38. Continue...Test Class for Triggers - 23 Nov 2021

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);

}

}

public class prefixApplicantHandler {

public static boolean flag = true;

public static void beforeInsert(List<Applicant\_\_c> appList){

prefixApplicantHelper.prefixApplicantFirstName(appList);

}

public static void beforeUpdate(List<Applicant\_\_c> appList){

prefixApplicantHelper.prefixApplicantFirstName(appList);

}

}

public class prefixApplicantHelper {

public static void prefixApplicantFirstName(List<Applicant\_\_c> appList){

for(Applicant\_\_c objApp : appList){

if(String.isNotBlank(objApp.First\_Name\_\_c)){

if((objApp.First\_Name\_\_c.startsWith('Mr.') || objApp.First\_Name\_\_c.startsWith('Ms.'))){

if(objApp.First\_Name\_\_c.startsWith('Mr.')){

objApp.First\_Name\_\_c = objApp.First\_Name\_\_c.remove('Mr.');

}

else{

objApp.First\_Name\_\_c = objApp.First\_Name\_\_c.remove('Ms.');

}

}

if(objApp.Gender\_\_c=='Male' && !objApp.First\_Name\_\_c.startsWith('Mr.')){

objApp.First\_Name\_\_c = 'Mr.'+objApp.First\_Name\_\_c;

}

else{

if(objApp.Gender\_\_c == 'Female' && !objApp.First\_Name\_\_c.startsWith('Ms.')){

objApp.First\_Name\_\_c = 'Ms.'+objApp.First\_Name\_\_c;

}

else{

if(objApp.Gender\_\_c == 'Transgender')

objApp.addError('Transgender is currently not allowed...!!!');

}

}

}

else{

objApp.First\_Name\_\_c.addError('Please enter first name');

}

}

}

}

@isTest

public class prefixApplicantTest {

public static testMethod void prefixMaleTest(){

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

Applicant\_\_c objApp = new Applicant\_\_c(First\_Name\_\_c='Raju',Last\_Name\_\_c='Modi', Gender\_\_c='Male', DOB\_\_c=dob);

insert objApp;

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

System.assertEquals('Mr.Raju', 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(){

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

Applicant\_\_c objApp = new Applicant\_\_c(First\_Name\_\_c='Pinki',Last\_Name\_\_c='Modi', Gender\_\_c='Female', DOB\_\_c=dob);

insert objApp;

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{

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

Applicant\_\_c objApp = new Applicant\_\_c(First\_Name\_\_c='Sangita',Last\_Name\_\_c='Modi', Gender\_\_c='Transgender', DOB\_\_c=dob);

insert objApp;

}

catch(Exception ex){

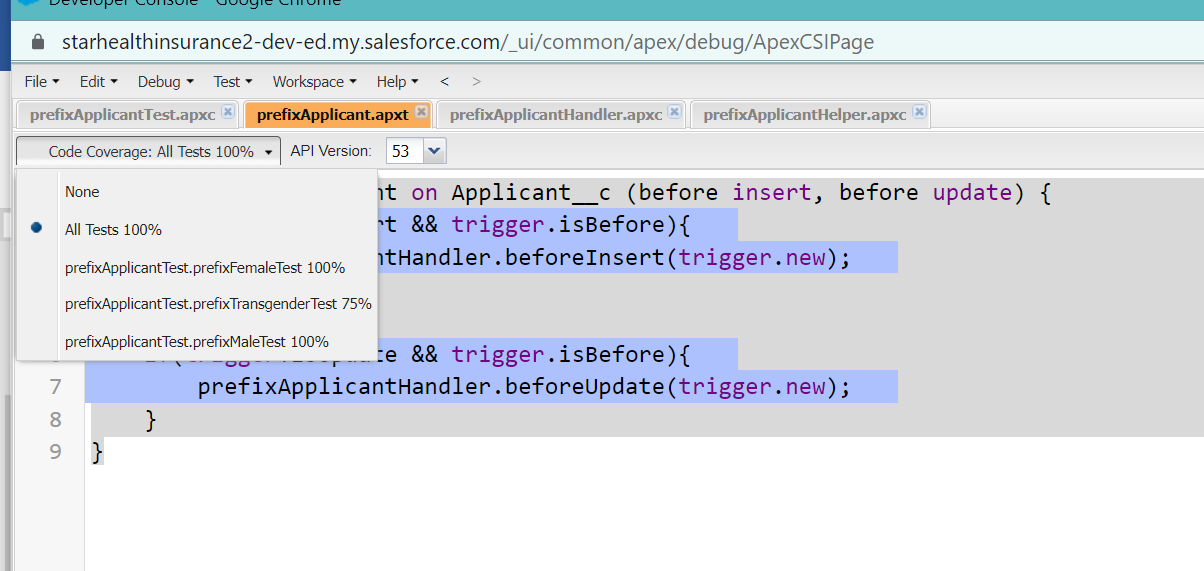
boolean result = ex.getMessage().contains('Transgender is currently not allowed...!!!') ? true : false;

System.assertEquals(true, result);

}

}

}



trigger ContactToAccountDesc on Contact (after insert) {

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

for(Contact objCon : trigger.new){

if(objCon.AccountId!=null)

accIdSet.add(objCon.AccountId);

}

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

if(!accIdSet.isEmpty()){

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

accMap.put(objAcc.Id, objAcc);

}

}

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

if(!accMap.isEmpty()){

for(Contact objCon : trigger.new){

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

if(accMap.get(objCon.AccountId).Description!=null){

accMap.get(objCon.AccountId).Description =accMap.get(objCon.AccountId).Description +' '+objCon.FirstName+' '+objCon.LastName ;

}

else{

accMap.get(objCon.AccountId).Description =objCon.FirstName+' '+objCon.LastName ;

}

}

}

accListUpdate.addAll(accMap.values());

}

if(!accListUpdate.isEmpty())

Database.update(accListUpdate,false);

}

@isTest

public class ContactToAccountDescTest {

public static testMethod void testContactToAccount(){

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);

}

String descriptionActual = [select id, Description from Account where Id=: objAcc.Id].Description;

String expectedStartWith='Bunty1 Modi1';

String expectedEndsWith='Bunty200 Modi200';

System.debug('@Description Values = '+descriptionActual);

boolean flag=false;

if(descriptionActual.startsWith(expectedStartWith) && descriptionActual.endsWith(expectedEndsWith) ){

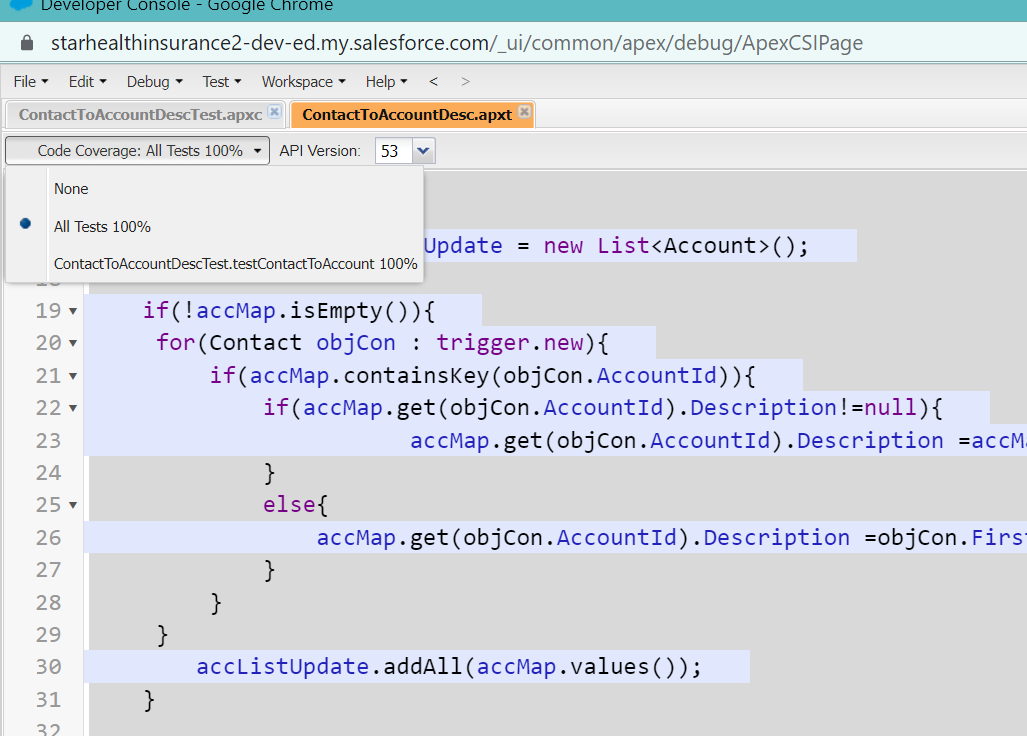
flag = true;

}

System.assert(flag);

}

}



trigger AccountToContact on Account (after update) {

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

for(Account objAcc : trigger.new){

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

if(objAcc.Type != trigger.oldMap.get(objAcc.Id).Type){

accMap.put(objAcc.Id,objAcc);

}

}

}

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

if(!accMap.isEmpty()){

for(Contact objCon : [select Id,LeadSource,Level\_\_c,AccountId from Contact where AccountID IN : accMap.keySet()]){

conList.add(objCon);

}

}

if(!conList.isEmpty()){

for(Contact objCon : conList){

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

if(accMap.get(objCon.AccountId).Type =='Technology Partner'){

objCon.LeadSource ='Web';

objCon.Level\_\_c='Primary';

}

}

}

Database.update(conList,false);

}

}

@isTest

public class AccountToContactTest {

@isTest

public static void postiveTestMethod(){

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);

}

Account objAccResult = [select Id, Type from Account where Id=: objAcc.Id];

objAccResult.Type='Technology Partner';

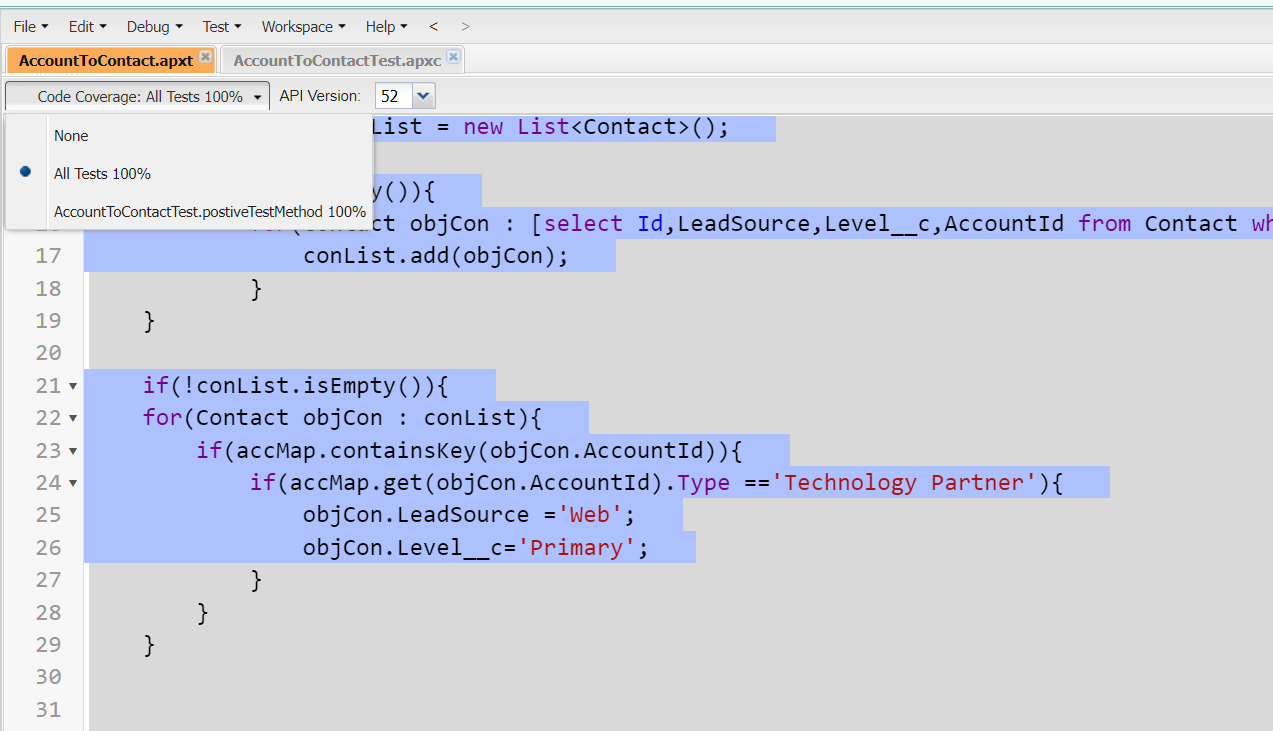
update objAccResult;

List<Contact> conListResult = [select Id, LeadSource, Level\_\_c from Contact where AccountId=: objAcc.Id and LeadSource='Web' and Level\_\_c='Primary'];

System.assertEquals(200, conListResult.Size());

}

}



trigger AccountBackup on Account (before delete) {

for(Account objAcc : trigger.old){

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

objAcc.addError('You can not delete Prospected Account...!!!!');

}

}

}

@isTest

public class AccountBackupTest {

public static testMethod void testPositiveMethod(){

try{

Account objAcc = new Account(Name='Cinemax', Type='Prospect');

insert objAcc;

delete objAcc;

}

catch(Exception ex){

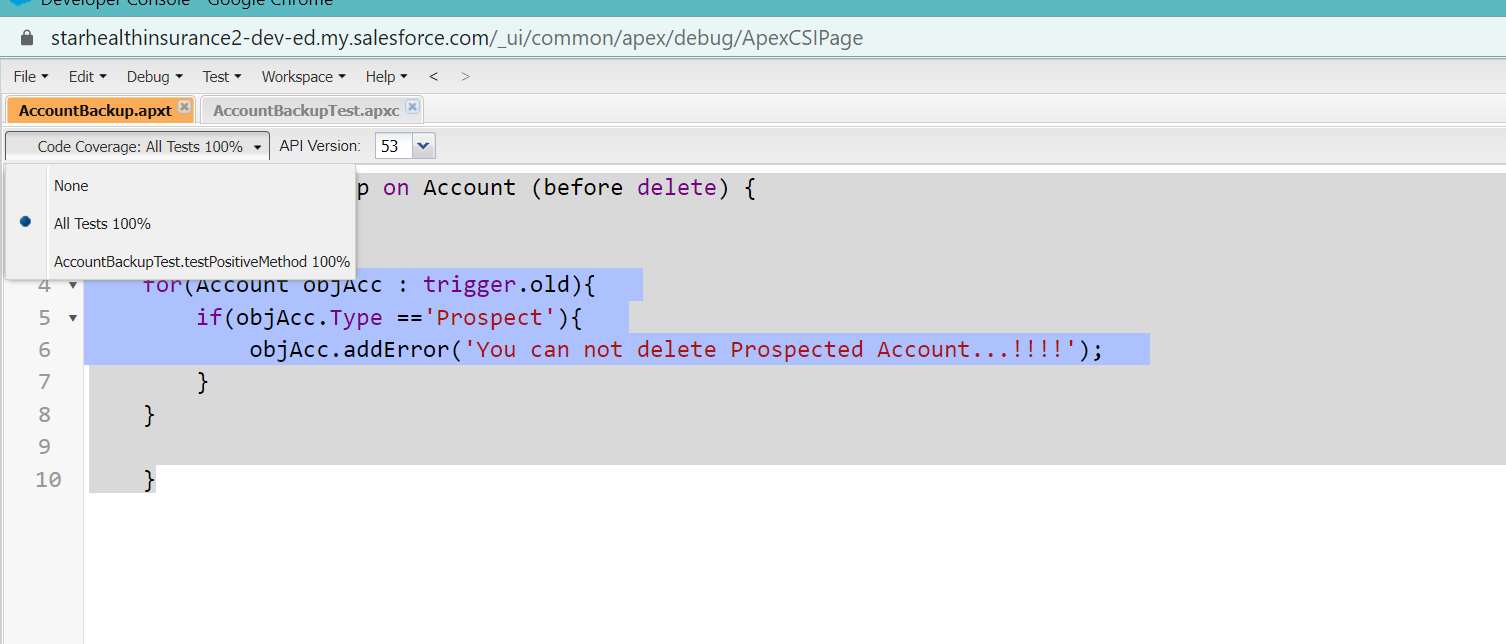
boolean result = ex.getMessage().contains('You can not delete Prospected Account...!!!!') ? true : false;

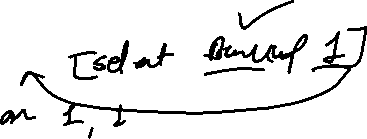
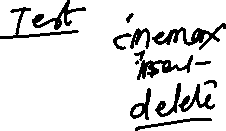
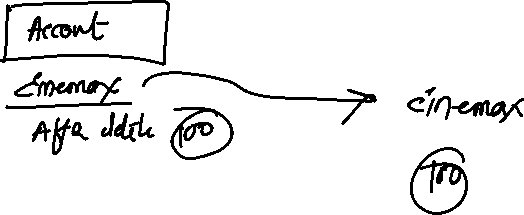
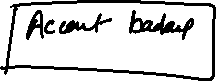
System.assertEquals(true, result);

}

}

}





trigger AccountBackup on Account (before delete) {

List<AccountBackup\_\_c> accBkList = new List<AccountBackup\_\_c>();

for(Account objAcc : trigger.old){

AccountBackup\_\_c objAccBackup = new AccountBackup\_\_c(Account\_Name\_Backup\_\_c=objAcc.Name, Account\_Rating\_Backup\_\_c=objAcc.Rating);

accBkList.add(objAccBackup);

}

if(!accBkList.isEmpty()){

Database.insert(accBkList, false);

}

}

@isTest

public class AccountBackupTest {

public static testMethod void testPositiveMethod(){

try{

Account objAcc = new Account(Name='Cinemax', Type='Prospect');

insert objAcc;

delete objAcc;

}

catch(Exception ex){

boolean result = ex.getMessage().contains('You can not delete Prospected Account...!!!!') ? true : false;

System.assertEquals(true, result);

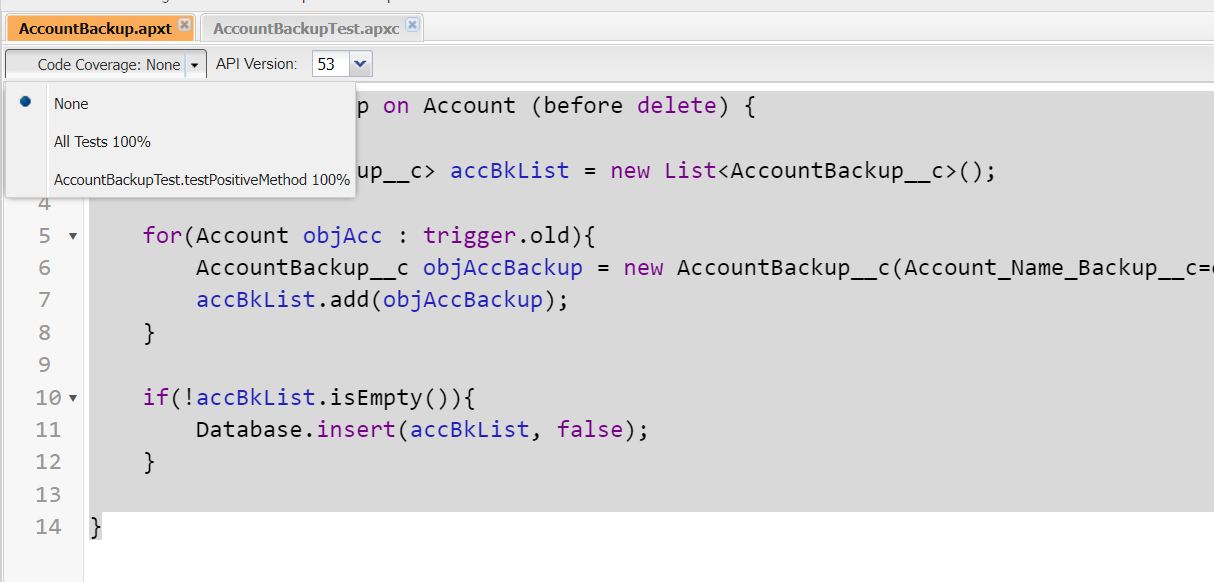
}

List<AccountBackup\_\_c> accBkList = [select id from AccountBackup\_\_c];

System.assertEquals(1, accBkList.size());

}

}



Ternary Operator:

?

Scenario : 1] If age is greater of equal to 18, then print ‘Eligible’, else ‘Not Eligible’

Integer age = 20;

if(age >= 18){

System.debug('Eligible');

}

else{

System.debug('Not Eligible');

}

2 Sec

Ternary Operator :

Integer age = 10;

String result = age >=18 ? 'Eligible' : 'Not Eligible';

System.debug(result);

0.23 MilliSec

Decimal Salary=200;

Decimal ROI = 19;

Decimal result = ROI >=10 ? (Salary+100) : (Salary-100);

System.debug(result);

Try – catch Block

* It used to handle the Exception

try{

//Business Logic

}

Catch(Exception ex){

}

try{

Integer x=1, y=0;

integer z = x / y;

System.debug(z);

}

catch(Exception ex){

System.debug(ex.getMessage());

}

try{

Integer x=1, y=0;

integer z = x / y;

System.debug(z);

}

catch(NoSuchElementException ex){

System.debug('NoSuch'+ex.getMessage());

}

catch(SearchException ex){

System.debug('Sear'+ex.getMessage());

}

catch(MathException ex){

System.debug('Math'+ex.getMessage());

}

catch(Exception ex){

System.debug(ex.getMessage());

}

Assignment:

User Defined Exception: