**TASK 01:-**

**Account having a salary field and category field, in category field we have a slab of salaries which indicate the category. Salary will decide the category of the user. (Like, salary is below or equal to 10,000 D category is selected, if salary is in between 10,000 to 30,000 C category is selected.)**

**:- Do it With Flow And Trigger Design Pattern.**

**Itrigger:-**

/\*\*

\* Interface containing methods Trigger Handlers must implement to enforce best practice

\* and bulkification of triggers.

\*/

public interface ITrigger

{

/\*\*

\* bulkBefore

\*

\* This method is called prior to execution of a BEFORE trigger. Use this to cache

\* any data required into maps prior execution of the trigger.

\*/

void bulkBefore();

/\*\*

\* bulkAfter

\*

\* This method is called prior to execution of an AFTER trigger. Use this to cache

\* any data required into maps prior execution of the trigger.

\*/

void bulkAfter();

/\*\*

\* beforeInsert

\*

\* This method is called iteratively for each record to be inserted during a BEFORE

\* trigger. Never execute any SOQL/SOSL etc in this and other iterative methods.

\*/

void beforeInsert(SObject so);

/\*\*

\* beforeUpdate

\*

\* This method is called iteratively for each record to be updated during a BEFORE

\* trigger.

\*/

void beforeUpdate(SObject oldSo, SObject so);

/\*\*

\* beforeDelete

\*

\* This method is called iteratively for each record to be deleted during a BEFORE

\* trigger.

\*/

void beforeDelete(SObject so);

/\*\*

\* afterInsert

\*

\* This method is called iteratively for each record inserted during an AFTER

\* trigger. Always put field validation in the 'After' methods in case another trigger

\* has modified any values. The record is 'read only' by this point.

\*/

void afterInsert(SObject so);

/\*\*

\* afterUpdate

\*

\* This method is called iteratively for each record updated during an AFTER

\* trigger.

\*/

void afterUpdate(SObject oldSo, SObject so);

/\*\*

\* afterDelete

\*

\* This method is called iteratively for each record deleted during an AFTER

\* trigger.

\*/

void afterDelete(SObject so);

/\*\*

\* andFinally

\*

\* This method is called once all records have been processed by the trigger. Use this

\* method to accomplish any final operations such as creation or updates of other records.

\*/

void andFinally();

}

**Trigger Factory:-**

public with sharing class TriggerFactoryAccountSalaryCategory

{

public static void SalaryCategory(Schema.sObjectType soType)

{

// Get a handler appropriate to the object being processed

ITrigger handler = getHandler(soType);

// Make sure we have a handler registered, new handlers must be registered in the getHandler method.

if (handler == null)

{

//throw new TriggerException('No Trigger Handler registered for Object Type: ' + soType);

system.debug('No Trigger Handler');

}

// Execute the handler to fulfil the trigger

execute(handler);

}

private static void execute(ITrigger handler)

{

// Before Trigger

if (Trigger.isBefore)

{

// Call the bulk before to handle any caching of data and enable bulkification

try

{

handler.bulkBefore();

}

catch(exception ex)

{

system.debug('ex===='+ ex);

//ErrorNotificationHelper.notifyError(ex,'Trigger Factory' ,string.valueof(handler));

}

// Iterate through the records to be deleted passing them to the handler.

if (Trigger.isDelete)

{

try

{

for (SObject so : Trigger.old)

{

handler.beforeDelete(so);

}

}

catch(exception ex)

{

//ErrorNotificationHelper.notifyError(ex,'Trigger Factory' ,string.valueof(handler));

system.debug('Error in isDelete func.');

}

}

// Iterate through the records to be inserted passing them to the handler.

else if (Trigger.isInsert)

{

try

{

for (SObject so : Trigger.new)

{

handler.beforeInsert(so);

}

}

catch(exception ex)

{

//ErrorNotificationHelper.notifyError(ex,'Trigger Factory' ,string.valueof(handler));

system.debug('Error in isInsert func.');

}

}

// Iterate through the records to be updated passing them to the handler.

else if (Trigger.isUpdate)

{

try

{

for (SObject so : Trigger.old)

{

handler.beforeUpdate(so, Trigger.newMap.get(so.Id));

}

}

catch(exception ex)

{

//ErrorNotificationHelper.notifyError(ex,'Trigger Factory' ,string.valueof(handler));

system.debug('Error in isUpdate func.');

}

}

}

else

{

// Call the bulk after to handle any caching of data and enable bulkification

handler.bulkAfter();

// Iterate through the records deleted passing them to the handler.

if (Trigger.isDelete)

{

try

{

for (SObject so : Trigger.old)

{

handler.afterDelete(so);

}

}

catch(exception ex)

{

//ErrorNotificationHelper.notifyError(ex,'Trigger Factory' ,string.valueof(handler));

system.debug('Error in isDelete func.');

}

}

// Iterate through the records inserted passing them to the handler.

else if (Trigger.isInsert)

{

try

{

for (SObject so : Trigger.new)

{

handler.afterInsert(so);

}

}

catch(exception ex)

{

//ErrorNotificationHelper.notifyError(ex,'Trigger Factory' ,string.valueof(handler));

system.debug('Error in isInsert func.');

}

}

// Iterate through the records updated passing them to the handler.

else if (Trigger.isUpdate)

{

try

{

for (SObject so : Trigger.old)

{

handler.afterUpdate(so, Trigger.newMap.get(so.Id));

}

}

catch(exception ex)

{

//ErrorNotificationHelper.notifyError(ex,'Trigger Factory' ,string.valueof(handler));

system.debug('Error in isUpdate func.');

}

}

}

// Perform any post processing

try

{

handler.andFinally();

}

catch(exception ex)

{

//ErrorNotificationHelper.notifyError(ex,'Trigger Factory' ,string.valueof(handler));

system.debug('Error in Post Processing');

}

}

private static ITrigger getHandler(Schema.sObjectType soType)

{

ITrigger handler;

if (soType == Account.sObjectType)

{

handler = new AccountSalaryCategoryHandler();

}

return handler;

}

}

**Trigger:-**

trigger TriggerAccountSalaryCategory on Account (before insert, before update, before delete, after insert, after update, after delete) {

TriggerFactoryAccountSalaryCategory.SalaryCategory(Account.sObjectType);

}

**Handler:-**

public class AccountSalaryCategoryHandler implements ITrigger

{

public void bulkBefore(){

system.debug('--- bulkbEfore is called');

}

public void bulkAfter(){

system.debug('--- bulkAfter is called');

}

public void beforeInsert(SObject so){

system.debug('--- beforeInsert is called');

AccountSalaryCategoryHelper.accountSalaryCategory((Account)so);

}

public void beforeUpdate(SObject oldSo, SObject so){

system.debug('--- beforeUpdate is called');

AccountSalaryCategoryHelper.accountSalaryCategory((Account)so);

}

public void beforeDelete(SObject so){

system.debug('--- beforeDelete is called');

}

public void afterInsert(SObject so){

system.debug('--- afterInsert is called');

}

public void afterUpdate(SObject oldSo, SObject so){

system.debug('--- afterUpdate is called');

}

public void afterDelete(SObject so){

system.debug('--- afterDelete is called');

}

public void andFinally(){

system.debug('--- andFinally called');

}

}

**Helper:-**

public class AccountSalaryCategoryHelper {

public static void accountSalaryCategory(Account account){

if(account.Salary\_\_c <= 10000){

account.Category\_\_c = 'D';

}

else if(account.Salary\_\_c >10000 && account.Salary\_\_c <=30000 ) {

account.Category\_\_c = 'C';

}

else{

account.Category\_\_c = 'A';

}

}

}

**TASK 02:-**

**Concatenate Opportunity Name And Accout Name.**

**:- Do it With Flow And Trigger Design Pattern.**

**Handler:-**

public class AccountHandlerConcate implements ITrigger

{

public void bulkBefore(){

system.debug('--- bulkbEfore is called');

}

public void bulkAfter(){

system.debug('--- bulkAfter is called');

}

public void beforeInsert(SObject so){

system.debug('--- beforeInsert is called');

ConcatenateOpportunityAccountHelper.Concatenation((Opportunity)so);

}

public void beforeUpdate(SObject oldSo, SObject so){

system.debug('--- beforeUpdate is called');

ConcatenateOpportunityAccountHelper.Concatenation((Opportunity)so);

}

public void beforeDelete(SObject so){

system.debug('--- beforeDelete is called');

}

public void afterInsert(SObject so){

system.debug('--- afterInsert is called');

}

public void afterUpdate(SObject oldSo, SObject so){

system.debug('--- afterUpdate is called');

}

public void afterDelete(SObject so){

system.debug('--- afterDelete is called');

}

public void andFinally(){

system.debug('--- andFinally called');

}

}

**Helper:-**

public class ConcatenateOpportunityAccountHelper {

public static void Concatenation( Opportunity ConcatenateOpportunityAccount){

Set<Id> setOfAccountIds = new Set<Id>(); // Initialize a list of Ids

if(ConcatenateOpportunityAccount.AccountId !=null){

setOfAccountIds.add(ConcatenateOpportunityAccount.AccountId);

}

Map<Id,String> setOfIdName = new Map<Id,String>(); // it is used to compare the two value.

List<Account> listOfAccount = [SELECT id, name FROM Account WHERE Id = : setOfAccountIds];

for(Account a: listOfAccount){

setOfIdName.put(a.Id, a.Name);

}

if(setOfIdName.containsKey(ConcatenateOppAcc.AccountId)){

ConcatenateOpportunityAccount.Name = ConcatenateOpportunityAccount.Name + '-' + setOfIdName.get(ConcatenateOpportunityAccount.AccountId);

}

}

}

**TASK 03:-**

**Write a trigger on Account when an account is update when account type changes send an email to all its contacts that your account information has been changed.**

**Subject: Account Update Info**

**Body: Your account information has been updated successfully.**

**Account Name: XYZ.**

**Complete it with flow and code.**

**Handler:-**

public class EmailSendToAccountHolderHandler implements ITrigger

{

public void bulkBefore(){

system.debug('--- bulkbEfore is called');

}

public void bulkAfter(){

system.debug('--- bulkAfter is called');

}

public void beforeInsert(SObject so){

system.debug('--- beforeInsert is called');

}

public void beforeUpdate(SObject oldSo, SObject so){

system.debug('--- beforeUpdate is called');

}

public void beforeDelete(SObject so){

system.debug('--- beforeDelete is called');

}

public void afterInsert(SObject so){

system.debug('--- afterInsert is called');

}

public void afterUpdate(SObject oldSo, SObject so){

system.debug('--- afterUpdate is called');

EmailSendToContactListHelper.SendEmailToContact((Account)so);

}

public void afterDelete(SObject so){

system.debug('--- afterDelete is called');

}

public void andFinally(){

system.debug('--- andFinally called');

}

}

**Helper:-**

public class EmailSendToContactListHelper {

public static void SendEmailToContact(Account account){

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

if (account.Type != null) {

accountIdSet.add(account.Id);

}

if (accountIdSet.size() > 0) {

List<Contact> contactList = [SELECT Id, Email FROM Contact WHERE AccountId IN :accountIdSet AND Email != null];

if (contactList.size() > 0) {

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

for (Contact contact : contactList) {

emailAddressList.add(contact.Email);

}

Messaging.SingleEmailMessage message = new Messaging.SingleEmailMessage();

message.setToAddresses(emailAddressList);

message.Subject = 'Account Update Info';

message.plaintextbody = 'Your account information has been updated successfully Deepak pundir.';

Messaging.SingleEmailMessage[] messages = new List<Messaging.SingleEmailMessage> {message};

Messaging.SendEmailResult[] results = Messaging.sendEmail(messages);

}

}

}

}

**TASK 04:-**

**Write a trigger on Opportunity, when an Opportunity will be inserting an Opportunity Line Item should be insert by default with any of the Product associated with Opportunity.**

**:- Do it With Flow And Trigger Design Pattern.**

**Handler:-**

public class TriggerHandlerProductAddInOpportunity implements ITrigger

{

public void bulkBefore(){

system.debug('--- bulkbEfore is called');

}

public void bulkAfter(){

system.debug('--- bulkAfter is called');

}

public void beforeInsert(SObject so){

system.debug('--- beforeInsert is called');

}

public void beforeUpdate(SObject oldSo, SObject so){

system.debug('--- beforeUpdate is called');

}

public void beforeDelete(SObject so){

system.debug('--- beforeDelete is called');

}

public void afterInsert(SObject so){

system.debug('--- afterInsert is called');

TriggerHelperProductAddInOpportunity.ProductAddInOpportunity((Opportunity)so);

}

public void afterUpdate(SObject oldSo, SObject so){

system.debug('--- afterUpdate is called');

}

public void afterDelete(SObject so){

system.debug('--- afterDelete is called');

}

public void andFinally(){

system.debug('--- andFinally called');

}

}

**Helper:-**

public class TriggerHelperProductAddInOpportunity {

public static void ProductAddInOpportunity(Opportunity NewOpportunity){

List<OpportunityLineItem> OpportunityLineItemList = new List<OpportunityLineItem>();

OpportunityLineItem OpportunityLineItem =new OpportunityLineItem();

OpportunityLineItem.OpportunityId = NewOpportunity.id;

OpportunityLineItem.PricebookEntryId = '01u5j000005UrLtAAK';

OpportunityLineItem.UnitPrice = 8000;

OpportunityLineItem.Quantity = 2;

OpportunityLineItemList.add(OpportunityLineItem);

insert OpportunityLineItemList;

}

}

**TASK 05:-**

**Contact having the email field and extract the domain name and check the domain is exist in account, if exist then link the contact to account.**

**Do this with bulkification.**

**Handler:-**

public class AddContactEmailHandler implements ITrigger

{

Map<String,Id> mapOfHandler=new Map<String,Id>();

public void bulkBefore(){

mapOfHandler=AddContactEmailHelper.onbulkBefore(Trigger.new);

}

public void bulkAfter(){

system.debug('--- bulkAfter is called');

}

public void beforeInsert(SObject so){

system.debug('--- beforeInsert is called');

Contact a=(Contact)so;//Typecasting to contact type

AddContactEmailHelper.checkDomainFromAccount(mapOfHandler,a);

}

public void beforeUpdate(SObject oldSo, SObject so){

system.debug('--- beforeUpdate is called');

}

public void beforeDelete(SObject so){

system.debug('--- beforeDelete is called');

}

public void afterInsert(SObject so){

system.debug('--- afterInsert is called');

TriggerHelperProductAddInOpportunity.ProductAddInOpportunity((Opportunity)so);

}

public void afterUpdate(SObject oldSo, SObject so){

system.debug('--- afterUpdate is called');

}

public void afterDelete(SObject so){

system.debug('--- afterDelete is called');

}

public void andFinally(){

system.debug('--- andFinally called');

}

}

**Helper:-**

public class AddContactEmailHelper {

public static Map<String,Id> onbulkBefore(List<Contact> listOfContacts)

{

Set<String> emailDomains = new Set<String>();

// Todo loop on list

for(Contact contactList :listOfContacts){

if(contactList.Email!=null){

String Domain=(contactList.Email.split('@').get(1));

emailDomains.add(Domain.split('\\.').get(0));

system.debug('domains'+emailDomains);

}

}

Map<String,Id> mapacc=new Map<String,Id>();

List<Account> listOfAccounts=[Select Id,Name from Account where Name in :emailDomains];

for(Account accountList:listOfAccounts)

{

mapacc.put(accountList.Name, accountList.Id);

}

return mapacc;

// system.debug('Print map'+mapacc);

}

public static void checkDomainFromAccount(Map<String,Id> mapacc,Contact listOfContacts)

{

String Domain=(listOfContacts.Email.split('@').get(1));

Domain=(Domain.split('\\.').get(0));

system.debug('domains'+Domain);

listOfContacts.AccountId=mapacc.get(Domain);

}

}

**TASK6:-**

**Prefix Account first name with Dr. when new Account is created or Account updated.**

**:- Do this with bulkification.**

**Handler:-**

public class AppendPrefixBulkifyHandler implements ITrigger

{

public void bulkBefore(){

system.debug('--- bulkbEfore is called');

AppendPrefixBulkifyHelper.AppendPrefixBulkifyHelperAddNewAccount(Trigger.New);

}

public void bulkAfter(){

system.debug('--- bulkAfter is called');

}

public void beforeInsert(SObject so){

system.debug('--- beforeInsert is called');

}

public void beforeUpdate(SObject oldSo, SObject so){

system.debug('--- beforeUpdate is called');

}

public void beforeDelete(SObject so){

system.debug('--- beforeDelete is called');

}

public void afterInsert(SObject so){

system.debug('--- afterInsert is called');

//TriggerHelperProductAddInOpportunity.ProductAddInOpportunity((Opportunity)so);

}

public void afterUpdate(SObject oldSo, SObject so){

system.debug('--- afterUpdate is called');

}

public void afterDelete(SObject so){

system.debug('--- afterDelete is called');

}

public void andFinally(){

system.debug('--- andFinally called');

}

}

**Helper:-**

public class AppendPrefixBulkifyHelper {

public static void AppendPrefixBulkifyHelperAddNewAccount(List<Account> newList){

for(Account accountList: newList){

if(accountList.First\_Name\_\_c!= null){

accountList.Salutation\_\_c = 'Dr.';

}

}

}

}

**TASK 7:-**

**Make two address fields in opportunity , correspondence address and permanent address.**

**We have to fill both the fields with different addresses, if anyone of them is empty then generate message, and if the checkbox is selected then both the addresses should be same.**

**Handler:-**

public class AddressMatchHandler implements ITrigger

{

public void bulkBefore(){

AddressMatchHelper.onbulkBefore(Trigger.new);

}

public void bulkAfter(){

system.debug('--- bulkAfter is called');

}

public void beforeInsert(SObject so){

system.debug('--- beforeInsert is called');

}

public void beforeUpdate(SObject oldSo, SObject so){

system.debug('--- beforeUpdate is called');

}

public void beforeDelete(SObject so){

system.debug('--- beforeDelete is called');

}

public void afterInsert(SObject so){

system.debug('--- afterInsert is called');

TriggerHelperProductAddInOpportunity.ProductAddInOpportunity((Opportunity)so);

}

public void afterUpdate(SObject oldSo, SObject so){

system.debug('--- afterUpdate is called');

}

public void afterDelete(SObject so){

system.debug('--- afterDelete is called');

}

public void andFinally(){

system.debug('--- andFinally called');

}

}

**Helper:-**

public class AddressMatchHelper {

public static void onbulkBefore(List<Opportunity> listOfOpportunity)

{

for(Opportunity opportunityRecord: listOfOpportunity){

if((opportunityRecord.Permanent\_Address\_\_c == null || opportunityRecord.Correspondence\_Address\_\_c == null) && (opportunityRecord.Match\_Billing\_Address\_\_c == False)){

opportunityRecord.addError(' Both Address Field are Mandatory,and cannot be null');

}

else if((opportunityRecord.Correspondence\_Address\_\_c != null) && (opportunityRecord.Match\_Billing\_Address\_\_c == True)){

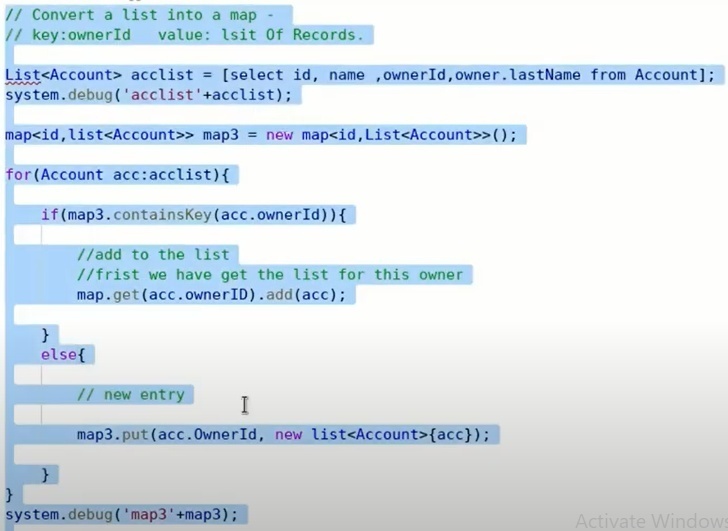
opportunityRecord.Permanent\_Address\_\_c = opportunityRecord.Correspondence\_Address\_\_c;

}

}

}

}



**Method1:**

List<Account> accList = [SELECT id,name, ownerId, owner.lastName FROM Account];

System.debug('accList' + accList);

Map<ID,list<Account>> IdToAccountMap = new Map<ID,list<Account>>();

for(Account acc: accList){

if(IdToAccountMap.containsKey(acc.ownerId)){

//add to the list

//First we have get the list for this owner

IdToAccountMap.get(acc.ownerId).add(acc);

}

else{

//new entry

IdToAccountMap.put(acc.ownerId, new list<Account>{acc});

}

}

system.debug('IdToAccountMap ' + IdToAccountMap );

**Method 2:**

How do we can convert below list to a map in Salesforce

list<contact> lstContacts = [SELECT Id,Name,Accountid,Phone FROM Contact];

System.debug('Contact List:'+lstContacts);

Map<id,list<contact>> mapAcccCOntacts = new Map<id,List<contact>>();

for(contact con : lstContacts){

if(mapAcccContacts.containsKey(con.accountid)){

list<contact> lstExistingCons = mapAcccContacts.get(con.accountid);

lstExistingCons.add(con);

mapAcccContacts.put(con.accountid,lstExistingCons);

}else{

list<contact> lstNewCons = new list<contact>();

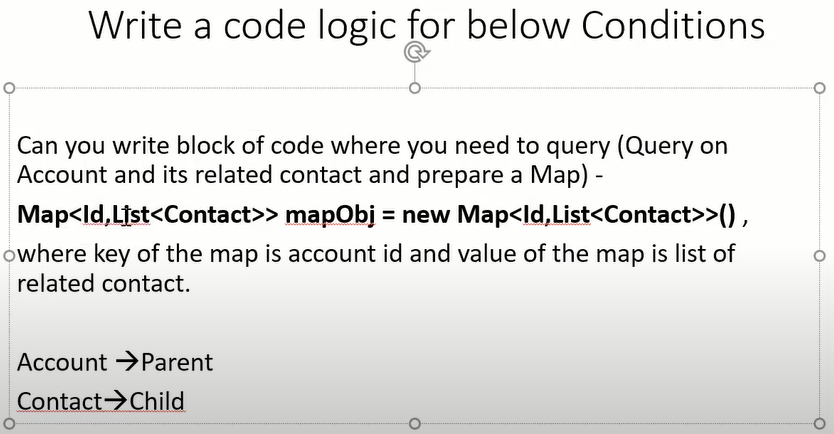
lstNewCons.add(con);

mapAcccContacts.put(con.accountid,lstNewCons);

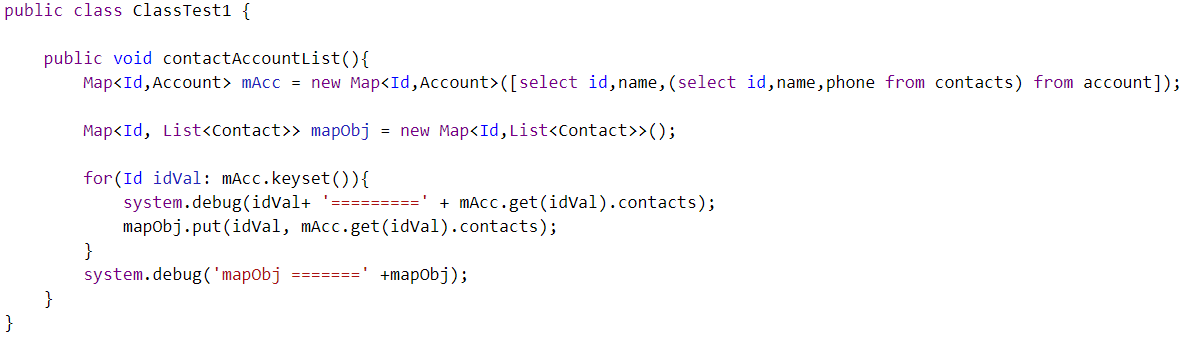
}

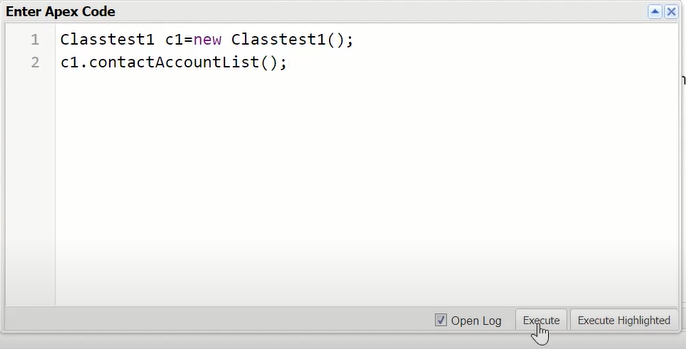
}

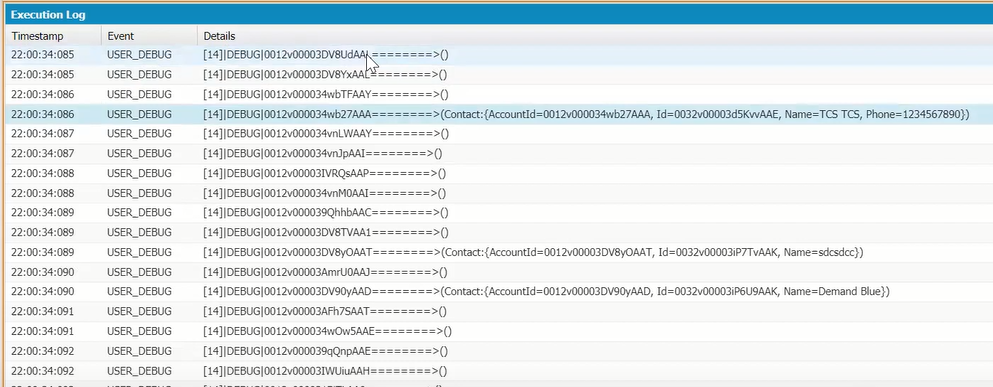
system.debug('Map Details :::' + mapAcccContacts);



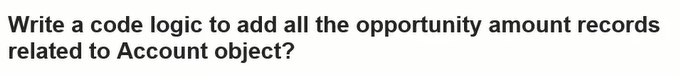


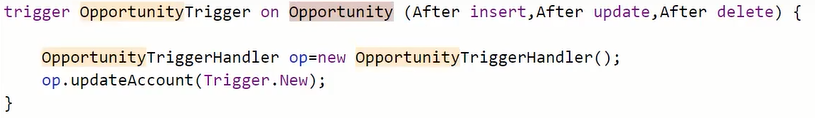


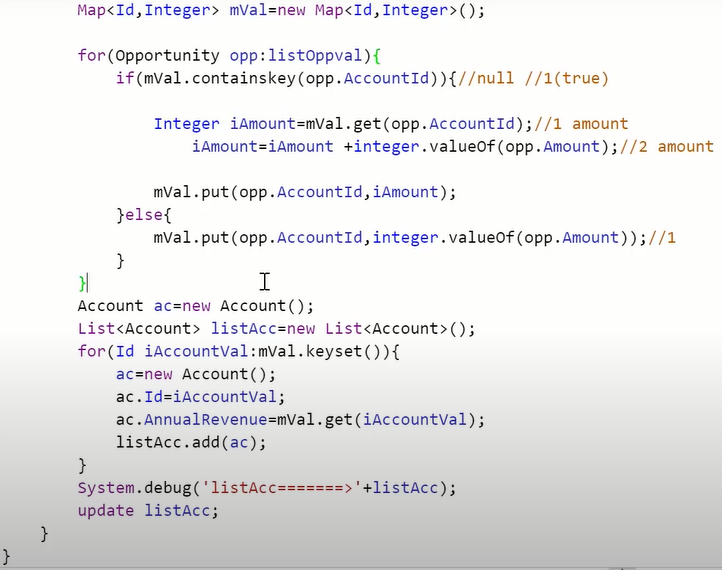
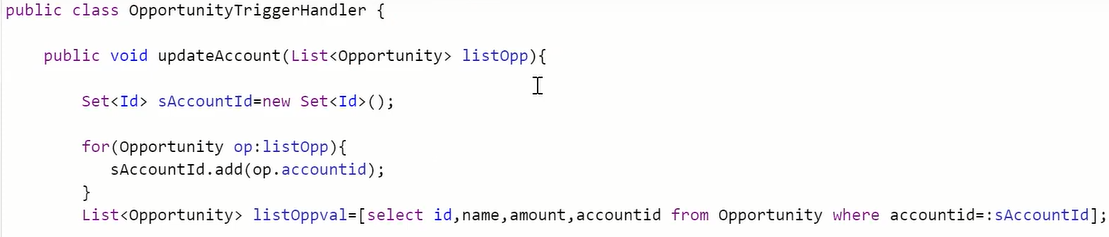


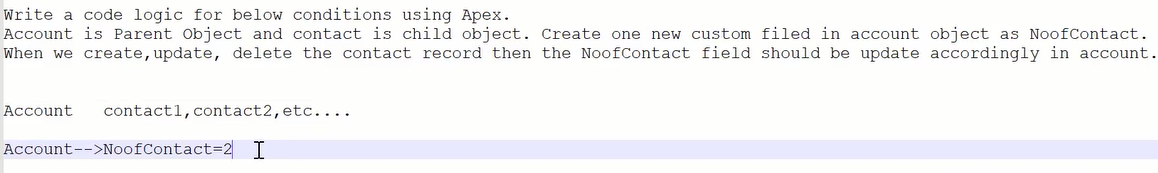


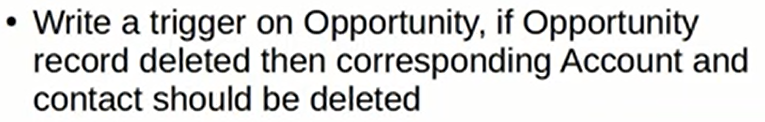
**Q. Write a Code logic to add all the opportunity amount records related to Account object?**

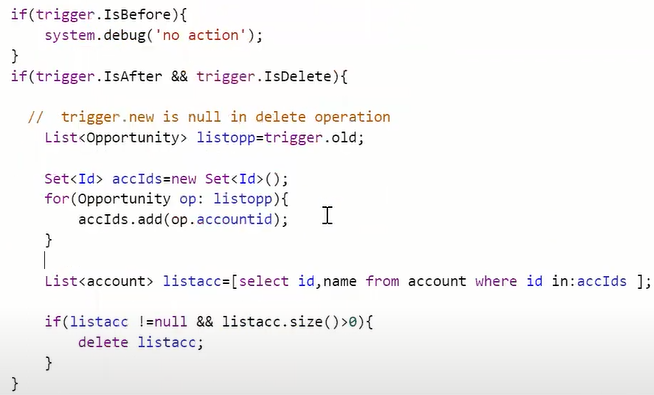
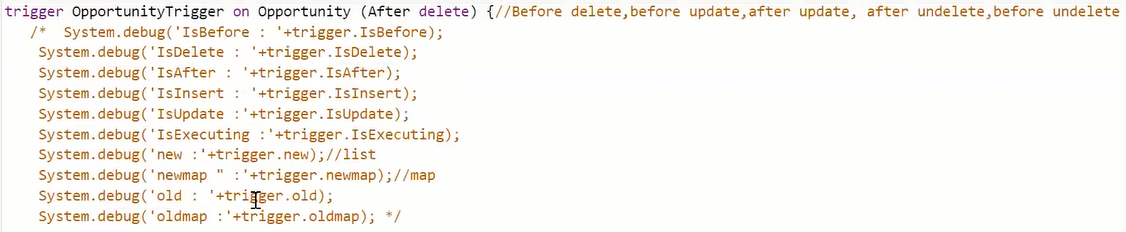
****

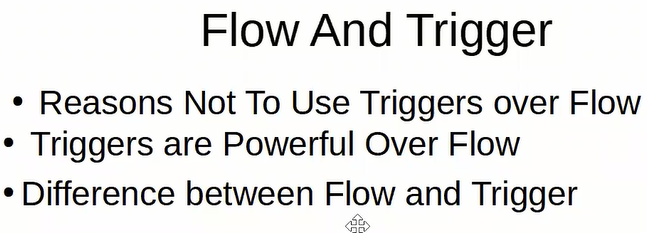




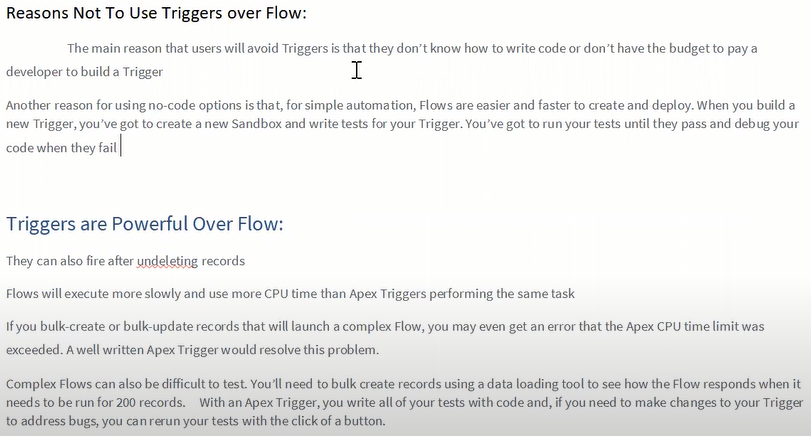


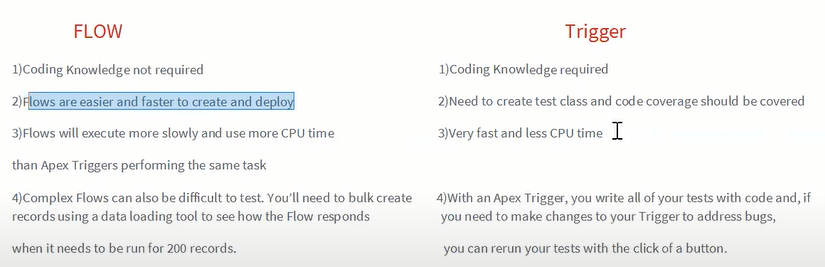


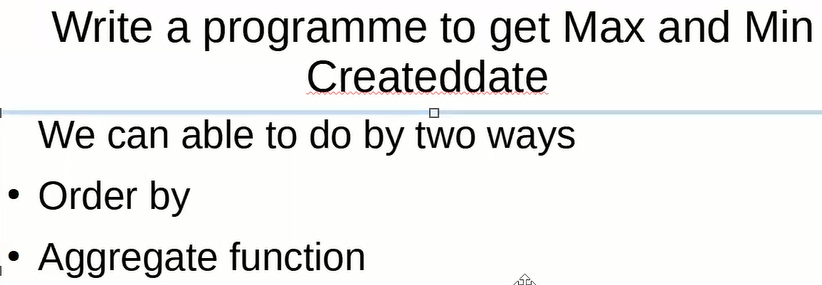


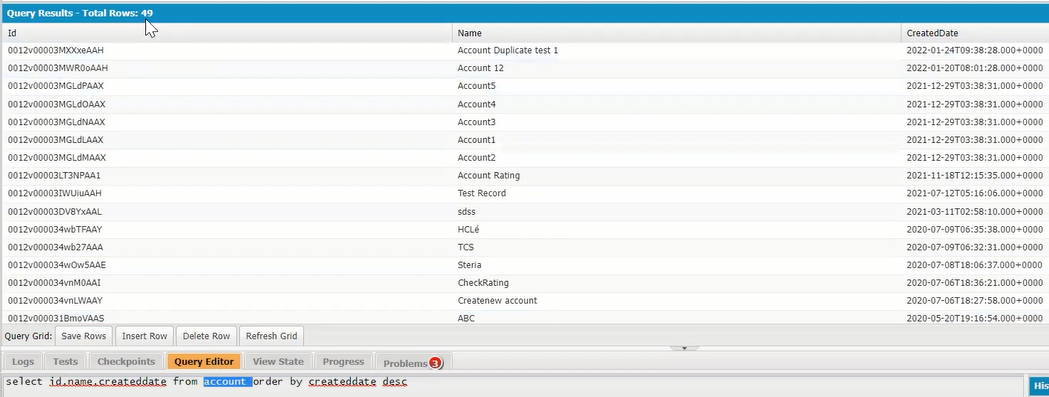








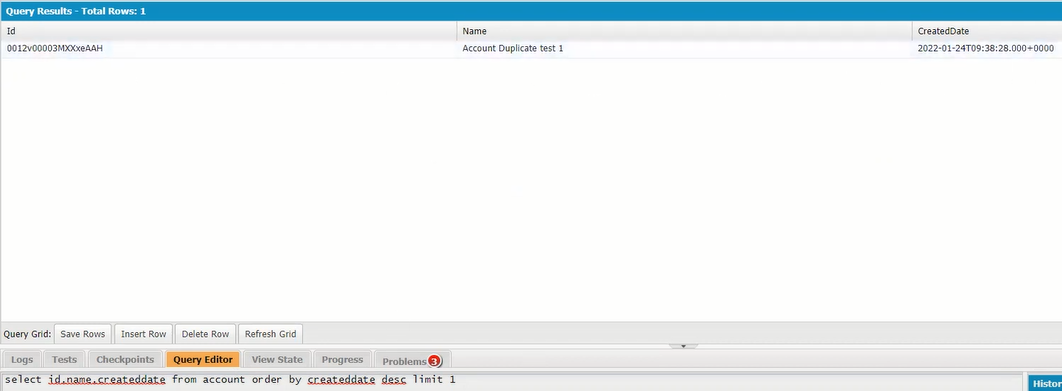




Select id,name,createddate from account order by createddate desc;

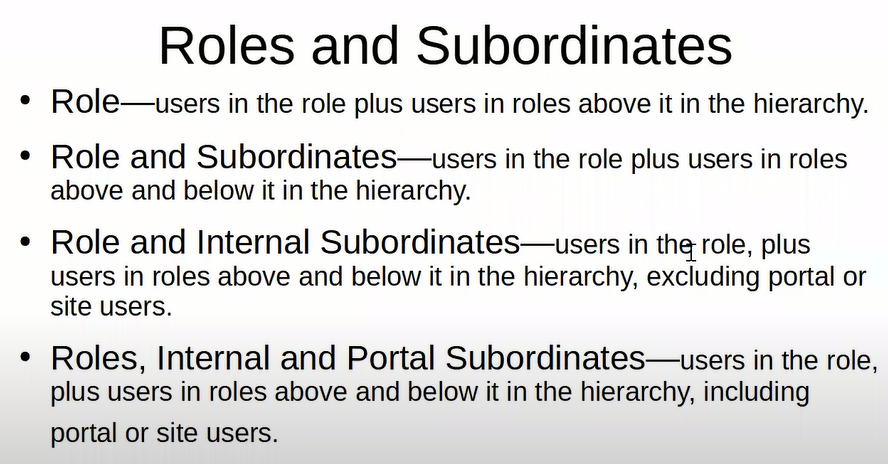
**How to get first record in salesforce?**

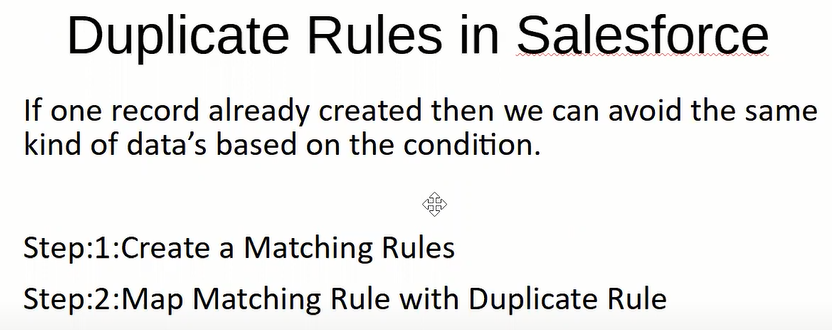
Select id,name,createddate from account order by createddate desc limit 1;

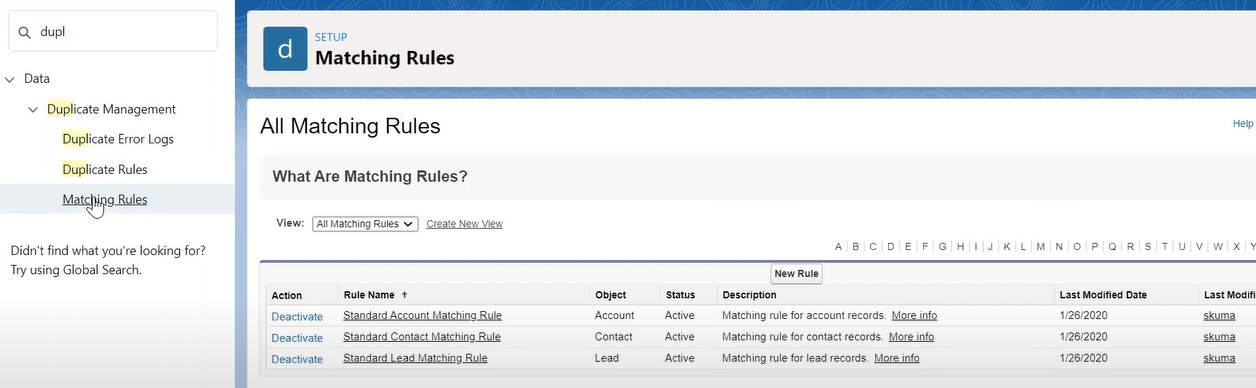


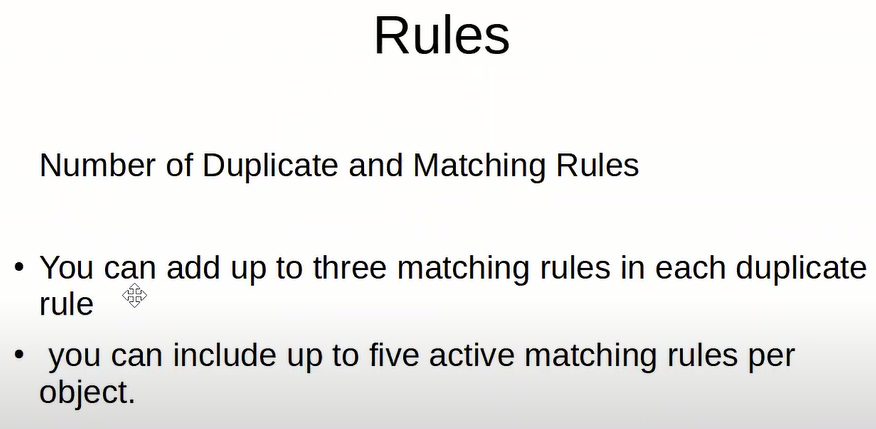
**How to get Last record in salesforce?**

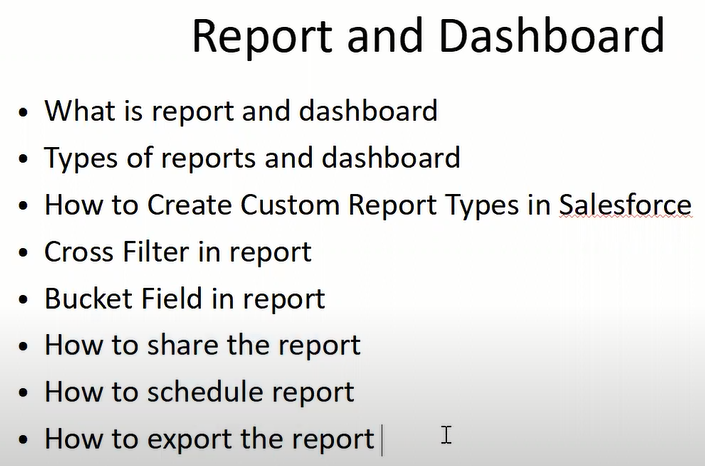
Select id,name,createddate from account order by createddate asc limit 1;











**Q. How to retrieve org deployed data of LWC?**

sfdx force:source:retrieve -x ./package.xml

**Q. Write a code logic whenever update or delete account phone number then account related all contact phone field should be updated with account phone number?**

