

## **Trigger:**

→**Validations**

→**VALIDATION AND INSERTIONS**

→**Update on same object or related object**

→**compute a value [summary count,sum,max,min]**

→**send an email**

Scenario1: Create a trigger that will prevent Inserting an account without the type field of an account object.

**Object:Account**

**Event:Before insert.**

trigger scenario1 on Account (before insert)

{

for(account ac:trigger.new)

{

if(ac.type==null)

{

ac.adderror('Account type is required');

ac.type.adderror('Please enter account type');

}

}

}

Scenario2: Create a trigger that will prevent Inserting an account without the type field and Industry field of an account object.

**Object:Account**

**Event:Before insert.**

```
trigger scenario2 on Account (before insert) {  
    {  
        for(account ac:trigger.new)  
        {  
            if(ac.type==null)  
  
            {  
                ac.type.adderror('Accounttype is required');  
  
            }  
            else if(ac.industry==null)  
            {  
                ac.industry.adderror('Please enter industry field');  
            }  
        }  
    }  
}
```

Scenario3: Create a trigger that will prevent Inserting an contact without the email field of a contact object.

**Object:**

**Event:**

```
trigger scenario3 on Contact (before insert) {  
    for (contact c:trigger.new)
```

```

{
    if(c.Email==null)
    {
        c.email.adderror('field is empty');
    }
}

```

Scenario4: Create a trigger that fires when the user inserts an account with the name 'axy1' then the rating would be 'High'.

**Object:**

**Event:**

trigger scenario4 on Account (before insert)

```

{
for(Account ac:trigger.new)
{
    if(ac.name=='axy1' )
    {
        ac.Rating='Hot';
    }
}
}

```

Scenario5: Create a trigger that will not allow the users to create a contact with their account name. Throw an error if the user tries to create the account.

**Object:Contact**

**Event:Before insert**

trigger scenario5 on Contact (before insert)

```
{
  for(contact ac:trigger.new)
  {
    if(ac.accountid==null) //accountid is field name and label is account name
    {
      ac.adderror('Account name is required');
      ac.accountid.adderror('Please enter account name');
    }
  }
}
```

Scenario6: Create a trigger that will not allow the users to create a duplicate account name. Throw an error if the user tries to create an account with the same name.

**Object:Account**

**Event:Before insert**

Testing:

1. Get the name of an account that the user insert.
2. Fetch the record from the database.
3. Compare and restrict it.

trigger scenario6 on Account (before insert)

```
{
```

```

list<string> str1=new list<string>();
for(account a:trigger.new)
{
    str1.add(a.name);
}

list<account> aclist=[select name from account where name in :str1]; // fetch
the match record from database

```

```

for(account a1:trigger.new)
{
    for(account a2:aclist)
    {
        if(a1.name==a2.name)
        {
            a1.adderror('duplicate record exists');
        }

    }
}

}

```

Scenario7:Create a trigger on account if Type is Prospect then rating would be 'High' if Type is customer - direct then rating would be warm.

```

trigger scenario7 on account (before insert)
{

```

```
for(account ac:trigger.new)
{
    if(ac.type=='prospect')
    {
        ac.rating='hot';

    }
    if(ac.type=='customer - direct')
    {
        ac.rating='warm';
    }
}
}
```

Scenario8:Write a trigger to update the account phone number to the contact phone number field,when user inserts and updates contact phone number and account phone number must be null.

Scenario7: Create a trigger on account if Type is Prospect then rating would be 'High' if Type is customer - direct then rating would be warm.

Scenario8: Write a trigger to update the account phone number to the contact phone number field, when user inserts and updates contact phone number and account phone number must be null.

Diagram illustrating the relationship between Contact and Account:

- Contact (phone: 3424324) is linked to Account (phone number: 3424324, Phone number must be null).
- Accountid[ contact] --> list
- contact name: Kiran, Account: XYZ

trigger scenario7 on Contact (after insert, after update)

```
{
```

```
list<string> ab=new list<string>();
```

```
for(contact c:trigger.new)
```

```
{
```

```
    ab.add(c.accountid);
```

```
}
```

```
list<account> aclist=[select phone from Account where id in :ab];
```

```
for(contact con:trigger.new)
```

```
{
```

```
    for(account ac:aclist)
```

```
    {
```

```
        if(ac.id==con.accountid && ac.phone==null)
```

```
        {
```

```

        ac.phone=con.phone;
    }
}

update aclist;
}

```

Scenario9: Write a trigger to insert or update the contact phone number, when the user inserts and updates Account phone number and contact phone number must be null.

trigger scenario9 on Account (after update)

```

{
list<string> strlist= new list<string>();
    for(account a:trigger.new)
    {
        strlist.add(a.id);
    }

    list<contact> clist=[select id,phone,accountid from contact where accountid in
: strlist];

    for(account acc:trigger.new)
    {
        for(contact conlist:clist)
        {
            if(acc.id==conlist.accountid && conlist.phone==Null)
            {
                conlist.phone=acc.phone;
            }
        }
    }
}

```



```

    }
  }
}
update clist;
}

```

Scenario10: Write a trigger to Insert contact name Tantul and last name when user creates an account with name Arizona.

Account: → Arizona

Contact: Name last name Accountname [insert ]

object: Account

Event: After Insert

trigger Scenarios10 on Account (after insert)

```

{
  list<contact> conlist= new list<contact>();
  conlist.add(new contact(firstname='Shweta',lastname='jha'));

  for(account ac:trigger.new){

    if(ac.name=='Arizona1')
  {
    insert conlist;

  }
}

```

}

Send a mail with the Trigger

**Compute a value of aggregate functions with the trigger.[sum,max,min]**

Scenario10: Create a trigger that will display the max and min opportunity of an account.

**Testing:**

The screenshot shows a Google Docs document titled "class\_codes" with the following content:

Send a mail with the Trigger

**Compute a value of aggregate functions with the trigger.[sum,max,min]**

Scenario10: Create a trigger that will display the max and min opportunity of an account.

**Testing:**

A diagram shows a relationship between "Account" and "Opportunity".

Id	AccountId
HCL LTD	Opp1 hcltd 8900
	opp2 hcltd 2300
	opp3 hcltd 1000
	max:8900
	min:1000
	count:3

Take a field of max and min in Account object  
Trigger :opportunity  
Event: after insert,after update, after delete

**Object :Opportunity**

**Event: After insert,AFTER update,After delete,After undelete**

trigger scenario11 on Opportunity (after insert,after update)

{

```

list<string> str2=new list<string>();
for(opportunity o:trigger.new)
{
    str2.add(o.accountid);
}

list<account> listacc=[select max_amount__c,min_amount__c from account
where id in :str2];
for(account ac:listacc)
{
    aggregatoresult res=[select max(amount) maximum ,min(amount) minimum
from opportunity where accountid =:ac.id];
    ac.max_amount__c=(decimal)res.get('maximum');
    ac.min_amount__c=(decimal)res.get('minimum');
}
update listacc;

}

```

---

Scenario12: Create a trigger that will display the sum of an amount of an opportunity of an account.

Scenario13: Create a trigger that will display the count opportunity of an account.

trigger scenario13 on Contact (after insert,after update)

{

list<string> str2=new list<string>();

for(contact o:trigger.new)

{

str2.add(o.accountid);

}

list<account> listacc=[select count\_contact\_\_c from account where id in :str2];

for(account ac:listacc)

{

aggregateresult res=[select count(accountid) countcon from contact where accountid =:ac.id];

ac.count\_contact\_\_c=(decimal)res.get('countcon');

}

update listacc;

}

**trigger scenario13 on Contact (after insert,after update,after delete,after undelete)**

```
{

if(trigger.isinsert||trigger.isundelete)
{
list<string> str2=new list<string>();
    for(contact o:trigger.new)
    {
        str2.add(o.accountid);
    }

    list<account> listacc=[select count_contact__c from account where id in
:str2];
    for(account ac:listacc)
    {
        aggregatoresult res=[select count(accountid) countcon from contact
where accountid =:ac.id];
        ac.count_contact__c=(decimal)res.get('countcon');

    }
    update listacc;
}

if(trigger.isdelete)
{
```

```
list<string> str2=new list<string>();
for(contact o:trigger.old)
{
    str2.add(o.accountid);
}
```

```
list<account> listacc=[select count_contact__c from account where id in
:str2];
for(account ac:listacc)
{
    aggregatorresult res=[select count(accountid) countcon from contact
where accountid =:ac.id];
    ac.count_contact__c=(decimal)res.get('countcon');

}

update listacc;
}

}
```

.....

**Scenario14:**Create a trigger to prevent deleting an account if that account is associated with contacts.

Object:Account

Event:Before Delete

.....

**Scenario15: Create a trigger to send a mail whenever a user inserts or deletes an account using apex class.**

**object:Account**

**EventT:After insert,after delete**

**Testing:**

**Call an apex class to send a mail**

```
public class emailmanager
{
    public static void sendemail(string address,string subject1,string body1)
    {
        list<messaging.SingleEmailMessage> mailist=new
list<messaging.SingleEmailMessage>();
        messaging.singleemailmessage mail=new
messaging.SingleEmailMessage();
        string[] toaddress= new string[]{address}; //setting an array of string to
address
        mail.settoaddresses(toaddress); // adding string array of toaddress to the
mail datatype of single message
        mail.setsubject(subject1);
        mail.setPlainTextBody(body1);
        mailist.add(mail); //add a subject to the list
        messaging.sendEmail(mailist);
    }
}
```

```
}
```

```
}
```

trigger scenario15 on Contact (after insert,after delete)

```
{
```

```
if(trigger.isinsert)
```

```
{
```

```
    integer count=trigger.new.size();
```

```
    for(contact c:trigger.new)
```

```
    {
```

```
        emailmanager.sendemail(c.email,'New contact added',count + 'Contact  
added sucessfully');
```

```
    }
```

```
}
```

```
if(trigger.isdelete)
```

```
{
```

```
    integer count=trigger.old.size();
```

```
    for(contact c:trigger.old)
```

```
    {
```

```
        emailmanager.sendemail(c.email,'contact deleted',count + 'Contact deleted  
sucessfully');
```

```
    }
```

```
}
```



}

**Challenge:** If a user inserts data from code through a list it will test a trigger.

**Scenario16:** Create a trigger to prevent adding more than one opportunity of an amount not more than 100000 per day.

**Object :**opportunity

**Event:**before insert

**On one day amount is not more than 100000**

**25 Opp1–25000 hcl**

**25 Opp2-45000 info**

**26 Opp3–95000 ----->wrong**

**Scenario17:** Create a trigger to prevent adding more than one opportunity of an amount not more than 100000 per account.

**Object :**opportunity

**Event:**before insert

**Opp1–25000 hcl**

**Opp2-45000 info**

**Opp3–95000 ----->hcl**

trigger scenario17 on Opportunity (before insert)

```
{

list<string> strlist= new list<string>();
for(Opportunity opp:trigger.new)
{
    strlist.add(opp.accountid);
}

list<Opportunity> opplist=[select name,id,accountid from opportunity
where CreatedDate=Today and accountid in :strlist];
for (Opportunity ops:trigger.new)
{
    list<opportunity> opp =new list<opportunity>();
    for(opportunity oo:opplist)
    {
        if(ops.accountid==oo.accountid)
        {
            opp.add(oo);
        }

        if(opp.size()>0)
        {
            ops.adderror('opportunity cannot be added related to an account
created in a one day');
        }
    }
}
```

```
}
```

```
}
```

**Scenario17:** Write a trigger to update the contact phone number whenever the user updates the account phone number field from the account object.

Object:Account

Event:After insert,After update [using new map]

trigger scenario16 on Account (after update)

```
{
```

```
map<id,account> nmap=new map<id,account>();
```

```
    nmap=trigger.newmap; //declaration of map
```

```
    list<contact> clist=[select id,phone,accountid from contact where accountid in :nmap.keySet()];
```

```
    for(contact c:clist)
```

```
    {
```

```
        account a=nmap.get(c.accountid);
```

```
        c.phone=a.phone;
```

```
    }
```

```
    update clist;
```

```
}
```

**Scenario18:**Write a trigger if a user tries to delete an account and the account is having some opportunity.[Using old map]

```
trigger scenario17 on Account (before delete)
{
    for(account a:[select id from account where id in(select accountid from
    opportunity) and id in :trigger.old])
    {
        trigger.oldmap.get(a.id).addError('cannot delete account becuase account is
        having opportunity');
    }
}
```

**Scenario18:**Write a trigger if a user tries to delete an account and the account is having some opportunity.[Using LIST]

```
trigger scenario18 on Account (before delete)
{
    list<string>x=new list<string>();
    for(account y:trigger.old)
    {
        x.add(y.id);
    }
    list<opportunity> z=[select accountid from opportunity where accountid in
    :x];
```

```

for( account a:trigger.old)

{
    for(opportunity h:z)
    {

        if(a.id==h.accountid)
        {
            a.adderror('account cannot be deleted because it is having a same
account name in a opportunity');
        }
    }

}
}

```

Scenario19:Write a trigger if a user tries to delete an account and the account is having the same contact.[Using old map]

**Scenario20** Create a Trigger that will make a checkbox true if an account has related contacts and uncheck it when there are no related contacts with this account.

**Object:contact**

**Event:After Insert**

Testing: Count field→Account object [Already created in last scenario of count contact added after insert]

trigger scenario15 on Contact (after insert,after delete, after undelete)

```
{
    if(trigger.isinsert||trigger.isundelete)
    {
        list<string> accid=new list<string>();
        for(contact c:trigger.new)
        {
            accid.add(c.accountid);
        }

        map<id,account> accmap= new map<id,account>([select id,(select id
from contacts) from account where id in :accid]);
        for(contact c:trigger.new)
        {

            if(accmap.get(c.accountid).contacts.size()>0)
            {
                accmap.get(c.accountid).has_related_contact__c=true;
            }
        }
        list<account> listacc=accmap.values();
        update listacc;
```

```
}
```

```
if(trigger.isdelete)
```

```
{
```

```
    list<string> accid=new list<string>();
```

```
    for(contact c:trigger.old)
```

```
    {
```

```
        accid.add(c.accountid);
```

```
    }
```

```
        map<id,account> accmap= new map<id,account>([select id,(select id  
from contacts) from account where id in :accid]);
```

```
        for(contact c:trigger.old)
```

```
        {
```

```
            if(accmap.get(c.accountid).contacts.size()==0)
```

```
            {
```

```
                accmap.get(c.accountid).has_related_contact__c=false;
```

```
            }
```

```
        }
```

```
        list<account> listacc=accmap.values();
```

```
        update listacc;
```

```
}
```

}