NAME: JAHNAVI MACHUPALLY

APEX TRIGGERS:

1. Get Started with Apex Triggers:

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

for(Account account:Trigger.New){

if(account.Match\_Billing\_Address\_\_c == True){

account.ShippingPostalCode=account.BillingPostalCode;

}

}

}

2. Bulk Apex Triggers:

trigger ClosedOpportunityTrigger on Opportunity (after insert, before 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;

}

}

APEX TESTING:

1. Get Started with Apex Unit Tests:

VerifyDate.apxc:

public class VerifyDate {

//method to handle potential checks against two dates

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

//if date2 is within the next 30 days of date1, use date2. Otherwise use the end of the month

if(DateWithin30Days(date1,date2)) {

return date2;

} else {

return SetEndOfMonthDate(date1);

}

}

//method to check if date2 is within the next 30 days of date1

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

//check for date2 being in the past

if( date2 < date1) { return false; }

//check that date2 is within (>=) 30 days of date1

Date date30Days = date1.addDays(30); //create a date 30 days away from date1

if( date2 >= date30Days ) { return false; }

else { return true; }

}

//method to return the end of the month of a given date

@TestVisible private static Date SetEndOfMonthDate(Date date1) {

Integer totalDays = Date.daysInMonth(date1.year(), date1.month());

Date lastDay = Date.newInstance(date1.year(), date1.month(), totalDays);

return lastDay;

}

}

TestVerifyDate.apxc:

@isTest

private class TestVerifyDate {

@isTest static void Test\_CheckDates\_case1(){

Date D=VerifyDate.CheckDates(date.parse('01/01/2020'), date.parse('01/05/2020'));

System.assertEquals(date.parse('01/05/2020'),D);

}

@isTest static void Test\_CheckDates\_case2(){

Date D=VerifyDate.CheckDates(date.parse('01/01/2020'), date.parse('05/05/2020'));

System.assertEquals(date.parse('01/31/2020'),D);

}

@isTest static void Test\_DateWithin30Days\_case1(){

Boolean flag = VerifyDate.DateWithin30Days(date.parse('01/01/2020'), date.parse('12/30/2019'));

System.assertEquals(false, flag);

}

@isTest static void Test\_DateWithin30Days\_case2(){

Boolean flag = VerifyDate.DateWithin30Days(date.parse('01/01/2020'), date.parse('02/02/2020'));

System.assertEquals(false, flag);

}

@isTest static void Test\_DateWithin30Days\_case3(){

Boolean flag = VerifyDate.DateWithin30Days(date.parse('01/01/2020'), date.parse('01/15/2020'));

System.assertEquals(true, flag);

}

@isTest static void Test\_SetEndOfMonthDate(){

Date returndate = VerifyDate.SetEndOfMonthDate(date.parse('01/01/2020'));

}

}

1. Test Apex Triggers:

TestRestrictContactByName.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().size()>0);

System.assertEquals('The Last Name "INVALIDNAME" is not allowed for DML', result.getErrors()[0].getMessage());

}

}

RestrictContactByName.apxc:

trigger RestrictContactByName on Contact(before insert, before update){

For ((Contact c: Trigger.New)){

if(c.LastName == 'INVALIDNAME'){

c.AddError('The Last Name "'+c.LastName+'" is not allowed for DML');

}

}

}

1. Create Test Data for Apex Tests:

public class RandomContactFactory {

public static List<Contact> generateRandomContacts(Integer numcnt, string lastname){

List<Contact> contacts = new List<Contact>();

for(Integer i=0;i<numcnt;i++){

Contact cnt=new Contact(FirstName = 'Test' +i, LastName = lastname);

contacts.add(cnt);

}

return contacts;

}

}

APEX INTEGRATION SERVICES:

1. Apex REST Callouts:

AnimalLocator.apxc:

public class AnimalLocator{

public static String getAnimalNameById (Integer i) {

Http http = new Http();

HttpRequest request = new HttpRequest();

request.setEndpoint('https://th-apex-http-callout.herokuapp.com/animals/'+i);

request.setMethod('GET');

HttpResponse response = http.send(request);

Map<String, Object> result = (Map<String, Object>)JSON.deserializeUntyped(response.getBody());

Map<String, Object> animal = (Map<String, Object>)result.get('animal');

System.debug('name: '+string.valueOf(animal.get('name')));

return string.valueOf(animal.get('name'));

}

}

AnimalLocatorTest.apxc:

@isTest

private class AnimalLocatorTest{

@isTest

static void animalLocatorTest1() {

// Set mock callout class

Test.setMock(HttpCalloutMock.class, new AnimalLocatorMock());

String actual = AnimalLocator.getAnimalNameById(1);

String expected = 'moose';

// Verify that the response received contains fake values

System.assertEquals(actual, expected);

}

}

AnimalLocatorMock.apxc:

@isTest

global class AnimalLocatorMock implements HttpCalloutMock {

// Implement this interface method

global HTTPResponse respond(HTTPRequest request) {

// Create a fake response

HttpResponse response = new HttpResponse();

response.setHeader('contentType', 'application/json');

response.setBody('{"animal":{"id":1,"name":"moose","eats":"plants","says":"bellows"}}');

response.setStatusCode(200);

return response;

}

}

1. Apex SOAP Callouts:

ParkLocator.apxc:

public class ParkLocator {

public static List < String > country(String country){

ParkService.ParksImplPort prkSvc = new ParkService.ParksImplPort();

return prkSvc.byCountry(country);

}

}

ParkService.apxc:

public class ParkService {

public class byCountryResponse {

public String[] return\_x;

private String[] return\_x\_type\_info = new String[]{'return','http://parks.services/',null,'0','-1','false'};

private String[] apex\_schema\_type\_info = new String[]{'http://parks.services/','false','false'};

private String[] field\_order\_type\_info = new String[]{'return\_x'};

}

public class byCountry {

public String arg0;

private String[] arg0\_type\_info = new String[]{'arg0','http://parks.services/',null,'0','1','false'};

private String[] apex\_schema\_type\_info = new String[]{'http://parks.services/','false','false'};

private String[] field\_order\_type\_info = new String[]{'arg0'};

}

public class ParksImplPort {

public String endpoint\_x = 'https://th-apex-soap-service.herokuapp.com/service/parks';

public Map<String,String> inputHttpHeaders\_x;

public Map<String,String> outputHttpHeaders\_x;

public String clientCertName\_x;

public String clientCert\_x;

public String clientCertPasswd\_x;

public Integer timeout\_x;

private String[] ns\_map\_type\_info = new String[]{'http://parks.services/', 'ParkService'};

public String[] byCountry(String arg0) {

ParkService.byCountry request\_x = new ParkService.byCountry();

request\_x.arg0 = arg0;

ParkService.byCountryResponse response\_x;

Map<String, ParkService.byCountryResponse> response\_map\_x = new Map<String, ParkService.byCountryResponse>();

response\_map\_x.put('response\_x', response\_x);

WebServiceCallout.invoke(

this,

request\_x,

response\_map\_x,

new String[]{endpoint\_x,

'',

'http://parks.services/',

'byCountry',

'http://parks.services/',

'byCountryResponse',

'ParkService.byCountryResponse'}

);

response\_x = response\_map\_x.get('response\_x');

return response\_x.return\_x;

}

}

}

ParkLocatorTest.apxc:

@isTest

private class ParkLocatorTest {

@isTest static void testCallout (){

Test.setMock(WebServiceMock.class, new ParkServiceMock());

String country = 'United States';

List<String> expectedParks = new List<String>{'Yosemite', 'Sequoia', 'Crater Lake'};

System.assertEquals(expectedParks , 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>{'Yosemite', 'Sequoia', 'Crater Lake'};

response.put('response\_x', response\_x);

}

}

1. Apex Web Services:

AccountManager.apxc:

@RestResource(urlMapping='/Accounts/\*/contacts')

global with sharing class AccountManager {

@HttpGet

global static Account getAccount(){

RestRequest request = RestContext.request;

String accountId = request.requestURI.substringBetween('Accounts/','/contacts');

Account result = [SELECT ID,Name,(SELECT ID, FirstName,LastName FROM Contacts)

FROM Account

WHERE Id = :accountId];

return result;

}

}

AccountManagerTest.apxc;

@isTest

private class AccountManagerTest {

@isTest

static void testGetAccount(){

Account a = new Account(Name='TestAccount');

insert a;

Contact c = new Contact(AccountId=a.Id,FirstName='Test',LastName='Test');

insert c;

RestRequest request = new RestRequest();

request.requestURI = 'https://yourInstance.salesforce.com/appexrest//Accounts/'+a.id+'/contacts';

request.httpMethod = 'GET';

REstContext.request = request;

Account myAcct = AccountManager.getAccount();

System.assert(myAcct !=null);

System.assertEquals('TestAccount', myAcct.Name);

}

}

ASYNCHRONOUS APEX

USE FUTURE METHODS:

AccountProcessor:

public class AccountProcessor {

@future

public static void countContacts(List<Id> accountIds){

List<Account> accountsToUpdate = new List<Account>();

List<Account> accounts = [Select Id, Name, (Select Id from Contacts) from Account Where Id in :accountIds];

For(Account acc:accounts){

List<Contact> contactList = acc.Contacts;

acc.Number\_Of\_Contacts\_\_c = contactList.size();

accountsToUpdate.add(acc);

}

update accountsToUpdate;

}

}

Apex code:

List<Id> accountIds = new List<Id>();

accountIds.add('0015j00000B39HSAAZ');

AccountProcessor.countContacts(accountIds);

AccountProcessorTest:

@isTest

private class AccountProcessorTest{

@isTest

private static void testCountContacts(){

Account newAccount = new Account(Name='Test Account');

insert newAccount;

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

insert newContact1;

Contact newContact2 = new Contact(FirstName='Jane',LastName='Doe',AccountId=newAccount.Id);

insert newContact2;

List<Id> accountIds =new List<Id>();

accountIds.add(newAccount.Id);

Test.startTest();

AccountProcessor.countContacts(accountIds);

Test.stopTest();

}

}

USE BATCH APEX:

Apex class:

global class LeadProcessor implements

Database.Batchable<sObject>, Database.Stateful {

// instance member to retain state across transactions

global Integer recordsProcessed = 0;

global Database.QueryLocator start(Database.BatchableContext bc) {

return Database.getQueryLocator('SELECT Id, LeadSource FROM Lead');

}

global void execute(Database.BatchableContext bc, List<Lead> scope){

// process each batch of records

List<Lead> leads = new List<Lead>();

for (Lead lead : scope) {

lead.LeadSource = 'Dreamforce';

// increment the instance member counter

recordsProcessed = recordsProcessed + 1;

}

update leads;

}

global void finish(Database.BatchableContext bc){

System.debug(recordsProcessed + ' records processed. Shazam!');

}

}

Apex Test Class:

@isTest

public class LeadProcessorTest {

@testSetup

static void setup() {

List<Lead> leads = new List<Lead>();

// insert 200 leads

for (Integer i=0;i<200;i++) {

leads.add(new Lead(LastName='Lead '+i,

Company='Lead', Status='Open - Not Contacted'));

}

insert leads;

}

static testmethod void test() {

Test.startTest();

LeadProcessor lp = new LeadProcessor();

Id batchId = Database.executeBatch(lp, 200);

Test.stopTest();

// after the testing stops, assert records were updated properly

System.assertEquals(200, [select count() from lead where LeadSource = 'Dreamforce']);

}

}

CONTROL PROCESSES WITH QUEUABLE APEX:

AddPrimaryContact:

public without sharing class AddPrimaryContact implements Queueable {

private contact contact;

private String state;

public AddPrimaryContact (Contact inputContact, String inputState){

this.contact=inputContact;

this.state=inputState;

}

public void execute(QueueableContext context){

List<Account> accounts = [SELECT Id FROM Account WHERE BillingState= :state LIMIT 200];

List<Contact> contacts =new List<Contact>();

for(Account acc : accounts){

Contact contactClone = contact.clone();

contactClone.AccountId = acc.Id;

contacts.add(contactClone);

}

insert contacts;

}

}

AddPrimaryContactTest:

@isTest

private class AddPrimaryContactTest {

@isTest

private static void testQueueableClas(){

List<Account> accounts = new List<Account>();

for(Integer i=0; i<500;i++){

Account acc = new Account(Name='Test Account');

if(i<250){

acc.BillingState = 'NY';

}else{

acc.BillingState = 'CA';

}

accounts.add(acc);

}

insert accounts;

Contact contact = new Contact(FirstName='Simon', LastName='Cannock');

insert contact;

Test.startTest();

Id jobId = System.enqueueJob(new AddPrimaryContact(contact, 'CA'));

Test.stopTest();

List<Contact> contacts = [SELECT Id FROM Contact WHERE Contact.Account.BillingState = 'CA'];

System.assertEquals(200, contacts.size(), 'ERROR: Incorrect number of Contact records found');

}

}

SCHEDULE JOBS USING THE APEX SCHEDULER:

Apex class:

global class DailyLeadProcessor implements Schedulable{

global void execute(SchedulableContext ctx){

List<Lead> leads = [SELECT Id, LeadSource FROM Lead WHERE LeadSource = ''];

if(leads.size() > 0){

List<Lead> newLeads = new List<Lead>();

for(Lead lead : leads){

lead.LeadSource = 'DreamForce';

newLeads.add(lead);

}

update newLeads;

}

}

}

Apex test class:

@isTest

private class DailyLeadProcessorTest{

//Seconds Minutes Hours Day\_of\_month Month Day\_of\_week optional\_year

public static String CRON\_EXP = '0 0 0 2 6 ? 2022';

static testmethod void testScheduledJob(){

List<Lead> leads = new List<Lead>();

for(Integer i = 0; i < 200; i++){

Lead lead = new Lead(LastName = 'Test ' + i, LeadSource = '', Company = 'Test Company ' + i, Status = 'Open - Not Contacted');

leads.add(lead);

}

insert leads;

Test.startTest();

// Schedule the test job

String jobId = System.schedule('Update LeadSource to DreamForce', CRON\_EXP, new DailyLeadProcessor());

// Stopping the test will run the job synchronously

Test.stopTest();

}

}

APEX BASICS & DATABASE

GET STARTED WITH APEX:

public class StringArrayTest {

public static List<String> generateStringArray(Integer N){

List<String> TestList = new List<String>();

for(Integer i=0;i<N;i++){

TestList.add('Test '+ i);

system.debug(TestList[i]);

}

return TestList;

}

}

MANIPULATE RECORDS WITH DML:

public class AccountHandler {

public static Account insertNewAccount(String AccountName){

try {

Account newacct = new Account(Name=AccountName);

insert newacct;

return newacct;

} catch (DmlException e) {

System.debug('A DML exception has occurred: ' +

e.getMessage());

return null;

}

}

}

WRITE SOQL QUERIES:

1. public class ContactSearch {

public static List<Contact> searchForContacts(String lastName, String mailingPostalCode) {

List<Contact> conList = [SELECT Id, Name, LastName, MailingPostalCode FROM Contact WHERE LastName = :lastName AND MailingPostalCode = :mailingPostalCode];

return conList;

}

}

WRITE SOSL QUERIES:

public class ContactAndLeadSearch {

public static List<List< sObject>> searchContactsAndLeads(String LastName) {

list<List< sObject>> ContactLeadList = [Find :lastName IN ALL FIELDS

RETURNING Contact(Name),

Lead(Name)];

return ContactLeadList;

}

}