Interview Apex trigger Solved Scenarios

accenture

tcs TATA CONSULTANCY SERVICES

cognizant

salesforce

Infosys

SWIPE

1.when ever a record is inserted to the account automatically inserted to the contact

```
        trigger SCENARIO1 on Account (after insert) {
           list<contact> c=new list<contact>();
           for(account a:trigger.new)
           {
              contact b=new contact();
              b.LastName=a.Name;
              b.AccountId=a.Id;
              c.add(b);



           }
           insert c;

        }
```

===

2.when ever a record is inserted to the contact automatically inserted to the account

```
trigger scenario2 on Contact (after insert) {
if(Recursive.flag)
{
Recursive.flag=false;
list<account>a=new list<account>();
for(contact c:trigger.new)
{
account a1=new account();
a1.Phone=c.Phone;
a1.Name=c.LastName;
a.add(a1);

}
insert a;
}
```

Recursive trigger fire:
3.avoid recursive trigger

```
public class Recursive {
public static boolean flag=true;

}
```

3.when ever a create opportunity object record updated total opportunies and total amount in account object

```
trigger scenario24 on Opportunity (after insert) {
set<id>ids=new set<id>();
for(Opportunity op:trigger.new)
{
ids.add(op.accountid);
}
list<account>ac=[select Total_opportunities__c,Total_Amount__c,(select id,Amount from Opportunities ) from a
ccount where id=:ids];
for(account a:ac)
{
a.Total_opportunities__c=a.opportunities.size();
decimal sum=0;
for(opportunity p:a.opportunities)
{
sum=sum+p.amount;
}
a.Total_Amount__c=sum;


}
update ac;
}
```

==
====================

4.contact object when ever department equal to cse automatically before inserted email field

```
trigger scenario4 on Contact (before insert) {
for(contact c:trigger.new)
{
if(c.Department=='CSE')
{
c.Email='naveengorentla1@gmail.com';
}
}

}
```

5.when ever we modify inputout object doctorname automatically update droppoff object text field no relation ship

```
trigger SCENARIO32 on Inputout__c (after update) {
list<Dropoffl__c>d=[select id,name,Text__c from Dropoffl__c where Text__c='naveen'];
string name;
for(Inputout__c c:trigger.new)
{
name=c.Doctor_Name__c;
}
for(Dropoffl__c dp:d)
{
dp.Text__c=name;
}
update d;

}
```

==

6.limit reached the records

```
trigger SCENARIO6 on Account (before insert,before update) {
integer count=0;
list<account>a=[select id,name from account where createddate=today or lastmodifieddate=today];
for(account ac:trigger.new)
{
count=a.size();
ac.NumberofLocations__c=count;
if(count>2)
{
ac.adderror('reached limit today');
}
}

}
```

==

==========

7.can not inser/update/delete that user account object records

```
trigger scenario30 on Account (before insert,before update,before delete) {
user u=[select id,name from user where username='naveensfdc98@gmail.com'];
if(u.id==userinfo.getUserId())
{
if(trigger.isdelete)
{
for(account a:trigger.old)
{
a.adderror('cant delete record');

}
}
if(trigger.isupdate)
{
for(account b:trigger.new)
{
b.adderror('can not update');
}
}
if(trigger.isinsert)
{
for(account c:trigger.new)
{
c.adderror('can not insert');
}
}
}
}
```
==

8.already existing records display error message

```
trigger scenario8 on Contact (before insert) {
list<string>st=new list<string>();

for(contact c:trigger.new)
{
list<contact>a=[select id,name,Email,lastname from contact where Email=:c.Email];
if(a.size()>0)
{
c.Email.adderror('already existing');
}
}

}
```

for loop without query
==========================

```
trigger duplicatetrigger on Inputout__c (before insert) {
set<string>s=new set<string>();
for(Inputout__c op:trigger.new)
{
s.add(op.Doctor_Name__c);
}
list<Inputout__c>d=[select id,Doctor_Name__c from Inputout__c where Doctor_Name__c=:s];
set<string>dupids=new set<string>();
for(Inputout__c don:d)
{
dupids.add(don.Doctor_Name__c);
}
for(Inputout__c c:trigger.new)
{
if(c.Doctor_Name__c!=null)
{
if(dupids.contains(c.Doctor_Name__c))
{
c.Doctor_Name__c.adderror('already existing record');
}
}
}
}
```

==

===
=====

9. count of related contacts and accounts field display size

```
public class rollupsummery {

public static void increment(list<contact>con)
{
set<id>ids=new set<id>();
for(contact c:con)
{
ids.add(c.accountid);
}
list<account>a=[select id,name,NumberOfEmployees,(select id,lastname from contacts) from account where id
=:ids];
for(account ac:a)
{
ac.NumberOfEmployees=ac.contacts.size();

}
update a;

}

}
```

trigger:
========

```
trigger scenario11 on Contact (after insert) {
rollupsummery.increment(trigger.new);

}
```

===
====

10. when ever opportunity stagename =closedwon automatically update the account field rating=hot

```
trigger scenario12 on Opportunity (after insert,after update) {
set<id>ids=new set<id>();
list<account>ac=new list<account>();
for(opportunity op:trigger.new)
{
ids.add(op.AccountId);

ac=[select id,name,rating from account where id=:ids];
if(op.StageName=='Closed won')
{

for(account a:ac)
{
a.Rating='hot';
}
update ac;
}
}


}
```

==

11.when ever account name is naveen automatically update the contact all lastnames

```
trigger scenario13 on Account (after update) {

string names;

list<contact>c=[select id,lastname,firstname from contact where lastname=:names ];
for(account a:trigger.new)
{
names=a.name;
}
for(contact con:c)
{
con.lastname=names;
}
update c;

}
```

==

==
====

12.when ever a opportunity created record amount field is calculated by account total field

```
trigger scenario21 on Opportunity (after insert,after update,after delete) {
set<id>ids=new set<id>();
map<id,opportunity>opp=new map<id,opportunity>();
Decimal oldVal;
Decimal newVal;
if(trigger.isinsert)
{
for(opportunity op:trigger.new)

{
ids.add(op.AccountId);
opp.put(op.AccountId, op);
}
list<account> acc=[select id,Total_Amount__c from account where id=:ids];
for(account a:acc)
{
if(a.Total_Amount__c==null )
{
a.Total_Amount__c=opp.get(a.Id).amount;
}
else
{

a.Total_Amount__c= a.Total_Amount__c+opp.get(a.Id).amount;
}
}
update acc;
}
if(trigger.isUpdate)
{
for(opportunity op:trigger.new)

{
ids.add(op.AccountId);
opp.put(op.AccountId, op);
newVal=op.Amount;
}
for(Opportunity ops:trigger.old){
oldVal=ops.Amount;
}
list<account> acc=[select id,Total_Amount__c from account where id=:ids];
for(account a:acc)
{
if(a.Total_Amount__c==null )
{
a.Total_Amount__c=opp.get(a.Id).amount;
}
else
{

a.Total_Amount__c= a.Total_Amount__c+opp.get(a.Id).amount-oldVal;
}
}
update acc;
}
}
}
```

==

13.when ever a create a lead object automatically converted account ,contact,opportunity

```
trigger scenario19 on Lead (after insert) {
list<account>acc=new list<account>();
list<contact>con=new list<contact>();
list<opportunity>op=new list<opportunity>();
for(lead l:trigger.new)
{
account a=new account();
a.Name=l.lastname;
a.Phone=l.Phone;
acc.add(a);
contact c=new contact();
c.LastName=l.Name;

con.add(c);
opportunity o=new opportunity();
o.Amount=l.AnnualRevenue;
o.CloseDate=system.today();
o.StageName='closed won';
op.add(o);
}
insert acc;
insert con;
insert op;

}
```

==

14.when ever create a contact automatically update opprtunity fields

```
trigger scenario17 on Contact (after insert) {
list<opportunity>op=[select id,name,stagename,Description,amount from opportunity limit 50];
for(contact c:trigger.new){
for(opportunity o:op)
{
if(o.amount<5000||o.Amount==null)
{
o.amount=5000;
o.Name=o.Name+'Mr';
o.StageName='prospecting';

}
else{
o.Amount=o.Amount+1000;
o.Name=o.Name+'Dr';
}
update o;
}
}

}
```

==

15. Whenever new account is created, its rating should be hot.

```
        trigger AccTrig on Account (before insert) {


           for (Account a:trigger.new){

              a.rating = 'Hot';
           }

        }
```

==

16. Make a Fax field mandatory in Account using trigger.

```
trigger AccTrig on Account (before insert) {

    for (Account a:trigger.new){
        if (a.fax == Null){
            a.fax.addError('Fax field is mandatory');
        }
    }
}
```

17. Whenever a new contact is created, prefix the firstname with Mr.

```
trigger Contrig on Contact (before insert) {

    for (contact c:Trigger.new){

        c.salutation = 'Mr';
    }

}
```

```
}
```

==

18. Prevent the insertion of an account, if you already have an account with the same name
    (Prevent the insertion of a duplicate account).

```
trigger AccountDuplicate on Account (before insert)
{

 Set<String> setName = new Set<String>();
 For(Account acc : trigger.new)
 {
  setName.add(acc.name);
 }

 if(setName.size() > 0 )
 {
  List<Account> lstAccount = [select name ,id from account where name in :setName ];

  Map<String ,Account> mapNameWiseAccount = new Map<String,Account>();
  For(Account acc: lstAccount)
  {
   mapNameWiseAccount.put(acc.name ,acc);
  }

  For(Account acc : trigger.new)
  {
   if(mapNameWiseAccount.containsKey(acc.name))
   {
    acc.Name.addError('Name already Exist ');
   }
  }
 }
}
```

==

19. Whenever a new contact is inserted in the system, update the accounts phone field with the contacts phone field.

TRIGGER :

```
        trigger contactTrigger on Contact (before insert, before update, before delete, after insert, after update, after d
elete, after undelete) {

        if(trigger.isBefore ){
            system.debug('I am before trigger ');
        }

        else if(trigger.isAfter){
            system.debug('I am after trigger ');
            if(trigger.isUpdate){
                contactTriggerHandler.afterUpdateHelper(trigger.new);
            }
        }
```

HANDLER CLASS :

```
        public class contactTriggerHandler {

         public static void afterUpdateHelper(List<Contact> conList){

             Set<Id> setId = new Set<Id>();
            for(Contact con:conList){
                setId.add(con.AccountId);
            }
            system.debug('setId ' + setId);

            List<Account> accItemList = [Select Id, Name, Phone, (Select Id, FirstName, LastName, Phone, Accoun
tId From Contacts) From Account Where Id IN:setId];
            for(Account a1:accItemList){
                for(Contact c1:a1.Contacts){
                    if(c1.Phone !=null){
                        a1.Phone = c1.Phone;
                        update accItemList;
                    }
                }
            }
         }
       }
```

==

,

==

20. Create a contact whenever a new account is created.

```
trigger AccTrig2 on Account (after insert) {

    set<id> Accid = new set<id>();
    list<Contact> ContList = new List<Contact>();

    for (Account a:Trigger.new){
        Accid.add(a.id);

        contact c = new contact();

        c.AccountId = a.id;
        c.lastName  = a.name;
        c.phone = a.phone;
        contList.add(c);
    }
    insert contList;
}
```

==

21. Prevent to delete account if they have related opportunity

```
trigger DeleteAccountOpportunity1 on Account (before delete)
{
    List<Opportunity> opp = [Select accountID from opportunity where accountid in :TRIGGER.OLD];
    for(Account a : Trigger.OLD)
    {
        for(Opportunity o : Opp)
        {
            if(a.id == o.accountId)
            {
                a.addError('Account have Opportunity ,so you can not delete this Account');
            }
        }
    }
}
```
==

---

23. Trigger will prevent the user to delete records from account

```
        trigger test6 on Account (before delete) {

           for (Account A:Trigger.old){

              A.addError(' You can not delete the record, Please contact your Administartator');
           }

        }
```

===

24. Prevent user from creating duplicate accounts with same name

```
trigger AccountDuplicate on Account (before insert)
{

 Set<String> setName = new Set<String>();
 For(Account acc : trigger.new)
 {
  setName.add(acc.name);
 }

 if(setName.size() > 0 )
 {
  List<Account> lstAccount = [select name ,id from account where name in :setName ];

  Map<String ,Account> mapNameWiseAccount = new Map<String,Account>();

  For(Account acc: lstAccount)
  {
   mapNameWiseAccount.put(acc.name ,acc);
  }

  For(Account acc : trigger.new)
  {
   if(mapNameWiseAccount.containsKey(acc.name))
   {
    acc.Name.addError('Name already Exist ');
   }
  }

 }
}
```

===

==

25. Add a prefix dr to all leads names whenever a record is updated or inserted

```
trigger Leadtrig on Lead (before insert, before update) {

   for (Lead l:Trigger.new){

      l.Firstname  = 'Dr. ' + l.FirstName;
   }
}
```

==

26. Prevent user to delete contact which is associated with any account.

```
trigger TrigContact on Contact (before delete) {

   for (contact c:Trigger.old){
      if(c.AccountId != Null){
         c.addError('Can not delete contact');
      }
   }
}
```

==

27. Throw an error if phone number is not 10 dight of length

```
trigger AccTrig23 on Account (before insert, BEFORE UPDATE) {

   for (Account a:Trigger.new){
      if (a.phone.length() != 10){
         a.phone.addError('Phone length is not 10');
      }
   }
}
```

==

28. Whenever contact is created update status field from it's associate account

==

29. Whenever new record is created in account object, before this record inserted in account object Account, delete all contact records avilable with the same name

==

30. when ever a record is inserted to the Account automatically inserted to the Contact

```
trigger SCENARIO1 on Account (after insert) {
list<contact> c=new list<contact>();
for(account a:trigger.new)
{
contact b = new contact();
b.LastName = a.Name;
b.AccountId = a.Id;
c.add(b);
}
insert c;

}
```

==

31. when ever a create opportunity object record updated total opportunies and total amount in account object

```
    trigger scenario24 on Opportunity (after insert) {
    set<id> ids = new set<id>();
    for(Opportunity op:trigger.new)
    {
    ids.add(op.accountid);
    }
    list<account>ac=[select Total_opportunities__c,Total_Amount__c,(select id,Amount from Opportunities ) fro
m a
    ccount where id=:ids];
    for(account a:ac)
    {
    a.Total_opportunities__c=a.opportunities.size();
    decimal sum=0;
    for(opportunity p:a.opportunities)
    {
    sum=sum+p.amount;
    }
    a.Total_Amount__c=sum;


    }
    update ac;
    }
```

==

32. can not inser/update/delete that user account object records

```
trigger AccountDuplicate on Account (before insert, before update, before delete){

    for (Account a:Trigger.new){
        a.addError('Need to contact Amin for access');
    }
}
```

==

33. whenever opportunity StageName = Closed Won automatically update the account field Rating = Hot

```
trigger updateAccountRating on opportunity (after insert, after update){
 list<Id> accIds = new list<Id>();
 list<Account> accounts = new list<account>();
 for(opportunity o:trigger.new){
  accIds.add(o.accountId);
 }
 for(account a:[select Id, Rating from account where Id IN :accIds]){
  for(opportunity opp:trigger.new){
   if(opp.stage=='closed won'){
    a.Rating='hot';
    accounts.add(a);
   }
  }
 }
   update accounts;
}
```

==

34. name field is updated on account then put name field as name + Phone

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

    for (Account a:Trigger.new){
        if (a.name != Trigger.oldMap.get(a.Id).name){
            a.name = a.name + trigger.oldMap.get(a.id).phone;
        }
    }
}
```

==

==

35.Whenever account is created with annual revenue more than 50000, then add Pranay mehare as contact Name :: DONE (Check 50 k and insert contact)

```
trigger test5 on Account (before insert) {

   list<Contact> ContList = new List<Contact>();

   // set<Id> AccIdsSet = new Set<Id>();

   for (Account Acc:Trigger.new){

      if (Acc.AnnualRevenue > 50000){
         Contact cont = new contact();

         Cont.AccountId = Acc.Id;
         Cont.Firstname = 'Pranay';
         Cont.Lastname = 'Mehare';
         ContList.add(Cont);
      }
   }
   Insert ContList;

}
```

==

36. Whenever Lead is created with lead Source as WEB then give rating as COLD otherwise HOT

```
trigger test56 on Lead (before insert) {

   for (Lead l:Trigger.New){

      if (l.leadSource == 'Web'){
         l.Rating = 'Cold';
      }
      Else{
         l.Rating = 'Hot';
      }
   }
}
```

==

37. Prevent Account from deleting if it has 2 or more contacts..

```
trigger Account1 on Account (before Delete) {

   set<Id> AccIds = new set<Id>();
   Map<Id, Account> AccMap = new Map<Id, Account>();     // need to store map

   for (Account A:Trigger.old){

      AccIds.add(A.Id);
   }
   for (Account A:[SELECT Id, Name, (SELECT Id, Firstname, LastName FROM Contacts) FROM ACCOU
NT WHERE Id IN :AccIds]){
      AccMap.put(A.Id, A);
   }
   for (Account A1:Trigger.Old){

      if (AccMap.get(A1.id).contacts.Size() >= 2){
         A1.addError('You Can not delete Account');
      }
   }

}
```

==

38. While creating or updating Lead check whether Email is already there in existing contact , if it is there then thro
w an error.

//Unrelated Objects ::

```
trigger Lead1 on Lead (before insert, before update) {
   map<string, contact> contMap = new map<string, contact>();
   list<contact> contEmail = [SELECT Id, Email FROM Contact];

   for (contact c:contEmail){

      contMap.put(c.Email, c);
   }
   for (lead l:Trigger.new){

      if ((l.email != null) && (trigger.isInsert || (l.email != Trigger.oldMap.get(l.Id).Email))){
         if (contMap.containsKey(l.email)){
            l.Email.addError('Email already exist');
         }
      }
   }
}
```

==

==

39. Whenever phone is modified on account object then update contact record with phone field (other phone with ol d value and home phone with new value)
associated with account record.

```
 trigger updatePhone on Account (after update) {

   Set<Id> accIdSet = new Set<Id>();
   for(Account acc : Trigger.New){
      if(acc.Phone!=trigger.oldmap.get(acc.Id).Phone){
          accIdSet.add(acc.Id);
      }
   }
   list<contact> conlist = new list<contact>();
   for(Contact con : [select Id,AccountId from Contact Where AccountId In : accIdSet]){

      con.HomePhone=trigger.newmap.get(con.AccountId).Phone;
      con.OtherPhone = trigger.oldmap.get(con.AccountId).Phone;
      conlist.add(con);
   }

   update conlist;

 }
```

==

40. Counts number of contacts on Account using trigger (Create Lookup Summary field using Trigger)

Note : Object CONTACT [Very importatnt]

TRIGGER :

```
trigger ContCount on Contact (After Delete, After Insert, After Update, After Undelete) {

If (Trigger.isInsert || Trigger.isDelete || Trigger.isUpdate || Trigger.isUndelete){

CountContact.ContContacts (Trigger.New, Trigger.Old);
}
}
```

HANDLER CLASS :

```
    Public class CountContact {
      Public static void ContContacts (list<Contact> NewContact, List<Contact> oldContact){

        set<id> ContAccIds = new set<Id>();

        try{
          if (NewContact != Null){
            for (Contact con:newContact){
              if(Con.AccountId != Null){
                ContAccIds.add(con.AccountId);
              }
            }
          }

          if (oldContact != Null){
            for (Contact con:oldContact){
              if(Con.AccountId != Null){
                ContAccIds.add(con.AccountId);
              }
            }
          }

          List<Account> AccList = [SELECT Id, ContCount__c, (SELECT Id FROM Contacts) FROM Accoun
t Where Id IN :ContAccIds];

          if (AccList != Null){
            for (Account AccValue:AccList){
              accvalue.ContCount__c = accValue.Contacts.Size();
            }
          }

          if(!AccList.isEmpty()){
            update AccList;
          }
        }

        Catch (Exception e){
          System.debug ('Get Message' + e.getMessage());
        }
      }
    }
```

==

===

41. Recursion in Trigger

TRIGGER :

```
Trigger TrigCont on Contact (Before insert){

   if (trigger.isinsert && Trigger.isBefore  && CreateCont.isFirst ){                    //TRIGGER WILL RUN A
FTER ISFRIST CHECK

      CreateCont = false;                                                                //TO AVOID RECURSION

      createCont.CreateConts(Trigger.New);

   }
}
```

HANDLER :

```
Public class CreateCont{

   public static Boolean isFirst = true;                                                 //TO AVOID RECURSION

   public static void (List<Contact> conList) {

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

      for (Contact c: conList){

         contact c = new Contact;
         c.firstName  = 'Recursive';
         c.LastName = 'Trigger';

         cList.add(c);

      }

      if (!cList.isEmpty()){
         insert cList
            }
   }

}
```

===******************==========