1. Write a trigger on Account, when an account is inserted, automatically account billing address should populate into the account shipping address.

When it comes to Account Billing and Shipping Address, we cannot directly enter the values to the field Address because the Address is split into Street, City, State, PostalCode, and Country. We are using ” before insert ” because we need to change the values of the account which is getting inserted. If we use ” after insert ” then the fields of the account we are inserting will become read-only and we will not be able to update that account. This is the reason we are using ” before insert “.

```
trigger changeShippingAddress on Account (before insert)
{
for(Account aObj : Trigger.new)
{
//We should first check whether the user is passing null data

if(aObj.BillingStreet!=null)
{
aObj.ShippingStreet=aObj.BillingStreet;
}
if(a.BillingCity!=null)
{
aObj.ShippingCity=aObj.BillingCity;
}
if(a.BillingState!=null)
{
aObj.ShippingState=aObj.BillingState;
}
if(a.BillingPostalCode!=null)
{
aObj.ShippingPostalCode=aObj.BillingPostalCode;
}
```

```
if(a.BillingCountry!=null)
{
aObj.ShippingCountry=aObj.BillingCountry;


}
}
}
```

2. Write a trigger on the Account when the Account is updated check all opportunities related to the account. Update all Opportunities Stage to close lost if an opportunity created date is greater than 30 days from today and stage not equal to close won.

Here we have to update Account related Opportunities based on a condition which is mentioned above in Question. So here we are using ” after update ” because once the Account is updated we are not making changes for the Account which is getting updated instead we are making changes for its related Object Opportunity.

First, we have to get the Account Id’s of the Account which is getting updated in a Set<Id> and write a query to get that particular Account Id’s Opportunities, then loop the list of that opportunities and check for the condition given in question then change the values according to the question and also check for the mandatory fields to update in opportunity. Later update the list of Opportunities for which the changes have been applied by adding those to a separate list and updating that list.

```
trigger OppoStageUpdate on Account (after update)
{
Set<Id> accountIds = new Set<Id>();
```

```
for(Account a:Trigger.new)
{

accountIds.add(a.Id);

}

//day30 is the date which is 30 days less than today

DateTime day30=system.now()-30;

List<Opportunity> oppListToUpdate=new List<Opportunity>();

//getting the opportunities whose account has been updated

List<Opportunity> oppList = [Select Id, AccountId, StageName, CreatedDate, CloseDate from
Opportunity where AccountId in :accountIds];

if(oppList.size()>0)
{
for(Opportunity opp : oppList)
{
```

```
//checking for condition if created date is greater than 30 days from today and stage not equal to close won


if(opp.CreatedDate<day30 && opp.StageName!='Closed Won')
{


opp.StageName='Closed Lost';    //This is a mandatory field when we update the CloseDate


opp.CloseDate=system.today();


oppListToUpdate.add(opp);  //putting the changed opportunity to separate list to update later
}


}


}


//checking if the opportunity list which has changed is having records or not


if(oppListToUpdate.size()>0)
{
```

```
update oppListToUpdate;



}
}
```

3. Once an Account is inserted an email should go to the System Admin user with specified text below.

An account has been created and the name is “Account Name”.

From the above question, we can clearly come to the conclusion that Email has to be sent after the Account is Inserted, so we use “after insert “. When there is bulk insertion then we have to send a list of emails that is why we are using List<Messaging.SingleEmailMessage>. Since the Email has to be sent to System Administrator, we should first get the Email of the Admin using the query. Then we have to loop the Accounts which are getting inserted, then check whether the email id of the Admin is null or not before creating a mail to send. Later for each Account, we have to add a single mail because when each Account is added the System Admin should get an email. Then pass the required fields for sending mail add it to the List of mails and check if the list of emails is empty or not. After that, we are ready to send the list of emails. We can also check whether the mail is sent or not, if not sent you can also check the errors in the logs using the last ” if ” condition which I have written below code.

```
trigger sendEmailToAdmin on Account (after insert)
{



//To send list of mails when there are insertion of list of Accounts



List<Messaging.SingleEmailMessage> mails = new List<Messaging.SingleEmailMessage>();
```

```
//Query to get the Email of a System Administrator

User userObj=[select Id,Profile.Name,Email from user where Profile.Name='System
Administrator'];

for(Account accObj:Trigger.new)
{

//Checking if the user email is not null

if(userObj.Email!=null)
{

//Assigning a single Mail to send

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

//Assigning the Sender Name for Mail

mail.setSenderDisplayName('Salesforce');

//We are make all this below fields as false because those are not needed for now
```

```
mail.setUseSignature(false);

mail.setBccSender(false);

mail.setSaveAsActivity(false);

//Assigning the receiver Mail Address

mail.toAddresses = new String[]{userObj.Email};

//Assign the Subject of the Mail

mail.setSubject('New Account was Created.');

//A variable to write the body of the Mail

String body = 'Dear System Administrator, <br/>';

body += 'An account has been created and name is '+accObj.Name+'.';

//Assigning the variable in which we wrote the body to the Mail Body
```

```
mail.setHtmlBody(body);

//Adding each single mail to be sent to the list of mails

mails.add(mail);

}

}

//Checking if the list of mails is not empty

if(mails.size()>0)
{

//" Messaging.sendEmail(mails) " is used to send the list of mails

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

//we are checking if the mails are sent or not.
```

```
if (results[0].success)

{

System.debug('The email was sent successfully.');

}
else
 {

System.debug('The email failed to send: '+ results[0].errors[0].message);

}
}
}
```

4. Once an Account will update then that Account will update with the total amount from All its Opportunities on the Account Level. The account field name would be ” Total Opportunity Amount “.

The first thing we have to see is which trigger to use ” before ” or ” after “, here we have to update the Account itself again so we cannot use the ” after ” trigger because once the Accounts are updated the fields of the Account becomes ” read-only ” i.e, we will not be able to do changes to the Account data. So if we use ” before update ” then we will be able to make changes to the Account fields later it will get updated by default. This is the reason we are using ” before update “.

In the below code first, we are getting the list of Account Id's to get its related Opportunities. We have created a Map<Id,Double> where Id contains the Account Id and Double contains the sum of Amount of a particular Account's related Opportunities so that it will be easy for us to insert the sum of amount according to the Account Id's. Then we are getting the "Aggregate Result" i.e, we use this when we use sum or average in a query. We are putting all the data of the Aggregate Result to the Map which we have written before. Later we are looping the Accounts which are getting updated and check if that Account Id is there in Map, if it is there then we are changing the value of the " Total Amount Opportunity " field in Account.

```
trigger totalAmountofRelatedOpp on Account (before update)
{

//Get the Account Id's of the List of Accounts getting updated to get its related Opportunities

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

for(Account acc:Trigger.new)
{

//before making changes see to that "Total Opportunity Amount" field is 0

acc.Total_Opportunity_Amount__c=0;

accId.add(acc.Id);

}
```

```
//map to get the Account Id and the sum of its related opportunities amount to insert later

Map<Id,Double> amountMap = new Map<Id,Double>();

//AggregateResult to get the sum of opportunities amount grouped by Account Id

List<AggregateResult> results=[select AccountId,sum(Amount)TotalAmount from opportunity
where AccountId in :accId group by AccountId];

if(results.size()>0)
{
for(AggregateResult a: results)
{

//getting the AccountId and sum(Amount) in separate variables and putting it to map

Id accountId = (Id)a.get('AccountId');

double TotalAmount = (double)a.get('TotalAmount');

amountMap.put(accountId,TotalAmount);
```

```
}
}
//Again looping the accounts which are getting updated and making changes in "Total Opportunity Amount" field
for(Account acc:Trigger.new)
{
if(amountMap.containskey(acc.Id))
{
acc.Total_Opportunity_Amount__c=amountMap.get(acc.Id);
}
}
}
```

5. Create a field on Account Named (Client Contact lookup to Contact). Once an Account is inserted a Contact will create with the name of the Account and that Contact will be the Client Contact on the Account.

Here we are using “after insert” because here the condition is first we should insert contact and then a related contact with the same name as account name, so to get the Account Id to make its related contact first the Account has to be inserted then only we will be able to get the Account Id. So we have to use ” after insert”.

We are looping the Accounts which are getting inserted and then also creating a Contact that has the same name as the Account and we are making it related to that particular Account only. Immediately after this, we are inserting those Contacts. Then we are getting the records of the inserted Account to put it into the Map<Id,Account> where Id contains Account Id and Account contains that particular Id’s records. After this, we are looping the inserted Contact and checking if the Contact’s Account Id is present in the Map. If present we are getting that particular Account’s record and given the Contact Id value to the field “Client Contact” of Account. Then we are adding it to a separate list to update it later. And finally, we are updating the Accounts for which the changes have been done.

trigger updateClientContactFieldOnAccount on Account (after insert)

```
{
//Account list to update after the doing changes

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

//Contacts to insert which has same name as Account Name

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

//Account Id list to update later after changes are done

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

//Map of Account Id and the Account record

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

for(Account accObj : Trigger.new)
{
//Contact which has to be inserted with same name as Account and putting that Contact as related Contact

Contact conObj =new Contact();
conObj.LastName = accObj.Name;
```

```
conObj.AccountId = accObj.Id;
conList.add(conObj);
accId.add(conObj.AccountId);
}
if(conList.size()>0)
{
insert conList;
}
//Getting list of Accounts to put into Map
List<Account> accList =[Select Id,Client_Contact__c from Account where Id=:accId];
if(accList.size()>0)
{
for(Account acc:accList)
{
mapVariable.put(acc.Id,acc);
}
}
if(conList.size()>0)
{
/*Looping inserted Contact and Check whether the Account Id is related and get its Account
record from map and change the value of the Client Contact with the current looping Contact*/
for(Contact cObj:conList)
{
if(mapVariable.containsKey(cObj.AccountId))
{
Account aObj=mapVariable.get(cObj.AccountId);
aObj.Client_Contact__c=cObj.Id;
accListUpdate.add(aObj);
}
}
```

```
}
//Update the Account in which chnages has been done
if(accListUpdate.size()>0)
{
update accListUpdate;
}
}
```

6. Create a field on Opportunity Line item(Serial No (Text)) and populate increment values once an Opportunity Line Item is added. Let's say if we add 3 products then the sequence would be "1,2,3". Now if we delete 2 and again add one more product this must be 4 irrespective of the deleted sequence number.

The very first thing we have to do is create a field with the name "Serial No" of datatype "Text" in an "Opportunity Product" (also called as Opportunity Line Item). Later check whether there is "Product" on the related tab of "Opportunity". So when you add a new "Product" in a particular Opportunity this Trigger is going to work. And also we will write a trigger on insert because the question tells on adding a product.

Here we are using "before insert" because when we add a new Product on a particular Opportunity then we also need to update the Serial No field of that particular OpportunityLineItem. And if we use " after insert " then the OpportunityLineItem field will become read-only and we will not be able to update that OpportunityLineItem.

In the below code we are getting all the OpportunityId in one Set so that we can get all the OpportunityLineItem of that particular Opportunity and the Serial No of that Opportunity Product. We are also using Map<Id, String> where Id contains OpportunityId and String contacts the new Serial No of that particular Opportunity Product. We are looping the Opportunity Line Item's which we got through the query and checking if the Serial NO field is null, if it is null then we are making the Serial No field as 1. If not then we are getting the last number of the old value of the Serial No and converting it to Integer because we need to Increment it and append it. Then we are making chances in the Serial No field then adding the OpportunityLineItem to List so that we can update it later, and also adding the values to the Map. After this, we are updating the Opportunity Line Item's which was previously present, and in which we made changes for the

Serial No. Now the only thing left is changing the Serial No field of the newly inserting OpportunityLineItem, so we loop the inserting List of OpportunityLineItem check if its OpportunityId is present in Map, if present then get that Map values and place it to Serial No field else we are just passing 1 to that Serial No field.

```
trigger updateSerialNoOnOppoLineItem on OpportunityLineItem(before insert)

{

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

for(OpportunityLineItem oppProd:Trigger.new)

{

oppId.add(oppProd.OpportunityId);

}

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

//Map of Opportunity Id and SeralNo of OpportunityLineItem

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

List<OpportunityLineItem> oppProdList=[Select Id,Serial_No__c,OpportunityId from
OpportunityLineItem where OpportunityId=:oppId];

if(oppProdList.size()>0)

{

for(OpportunityLineItem oppoPrd:oppProdList)

{

if(oppoPrd.Serial_No__c!=null)

{

//Getting the last number of the SerialNo field

String lastword = oppoPrd.Serial_No__c.right(1);

//converting String into Integer so that I can increment the last number which is already there in
SerialNo field

Integer num = Integer.valueOf(lastword);

num++;

//appending the incremented number
```

```
oppoPrd.Serial_No__c=oppoPrd.Serial_No__c+','+(num);

oppProdListTOUpdate.add(oppoPrd);

//adding it to Map so that i can make changes for the OpportunityLineItem which is newly getting inserted

oppIdSerialNoMap.put(oppoPrd.OpportunityId,oppoPrd.Serial_No__c);

}

Else

{

//if the SerialField is null then it means there is no OpportunityLineItem present so by default we are passing 1

oppoPrd.Serial_No__c='1';

oppProdListTOUpdate.add(oppoPrd);

}

}

}

if(oppProdListTOUpdate.size()>0)

{

update oppProdListTOUpdate;

}

for(OpportunityLineItem oppProd:Trigger.new)

{

/*Checking if the OpportunityId of newly inserting OpportunityLineItem is present in the Map

If present we are passing that SerialNo to the newly inserting OpportunityLineItem else we are just adding 1 because

this is the first OpportunityLineItem getting added*/

if(oppIdSerialNoMap.containsKey(oppProd.OpportunityId))

{

oppProd.Serial_No__c = oppIdSerialNoMap.get(oppProd.OpportunityId);

}

Else

{
```

```
oppProd.Serial_No__c='1';
}
}
}
```

Apex

Trigger Scenarios in Salesforce with Solution – Part 4

May 21, 2022 Chaitra CS

Rollup summary trigger for lookup relationship

Apex

Rollup Summary Trigger for Lookup Relationship to count child records

March 27, 2022 Chaitra CS

Post navigation

Previous:

Collection Filter in Salesforce Flow Builder – Spring22 Update

Next:

Trigger Scenarios in Salesforce with Solution – Part 2

5 thoughts on “36 Trigger Scenarios in Salesforce with Solution – Part 1”

Pingback: 36 Trigger scenarios for practice in Salesforce: Basic to Advance – Salesforce Geek

harold JOSOA says:

July 25, 2022 at 11:34 am

How to write a trigger to update the field City, Street, Country, Postal Code in all related opportunities,

when the same corresponding fields (from Billing Address) is updated in an account?

Reply

Chaitra CS says:

July 26, 2022 at 10:03 am

Hi Harold,

Write a Trigger on Account for after update and check if there is change in Billing Address of Account if yes then assign the updating Account Ids in a set. Then get all the Opportunities related to those Account Ids then loop all those opportunities and assign its particular account's Billing address to those fields of Opportunity. Later update those Opportunities whose values are changing.

Thank You.

Reply

Rahul says:

July 26, 2022 at 5:01 am

// could you please tell me why my code is not working

I am new in Salesforce and I am trying to learn salesforce concepts

Please tell me

When Primary Contact is True then only selected Contact address is same as Account address

```
trigger UpdateAddress on Contact (after update)
 {
Set contactIds = new Set();
```

```
for (Contact a : Trigger.new)
{
Contact old = Trigger.oldMap.get(a.Id);
if (a.BillingStreet != old.BillingStreet || …)
{
contactIds.add(a.Id);
}
}
if (contactIds.size() > 0)
{
Account[] updates = [select Id, AccountId from Account where AccountId in :contactIds];
for (Account c : updates)
{
Contact a = Trigger.newMap.get(c.ContactId);
c.BillingStreet = a.BillingStreet;
}
update updates;
}
}
```

Hi Rahul,

In Your trigger you are getting the Contact Ids in “contactIds” variable and In the query your passing It for Account. And the query is Completely wrong because in Account there isn’t any field called AccountId. Check those again and re-write it.

Thank You

Reply

Leave a Reply

Your email address will not be published. Required fields are marked *

Trigger Scenario 1 :*When ever a case is created with origin as email then set status as new and Priority as Medium.*

Object : Case

Trigger: Before Insert

Trigger Code: CaseOrigin.apxt

```
trigger CaseOrigin on Case (before insert)
 {
for(case c : trigger.new)
{
 if(c.origin == 'Email')
{
          c.status = 'New';
          c.priority = 'Medium';
        }
    }
}
```

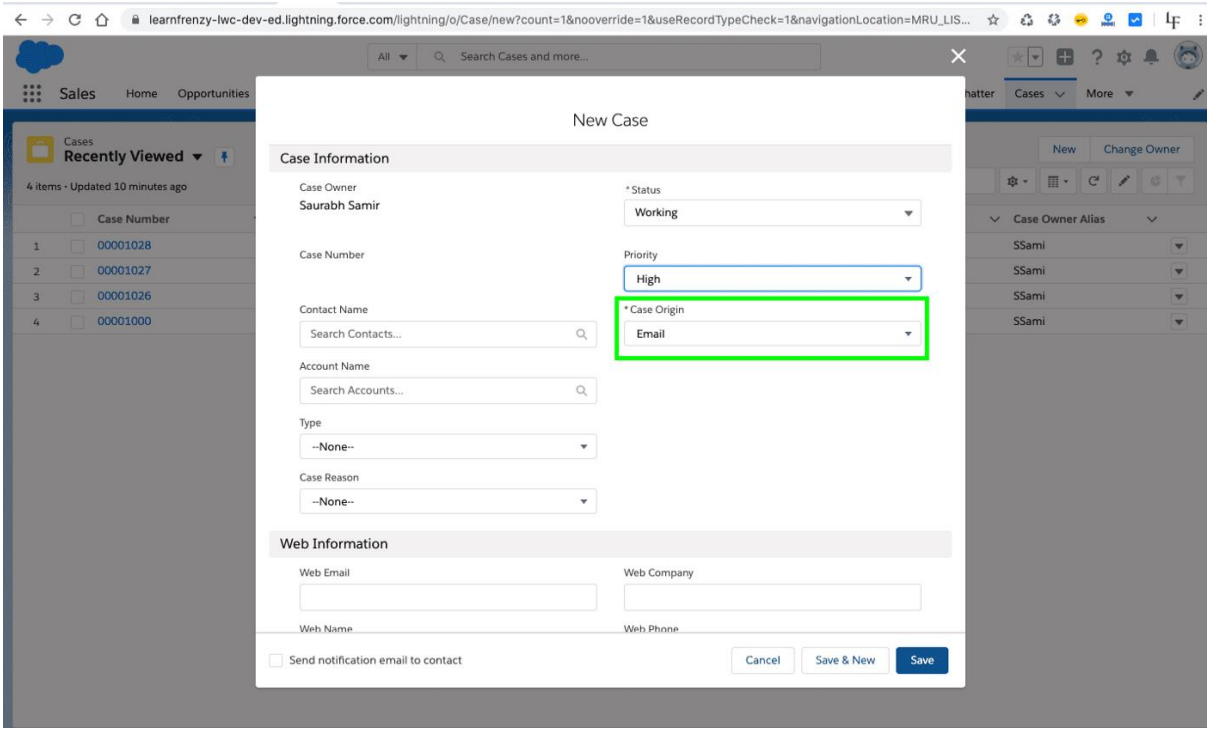
Output :

–> Case is created with Origin as “Email” :

Status : Working

Priority : High

Case Origin : Email



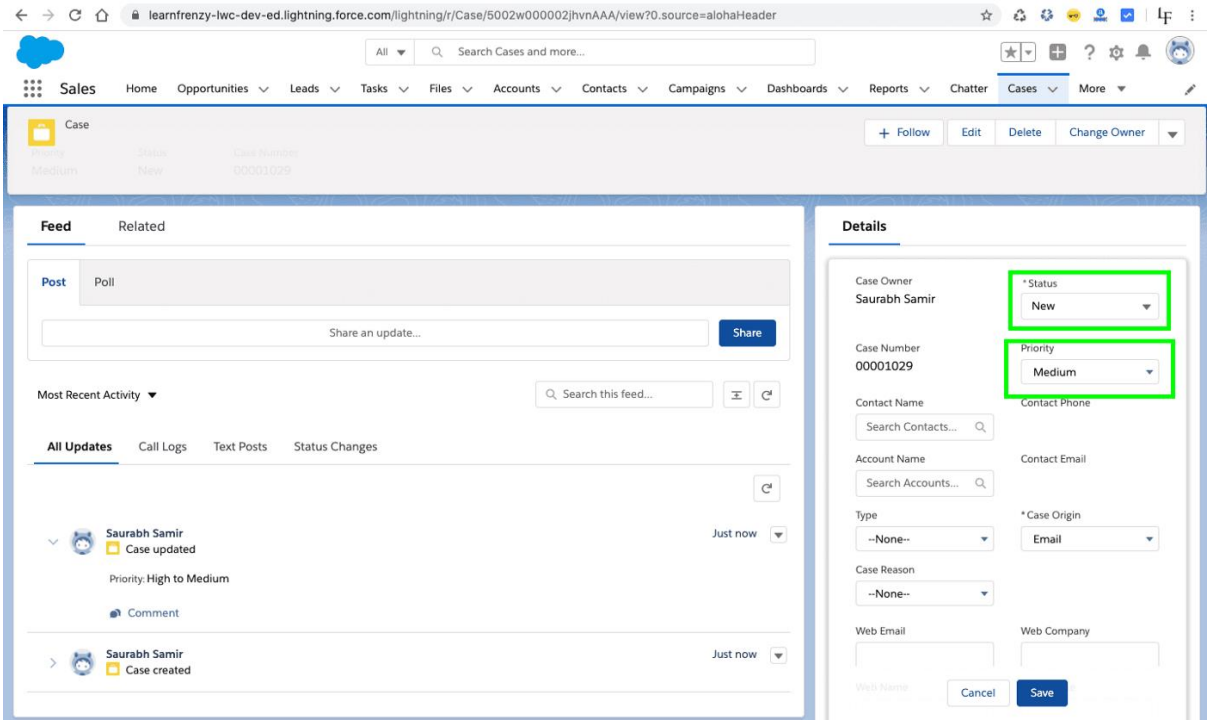
–> Before Insert :

As per the requirement, we are performing an operation on the trigger when the user save the new case that means we need to use as before insert trigger.

Status : New

Priority : Medium

Case Origin : Email



Trigger Scenario 2 :

When ever Lead is created with LeadSource as Web then give rating as cold otherwise hot.

Object : Lead

Trigger: Before Insert

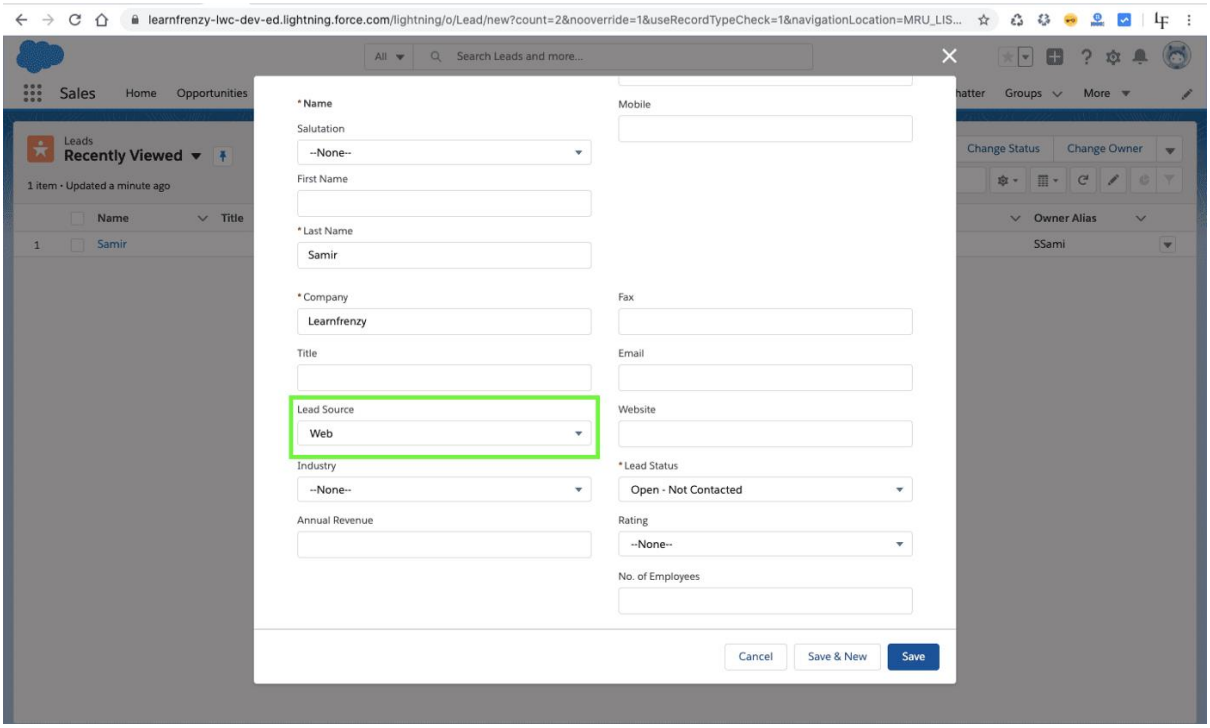
Trigger Code: LeadScenario.apxt

```
trigger LeadScenario on Lead (before insert)
 {
for(lead ld : trigger.new){
if(ld.leadsource == 'Web')
 {
  ld.Rating = 'Cold';
 }
Else
{
ld.Rating = 'Hot';
```

```
  }
 }
}
```

Output:

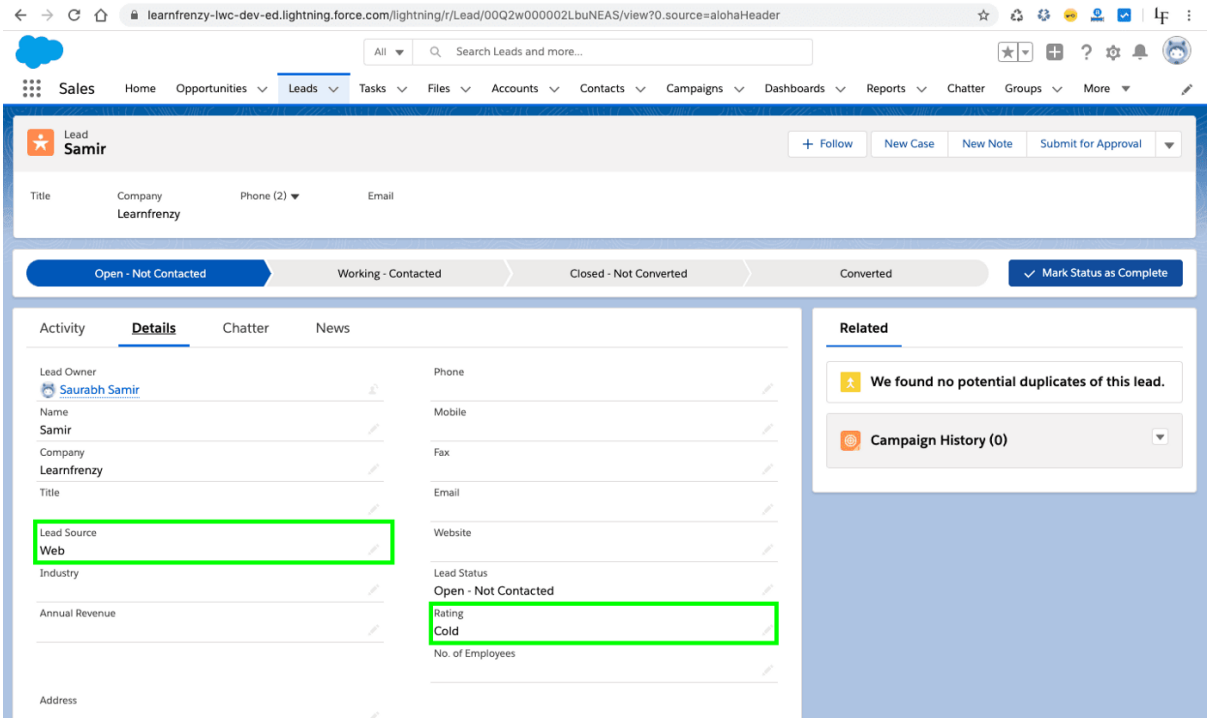
–> Lead is created with LeadSource as Web :



–> Before Insert :

As per the requirement, we are performing an operation on the trigger when the user save the new lead that means we need to use as before insert trigger.

Here the user has selected the lead source as ‘web’, so the rating will be ‘cold’



Trigger Scenario 3 : ***Whenever New Account Record is created then needs to create associated Contact Record automatically***.

Object : Account

Trigger: After Insert

Description : When ever new Account record is successfully created, then create the corresponding contact record for the account with:

•account name as contact lastname

•account phone as contact phone

Trigger Code: AccountAfter.apxt

```
trigger AccountAfter on Account (after insert)
 {
List<contact> cons=new List<contact>();
    for(Account acc: Trigger.New)
{
        Contact c=new Contact();
        c.accountid=acc.id;
        c.lastname=acc.name;
        c.phone=acc.phone;
```

```
            cons.add(c)
          }
  insert cons;
}
```

Test Class:

```
@isTest
private class AccountAfterHandler
 {
@isTest
static void testme()
{
Integer count=[select count() from Account];
Integer size=[select count() from Contact];
Account acc=new Account(Name='LearnFrenzy',phone='022-845454');
    Try
{
       insert acc;
       }
    catch(Exception e)
{
       System.debug(e);
      }
Integer newCount=[select count() from Account];
Integer newsize=[select count() from Contact];
Contact c=[select lastname,phone from Contact where accountid=:acc.id];
System.assertEquals(c.lastname,acc.name);
System.assertEquals(c.phone,acc.phone);
  }
}
```

Trigger Scenario 4 :

When ever the Account is created with Industry as Banking then create a contact for account, Contact Lastname as Account name and contact phone as account phone.

Object : Account

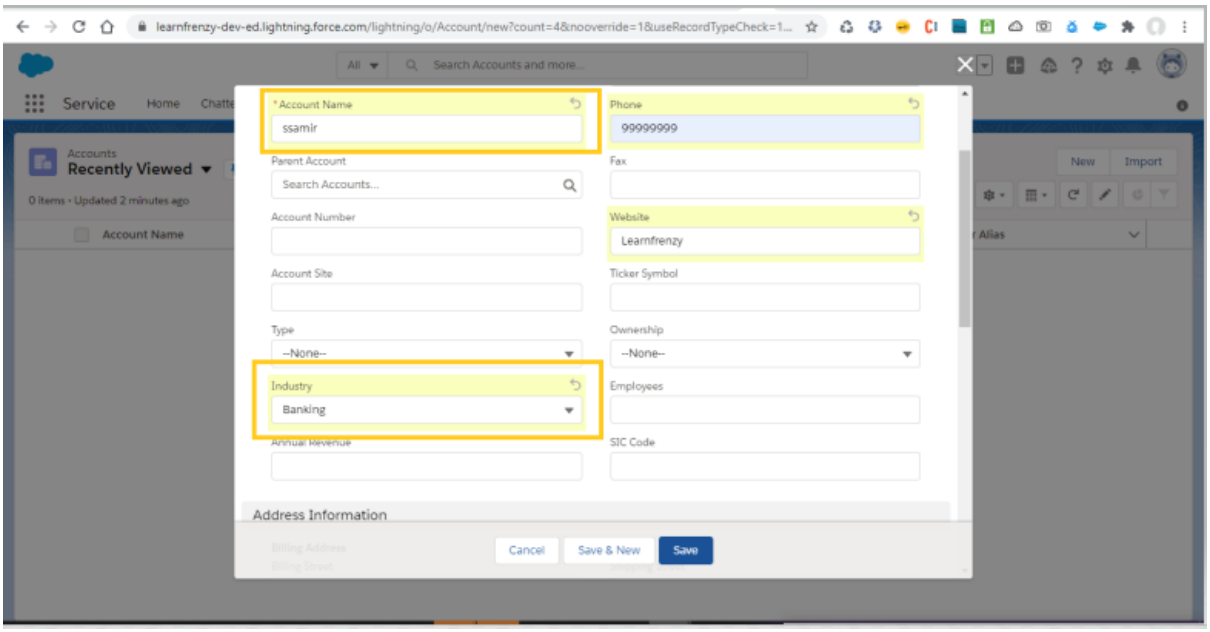
Trigger: After Insert

Trigger Code: CreateAccountContact.apxt

```
trigger CreateAccountContact on Account (after insert)
{
list<contact> contacts = new list<contact>();
   for(account acc : trigger.new)
{
      if(acc.Industry == 'Banking')
{
           contact con = new contact();
           con.LastName = acc.name;
           con.Phone = acc.Phone;
           con.AccountId = acc.Id;
           contacts.add(con);
       }
    }
insert contacts;
}
```

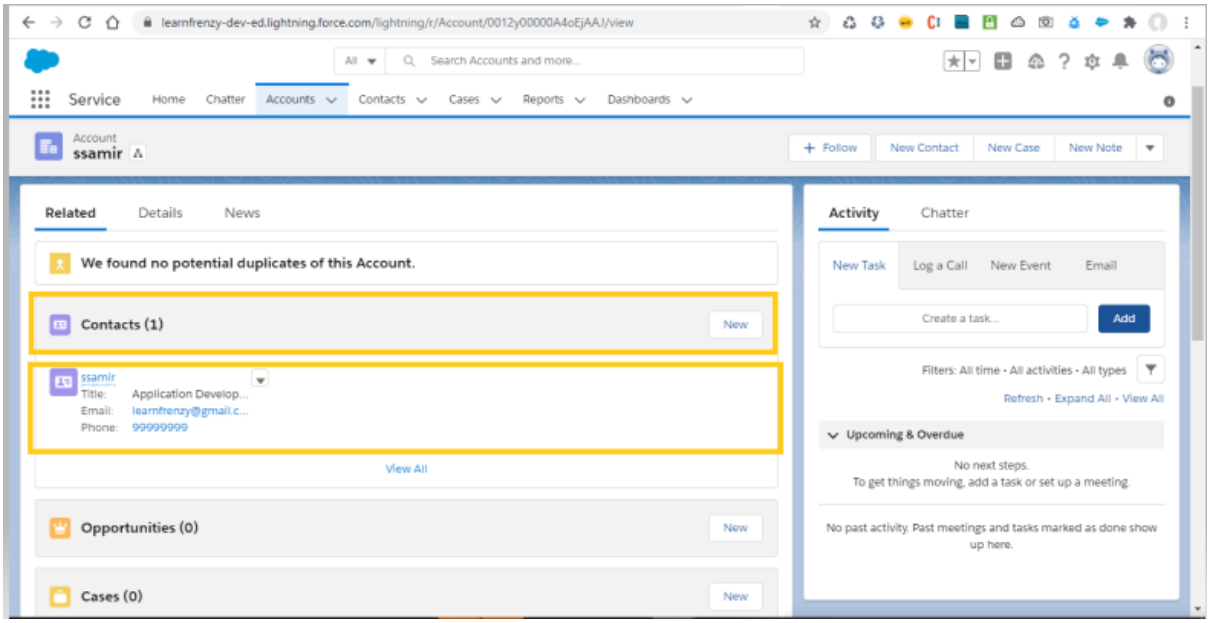
Output:

–> New Account is created with Industry as Banking :



–> After Insert :

As per the requirement, we are performing an operation on the trigger when the user save the new account that means we need to use as after insert trigger.



AFTER triggers are usually used when information needs to be updated in a separate table/object due to a change. They run after changes have been made to the database (not necessarily committed).

Trigger Scenario 5 :

Creates the number of contacts which are equal to the number which we will enter in the Number of Locations field on the Account Object.

Create Custom field called “Number of Locations” on the Account Object (Data Type=Number)

Object : Account

Trigger: After Insert

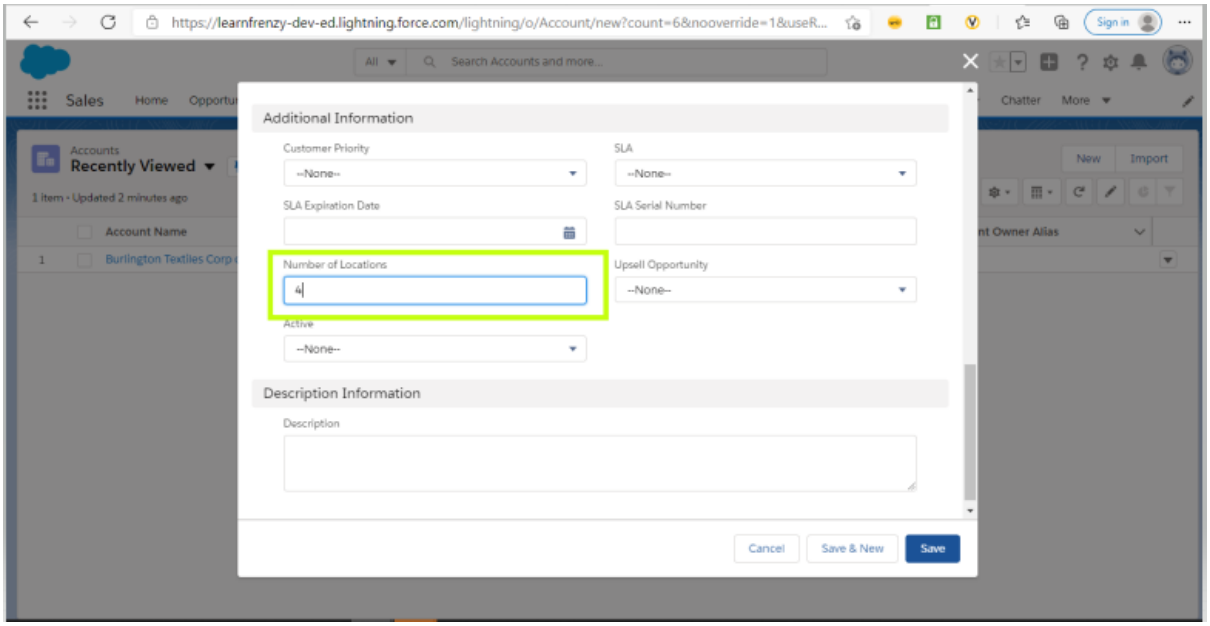
Trigger Code: ContactsCreation.apxt

```
trigger ContactsCreation on Account (after insert)
 {
list<contact> listContact = new list<contact>();
map<id,decimal> mapAcc = new map<id,decimal>();
 for(Account acc:trigger.new)
{
    mapAcc.put(acc.id,acc.NumberofLocations__c);
   }
      if(mapAcc.size()>0 && mapAcc!=null)
{
         for(Id accId:mapAcc.keyset())
{
           for(integer i=0;i<mapAcc.get(accId);i++)
{
             contact newContact=new contact();
             newContact.accountid=accId;
             newContact.lastname='contact'+i;
             listContact.add(newContact);
             }
          }
      }
if(listContact.size()>0 && listContact!=null)
insert listContact;
}
```

Output:

–> Enter in the Number of Locations field on the Account Object. :

Here the user has created a new account ‘LearnFrenzy’ and the Number of Locations is 4.



–> After Insert :

As per the requirement, we are performing an operation on the trigger when the user creates the number of contacts which are equal to the number which we will enter in the Number of Locations field on the Account Object. That means we need to use as after insert trigger.

Comment *