APEXTRIGGERS

Apex triggersenables us to perform some actions before or after some events such as insertions,updates or deletion.It helps in maintainig records.

AccountAddressTrigger.apxt

```
triggerAccountAddressTrigger on Account(beforeinsert, before

update){for(Account account:Trigger.new){

if(account.Match_Billing_Address c == True){

account.ShippingPostalCode = account.BillingPostalCode;

}
}
}
```

Explanation:

AccountAddressTrigger sets an account's ShippingPostal Code to match the Billing PostalCode.If the Match BillingAddress option is selected. Trigger is fired before inserting anaccount or updatingan account.

ClosedOpportunityTrigger.apxt

```
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;
  }

}
```

Explanation:

ClosedOpportunityTrigger is a apex trigger which fire trigger when StageName is ClosedWonand add Follow Up Test task afterinserting or updatingan opportunity.

APEX TESTING

VerifyDate.apxc

```
public classVerifyDate {

        / method to handle potential checks against two dates
        publicstatic 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);

    / methodtocheck 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
    pastif(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 staticvoid Test_DateWithin30Days_case1(){
        Booleanflag = VerifyDate.DateWithin30Days(date.parse('01/01/2020'),
date.parse('12/30/2019'));
        System.assertEquals(false, flag);
    }
    @isTest static void Test_DateWithin30Days_case2(){


 Booleanflag = VerifyDate.DateWithin30Days(date.parse('01/01/2020'),
date.parse('02/02/2020'));
        System.assertEquals(false, flag);
    }
    @isTest static void Test_DateWithin30Days_case3(){
        Booleanflag = 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'));
    }


}
```

Explanation:

TestVerifyDate isa apex class to test if a date is withina proper range and if not, returns a date that occurs at the end of the monthwithinthe range.

RestrictContactBy Name.apxc

```
trigger RestrictContactByName on Contact (beforeinsert, before update){

        / check contactsprior to insertor update for
        invalid dataFor (Contactc : Trigger.New) {
                if(c.LastName == 'INVALIDNAME') { / invalidname is invalid
                        c.AddError('The Last Name "'+c.LastName+'" is not allowedfor
                        DML');
                }

        }
}
```

TestRestrictContactBy Name.apxc

```
@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
      ().s ize() > 0);
       System.assertEquals('The Last Name "INVALIDNAME" is not allowedfor
DML',result.getErrors()[0].getMessage());
   }
```

```
}
```

Explanation:

TestRestrictContactByName is a Apex trigger which blocks inserts and updates any contactwith a last name of 'INVALIDNAME'

RandomContactFactory.apxc

```
public class RandomContactFactory{

  publicstatic 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;
  }

}
```

Explanation:

RandomContactFactory is an Apex class that returns a list of contacts based on two incoming parameters: the numberof contacts to generate and the last name.

Asy nchronous Apex

AccountProcessor.apxc

```
publicclass AccountProcessor {
```

```
  @future
  public static void countContacts(List<Id> accountIds){
    List<Account> accountsToUpdate = new List<Account>();
    List<Account> accounts= [Select Id, Name, (SelectIdfrom Contacts)from
AccountWhere 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.apxc

```
@IsTest
private class AccountProcessorTest
  {@IsTest
  Private static void testCountContacts(){
    Account newAccount = new Account(Name =
    'TestAccount');insert newAccount;

    ContactnewContact1 = new Contact(FirstName='John',LastName='Doe',AccountId =
newAccount.Id);
    insert newContact1;

    ContactnewContact2 = 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();
    }

}
```

LeadProcessor.apxc

```
globalclass LeadProcessor implements
    Database.Batchable<sObject> { global Integercount=0;
    global Database.QueryLocator start(Database.BatchableContext bc){
        return Database.getQueryLocator('SELECT ID, LeadSource FROM
        Lead');
    }


    globalvoid 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;
    }


    globalvoid finish(Database.BatchableContext bc){
        System.debug('count = '+count);
    }
```

}

LeadProcessorTest.apxc

```
@isTest
public class LeadProcessorTest {

   @isTest
   public staticvoid 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(l
      p);Test.stopTest();
   }

}
```

AddPrimaryContact.apxc

```
publicclass AddPrimaryContact implements

  Queueable{ private Contactcon;

  private String state;

  publicAddPrimaryContact(Contact con, String
    State){ this.con = con;
    this.state = state;
  }
  public void execute(QueueableContext context){
  List<Account> accounts = [Select Id, Name, (SelectFirstName,
    LastName,Id from contacts)
    from Account where BillingState = :state Limit200];
    List<Contact> primaryContacts = new List<Contact>();

    for(Account
      acc:accounts){Conta
      ctc = con.clone();
      c.AccountId = acc.Id;
      primaryContacts.add
      (c
      );
    }

    if(primaryContacts.size()
      >0){insert
      primaryContacts;
    }
  }
}
```

AddPrimary ContactTest.apxc

```
@isTest
public class AddPrimaryContactTest {
   static testmethod void
   testQueueable(){
      List<Account>testAccounts = new List<Account>();
      for(Integeri=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');inserttestContact;
      AddPrimaryContact addit = new addPrimaryContact(testContact,

      'CA');Test.startTest();

      System.enqueueJob(addit);
      Test.stopTest();

      System.assertEquals(50,[Select count() from Contact where accountIdin (Select Id from
Accountwhere BillingState='CA')]);

   }
```

Daily LeadProcessor.apxc

```
global class DailyLeadProcessor implements
```

```
  Schedulable{global void
  execute(SchedulableContext ctx) {
    List<Lead> leadstoupdate = new List<Lead>();
    List<Lead> leads = [Select id From Lead Where LeadSource= NULL Limit200];
    for(Lead l: leads) {

      l.LeadSource =
      'Dreamforce';
      leadstoupdate.add(l);
    }
    update leadstoupdate;
  }
}
```

DailyLeadProcessorTest.apxc

```
@isTest
private class DailyLeadProcessorTest {
        public static StringCRON_EXP = '0 0 0 15 3 ?
  2024';static testmethod void testScheduledJob() {
    List<Lead> leads = new List<Lead>();
    for(Integer i = 0; i < 200; i++) {
      Lead l = new Lead(
        FirstName = 'First' + i,
        LastName =
        'LastName',Company=
        'The Inc'
      );
      leads.add(l);
    }
    insert leads;
    Test.startTest(
    );
```

```
        StringjobId =
            System.schedule('ScheduledApexTest',CRON_EXP,new
            DailyLeadProcessor()); Test.stopTest();
        List<Lead> checkleads= new List<Lead>();
        checkleads = [SelectId From Lead Where LeadSource = 'Dreamforce' and Company =
'The Inc'];
        System.assertEquals(200,checkleads.size(),'Leads were not created');
    }
}
```

Apex Integration Services

AnimalLocator.apxc

```
public class AnimalLocator{


    publicstatic 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');
    }
    }
```

AnimalLocatorMock.apxc

```
@isTest
global class AnimalLocatorMock implements HttpCalloutMock {
    /   Implementthis interface method
   global HTTPResponse respond(HTTPRequest request) {
       / Create a fake response
       HttpResponse response = new
       HttpResponse();
       response.setHeader('Content-Type',
       'application/json');
       response.setBody('{"animals": ["majestic badger", "fluffy bunny", "scary bear", "chicken",
"mighty moose"]}');
       response.setStatusCode(20
       0); return response;
   }
}
```

AnimalLocatorTest.apxc

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

ParkLocator.apxc

```
public class ParkLocator {
```

```
    public static string[] country(String country) {
        parkService.parksImplPort park = new
        parkService.parksImplPort();return park.byCountry(country);
    }
}
```

ParkLocatorMock.apxc

```
@isTest
globalclass ParkServiceMock implements WebServiceMock {
    global void doInvoke(
            Object
            stub,
            Object
            request,
            Map<String, Object>
            response,String endpoint,
            String soapAction,
            String
            requestName,
            String
            responseNS,
            StringresponseN
            ame, String
            responseType) {
        parkService.byCountryResponse response_x = new parkService.byCountryResponse();
        response_x.return_x= new List<String>{'Hamburg Wadden Sea NationalPark', 'Hainich
National Park', 'Bavarian ForestNational
        Park'};response.put('response_x', response_x);
    }
}
```

ParkLocatorTest.apxc

```
@isTest
```

```
private class ParkLocatorTest {
   @isTest static void
   testCallout() {
      Test.setMock(WebServiceMock.class, new ParkServiceMock());
      String country='Germany';
      String[] result = ParkLocator.Country(country);


      System.assertEquals(new List<String>{'Hamburg WaddenSea National Park','Hainich
NationalPark', 'Bavarian Forest National Park'},result);
   }
}
```

AccountManager.apxc

```
@RestResource(urlMapping='/Accounts/*/contacts')
global with sharing classAccountManager {



   @HttpGet
   global static account getAccount() {

      RestRequest request =

      RestContext.request;

      StringaccountId = request.requestURI.substring(request.requestURI.lastIndexOf('/')-18,
        request.requestURI.lastIndexOf('/'));
      List<Account> a = [selectid, name, (selectid, name from contacts)from account where id
      =
:accountId];
      List<contact>co = [select id, name from contact where account.id= :accountId];
      system.debug('** a[0]= '+ a[0]);
      return a[0];


   }
```

}

AccountManagerTest.apxc

```
@istest
public class AccountManagerTest {
@istest static void
testGetContactsByAccountId() { Id
recordId=createTestRecord();
/  Set up a test request
RestRequest request = new RestRequest();
request.requestUri =
'https:/yourInstance.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);
}

/ Helper method
static Id createTestRecord() {

/ Create test record
Account accountTest =
new
Account(Name='Test record');
   insert  accountTest;
Contact contactTest =
new
Contact(FirstName='Jo
```

```
hn',
LastName='Doe',
AccountId=accountTest.

Id
);
return accountTest.Id;
}
}
```

Apex Specialistsuper badge

Challenge-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 == 'RoutineMaintenance'){
               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,
Equipmentc, Equipment r.Maintenance_Cycle c,(SELECT Id,Equipment c,Quantity c
FROM Equipment_Maintenance_Items r)

                                        FROMCase 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 :ValidIdsGROUP 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 =
          'RoutineMaintenance',
          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

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

Challenge-2

WarehouseCalloutService.apxc

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

  / class that makes a REST callout to an externalwarehouse system to get a list of
  equipment
that needs to be updated.
  / The callout's JSON response returns the equipmentrecordsthat you upsertin Salesforce.

  @future(callout=tru
  e)publicstatic 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 warehouseSKU
        / warehouse SKU will be external ID for identifying which equipment recordstoupdate
withinSalesforce
        for (Objecteq : jsonResponse){
            Map<String,Object> mapJson=
            (Map<String,Object>)eq; Product2 myEq =
            newProduct2();
            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){
            upsertwarehouseEq;
            System.debug('Your equipmentwas synced with the warehouse one');
        }
    }
  }


  publicstatic void execute (QueueableContext
    context){runWarehouseEquipmentSync();
  }
}
```

Challenge-3

WarehouseSyncSchedule.apxc

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

Challenge-4

MaintenanceRequestHelperTest.apxc

```
@istest
public with sharing class MaintenanceRequestHelperTest {

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

  PRIVATE STATICVehicle  c createVehicle(){
    Vehicle c Vehicle= new Vehicle C(name = 'SuperTruck');
    returnVehicle;
  }

  PRIVATE STATIC Product2 createEq(){
    product2 equipment= new
    product2(name =
                'SuperEquipment',lifespan_months C = 10,
                maintenance_cycle C = 10,
                replacement_partc = true);
    return equipment;
```

```
}

PRIVATESTATIC Case createMaintenanceRequest(id vehicleId,id equipmentId){
  case cs = new case(Type=REPAIR,
            Status=STATUS_NEW,
            Origin=REQUEST_ORIGIN,

            Subject=REQUEST_SUBJECT,

            Equipment
            c=equipmentId,
            Vehiclec=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
privatestatic void
  testMaintenanceRequestPositive(){ Vehiclec
  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   cworkPart = [selectid
                        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.Equipmentc, equipmentId);
    SYSTEM.assertEquals(newReq.Vehicle c,vehicleId);
    SYSTEM.assertEquals(newReq.Date_Reported   c, system.today());
  }

  @istest
  private static void
```

```
    testMaintenanceRequestNegative(){Vehicle
    Cvehicle= 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);
    insertworkP;



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


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


    Equipment_Maintenance_Item   cworkPart = [selectid
                                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(casereq :
     requestList){ req.Status
     = CLOSED;
```

```
        oldRequestIds.add(re
        q.Id);
      }
      updaterequestList;test.stopT
      est();

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



      list<Equipment_Maintenance_Item   c>workParts = [selectid
                                      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 == 'RoutineMaintenance'){
            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,
Equipmentc, Equipment r.Maintenance_Cycle c,(SELECT Id,Equipment c,Quantity c
FROM Equipment_Maintenance_Items r)
                                    FROMCase 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 :ValidIdsGROUP 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 =
          'RoutineMaintenance',
          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

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

Challenge-5

WarehouseCalloutService.apxc

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


   / class that makes a REST callout to an externalwarehouse system to get a list of
equipmentthat needs to be updated.
   / The callout's JSON response returns the equipmentrecordsthat you upsertin Salesforce.


   @future(callout=tru
   e)publicstatic 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 followingfields: replacement part (always true),cost, current
inventory, lifespan, maintenance cycle, and warehouseSKU
        / warehouse SKU will be external ID for identifying which equipment recordstoupdate
withinSalesforce
        for (Objecteq : jsonResponse){
            Map<String,Object> mapJson=
            (Map<String,Object>)eq; Product2 myEq =
            newProduct2();


            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){
            upsertwarehouseEq;
            System.debug('Your equipmentwas synced with the warehouse one');
        }
    }
  }


  publicstatic void execute (QueueableContext
    context){runWarehouseEquipmentSync();
  }


}
```

WarehouseCalloutServiceTest.apxc

```
@IsTest
private class WarehouseCalloutServiceTest {
  / implement your mock callouttest here
        @isTest
  staticvoid testWarehouseCallout() {
    test.startTest();
    test.setMock(HttpCalloutMock.class, new WarehouseCalloutServiceMock());
    WarehouseCalloutService.execute(null);
    test.stopTest();


    List<Product2> product2List = new
    List<Product2>();product2List = [SELECTProductCode
    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);
  }
}
```

WarehouseCalloutServiceMock.apxc

```
@isTest
global class WarehouseCalloutServiceMock implements HttpCalloutMock {
  /  implementhttp mock callout
```

```
   global static HttpResponse respond(HttpRequestrequest){
      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,"n
ame  ": "Generator 1000
kW","maintenanceperiod":365,"lifespan":120,"cost":5000,"sku":"100003"}]');
      response.setStatusCode(20
      0); return response;
   }
}
```

Challenge-6

WarehouseSyncSchedule.apxc

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

WarehouseSyncScheduleTest.apxc

```
@isTest
```

```
public class WarehouseSyncScheduleTest {


   @isTeststatic void WarehousescheduleTest(){
      String scheduleTime = '00 00 01 * * ?';
      Test.startTest();
      Test.setMock(HttpCalloutMock.class, new WarehouseCalloutServiceMock());

      String jobID=System.schedule('Warehouse Time To Scheduleto Test', scheduleTime, new
WarehouseSyncSchedule());
      Test.stopTest();

      / Contains scheduleinformation for a scheduledjob. CronTrigger is similar to a cron job
on UNIX systems.
      / This object is available in API version17.0 and later.

      CronTrigger a=[SELECT Id FROM CronTrigger where NextFireTime > today];
      System.assertEquals(jobID, a.Id,'Schedule ');



   }
}
```

APEX SUPERBADGES