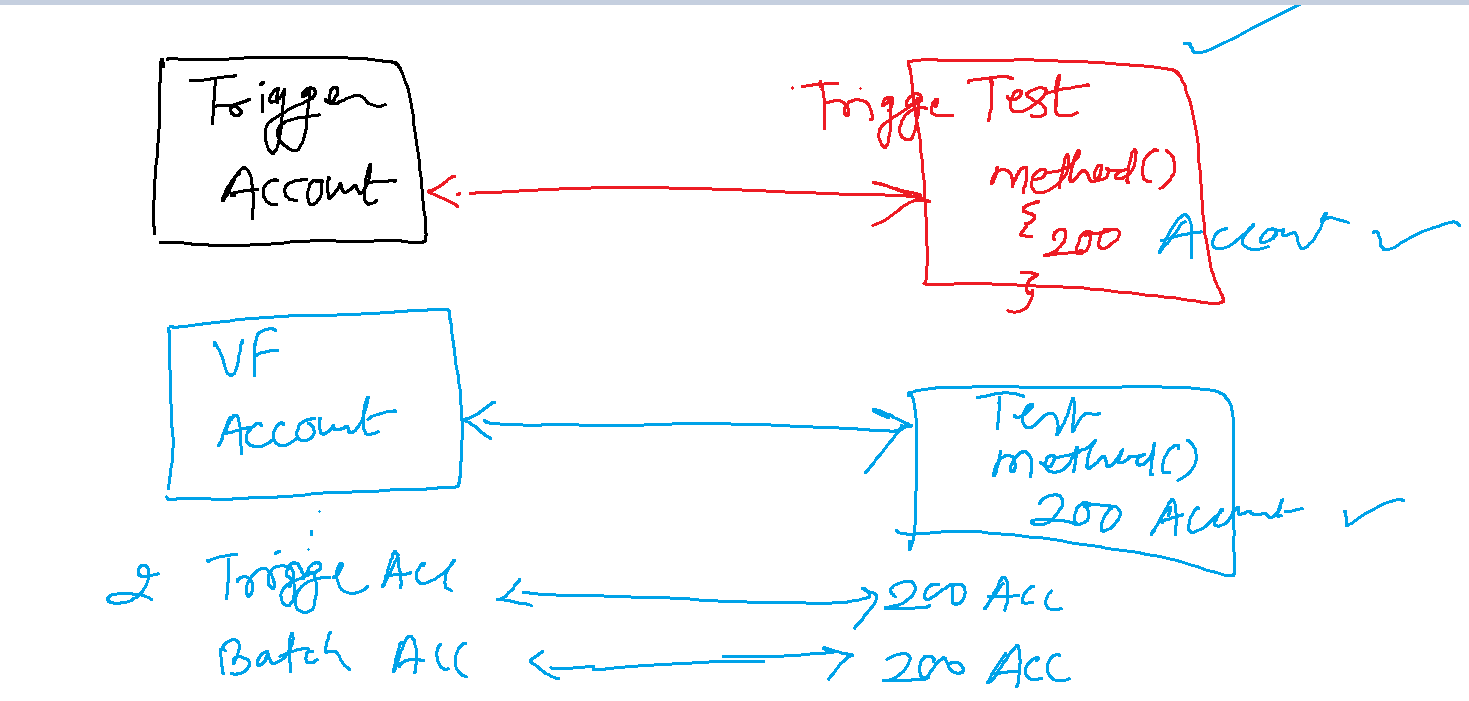
40. Test Data Factory Class - 25 Nov 2021

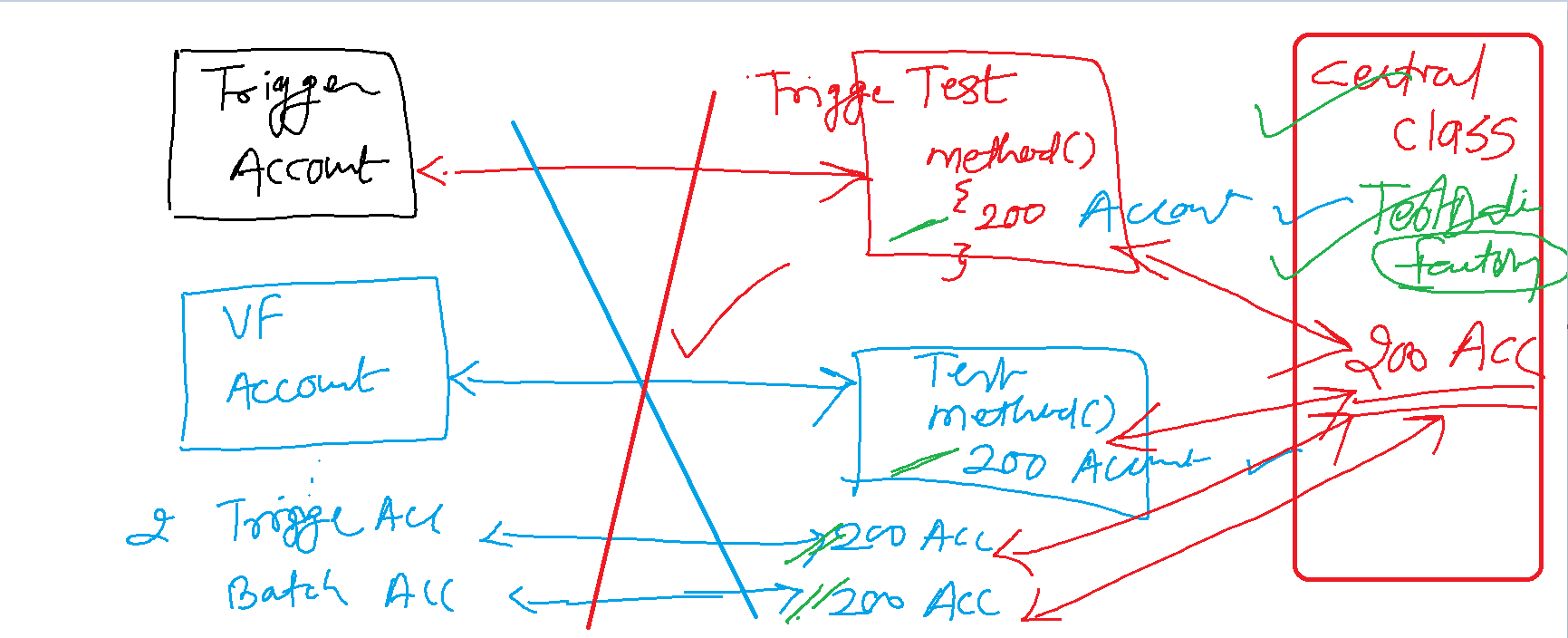
The TestDataFactory class IS ANNOTED as @isTest is used to setting up test data at central location.

OR

The TestDataFactory class is **a special type of class**—it is a public class that is annotated with isTest and can be accessed only from a running test. Test utility classes contain methods that can be called by test methods to perform useful tasks, such as setting up test data.



Test Data Factory:



@isTest

public class TestDataFactory {

public static List<Account> getAccounts(String accountType){

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

for(Integer i=1; i<=200; i++){

Account objAcc = new Account(Name='Cinemax'+i, Type=accountType);

accList.add(objAcc);

}

if(!accList.isEmpty()){

Database.insert(accList, false);

}

return accList;

}

}

Batch Class:

global class BatchAccount implements Database.Batchable<sObject> {

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

return Database.getQueryLocator('SELECT Description,Name,Type FROM Account WHERE Type = \'Prospect\' or Type=\'Customer - Direct\' ');

}

global void execute(Database.BatchableContext BC, List<Account> accountList){

if(!accountList.isEmpty()){

for(Account objAcc : accountList){

if(objAcc.Type == 'Prospect'){

objAcc.Description='Mere pass Bungla Hai, Gadi Hai, Bank Balance Hai, Tere Pass kya Hai?';

}

else{

if(objAcc.Type == 'Customer - Direct'){

objAcc.Description='Mere Pass Mask Hai';

}

}

}

Database.update(accountList,false);

}

}

global void finish(Database.BatchableContext BC){

}

}

@isTest

public class BatchAccountTest {

public static testMethod void prospectTestMethod(){

TestDataFactory.getAccounts('Prospect');

Test.startTest();//To refresh the governor LIMIT

Database.executeBatch(new BatchAccount(), 200);

Test.stopTest();

List<Account> accListResult = [select id, Description from Account];

for(Account objAcc : accListResult){

System.assert(objAcc.Description.contains('Mere pass Bungla Hai, Gadi Hai, Bank Balance Hai, Tere Pass kya Hai?'));

}

}

public static testMethod void CustomerDirectTestMethod(){

TestDataFactory.getAccounts('Customer - Direct');

Test.startTest();//To refresh the governor LIMIT

Database.executeBatch(new BatchAccount(), 200);

Test.stopTest();

List<Account> accListResult = [select id, Description from Account];

for(Account objAcc : accListResult){

System.assert(objAcc.Description.contains('Mere Pass Mask Hai'));

}

}

}

@isTest

public class TestDataFactory {

public static List<Account> getAccounts(String accountType){

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

for(Integer i=1; i<=200; i++){

Account objAcc = new Account(Name='Cinemax'+i, Type=accountType);

accList.add(objAcc);

}

if(!accList.isEmpty()){

Database.insert(accList, false);

}

return accList;

}

public static List<Contact> getContactRecords(){

Account objAcc = new Account(Name='Cinemax', Description='Babu');

insert objAcc;

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

for(Integer i=1; i<=200; i++){

Contact objCon = new Contact(FirstName='Bunty'+i, LastName='Modi'+i, AccountId=objAcc.Id);

conList.add(objCon);

}

if(!conList.isEmpty()){

Database.insert(conList,false);

}

return conList;

}

}

global class BatchContactAccount implements Database.Batchable<sObject>{

integer count=0;

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

return Database.getQueryLocator('select id, FirstName, LastName, LeadSource, Account.Description from Contact');

}

global void execute(Database.BatchableContext BC, List<Contact> conList){//3

count++;//6

Set<Id> accIdSet = new Set<Id>();

for(Contact objCon : conList){

accIdSet.add(objCon.AccountId);

}

Map<Id,Account> accMap = new Map<Id,Account>();

for(Account objAcc : [select Id, Description from Account where Id IN : accIdSet]){

accMap.put(objAcc.Id, objAcc);

}

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

for(Contact objCon : conList){

if(accMap.containsKey(objCon.AccountId)){

if(String.isNotBlank(accMap.get(objCon.AccountId).Description) || accMap.get(objCon.AccountId).Description !=null || accMap.get(objCon.AccountId).Description != ''){

accMap.get(objCon.AccountId).Description = accMap.get(objCon.AccountId).Description+ ' \n Contact '+objCon.FirstName+' '+objCon.LastName+' has Lead Source as Web';

}

else{

accMap.get(objCon.AccountId).Description = 'Contact '+objCon.FirstName+' '+objCon.LastName+' has Lead Source as Web';

}

}

}

accListUpdate.addAll(accMap.values());

Database.update(accListUpdate,false);

}

global void finish(Database.BatchableContext BC){

System.debug('#count = '+count);

}

}

@isTest

public class BatchContactAccountTest {

public static testMethod void testwithDescriptionMethod(){

TestDataFactory.getContactRecords();

Test.startTest();

Database.executeBatch(new BatchContactAccount(), 200);

Test.stopTest();

}

public static testMethod void testwithoutDescriptionMethod(){

Account objAcc = new Account(Name='Cinemax');

insert objAcc;

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

for(Integer i=1; i<=200; i++){

Contact objCon = new Contact(FirstName='Bunty'+i, LastName='Modi'+i, AccountId=objAcc.Id);

conList.add(objCon);

}

if(!conList.isEmpty()){

Database.insert(conList,false);

}

Test.startTest();

Database.executeBatch(new BatchContactAccount(), 200);

Test.stopTest();

}

}

@isTest

public class TestDataFactory {

public static List<Account> getAccounts(String accountType){

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

for(Integer i=1; i<=200; i++){

Account objAcc = new Account(Name='Cinemax'+i, Type=accountType);

accList.add(objAcc);

}

if(!accList.isEmpty()){

Database.insert(accList, false);

}

return accList;

}

public static List<Contact> getContactRecords(String accName, String accDescription, String accType){

Account objAcc = new Account(Name=accName, Description=accDescription, Type=accType);

insert objAcc;

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

for(Integer i=1; i<=200; i++){

Contact objCon = new Contact(FirstName='Bunty'+i, LastName='Modi'+i, AccountId=objAcc.Id);

conList.add(objCon);

}

if(!conList.isEmpty()){

Database.insert(conList,false);

}

return conList;

}

}

global class BatchAccountContact implements Database.Batchable<sObject> {

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

return Database.getQueryLocator('select id, LeadSource,Account.Type from Contact where Account.Type= \'Prospect\' or Account.Type= \'Other\' ');

}

global void execute(Database.BatchableContext BC, List<Contact> conList){

if(!conList.isEmpty()){

for(Contact objCon : conList){

if(objCon.Account.Type=='Prospect'){

objCon.LeadSource = 'Web';

}

else{

if(objCon.Account.Type=='Other'){

objCon.LeadSource = 'Phone Inquiry';

}

}

}

Database.update(conList,false);

}

}

global void finish(Database.BatchableContext BC){

}

}

@isTest

public class BatchAccountContactTest {

@isTest

public static void contactLeadSourceUpdate(){

TestDataFactory.getContactRecords('Cinemax', 'My Desc', 'Prospect');

Test.startTest();

BatchAccountContact objBatchAccCon = new BatchAccountContact();

Database.executeBatch(objBatchAccCon,200);

Test.stopTest();

List<Contact> conListReturned = [select id, LeadSource from Contact where LeadSource='Web' and AccountId=:objAcc.Id];

System.assertEquals(200, conListReturned.size());

}

@isTest

public static void contactLeadSourceOtherUpdate(){

TestDataFactory.getContactRecords('Cinemax', null, 'other');

Test.startTest();

BatchAccountContact objBatchAccCon = new BatchAccountContact();

Database.executeBatch(objBatchAccCon,200);

Test.stopTest();

List<Contact> conListReturned = [select id, LeadSource from Contact where LeadSource='Phone Inquiry' and AccountId=:objAcc.Id];

System.assertEquals(200, conListReturned.size());

}

}

Test Data Factory Class:

@isTest

public class TestDataFactory {

public static List<Account> getAccounts(String accountType){

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

for(Integer i=1; i<=200; i++){

Account objAcc = new Account(Name='Cinemax'+i, Type=accountType);

accList.add(objAcc);

}

if(!accList.isEmpty()){

Database.insert(accList, false);

}

return accList;

}

public static List<Contact> getContactRecords(String accName, String accDescription, String accType){

Account objAcc = new Account(Name=accName, Description=accDescription, Type=accType);

insert objAcc;

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

for(Integer i=1; i<=200; i++){

Contact objCon = new Contact(FirstName='Bunty'+i, LastName='Modi'+i, AccountId=objAcc.Id);

conList.add(objCon);

}

if(!conList.isEmpty()){

Database.insert(conList,false);

}

return conList;

}

public static Applicant\_\_c getApplicantRecords(String fName, String lName, String appGender){

Date dob = Date.parse('03/15/2000');

Applicant\_\_c objApp = new Applicant\_\_c(First\_Name\_\_c=fName,Last\_Name\_\_c=lName, Gender\_\_c=appGender, DOB\_\_c=dob);

insert objApp;

return objApp;

}

}

Trigger:

trigger prefixApplicant on Applicant\_\_c (before insert, before update) {

if(trigger.isInsert && trigger.isBefore){

prefixApplicantHandler.beforeInsert(trigger.new);

}

if(trigger.isUpdate && trigger.isBefore){

prefixApplicantHandler.beforeUpdate(trigger.new);

}

}

Trigger Test:

@isTest

public class prefixApplicantTest {

public static testMethod void prefixMaleTest(){

TestDataFactory.getApplicantRecords('Bunty', 'Modi', 'Male');

String actualFirstName = [select First\_Name\_\_c from Applicant\_\_c where Id=:objApp.Id].First\_Name\_\_c;

System.assertEquals('Mr.Bunty', actualFirstName);

//Update

Applicant\_\_c appData = [select First\_Name\_\_c,Gender\_\_c from Applicant\_\_c where Id=:objApp.Id LIMIT 1];

appData.First\_Name\_\_c='Ms.Chachu';

update appData;

String actualFirstNameUpdate = [select First\_Name\_\_c from Applicant\_\_c where Id=:appData.Id].First\_Name\_\_c;

System.assertEquals('Mr.Chachu', actualFirstNameUpdate);

}

public static testMethod void prefixFemaleTest(){

TestDataFactory.getApplicantRecords('Pinki', 'Modi', 'Female');

String actualFirstName = [select First\_Name\_\_c from Applicant\_\_c where Id=:objApp.Id].First\_Name\_\_c;

System.assertEquals('Ms.Pinki', actualFirstName);

//Update

Applicant\_\_c appData = [select First\_Name\_\_c,Gender\_\_c from Applicant\_\_c where Id=:objApp.Id LIMIT 1];

appData.First\_Name\_\_c='Mr.Pinki';

update appData;

String actualFirstNameUpdate = [select First\_Name\_\_c from Applicant\_\_c where Id=:appData.Id].First\_Name\_\_c;

System.assertEquals('Ms.Pinki', actualFirstNameUpdate);

}

public static testMethod void prefixTransgenderTest(){

try{

TestDataFactory.getApplicantRecords('Sangita', 'Modi', 'Transgender');

}

catch(Exception ex){

boolean result = ex.getMessage().contains('Transgender is currently not allowed...!!!') ? true : false;

System.assertEquals(true, result);

}

}

}

Interview:

1] Have to worked on Test Classes: Yes

2] Best practices Test Class:

- 75% for Apex Class, 1% Trigger 🡪 But we should try for 100%.

- Use TestDataFactory class

- Minimum 200 records

- Test.startTest and Test.StopTest

- Use System.assert

- Do not use SeeAllData=true (It is not only fetch the data from Virtual Database, but also from The actual Dataabase)

<https://www.levelupsalesforce.com/using-test-isrunningtest-in-apex>

//Check if a test is running

if(Test.isRunningTest()) {

//Run specific logic for Tests

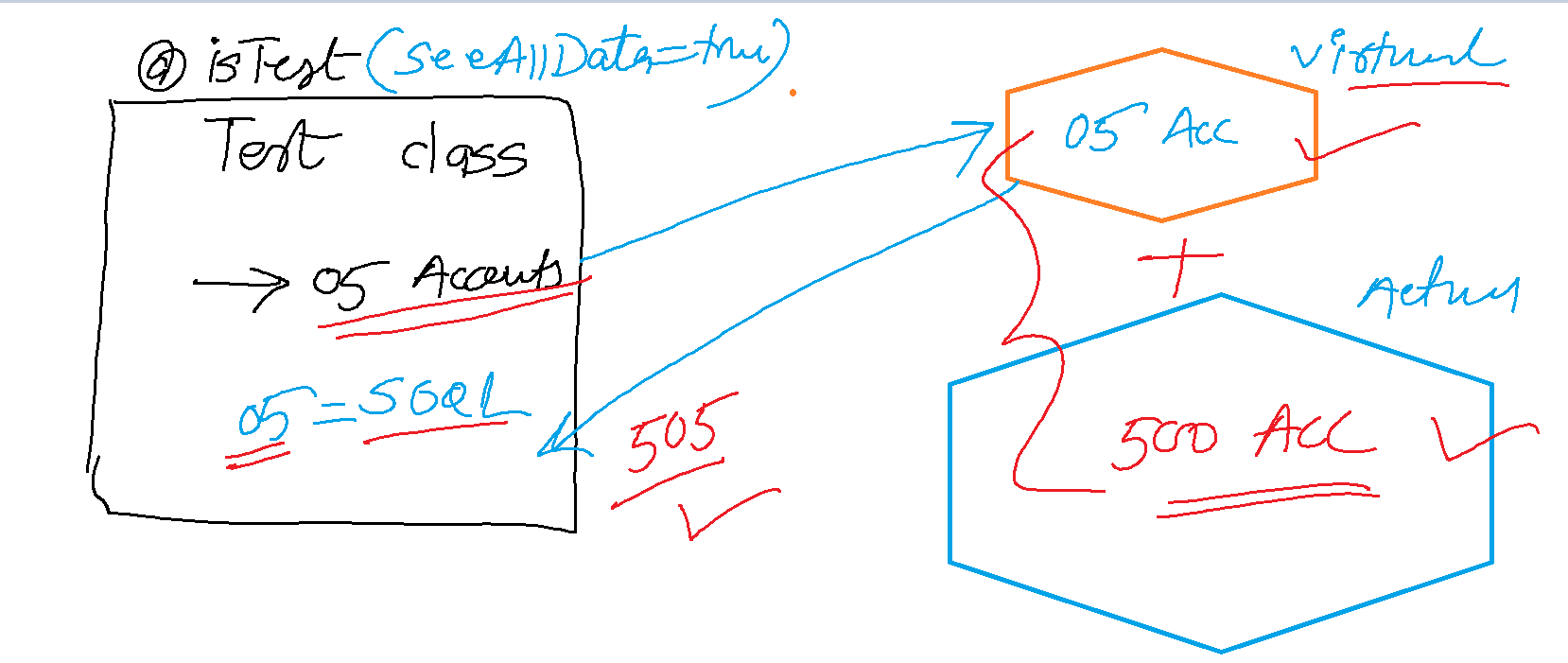
}

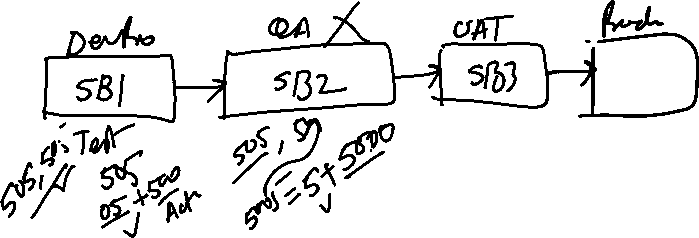
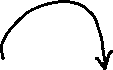
//Check if test is NOT Running

if(!Test.isRunningTest()){

//Run specific logic that wont run in tests

}





Mixed DML Statement Error: 01

Custom Settings and Custom Metadata – 03

Next Class on Monday :

Test Setup Method

