new List<Product2>();

        if (response.getStatusCode() == 200){
            List<Object> jsonResponse =
(List<Object>)JSON.deserializeUntyped(response.getBody());
            System.debug(response.getBody());

            for (Object eq : jsonResponse){
                Map<String,Object> mapJson = (Map<String,Object>)eq;
                Product2 myEq = new Product2();
                myEq.Replacement_Part__c = (Boolean) mapJson.get('replacement');
                myEq.Name = (String) mapJson.get('name');
                myEq.Maintenance_Cycle__c = (Integer) mapJson.get('maintenanceperiod');
                myEq.Lifespan_Months__c = (Integer) mapJson.get('lifespan');
                myEq.Cost__c = (Decimal) mapJson.get('lifespan');
```

```
            myEq.Warehouse_SKU__c = (String) mapJson.get('sku');
            myEq.Current_Inventory__c = (Double) mapJson.get('quantity');
            warehouseEq.add(myEq);
        }

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

    }
  }
}
```

WarehouseCalloutServiceTest.apxc :-

```
@isTest
private class WarehouseCalloutServiceTest {
  @isTest
  static void testWareHouseCallout(){
    Test.startTest();
    // implement mock callout test here
    Test.setMock(HTTPCalloutMock.class, new WarehouseCalloutServiceMock());
    WarehouseCalloutService.runWarehouseEquipmentSync();
    Test.stopTest();
    System.assertEquals(1, [SELECT count() FROM Product2]);
  }
}
```

WarehouseCalloutServiceMock.apxc :-

```
@isTest
global class WarehouseCalloutServiceMock implements HttpCalloutMock {
  // implement http mock callout
  global static HttpResponse respond(HttpRequest request){

    System.assertEquals('https://th-superbadge-apex.herokuapp.com/equipment',
```

```
request.getEndpoint());
        System.assertEquals('GET', request.getMethod());

        // Create a fake response
        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"}]');
        response.setStatusCode(200);
        return response;
    }
}
```

Test scheduling logic

WarehouseSyncSchedule.apxc :-

```
global class WarehouseSyncSchedule implements Schedulable {
    global void execute(SchedulableContext ctx) {

        WarehouseCalloutService.runWarehouseEquipmentSync();
    }
}
```

WarehouseSyncScheduleTest.apxc :-

```
@isTest
public class WarehouseSyncScheduleTest {

    @isTest static void WarehousescheduleTest(){
        String scheduleTime = '00 00 01 * * ?';
        Test.startTest();
        Test.setMock(HttpCalloutMock.class, new WarehouseCalloutServiceMock());
        String jobID=System.schedule('Warehouse Time To Schedule to Test', scheduleTime, new
WarehouseSyncSchedule());
        Test.stopTest();
        //Contains schedule information for a scheduled job. CronTrigger is similar to a cron job on
UNIX systems.
```

```
        // This object is available in API version 17.0 and later.
        CronTrigger a=[SELECT Id FROM CronTrigger where NextFireTime > today];
        System.assertEquals(jobID, a.Id,'Schedule ');


    }
}
```

Apex Integration Services

Apex Web Services

AccountManager.apxc

```
@RestResource(urlMapping='/Accounts/*/contacts')
global class AccountManager {

   @HttpGet
   global static Account getAccount() {
       RestRequest request = RestContext.request;
      String accId = request.requestURI.substringBetween('Accounts/','/contacts');

    List<Account> acc = [SELECT Id, Name, (SELECT Id, Name FROM Contacts) FROM Account
WHERE Id = :accId];

   return acc.get(0);
   }
}
```

AccountManagerTest.apxc

```
@isTest
public class AccountManagerTest {
@isTest(SeeAllData=true)
   public static void getAccountTest(){
      //create Test Data
      Account acc=new Account(Name='testAccount1');
      Contact con=new Contact(Account=acc,AssistantName='vidhyaTest',LastName='mukmuk');
            insert acc;
      Id recordId=acc.Id;
      insert con;


      RestRequest request = new RestRequest();

       request.requestUri = 'https://sadas-dev-
ed.my.salesforce.com/services/apexrest/Accounts/'+recordId+'/mukmuk';
      request.httpMethod = 'GET';
       RestContext.request = request;

       // Call the method to test
```

```
      Account thisAccount = AccountManager.getAccount();
       // Verify results
       System.assert(thisAccount != null);
   }

}
```

Apex SOAP Callouts

ParkLocator.apxc

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

ParkLocatorTest.apxc

```
@isTest
private class ParkLocatorTest {
   @isTest static void testCallout() {
      // This causes a fake response to be generated
      Test.setMock(WebServiceMock.class, new ParkServiceMock());
      // Call the method that invokes a callout


      String country = 'India';




      System.assertEquals(new List<String>{'Lal Bhag', 'Cubbon Park', 'Pazhassi Dam'},
ParkLocator.country(country));
   }
}
```

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) {
      // start - specify the response you want to send
      parkService.byCountryResponse response_x = new parkService.byCountryResponse();
      response_x.return_x = new List<String>{'Lal Bhag', 'Cubbon Park', 'Pazhassi Dam'};
          // end
      response.put('response_x', response_x);
  }
}
```

Apex REST Callouts

AnimalLocator.apxc

```
public with sharing class AnimalLocator {
   public static String getAnimalNameById(Integer animalNameId) {
      String animalName = '';
      //New Http 'GET' Request
      Http http = new Http();
      HttpRequest request = new HttpRequest();
      request.setEndpoint('https://th-apex-http-callout.herokuapp.com/animals/:id');
      request.setHeader('Content-Type', 'application/json;charset=UTF-8');
      request.setMethod('GET');
      //Get response
      HttpResponse response = Http.send(request);
      //Parse JSON from the response body
      JSONParser parser = JSON.createParser(response.getBody());
      while (parser.nextToken() != null) {
         // Read entire JSON object
         if (parser.getCurrentToken() == JSONToken.START_OBJECT) {
            AnimalLocator.AnimalList animalList = (AnimalLocator.AnimalList)
```

```
parser.readValueAs(AnimalLocator.AnimalList.class);
            System.debug(animalList.animal.size());
            //Sort through the list of animals to find one with the matching ID
            //Set the animal name
            for (Integer i = 0; i < animalList.animal.size() ; i++) {
               if (animalList.animal[i].id == animalNameId){
                  animalName = animalList.animal[i].name;
                  break;
               } else{
                  animalName = 'Could not find an Animal with a matching ID';
               }
            }

         }
      }
      return animalName;
   }

   public class AnimalList {
      public List<animal> animal; //This has to be the same name thats in the JSON file.
   }
   //animal Object Wrapper
   public class animal {
      public Integer id;
      public String name;
      public String eats;
      public String says;
   }
}
```

AnimalLocatorTest.apxc

```
@isTest
public with sharing class AnimalLocatorTest {
   @isTest
    static void testGetCallout() {
      Test.setMock(HttpCalloutMock.class, new AnimalLocatorMock());
      String result = AnimalLocator.getAnimalNameById(1);
      String expectedResult = 'Chicken';
      System.assertEquals(result,expectedResult);
      result = AnimalLocator.getAnimalNameById(4);
      expectedResult = 'Could not find an Animal with a matching ID';
```

```
        System.assertEquals(result,expectedResult);
    }
}
```

AnimalLocatorMock.apxc

```
@isTest
global class AnimalLocatorMock implements HttpCalloutMock{
    global HttpResponse respond(HttpRequest request){
        //Create Fake Response
        HttpResponse response = new HttpResponse();
        response.setHeader('Content-Type', 'application/json;charset=UTF-8');
        response.setStatusCode(200);
        response.setBody('
{"animal":[{"id":1,"name":"Chicken","eats":"Grain","says":"Cluck"},{"id":2,"name":"Dog","eats":"Chicken
","says":"Woof"}]} ');
        return response;
    }
}
```

Asynchronous Apex

Use Future Methods

AccountProcessor.apxc

```
public class AccountProcessor {

   //Writting the countContacts method and marking it whit the @future label.
   @future
   public static void countContacts(Set<Id> accountIDs) {

      // Creating a list that will contain all those accounts that are referenced through the accounIDs list.
      List<Account> accounts = [SELECT Id, Number_of_Contacts__c, (SELECT id FROM Contacts) from Account where id in :accountIDs];

      //Assigment from the total contact number to the Number_of_Contacts__c field for each account at accounts list.
      for( Account account : accounts ) {
        account.Number_of_Contacts__c = account.contacts.size();
      }

      //Updating all accounts in list
      update accounts;

   }

}
```

AccountProcessorTest.apxc

```
@isTest
public class AccountProcessorTest {

   @isTest
   public static void countContactsTest(){
      //Creating an account and inserting it
      Account account = New Account(Name = 'Account Number 1');
      insert account;

      //Creating some contacts related to the account and inserting them
      List<Contact> contacts = new List<Contact>();
      contacts.add(New Contact(lastname = 'Related Contact 1', AccountId = account.Id));
```

```
        contacts.add(New Contact(lastname = 'Related Contact 2', AccountId = account.Id));
        contacts.add(New Contact(lastname = 'Related Contact 3', AccountId = account.Id));
        contacts.add(New Contact(lastname = 'Related Contact 4', AccountId = account.Id));
        insert contacts;

        //Creating a List with account Ids to pass them throught the
AccountProcessor.countContacts method
        Set<Id> accountIds = new Set<Id>();
        accountIds.add(account.id);

        //Starting Test:
        Test.startTest();

        //Calling the AccountProcessor.countContacts method
        AccountProcessor.countContacts(accountIds);

        //Finishing Test:
        Test.stopTest();
        Account ACC = [SELECT Number_of_Contacts__c FROM Account WHERE id = :account.Id
LIMIT 1];

        //Setting Assert (We have to parse the account.Number_of_Contacts__c
        //to integer to avoid some comparasion error between decimal and integer)
        System.assertEquals( Integer.valueOf(ACC.Number_of_Contacts__c) , 4);
    }


}
```

Use Batch Apex

LeadProcessor.apxc

```
public class LeadProcessor  implements
    Database.Batchable<sObject>, Database.Stateful {
    // instance member to retain state across transactions
    public Integer recordsProcessed = 0;
    public Database.QueryLocator start(Database.BatchableContext bc) {
        return Database.getQueryLocator('SELECT ID, LeadSource from Lead');
```

```
    }
public void execute(Database.BatchableContext bc, List<Lead> scope){
        // process each batch of records
      //  List<Lead> lList = new List<Lead>();
        for (Lead lList : scope) {
              lList.leadsource='Dreamforce';



                }

        update scope;
    }
    public void finish(Database.BatchableContext bc){

    }
}
```

LeadProcessorTest.apxc

```
@isTest
public class LeadProcessorTest {
 @testSetup
    static void setup() {
        List<Lead> llist = new List<Lead>();
            // insert 10 accounts
        for (Integer i=0;i<200;i++) {
           llist.add(new Lead(FirstName='Lead '+i,LastName='last', Company ='demo'+i));
        }
        insert llist;
        // find the account just inserted. add contact for each

    }
    @isTest static void test() {
        Test.startTest();
        LeadProcessor lpt = new LeadProcessor();
        Id batchId = Database.executeBatch(lpt);
        Test.stopTest();
        // after the testing stops, assert records were updated properly
        System.assertEquals(200, [select count() from lead where Leadsource = 'Dreamforce']);
```

```
    }
}
```

Control Processes with Queueable Apex

AddPrimaryContact.apxc

```
public class AddPrimaryContact implements Queueable
{
    private Contact c;
    private String state;
    public  AddPrimaryContact(Contact c, String state)
    {
        this.c = c;
        this.state = state;
    }
    public void execute(QueueableContext context)
    {
         List<Account> ListAccount = [SELECT ID, Name ,(Select id,FirstName,LastName from
contacts ) FROM ACCOUNT WHERE BillingState = :state LIMIT 200];
         List<Contact> lstContact = new List<Contact>();
         for (Account acc:ListAccount)
         {
               Contact cont = c.clone(false,false,false,false);
               cont.AccountId =  acc.id;
               lstContact.add( cont );
         }

         if(lstContact.size() >0 )
         {
            insert lstContact;
         }

    }

}
```

AddPrimaryContactTest.apxc

```
@isTest
public class AddPrimaryContactTest
{
    @isTest static void TestList()
    {
        List<Account> Teste = new List <Account>();
        for(Integer i=0;i<50;i++)
        {
            Teste.add(new Account(BillingState = 'CA', name = 'Test'+i));
        }
        for(Integer j=0;j<50;j++)
        {
            Teste.add(new Account(BillingState = 'NY', name = 'Test'+j));
        }
        insert Teste;
        Contact co = new Contact();
        co.FirstName='demo';
        co.LastName ='demo';
        insert co;
        String state = 'CA';
         AddPrimaryContact apc = new AddPrimaryContact(co, state);
         Test.startTest();
           System.enqueueJob(apc);
         Test.stopTest();
    }
}
```

Schedule Jobs Using the Apex Scheduler

DailyLeadProcessor .apxc

```
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.apxc

```
@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());
        }
}
```