CODES :

1.

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

    for(Account a: Trigger.New){

        if(a.Match_Billing_Address__c == true && a.BillingPostalCode!= null){

            a.ShippingPostalCode=a.BillingPostalCode;

        }

    }

}
```

2.

```
trigger ClosedOpportunityTrigger on Opportunity (after insert, after update) {
   List<Task> taskList = new List<Task>();
   for(Opportunity opp : [SELECT Id, StageName FROM Opportunity WHERE StageName='Closed Won' AND Id IN : Trigger.New]){
      taskList.add(new Task(Subject='Follow Up Test Task', WhatId = opp.Id));
   }
   if(taskList.size()>0){
      insert tasklist;
   }
}
```

1.

```
1  Verifydata
2  //method to handle potential checks against two dates
3
4    public static Date CheckDates(Date date1, Date date2) {
5
6      //if date2 is within the next 30 days of date1, use
   date2.  Otherwise use the end of the month
7
8      if(DateWithin30Days(date1,date2)) {
9
10       return date2;
11
12     } else {
13
14       return SetEndOfMonthDate(date1);
15
16     }
17
18   }
19
20
21   //method to check if date2 is within the next 30 days of
   date1
22
23   private static Boolean DateWithin30Days(Date date1, Date
   date2) {
24
25     //check for date2 being in the past
26
27           if( date2 < date1) { return false; }
28
29
30           //check that date2 is within (>=) 30 days of
```

```
    date1
31
32          Date date30Days = date1.addDays(30); //create a
    date 30 days away from date1
33
34      if( date2 >= date30Days ) { return false; }
35
36      else { return true; }
37
38    }
39
40
41
42    //method to return the end of the month of a given date
43
44    private static Date SetEndOfMonthDate(Date date1) {
45
46      Integer totalDays = Date.daysInMonth(date1.year(),
    date1.month());
47
48      Date lastDay = Date.newInstance(date1.year(),
    date1.month(), totalDays);
49
50      return lastDay;
51
52    }
53
54
55
56 }
```

```
TestVerifyDate
@isTest

public class TestVerifyDate

{
```

```
    static testMethod void testMethod1()

    {

        Date d =
VerifyDate.CheckDates(System.today(),System.today()+1);

        Date d1 =
VerifyDate.CheckDates(System.today(),System.today()+60);

    }

}
```

```
1  Restrictcontactbyname
2
3  trigger RestrictContactByName on Contact (before
   insert, before update) {
4
5    //check contacts prior to insert or update for
   invalid data
6
7    For (Contact c : Trigger.New) {
8
9        if(c.LastName == 'INVALIDNAME') {
   //invalidname is invalid
10
11           c.AddError('The Last Name
   ed for DML');
12       }
```

```
 }
}

wonder studies
Pinned by wonder studies
wonder studies
11 days ago
1.restrictcontactbyname
**********



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 ed for DML');

     }


```

```
40
41  }
42
43
44
45}
46
47
48
49
50
51
52Testrestrictcontactname
53private class TestRestrictContactByName {
54
55
56    static testMethod void  metodoTest()
57
58    {
59
60        List<Contact> listContact= new
  List<Contact>();
61
62        Contact c1 = new
  Contact(FirstName='Francesco', LastName='Riggio' ,
  email='Test@test.com');
63
64        Contact c2 = new
  Contact(FirstName='Francesco1', LastName =
  'INVALIDNAME',email='Test@test.com');
65
```

```
66          listContact.add(c1);
67
68          listContact.add(c2);
69
70          Test.startTest();
71
72              try
73
74              {
75
76                  insert listContact;
77
78              }
79
80              catch(Exception ee)
81
82              {
83
84              }
85
86          Test.stopTest();
87
88      }
89
90}
```

**3.**

```
1   Public class RandomContactFactory {
2
3      public static List<Contact> generateRandomContacts(Integer
   numContactsToGenerate, String FName) {
```

```
4
5           List<Contact> contactList = new List<Contact>();
6
7
8       for(Integer i=0;i<numContactsToGenerate;i++) {
9
10              Contact c = new Contact(FirstName=FName + ' ' + i,
    LastName = 'Contact '+i);
11
12              contactList.add(c);
13
14              System.debug(c);
15
16          }
17
18          //insert contactList;
19
20          System.debug(contactList.size());
21
22          return contactList;
23
24      }
25  }
```

1.

```
1   AccountProcessor
2
3   public class AccountProcessor {
4
5       public static void countContacts(List<Id> accountIds){
6
7           List<Account> accounts = [Select Id, Name from Account
    Where Id IN : accountIds];
8
9           List<Account> updatedAccounts = new List<Account>();
10
11          for(Account account : accounts){
```

```
12
13            account.Number_of_Contacts__c = [Select count() from
  Contact Where AccountId =: account.Id];
14
15             System.debug('No Of Contacts = ' +
  account.Number_of_Contacts__c);
16
17             updatedAccounts.add(account);
18
19         }
20
21         update updatedAccounts;
22
23     }
24
25 }
26
27  AccountProcessorTest
28
29 public class AccountProcessorTest {
30
31     public static void testNoOfContacts(){
32
33         Account a = new Account();
34
35         a.Name = 'Test Account';
36
37         Insert a;
38
39
40         Contact c = new Contact();
41
42         c.FirstName = 'Bob';
43
44         c.LastName =  'Willie';
45
46         c.AccountId = a.Id;
47
48
49         Contact c2 = new Contact();
```

```
50
51          c2.FirstName = 'Tom';
52
53          c2.LastName = 'Cruise';
54
55          c2.AccountId = a.Id;
56
57
58          List<Id> acctIds = new List<Id>();
59
60          acctIds.add(a.Id);
61
62
63          Test.startTest();
64
65          AccountProcessor.countContacts(acctIds);
66
67          Test.stopTest();
68
69      }
70
71  }
```

**2.**

```
1   LeadProcessor
2   public class LeadProcessor implements Database.Batchable<sObject>
    {
3
4
5       public Database.QueryLocator start(Database.BatchableContext
    bc) {
6
7           // collect the batches of records or objects to be passed
    to execute
8
9             return Database.getQueryLocator([Select LeadSource From
    Lead ]);
```

```
10
11      }
12
13      public void execute(Database.BatchableContext bc, List<Lead>
    leads){
14
15           // process each batch of records
16
17              for (Lead Lead : leads) {
18
19                  lead.LeadSource = 'Dreamforce';
20
21              }
22
23          update leads;
24
25      }
26
27      public void finish(Database.BatchableContext bc){
28
29        }
30
31  }
32  .
33
34  LeadProcessorTest
35
36  public class LeadProcessorTest {
37
38          @testSetup
39
40      static void setup() {
41
42          List<Lead> leads = new List<Lead>();
43
44          for(Integer counter=0 ;counter <200;counter++){
45
46              Lead lead = new Lead();
47
48              lead.FirstName ='FirstName';
```

```
49
50              lead.LastName ='LastName'+counter;
51
52              lead.Company ='demo'+counter;
53
54              leads.add(lead);
55
56          }
57
58          insert leads;
59
60      }
61
62
63       static void test() {
64
65          Test.startTest();
66
67          LeadProcessor leadProcessor = new LeadProcessor();
68
69          Id batchId = Database.executeBatch(leadProcessor);
70
71          Test.stopTest();
72
73      }
74
75 }
```

3.

```
1 AddPrimaryContac
2
3 public class AddPrimaryContact implements Queueable
4
5 {
6
7     private Contact c;
```

```
8
9       private String state;
10
11      public  AddPrimaryContact(Contact c, String
  state)
12
13      {
14
15          this.c = c;
16
17          this.state = state;
18
19      }
20
21      public void execute(QueueableContext context)
22
23      {
24
25          List<Account> ListAccount = [SELECT ID,
  Name ,(Select id,FirstName,LastName from contacts )
  FROM ACCOUNT WHERE BillingState = :state LIMIT
  200];
26
27          List<Contact> lstContact = new
  List<Contact>();
28
29          for (Account acc:ListAccount)
30
31          {
32
33                  Contact cont =
```

```
  c.clone(false,false,false,false);
34
35                  cont.AccountId =  acc.id;
36
37                  lstContact.add( cont );
38
39          }
40
41
42          if(lstContact.size() >0 )
43
44          {
45
46              insert lstContact;
47
48          }
49
50
51      }
52
53
54
55}
56
57
58 AddPrimaryContactTest
59
60public class AddPrimaryContactTest
61
62{
63
```

```
64      @isTest static void TestList()
65
66      {
67
68          List<Account> Teste = new List
  <Account>();
69
70          for(Integer i=0;i<50;i++)
71
72          {
73
74              Teste.add(new Account(BillingState =
  'CA', name = 'Test'+i));
75
76          }
77
78          for(Integer j=0;j<50;j++)
79
80          {
81
82              Teste.add(new Account(BillingState =
  'NY', name = 'Test'+j));
83
84          }
85
86          insert Teste;
87
88
89
90          Contact co = new Contact();
91
```

```
92          co.FirstName='demo';
93
94          co.LastName ='demo';
95
96          insert co;
97
98          String state = 'CA';
99
100
101           AddPrimaryContact apc = new
  AddPrimaryContact(co, state);
102
103           Test.startTest();
104
105             System.enqueueJob(apc);
106
107           Test.stopTest();
108
109       }
110
111 }
```

4.

```
1 DailyLeadProcessor
2
3 public class DailyLeadProcessor implements
  Schedulable  {
4
5     Public void execute(SchedulableContext SC){
6
```

```
7         List<Lead> LeadObj=[SELECT Id from Lead
  where LeadSource=null limit 200];
8
9          for(Lead l:LeadObj){
10
11             l.LeadSource='Dreamforce';
12
13             update l;
14
15         }
16
17     }
18
19}
20
21
22
23DailyLeadProcessorTest
24
25private class DailyLeadProcessorTest {
26
27 static testMethod void testDailyLeadProcessor() {
28
29      String CRON_EXP = '0 0 1 * * ?';
30
31      List<Lead> lList = new List<Lead>();
32
33      for (Integer i = 0; i < 200; i++) {
34
35          lList.add(new
  Lead(LastName='Dreamforce'+i, Company='Test1 Inc.',
```

```
  Status='Open - Not Contacted'));
36
37      }
38
39      insert lList;
40
41      Test.startTest();
42
43      String jobId =
  System.schedule('DailyLeadProcessor', CRON_EXP, new
  DailyLeadProcessor());
44
45 }
46
47}
```

1.

```
1 AnimalLocator
2 public class AnimalLocator{
3
4     public static String getAnimalNameById(Integer
  x){
5
6         Http http = new Http();
7
8         HttpRequest req = new HttpRequest();
9
10        req.setEndpoint('https://th-apex-http-
```

```
11
12          req.setMethod('GET');
13
14          Map<String, Object> animal= new Map<String,
  Object>();
15
16          HttpResponse res = http.send(req);
17
18              if (res.getStatusCode() == 200) {
19
20          Map<String, Object> results = (Map<String,
  Object>)JSON.deserializeUntyped(res.getBody());
21
22        animal = (Map<String, Object>)
  results.get('animal');
23
24          }
25
26return (String)animal.get('name');
27
28      }
29
30}
31
32 AnimalLocatorTest
33
34private class AnimalLocatorTest{
35
36      @isTest static void AnimalLocatorMock1() {
37
38          Test.setMock(HttpCalloutMock.class, new
```

```
  AnimalLocatorMock());
39
40          string result =
  AnimalLocator.getAnimalNameById(3);
41
42          String expectedResult = 'chicken';
43
44          System.assertEquals(result,expectedResult
  );
45
46      }
47
48}
49
50 AnimalLocatorMock
51
52
53global class AnimalLocatorMock implements
  HttpCalloutMock {
54
55       // Implement this interface method
56
57      global HTTPResponse respond(HTTPRequest
  request) {
58
59          // Create a fake response
60
61          HttpResponse response = new HttpResponse();
62
63          response.setHeader('Content-Type',
  'application/json');
```

```
64
65          response.setBody('{"animals": ["majestic


66
67          response.setStatusCode(200);
68
69          return response;
70
71      }
72
73}
```

2.

```
1 3.1 ParkLocator
2
3 ****************
4 public class ParkLocator {
5
6      public static string[] country(string
  theCountry) {
7
8          ParkService.ParksImplPort  parkSvc = new
  ParkService.ParksImplPort(); // remove space
9
10         return parkSvc.byCountry(theCountry);
11
12     }
13
14}
15
```

```
16 ParkLocatorTest
17
18private class ParkLocatorTest {
19
20     static void testCallout() {
21
22         Test.setMock(WebServiceMock.class, new
  ParkServiceMock ());
23
24         String country = 'United States';
25
26         List<String> result =
  ParkLocator.country(country);
27
28         List<String> parks = new
  List<String>{'Yellowstone', 'Mackinac National

29
30          System.assertEquals(parks, result);
31
32     }
33
34}
35
36
37
38 ParkServiceMock
39
40global class ParkServiceMock implements
  WebServiceMock {
41
```

```
42    global void doInvoke(
43
44            Object stub,
45
46            Object request,
47
48            Map<String, Object> response,
49
50            String endpoint,
51
52            String soapAction,
53
54            String requestName,
55
56            String responseNS,
57
58            String responseName,
59
60            String responseType) {
61
62        // start - specify the response you want to
  send
63
64        ParkService.byCountryResponse response_x =
  new ParkService.byCountryResponse();
65
66        response_x.return_x = new
  List<String>{'Yellowstone', 'Mackinac National

67
68        // end
```

```
69
70          response.put('response_x', response_x);
71
72      }
73
74}
```

3.

```
1 AccountManager
2
3 (urlMapping='/Accounts/*/contacts')
4
5 global class AccountManager {
6
7
8      global static Account getAccount() {
9
10         RestRequest req = RestContext.request;
11
12         String accId =
  req.requestURI.substringBetween('Accounts/',
  '/contacts');
13
14         Account acc = [SELECT Id, Name, (SELECT Id,
  Name FROM Contacts)
15
16                        FROM Account WHERE Id =
  :accId];
17
```

```
18         return acc;
19
20     }
21
22}
23
24AccountManagerTest
25
26
27private class AccountManagerTest {
28
29
30
31     private static testMethod void
   getAccountTest1() {
32
33         Id recordId = createTestRecord();
34
35         // Set up a test request
36
37         RestRequest request = new RestRequest();
38
39         request.requestUri =
   'https://na1.salesforce.com/services/apexrest/Accou

40
41         request.httpMethod = 'GET';
42
43         RestContext.request = request;
44
45         // Call the method to test
```

```
46
47          Account thisAccount =
  AccountManager.getAccount();
48
49          // Verify results
50
51          System.assert(thisAccount != null);
52
53          System.assertEquals('Test record',
  thisAccount.Name);
54
55
56      }
57      // Helper method
58
59          static Id createTestRecord() {
60
61          // Create test record
62
63          Account TestAcc = new Account(
64
65            Name='Test record');
66
67          insert TestAcc;
68
69          Contact TestCon= new Contact(
70
71          LastName='Test',
72
73          AccountId = TestAcc.id);
74
```

```
        return TestAcc.Id;

    }

}
```

```
MaintenanceRequestHelper

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){
```

```
12
13              if
  (nonUpdCaseMap.get(c.Id).Status !=
  'Closed' && c.Status == 'Closed'){
14
15                  if (c.Type == 'Repair'
  || c.Type == 'Routine Maintenance'){
16
17                      validIds.add(c.Id);
18
19
20
21                  }
22
23              }
24
25          }
26
27
28          if (!validIds.isEmpty()){
29
30              List<Case> newCases = new
  List<Case>();
31
```

```
32              Map<Id,Case> closedCasesM =
  new Map<Id,Case>([SELECT Id,
  Vehicle__c, Equipment__c,
  Equipment__r.Maintenance_Cycle__c,(SELE

  Equipment_Maintenance_Items__r)
33
34
  FROM Case WHERE Id IN :validIds]);
35
36              Map<Id,Decimal>
  maintenanceCycles = new
  Map<ID,Decimal>();
37
38              AggregateResult[] results =
  [SELECT Maintenance_Request__c,
  MIN(Equipment__r.Maintenance_Cycle__c)cy

  WHERE Maintenance_Request__c IN
  :ValidIds GROUP BY
  Maintenance_Request__c];
39
40
41          for (AggregateResult ar :
```

```
  results){
42
43              maintenanceCycles.put((Id)
  ar.get('Maintenance_Request__c'),
  (Decimal) ar.get('cycle'));
44
45          }
46
47
48              for(Case cc :
  closedCasesM.values()){
49
50                  Case nc = new Case (
51
52                      ParentId = cc.Id,
53
54                  Status = 'New',
55
56                      Subject = 'Routine

57
58                      Type = 'Routine

59
```

```
60                      Vehicle__c =
  cc.Vehicle__c,
61
62                      Equipment__c
  =cc.Equipment__c,
63
64                      Origin = 'Web',
65
66                      Date_Reported__c =
  Date.Today()
67
68
69                  );
70
71
72                  If
  (maintenanceCycles.containskey(cc.Id)){
73
74                      nc.Date_Due__c =
  Date.today().addDays((Integer)
  maintenanceCycles.get(cc.Id));
75
76                  } else {
77
```

```
78                      nc.Date_Due__c =
  Date.today().addDays((Integer)
  cc.Equipment__r.maintenance_Cycle__c);
79
80                  }
81
82
83                  newCases.add(nc);
84
85              }
86
87              insert newCases
88
  List<Equipment_Maintenance_Item__c>
  clonedWPs = new
  List<Equipment_Maintenance_Item__c>();
89
90              for (Case nc : newCases){
91
92                  for
  (Equipment_Maintenance_Item__c wp :
  closedCasesM.get(nc.ParentId).Equipment_

93
```

```
94
  Equipment_Maintenance_Item__c wpClone =
  wp.clone();
95
96
  wpClone.Maintenance_Request__c = nc.Id;
97
98
  ClonedWPs.add(wpClone);
99
100
101                     }
102
103                 }
104
105                 insert ClonedWPs;
106
107             }
108
109         }
110
111     }
112
113
```

```
114
115     MaitenanceRequest
116
117     trigger MaintenanceRequest on Case
  (before update, after update) {
118
119
120
121         if(Trigger.isUpdate &&
  Trigger.isAfter){
122         }
123     }
```

2.

```
1  WarehouseCalloutService
2
3
4 public with sharing class
  WarehouseCalloutService implements Queueable
  {
5
6     private static final String
  WAREHOUSE_URL = 'https://th-superbadge-
```

```
7
8
9       //class that makes a REST callout to an external warehouse system to get a list of equipment that needs to be updated.
10
11       //The callout's JSON response returns the equipment records that you upsert in Salesforce.
12
13       public static void runWarehouseEquipmentSync(){
14
15           Http http = new Http();
16
17           HttpRequest request = new HttpRequest();
18
19
20           request.setEndpoint(WAREHOUSE_URL);
21
22           request.setMethod('GET');
23
24           HttpResponse response = http.send(request);
25
26
```

```
27          List<Product2> warehouseEq = new
  List<Product2>();
28
29
30          if (response.getStatusCode() ==
  200){
31
32              List<Object> jsonResponse =
  (List<Object>)JSON.deserializeUntyped(respons

33
34
  System.debug(response.getBody());
35
36
37              //class maps the following
  fields: replacement part (always true),
  cost, current inventory, lifespan,
  maintenance cycle, and warehouse SKU
38
39              //warehouse SKU will be external
  ID for identifying which equipment records
  to update within Salesforce
40
41              for (Object eq : jsonResponse){
42
43                  Map<String,Object> mapJson =
```

```
  (Map<String,Object>)eq;
44
45                  Product2 myEq = new
  Product2();
46
47                  myEq.Replacement_Part__c =
  (Boolean) mapJson.get('replacement');
48
49                  myEq.Name = (String)
  mapJson.get('name');
50
51                  myEq.Maintenance_Cycle__c =
  (Integer) mapJson.get('maintenanceperiod');
52
53                  myEq.Lifespan_Months__c =
  (Integer) mapJson.get('lifespan');
54
55                  myEq.Cost__c = (Integer)
  mapJson.get('cost');
56
57                  myEq.Warehouse_SKU__c =
  (String) mapJson.get('sku');
58
59                  myEq.Current_Inventory__c =
  (Double) mapJson.get('quantity');
60
61                  myEq.ProductCode = (String)
```

```
  mapJson.get('_id');
62
63                  warehouseEq.add(myEq);
64
65              }
66
67
68              if (warehouseEq.size() > 0){
69
70                  upsert warehouseEq;
71
72                  System.debug('Your equipment
73
74              }
75
76          }
77
78      }
79
80
81      public static void execute
  (QueueableContext context){
82
83          runWarehouseEquipmentSync();
84
85      }
```

```
86
87
88 }
89
90 After saving the code open execute anonymous
   window ( CTRl+E ) and run this method ,
91
92 System.enqueueJob(new
   WarehouseCalloutService());
```

2.

```
1   WarehouseSyncShedule
2
3  global with sharing class WarehouseSyncSchedule implements
   Schedulable{
4
5      global void execute(SchedulableContext ctx){
6
7          System.enqueueJob(new WarehouseCalloutService());
8
9      }
10
11 }
```

3.

```
1  MaintenanceRequestHelperTest
2  ********************************
3
4
5  @istest
```

```
6
7  public with sharing class MaintenanceRequestHelperTest {
8
9
10     private static final string STATUS_NEW = 'New';
11
12     private static final string WORKING = 'Working';
13
14     private static final string CLOSED = 'Closed';
15
16     private static final string REPAIR = 'Repair';
17
18     private static final string REQUEST_ORIGIN = 'Web';
19
20     private static final string REQUEST_TYPE = 'Routine

21
22     private static final string REQUEST_SUBJECT = 'Testing

23
24
25     PRIVATE STATIC Vehicle__c createVehicle(){
26
27         Vehicle__c Vehicle = new Vehicle__C(name = 'SuperTruck');
28
29         return Vehicle;
30
31     }
32
33
34     PRIVATE STATIC Product2 createEq(){
35
36         product2 equipment = new product2(name =
   'SuperEquipment',
37
38                                           lifespan_months__C = 10,
39
40                                           maintenance_cycle__C =
   10,
41
```

```
                                        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;

    }
```

```
77
78
79
80      @istest
81
82      private static void testMaintenanceRequestPositive(){
83
84          Vehicle__c vehicle = createVehicle();
85
86          insert vehicle;
87
88          id vehicleId = vehicle.Id;
89
90
91          Product2 equipment = createEq();
92
93          insert equipment;
94
95          id equipmentId = equipment.Id;
96
97
98          case somethingToUpdate =
    createMaintenanceRequest(vehicleId,equipmentId);
99
100          insert somethingToUpdate;
101
102
103          Equipment_Maintenance_Item__c workP =
    createWorkPart(equipmentId,somethingToUpdate.id);
104
105          insert workP;
106
107
108          test.startTest();
109
110          somethingToUpdate.status = CLOSED;
111
112          update somethingToUpdate;
113
114          test.stopTest();
```

```
115
116
117         Case newReq = [Select id, subject, type, Equipment__c,
    Date_Reported__c, Vehicle__c, Date_Due__c
118
119                         from case
120
121                         where status =:STATUS_NEW];
122
123
124         Equipment_Maintenance_Item__c workPart = [select id
125
126                                                 from
    Equipment_Maintenance_Item__c
127
128                                                 where
    Maintenance_Request__c =:newReq.Id];
129
130
131         system.assert(workPart != null);
132
133         system.assert(newReq.Subject != null);
134
135         system.assertEquals(newReq.Type, REQUEST_TYPE);
136
137         SYSTEM.assertEquals(newReq.Equipment__c, equipmentId);
138
139         SYSTEM.assertEquals(newReq.Vehicle__c, vehicleId);
140
141         SYSTEM.assertEquals(newReq.Date_Reported__c,
    system.today());
142
143     }
144
145
146     @istest
147
148     private static void testMaintenanceRequestNegative(){
149
150         Vehicle__C vehicle = createVehicle();
```

```
151
152         insert vehicle;
153
154         id vehicleId = vehicle.Id;
155
156
157         product2 equipment = createEq();
158
159         insert equipment;
160
161         id equipmentId = equipment.Id;
162
163
164         case emptyReq =
   createMaintenanceRequest(vehicleId,equipmentId);
165
166         insert emptyReq;
167
168
169         Equipment_Maintenance_Item__c workP =
   createWorkPart(equipmentId, emptyReq.Id);
170
171         insert workP;
172
173
174         test.startTest();
175
176         emptyReq.Status = WORKING;
177
178         update emptyReq;
179
180         test.stopTest();
181
182
183         list<case> allRequest = [select id
184
185                                  from case];
186
187
188         Equipment_Maintenance_Item__c workPart = [select id
```

```
189
190                                                     from
   Equipment_Maintenance_Item__c
191
192                                                     where
   Maintenance_Request__c = :emptyReq.Id];
193
194
195         system.assert(workPart != null);
196
197         system.assert(allRequest.size() == 1);
198
199     }
200
201
202     @istest
203
204     private static void testMaintenanceRequestBulk(){
205
206         list<Vehicle__C> vehicleList = new list<Vehicle__C>();
207
208         list<Product2> equipmentList = new list<Product2>();
209
210         list<Equipment_Maintenance_Item__c> workPartList = new
   list<Equipment_Maintenance_Item__c>();
211
212         list<case> requestList = new list<case>();
213
214         list<id> oldRequestIds = new list<id>();
215
216
217         for(integer i = 0; i < 300; i++){
218
219            vehicleList.add(createVehicle());
220
221             equipmentList.add(createEq());
222
223         }
224
225         insert vehicleList;
```

```
226
227         insert equipmentList;
228
229
230         for(integer i = 0; i < 300; i++){
231
232
   requestList.add(createMaintenanceRequest(vehicleList.get(i).id,
   equipmentList.get(i).id));
233
234         }
235
236         insert requestList;
237
238
239         for(integer i = 0; i < 300; i++){
240
241
   workPartList.add(createWorkPart(equipmentList.get(i).id,
   requestList.get(i).id));
242
243         }
244
245         insert workPartList;
246
247
248         test.startTest();
249
250         for(case req : requestList){
251
252             req.Status = CLOSED;
253
254             oldRequestIds.add(req.Id);
255
256         }
257
258         update requestList;
259
260         test.stopTest();
261
```

```
262
263         list<case> allRequests = [select id
264
265                                   from case
266
267                                   where status =: STATUS_NEW];
268
269
270         list<Equipment_Maintenance_Item__c> workParts = [select
    id
271
272                                                          from
    Equipment_Maintenance_Item__c
273
274                                                          where
    Maintenance_Request__c in: oldRequestIds];
275
276
277         system.assert(allRequests.size() == 300);
278
279     }
280
281 }
282
283
284
285
286
287 5.2MaintenanceRequestHelper
288 ****************************
289
290 public with sharing class MaintenanceRequestHelper {
291
292     public static void updateworkOrders(List<Case>
    updWorkOrders, Map<Id,Case> nonUpdCaseMap) {
293
294         Set<Id> validIds = new Set<Id>();
295
296
297
```

```
298         For (Case c : updWorkOrders){
299 
300             if (nonUpdCaseMap.get(c.Id).Status != 'Closed' &&
    c.Status == 'Closed'){
301 
302                 if (c.Type == 'Repair' || c.Type == 'Routine
303 
304                     validIds.add(c.Id);
305 
306 
307 
308                 }
309 
310             }
311 
312         }
313 
314 
315         if (!validIds.isEmpty()){
316 
317             List<Case> newCases = new List<Case>();
318 
319             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)
320 
321                                                           FROM
    Case WHERE Id IN :validIds]);
322 
323             Map<Id,Decimal> maintenanceCycles = new
    Map<ID,Decimal>();
324 
325             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];
326 
```

```
327
328         for (AggregateResult ar : results){
329
330             maintenanceCycles.put((Id)
   ar.get('Maintenance_Request__c'), (Decimal) ar.get('cycle'));
331
332         }
333
334
335             for(Case cc : closedCasesM.values()){
336
337                 Case nc = new Case (
338
339                     ParentId = cc.Id,
340
341                 Status = 'New',
342
343                     Subject = 'Routine Maintenance',
344
345                     Type = 'Routine Maintenance',
346
347                     Vehicle__c = cc.Vehicle__c,
348
349                     Equipment__c =cc.Equipment__c,
350
351                     Origin = 'Web',
352
353                     Date_Reported__c = Date.Today()
354
355
356                 );
357
358
359                 If (maintenanceCycles.containskey(cc.Id)){
360
361                     nc.Date_Due__c =
   Date.today().addDays((Integer) maintenanceCycles.get(cc.Id));
362
363                 }
364
```

```
365
366                 newCases.add(nc);
367
368             }
369
370
371             insert newCases;
372
373
374             List<Equipment_Maintenance_Item__c> clonedWPs = new
    List<Equipment_Maintenance_Item__c>();
375
376             for (Case nc : newCases){
377
378                 for (Equipment_Maintenance_Item__c wp :
    closedCasesM.get(nc.ParentId).Equipment_Maintenance_Items__r){
379
380                     Equipment_Maintenance_Item__c wpClone =
    wp.clone();
381
382                     wpClone.Maintenance_Request__c = nc.Id;
383
384                     ClonedWPs.add(wpClone);
385
386
387                 }
388
389             }
390
391             insert ClonedWPs;
392
393         }
394
395     }
396
397 }
398
399
400
401 5.3 MaintenanceRequest
```

```
402 **********************
403
404 trigger MaintenanceRequest on Case (before update, after update)
    {
405
406     if(Trigger.isUpdate && Trigger.isAfter){
407
408         MaintenanceRequestHelper.updateWorkOrders(Trigger.New,
    Trigger.OldMap);
409
410     }
411
412 }
```

## 5.

```
1  WarehouseCalloutService
2  ***************************
3
4
5  public with sharing class WarehouseCalloutService {
6
7
8
9      private static final String WAREHOUSE_URL = 'https://th-

10
11
12     //@future(callout=true)
13
14     public static void runWarehouseEquipmentSync(){
15
16
17         Http http = new Http();
18
19         HttpRequest request = new HttpRequest();
20
21
```

```
22          request.setEndpoint(WAREHOUSE_URL);
23
24          request.setMethod('GET');
25
26          HttpResponse response = http.send(request);
27
28
29
30          List<Product2> warehouseEq = new List<Product2>();
31
32
33          if (response.getStatusCode() == 200){
34
35              List<Object> jsonResponse =
    (List<Object>)JSON.deserializeUntyped(response.getBody());
36
37              System.debug(response.getBody());
38
39
40              for (Object eq : jsonResponse){
41
42                  Map<String,Object> mapJson =
    (Map<String,Object>)eq;
43
44                  Product2 myEq = new Product2();
45
46                  myEq.Replacement_Part__c = (Boolean)
    mapJson.get('replacement');
47
48                  myEq.Name = (String) mapJson.get('name');
49
50                  myEq.Maintenance_Cycle__c = (Integer)
    mapJson.get('maintenanceperiod');
51
52                  myEq.Lifespan_Months__c = (Integer)
    mapJson.get('lifespan');
53
54                  myEq.Cost__c = (Decimal) mapJson.get('lifespan');
55
56                  myEq.Warehouse_SKU__c = (String)
```

```
    mapJson.get('sku');
57
58                  myEq.Current_Inventory__c = (Double)
    mapJson.get('quantity');
59
60                  warehouseEq.add(myEq);
61
62              }
63
64
65              if (warehouseEq.size() > 0){
66
67                  upsert warehouseEq;
68
69                  System.debug('Your equipment was synced with the

70
71                  System.debug(warehouseEq);
72
73              }
74
75
76          }
77
78      }
79
80  }
81
82
83
84  6.2 WarehouseCalloutServiceTest
85  **********************************
86  @isTest
87
88
89
90  private class WarehouseCalloutServiceTest {
91
92      @isTest
93
```

```
94      static void testWareHouseCallout(){
95
96          Test.startTest();
97
98          // implement mock callout test here
99
100          Test.setMock(HTTPCalloutMock.class, new
    WarehouseCalloutServiceMock());
101
102          WarehouseCalloutService.runWarehouseEquipmentSync();
103
104          Test.stopTest();
105
106          System.assertEquals(1, [SELECT count() FROM Product2]);
107
108      }
109
110 }
111
112
113
114 6.3 WarehouseCalloutServiceMock
115 ********************************
116
117 @isTest
118
119 global class WarehouseCalloutServiceMock implements
    HttpCalloutMock {
120
121      // implement http mock callout
122
123      global static HttpResponse respond(HttpRequest request){
124
125
126          System.assertEquals('https://th-superbadge-
    ));
127
128          System.assertEquals('GET', request.getMethod());
129
130
```

```
131         // Create a fake response
132
133         HttpResponse response = new HttpResponse();
134
135         response.setHeader('Content-Type', 'application/json');
136
137
   response.setBody('[{"_id":"55d66226726b611100aaf741","replacement
138
139         response.setStatusCode(200);
140
141         return response;
142
143     }
144
145 }
```

## 5.

```
1  WarehouseSyncSchedule
2
3  global class WarehouseSyncSchedule implements Schedulable {
4      global void execute(SchedulableContext ctx) {
5          WarehouseCalloutService.runWarehouseEquipmentSync();
6      }
7  }
8  WarehouseSyncScheduleTest
9
10 public class WarehouseSyncScheduleTest {
11     t static void WarehousescheduleTest(){
12         String scheduleTime = '00 00 01 * * ?';
13         Test.startTest();
14         Test.setMock(HttpCalloutMock.class, new
   WarehouseCalloutServiceMock());
15         String jobID=System.schedule('Warehouse Time To Schedule
```

```
        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 ');
    }
}
```