Apex Trigger
Solved Scenarios

(Improve Your Salesforce Development Skills)

Learn Salesforce Live & Free!

Join Ongoing Admin & Dev Bootcamp on Youtube!

1000+ folks are learning Salesforce together!

(January - June 2023)

Sanjay Gupta Tech School

Upscale Your Knowledge

https://studysalesforce.com

Sanjay Gupta Tech School

Join Free & Live Salesforce Admin & Dev Bootcamp on Youtube

Trigger Use Cases with Solutions

1. Upon Account Creation if Industry is not null and having value as ‘Media’ then populate Rating as Hot.

   Solution:

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

    if(Trigger.isInsert){
        if(Trigger.isBefore){
            AccountTriggerHandler.updateRating(Trigger.New);
        }
    }
}

public class AccountTriggerHandler {

    public static void updateRating(List<Account> accList){
        for(Account acc:accList){
            if(acc.Industry!=null && acc.Industry=='Media'){
                acc.Rating='Hot';
            }
        }
    }
}
```

2. Upon Opportunity Creation if Amount is not null and is greater than 100000 then populate ‘Hot Opportunity’ in description field.

   Solution:

```
trigger OpportunityTrigger on Opportunity (before insert) {

    if(Trigger.isInsert){
        if(Trigger.isBefore){
            OpportunityTriggerHandler.updateDesc(Trigger.New);
        }
    }
}

public class OpportunityTriggerHandler {

    public static void updateDesc(List<Opportunity> oppList){
        for(Opportunity opp:oppList){
            if(opp.Amount!=null && opp.Amount>100000){
                opp.Description='Hot Opportunity';
            }
        }
    }
}
```

3. When an account inserts and CopyBillingToShipping (Custom Field) checkbox is checked then automatically copy account billing address into account shipping address.

Solution:

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

    if(Trigger.isInsert){
        if(Trigger.isBefore){
            AccountTriggerHandler.updateRating(Trigger.New);
        }
    }
}

public class AccountTriggerHandler {

    public static void updateAddres(List<Account> accList){
        for(Account acc:accList){
            if(acc.CopyBillingToShipping__c && acc.acc.BillingCity!= null
            && acc.BillingCountry!= null && acc.BillingPostalCode != null
            && acc.BillingState != null &&  acc.BillingStreet != null){
                acc.ShippingCity=acc.BillingCity;
                acc.ShippingCountry=acc.BillingCountry;
                acc.ShippingPostalCode=acc.BillingPostalCode;
                acc.ShippingState=acc.BillingState;
                acc.ShippingStreet=acc.BillingStreet;
            }
        }
    }
}
```

4. Upon Creation of Position (Custom Object) if it is a New Position and Open Date, Min Pay & Max Pay are not populated then populated them with below values:
   a. Open Date = Today's Date
   b. Min Pay = 10000
   c. Max Pay = 15000

Solution:

```
trigger PositionTrigger on Position__c (before insert) {

    if(Trigger.isInsert){
        if(Trigger.isBefore){
            PositionTriggerHandler.populateDateAndPay(Trigger.New);
        }
    }
}
public class PositionTriggerHandler {

    public static void populateDateAndPay(List<Position__c> posList){
        for(Position__c pos:posList){
            if(pos.status__c=='New Position' && pos.Min_Pay__c ==null
               && pos.Max_Pay__c==null &&
               pos.Open_Date__c==null){
                pos.Open_Date__c=System.today();
                pos.Min_Pay__c=10000;
                pos.Max_Pay__c=15000;
            }
        }
    }
}
```

5. Create a related Contact when an Account is created.

Solution:

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

    if(Trigger.isInsert){
        if(Trigger.isAfter){
            AccountTriggerHandler.createContact(Trigger.New);
        }
    }
}

public class AccountTriggerHandler {

    public static void createContact(List<Account> accList){
        List<Contact> conList = new List<Contact>();
        for(Account acc: accList){
            Contact con= new Contact();
            con.FirstName = acc.Name + 'FN';
            con.LastName = acc.Name + 'LN';
            con.AccountId = acc.Id;
            conList.add(con);
        }
        if(!conList.isEmpty()){
            insert conList;
        }
    }
}
```

6. Create a related Opportunity when an Account is created.

Solution:

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

    if(Trigger.isInsert){
        if(Trigger.isAfter){
            AccountTriggerHandler.createRelatedOpp(Trigger.New);
        }
    }
}

public class AccountTriggerHandler {

    public static void createRelatedOpp(List<Account> accList){
        List<Opportunity> oppList = new List<Opportunity>();
            for(Account acc: accList){
                Opportunity opp = new Opportunity();
                opp.Name = acc.Name + 'opp';
                opp.AccountId = acc.Id;
                opp.StageName = 'Prospecting';
                opp.CloseDate = System.today();
                oppList.add(opp);
            }
            if(!oppList.isEmpty()){
                insert oppList;
            }
        }
    }
}
```

7. When a Case is created on any Account, put the latest case number on the Account in the 'Latest Case Number' field.

Solution:

```
trigger CaseTrigger on Case (after insert) {

    if(Trigger.isInsert){
        if(Trigger.isAfter){
            CaseTriggerHandler.populateLatestCaseNum(Trigger.New);
        }
    }
}

public class CaseTriggerHandler {

    public static void populateLatestCaseNum(List<Case> caseList){
    List<Account> accLIst = new List<Account>();
        for(Case cs : caseList){
            if(cs.AccountId != null){
                Account acc = new Account();
                acc.id = cs.AccountId;
                acc.Latest_Case_Number__c = cs.CaseNumber;
                accList.add(acc);
            }
        }
        if(!accList.isEmpty()){
            update accList;
        }
    }
}
```

8. Account records should have a field named ‘Recent Opportunity Amount’. It should contain the opportunity amount of the latest created opportunity on account.

Solution:

```
trigger OpportunityTrigger on Opportunity (after insert) {

	if(Trigger.isInsert){
		if(Trigger.isAfter){
			OpportunityTriggerHandler.populateAmount(Trigger.New);
		}
	}
}

public class OpportunityTriggerHandler {

	public static void populateAmount(List<Opportunity> oppList){
		List<Account> accList= new List<Account>();
		for(Opportunity opp:oppList){
			if(opp.Amount!=null && opp.AccountId!=null){
				Account acc = new Account();
				acc.Id=opp.AccountId;
				acc.Recent_Opp_Amount__c=opp.Amount;
				accList.add(acc);
			}
		}
		if(!accList.isEmpty()){
			update accList;
		}
	}
}
```

9. On Account create two checkbox fields labeled as Contact and Opportunity. Now when a new Account record is created and if a particular Contact or Opportunity checkbox is checked then create that related record. Also Opportunity record should be created only if the Account record Active picklist is populated with a Yes.

Solution:

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

    if(Trigger.isInsert){
        if(Trigger.isAfter){
            AccountTriggerHandler.createContactOrOpp(Trigger.New);
        }
    }
}

public class AccountTriggerHandler {

    public static void createContactOrOpp(List<Account> accList){
        List<Contact> conList= new List<Contact>();
        List<Opportunity> oppList= new List<Opportunity>();
        for(Account acc:accList){
            if(acc.Contact__c){
                Contact con = new Contact();
                con.FirstName = 'con1';
                con.LastName = 'last';
                con.AccountId = acc.Id;
                conList.add(con);
            }
            if(acc.Opportunity__c && acc.Active__c == 'Yes'){
```

```
                        Opportunity opp = new Opportunity();
                        opp.AccountId = acc.Id;
                        opp.StageName = 'Prospecting';
                        opp.CloseDate = System.today();
                        opp.Name = 'Opp1';
                        oppList.add(opp);
                    }
                }

                if(oppList.size()>0){
                    insert oppList;
                }
                if(conList.size()>0){
                    insert conList;
                }

        }
}
```

10. If the Account phone is updated then populate below message in description.

Description = Phone is Updated! Old Value : XXX & New Value : XXX

Solution:

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

    if(Trigger.isUpdate){
        if(Trigger.isBefore){
            AccountTriggerHandler.updateDescription(Trigger.New,
            Trigger.oldMap);
        }
    }
}

public class AccountTriggerHandler {

    public static void updateDescription(List<Account>
        accList,Map<Id,Account> oldMap){
        for(Account acc:accList){
            if(acc.Phone!=oldMap.get(acc.Id).Phone){
                acc.Description='Phone is updated! Old Value :
                    '+oldMap.get(acc.Id).Phone+' & New Value :
                    '+acc.Phone;
            }
        }
    }
}
```

11. When an account is inserted or updated and the CopyBillingToShipping checkbox is checked then automatically copy the account billing address into account shipping address.

Solution:

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

    if(Trigger.isUpdate){
        if(Trigger.isBefore){
            AccountTriggerHandler.copyBillToShip(Trigger.New,
            Trigger.oldMap);
        }
    }
}
public class AccountTriggerHandler {
    public static void copyBillToShip(List<Account> accList,Map<Id,Account>
        oldMap){
        for(Account acc:accList){
            if((oldMap==null && acc.CopyBillingToShipping__c) ||
                (!oldMap.get(acc.Id).CopyBillingToShipping__c &&
                acc.CopyBillingToShipping__c)){
                acc.ShippingCity=acc.BillingCity;
                acc.ShippingCountry=acc.BillingCountry;
                acc.ShippingPostalCode=acc.BillingPostalCode;
                acc.ShippingState=acc.BillingState;
                acc.ShippingStreet=acc.BillingStreet;
            }
        }
    }
}
```

12. Upon Account Creation or updation if Industry is not null and having value as ‘Media’ then populate Rating as Hot.

Solution:

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

    if(Trigger.isUpdate){
        if(Trigger.isBefore){
            AccountTriggerHandler.updateIndustryRating(Trigger.New,
            Trigger.oldMap);
        }
    }
}
public class AccountTriggerHandler {

    public static void updateIndustryRating(List<Account>
    accList,Map<Id,Account> oldMap){
        for(Account acc:accList){
            if((oldMap==null &&acc.Industry=='Media') ||
            (acc.Industry=='Media' && acc.Industry !=
            oldMap.get(acc.Id).Industry)){
                acc.Rating='Hot';
            }
        }
    }
}
```

13. If opportunity Stage is updated upon its creation or update then update description as either 'Opp is Closed Lost' or 'Opp is Closed Won' or 'Opp is Open'.

Solution:

```
trigger OpportunityTrigger on Opportunity (before update) {
    if(Trigger.isUpdate){
        if(Trigger.isBefore){
            OpportunityTriggerHandler.updateDesc(Trigger.New,
            Trigger.oldMap);
        }
    }
}
public class OpportunityTriggerHandler {
    public static void updateDesc(List<Opportunity>
    oppList,Map<Id,Opportunity> oldMap){
        for(Opportunity opp:oppList){
            if((oldMap==null)||(opp.StageName!=oldMap.get(opp.Id).Stage
            Name)){
                if(opp.StageName=='Closed Won'){
                    opp.Description='Opportunity is Closed won';
                }else if(opp.StageName=='Closed Lost'){
                    opp.Description='Opportunity is closed lost';
                }else{
                    opp.Description='Opportunity is open';
                }
            }
        }
    }
}
```

14. If the Account phone is updated then populate the phone number on all related Contacts (Home Phone field). **[Using Map]**

Solution:

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

    if(Trigger.isUpdate){
        if(Trigger.isAfter){
            AccountTriggerHandler.updateRelatedConts(Trigger.New,
            Trigger.oldMap);
        }
    }
}
public class AccountTriggerHandler {
    public static void updateRelatedConts(List<Account>
    accList,Map<Id,Account> oldMap){
        List<Contact> conList=new List<Contact>();
        Map<Id,Account> accToAccountMap= new Map<Id,Account>();
        for(Account acc:accList){
            if((acc.Phone!=null && acc.Phone!=(oldMap.get(acc.Id).Phone))
            && oldMap!=null){
                accToAccountMap.put(acc.Id,acc);
            }
        }

        for(Contact cont:[SELECT Id, HomePhone, AccountId FROM Contact
        WHERE AccountId IN: accTOAccountMap.keySet()]){
            if(accToAccountMap.containsKey(cont.AccountId)){
                cont.HomePhone=accToAccountMap.get(cont.AccountId).
                Phone;
```

```
                        conList.add(cont);
                    }
                }
                if(!conList.isEmpty()){
                    update conList;
                }
                OR
                if(conList.size() > 0){
                    update conList;
                }
        }
}
```

15. If the Account phone is updated then populate the phone number on all related Contacts (Home Phone field). **[Using Parent-Child SOQL]**

Solution:

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

    if(Trigger.isUpdate){
        if(Trigger.isAfter){
            AccountTriggerHandler.updateRelatedConts(Trigger.New,
            Trigger.oldMap);
        }
    }
}
public class AccountTriggerHandler {

    public static void updateRelatedContsWithoutMap(List<Account>
    accList,Map<Id,Account> oldMap){
        List<Contact> conList=new List<Contact>();
        Set<Id> accIds = new Set<Id>();
        for(Account acc:accList){
            if((acc.Phone!=null && acc.Phone!=(oldMap.get(acc.Id).Phone))
            && oldMap!=null){
                accIds.add(acc.Id);
            }
        }

        for(Account acc : [SELECT Id, Phone, (SELECT HomePhone FROM
        Contacts) FROM Account WHERE Id IN: accIds]){
            if(acc.Contacts != null){
                for(Contact con : acc.Contacts){
                    con.HomePhone=acc.Phone;
```

```
                                conList.add(con);
                            }
                        }
                    }
                    if(!conList.isEmpty()){
                        update conList;
                    }
                }
            }
```

16. If the Account billing address is updated then update related contacts mailing address. **[Using Map]**

Solution:

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

    if(Trigger.isUpdate){
        if(Trigger.isAfter){
            AccountTriggerHandler.updateRelatedConts(Trigger.New,
            Trigger.oldMap);
        }
    }
}
public class AccountTriggerHandler {
    public static void updateRelatedContactMail(List<Account>
    accList,Map<Id,Account> oldMap){
        List<Contact> conList=new List<Contact>();
        Map<Id,Account> accToAccountMap= new Map<Id,Account>();

        for(Account acc:accList){
            if( (!acc.BillingCity.equals(oldMap.get(acc.Id).BillingCity) ||
            !acc.BillingCountry.equals(oldMap.get(acc.Id).BillingCountry) ||
            !acc.BillingPostalCode.equals(oldMap.get(acc.Id).BillingPostalC
            ode) ||c!acc.BillingState.equals(oldMap.get(acc.Id).BillingState)
            || !acc.BillingStreet.equals(oldMap.get(acc.Id).BillingStreet)  )
            && oldMap!=null){
                accToAccountMap.put(acc.Id,acc);
            }
        }
```

```
            for(Contact con:[SELECT Id,AccountId  FROM Contact WHERE
            AccountId IN: accTOAccountMap.keySet()]){

            if(accToAccountMap.containsKey(cont.AccountId)){
                  con.MailingCountry =
                  accToAccountMap.get(cont.AccountId).BillingCountry;
                  con.MailingCity =
                  accToAccountMap.get(cont.AccountId).BillingCity;
                  con.MailingState =
                  accToAccountMap.get(cont.AccountId).BillingState;
                  con.MailingPostalCode =
                  accToAccountMap.get(cont.AccountId).BillingPostalCode;
                  con.MailingStreet =
                  accToAccountMap.get(cont.AccountId).BillingStreet;
                  conList.add(con);
            }
      }
      if(!conList.isEmpty()){
            update conList;
      }
   }
}
```

17. If the Account billing address is updated then update related contacts mailing address. **[Using Parent-Child SOQL]**

Solution:

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

    if(Trigger.isUpdate){
        if(Trigger.isAfter){
            AccountTriggerHandler.updateRelatedConts(Trigger.New,
            Trigger.oldMap);
        }
    }
}
public class AccountTriggerHandler {

    public static void updateRelatedContactMailWithoutMap(List<Account>
    accList,Map<Id,Account> oldMap){
        List<Contact> conList=new List<Contact>();
        Set<Id> idSet= new Set<Id>();
        for(Account acc:accList){
            if((!acc.BillingCity.equals(oldMap.get(acc.Id).BillingCity) ||
            !acc.BillingCountry.equals(oldMap.get(acc.Id).BillingCountry) ||
        !acc.BillingPostalCode.equals(oldMap.get(acc.Id).BillingPostalCode)
|| !acc.BillingState.equals(oldMap.get(acc.Id).BillingState) ||
!acc.BillingStreet.equals(oldMap.get(acc.Id).BillingStreet)  ) && oldMap!=null){
                idSet.add(acc.Id);
            }
        }

        for(Account acc:[SELECT Id, BillingCountry, BillingCity, BillingState,
    BillingPostalCode, BillingStreet, (SELECT Id FROM Contacts)  FROM
    Account WHERE Id IN:idSet]) {
```

```
                    if(acc.Contacts!=null){
                        for(Contact cont:acc.Contacts){
                            cont.MailingCountry=acc.BillingCountry;
                            cont.MailingCity= acc.BillingCity;
                            cont.MailingState=acc.BillingState;
                            cont.MailingPostalCode=acc.BillingPostalCode;
                            cont.MailingStreet=acc.BillingStreet;
                            conList.add(cont);
                        }
                    }
                }
                if(!conList.isEmpty()){
                    update conList;
                }
        }
}
```

18. When a Opportunity Stage (field) is changed, create a Task record on Opportunity and assign it to Logged In User/Opportunity Owner / Any User.

Solution:

```
trigger OpportunityTrigger on Opportunity (after update) {
    if(Trigger.isUpdate){
        if(Trigger.isAfter){
            OpportunityTriggerHandler.createTask(Trigger.New,
            Trigger.oldMap);
        }
    }
}
public class OpportunityTriggerHandler {
    public static void createTask(List<Opportunity>
    oppList,Map<Id,Opportunity> oldMap){
        List<Task> tList= new List<Task>();
        for(Opportunity opp:oppList){
            if(opp.StageName!=oldMap.get(opp.Id).StageName){
                Task t = new Task();
                t.WhatId = opp.Id;
                t.Subject = 'Email';
                t.Priority = 'Normal';
                T.status = 'Not Started';
                t.OwnerId = UserInfo.getUserId();
                tList.add(t);
            }
        }
        If(tList.size( ) > 0){
            insert tList;
        }
    }}
```

19. Write a trigger on Account when Account Active field is updated from ‘Yes’ to ‘No’ then check all opportunities associated with the account. Update all Opportunities Stage to close lost if stage not equal to close won.

Solution:

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

    if(Trigger.isUpdate){
        if(Trigger.isAfter){
            AccountTriggerHandler.updateOpportunityStage(Trigger.New,
            Trigger.oldMap);
        }
    }
}
public class AccountTriggerHandler {

    public static void updateOpportunityStage(List<Account>
     accList,Map<Id,Account> oldMap){
        List<Opportunity> oppList=new List<Opportunity>();
        Set<Id> idSet= new Set<Id>();
        for(Account acc:accList){
            if(acc.Active__c == 'No' && acc.Active__c !=
             oldMap.get(acc.Id).Active__c){
                idSet.add(acc.Id);
            }
         }
        for(Account a:[SELECT Id,Active__c,(SELECT Id,StageName FROM
         Opportunities) FROM Account WHERE Id IN:idSet]){
            if(acc.Opportunities!=null){
                for(Opportunity opp:a.Opportunities){
```

```
                                    if(opp.StageName!='Closed
                                     Won'&&opp.StageName!='Closed Lost'){
                                            opp.StageName='Closed Lost';
                                            oppList.add(opp);
                                    }
                            }
                    }
            }
            if(oppList.size( ) > 0){
                    update oppList;
            }
    }
}
```

20. Account records cannot be deleted if active is Yes.

Solution:

```
trigger AccountTrigger on Account (before delete) {
	if(Trigger.isDelete){
		if(Trigger.isBefore){
			AccountTriggerHandle.preventDel(Trigger.old);
		}
	}
}

public class AccountTriggerHandle{
	public static void preventDel(List<Account> accList){
		for(Account acc : accList){
			if(acc.Active__c == 'Yes'){
				acc.addError(Label.Prevent_Account_Deletion);
			}
		}
	}
}
```

21. Prevent account record from being edited if the record is created 7 days back.

Solution:

```
trigger AccountTrigger on Account (before update) {
	if(Trigger.isUpdate){
		if(Trigger.isBefore){
			AccountTriggerHandle.preventAccEdit(Trigger.new);
		}
	}
}

public class AccountTriggerHandle{
	public static void  preventAccEdit(List<Account> accList){
		for(Account acc:accList){
			if(acc.CreatedDate<System.today()-6){
				acc.addError('You cannot update account created 7 days
				back');
			}
		}
	}
}
```

22. Apply validation using addError( ) method in trigger. While Creation of Opportunity is Amount is null then throw an error message.

Solution:

```
trigger OpportunityTrigger on Opportunity (before insert) {
    if(Trigger.isInsert){
        if(Trigger.isBefore){
            OpportunityTriggerHandler.validateAmount(Trigger.New);
        }
    }
}
public class OpportunityTriggerHandler {

    public static void validateAmount(List<Opportunity> oppList){
        for(Opportunity opp:oppList){
            if(opp.Amount == null){
                opp.addError('Amount field can not be null');
            }
        }
    }
}
```

23. When an opportunity is updated to Closed Lost and Closed Lost Reason (field) is not populated then throw validation error that 'Please populate Closed Lost Reason' on opportunity. [before update]

Solution:

```
trigger OpportunityTrigger on Opportunity (before update) {
    if(Trigger.isUpdate){
        if(Trigger.isBefore){
            OpportunityTriggerHandler.populateClosedReason(Trigger.New
            , Trigger.oldMap);
        }
    }
}
public class OpportunityTriggerHandler {
    public static void populateClosedReason(List<Opportunity> oppList,
    Map<Id,Opportunity> oldMap){
        for(Opportunity opp:oppList){
            if(opp.StageName == 'Closed Lost' && opp.StageName !=
            oldMap.get(opp.Id).StageName &&
            opp.Closed_Lost_Reason__c == null){
                opp.addError('Please populate Closed Lost Reason');
            }
        }
    }
}
```

24. Write a trigger on Account and check only System Administrator profile users should be able to delete an account.

Solution:

```
trigger AccountTrigger on Account (before delete) {
    if(Trigger.isDelete){
        if(Trigger.isBefore){
            AccountTriggerHandler.checkProfileForDeletion(Trigger.old);
        }
    }
}

public class AccountTriggerHandler{

    public static void checkProfileForDeletion(List<Account> accList){

        Profile p = [SELECT Id FROM Profile WHERE Name = 'System
        Administrator'];
        for(Account acc:accList){
            if(UserInfo.getProfileId() != p.Id){
                acc.addError('Only System Administrator can delete
                Account');
            }
        }
    }
}
```

25. If an opportunity is closed then, no one should be able to delete it except the user having a System Administrator profile.

Solution:

```
trigger OpportunityTrigger on Opportunity (before delete) {
    if(Trigger.isDelete){
        if(Trigger.isBefore){
            OpportunityTriggerHandler.checkProfileForDeletion(Trigger.old);
        }
    }
}
public class OpportunityTriggerHandler {

    public static void checkProfileForDeletion(List<Opportunity> oppList){

        Profile p = [SELECT Id FROM Profile WHERE Name = 'System
            Administrator'];
        for(Opportunity opp:oppList){
            if(opp.StageName == 'Closed Won' || opp.StageName ==
                'Closed Lost')
            if(UserInfo.getProfileId() != p.Id){
                opp.addError('Only System administrator can delete
                    opportunity');
            }
        }
    }
}
```

26. Prevent deletion of an account if there is any opportunity related to that account.

Solution:

```
trigger AccountTrigger on Account (before delete) {
    if(Trigger.isDelete){
        if(Trigger.isBefore){
            AccountTriggerHandler.preventDelIfHasRelatedOpp(Trigger.old);
        }
    }
}

public class AccountTriggerHandler{

    public static void preventDelIfHasRelatedOpp(List<Account> accList){
        Set<Id> idSet= new Set<Id>();

        for(Account acc:accList){
            idSet.add(acc.Id);
        }
        for(Account acc:[SELECT Id,(SELECT Id FROM Opportunities)FROM
            Account WHERE Id IN:idSet]){
            if(acc.Opportunities.size( ) > 0){
                acc.addError('You can not delete account where
                    opportunities are available ');
            }
        }
    }
}
```

27. Prevent deletion of an account if there is any case related to that account.

Solution:

```
trigger AccountTrigger on Account (before delete) {
    if(Trigger.isDelete){
        if(Trigger.isBefore){
            AccountTriggerHandler.preventDelIfHasRelatedOpp(Trigger.old);
        }
    }
}

public class AccountTriggerHandler{

    public static void preventDelIfHasRelatedCase(List<Account> accList){
        Set<Id> idSet= new Set<Id>();

        for(Account acc:accList){
            idSet.add(acc.Id);
        }
        for(Account acc:[SELECT Id,(SELECT Id FROM Cases)FROM Account
            WHERE Id IN:idSet]){
            if(acc.Cases.size() > 0){
                acc.addError('You can not delete account where cases are
                    available ');
            }
        }
    }
}
```

28. When the Employee record is deleted then update ‘Left Employee Count’ on Account.

Solution:

```
trigger EmployeeTrigger on Employee__c (After delete) {
    if(Trigger.isDelete){
        if(Trigger.isAfter){
            EmployeeTriggerHandler.leftEmpCount(Trigger.old);
        }
    }
}

public class EmployeeTriggerHandler{

    public static void leftEmpCount(List<Employee__c> oldEmpList){
        Set<Id> accIds=new Set<Id>();
        List<Account> accToBeUpdated=new List<Account>();
        Map<Id,Account> accIdToAccMap;
        List<Employee__c> empList= new List<Employee__c>();
        Map<Id,Decimal> accIdToTotalCount= new Map<Id,Decimal>();

        for(Employee__c emp:oldEmpList){
            if(emp.Account__c!=null){
                accIds.add(emp.Account__c);
                empList.add(emp);
            }
        }
```

```
            if(!accIds.isEmpty()){
                accIdToAccMap= new Map<Id,Account>([SELECT
                 Id,Left_Employee_Count__c FROM Account WHERE Id
                 IN:accIds]);
            }
            if(!empList.isEmpty()){
                for(Employee__c emp:empList){
                    if(accIdToAccMap.containsKey(emp.Account__c)){
                        if(accIdToTotalCount.containsKey(emp.Account__c)){
                                Decimal count =
                                accIdToTotalCount.get(emp.Account__c)+1;
                            accIdToTotalCount.put(emp.Account__c,count);
                          }else{
                           accIdToTotalCount.put(emp.Account__c,accIdToAcc
                         Map.get(emp.Account__c).Left_Employee_Count__c+1);
                            }
                        }
                }
            for(Id accId:accIdToTotalCount.keySet()){
                Account acc= new Account();
                acc.Id=accId;
                acc.Left_Employee_Count__c=accIdToTotalCount.get(accId);
                accToBeUpdated.add(acc);
             }
            if(!accToBeUpdated.isEmpty()){
                update  accToBeUpdated;
            }
        }
}
```

29. Undelete Employee record and set Active as true.

Solution:

```
trigger EmployeeTrigger on Employee__c (After undelete) {
    if(Trigger.isUnDelete){
        if(Trigger.isAfter){
            EmployeeTriggerHandler.unDeletionofEmp(Trigger.New);
        }
    }
}

public class EmployeeTriggerHandler{

    public static void unDeletionofEmp(List<Employee__c> empList){
        List<Employee__c> empListToBeUpdated = new
            List<Employee__c>();
        for(Employee__c emp : empList){
            Employee__c e = new Employee__c();
            e.Id = emp.Id;
            e.Active__c = true;
            empListToBeUpdated.add(e);
        }
        if(!empListToBeUpdated.isEmpty()){
            update empListToBeUpdated;
        }
    }
}
```

30. When the Employee record is undeleted then update 'Left Employee Count' on Account.

Solution:

```
trigger EmployeeTrigger on Employee__c (After undelete) {
 if(Trigger.isUnDelete){
    if(Trigger.isAfter){
       EmployeeTriggerHandler.updateLeftEmpCountUndeletedOnes(Trigger.New);
     }
  }
}

 public class EmployeeTriggerHandler{

        public static void updateLeftEmpCountUndeletedOnes(List<Employee__c>
 oldEmpList){
        Set<Id> accIds=new Set<Id>();
        List<Account> accToBeUpdated=new List<Account>();
        Map<Id,Account> accIdToAccMap;
        List<Employee__c> empList= new List<Employee__c>();
        Map<Id,Decimal> accIdToTotalCount= new Map<Id,Decimal>();

        for(Employee__c emp:oldEmpList){
           if(emp.Account__c!=null){
              accIds.add(emp.Account__c);
              empList.add(emp);
           }
        }

        if(!accIds.isEmpty()){
```

```
        accIdToAccMap= new Map<Id,Account>([SELECT
Id,Left_Employee_Count__c FROM Account WHERE Id IN:accIds]);
      }
      if(!empList.isEmpty()){
        for(Employee__c emp:empList){
          if(accIdToAccMap.containsKey(emp.Account__c)){
            if(accIdToTotalCount.containsKey(emp.Account__c)){
              Decimal count=accIdToTotalCount.get(emp.Account__c)-1;
              accIdToTotalCount.put(emp.Account__c,count);
            }else{

accIdToTotalCount.put(emp.Account__c,accIdToAccMap.get(emp.Account__c).L
eft_Employee_Count__c-1);
            }
          }
        }
      }

      for(Id accId:accIdToTotalCount.keySet()){

        Account acc= new Account();
        acc.Id=accId;
        acc.Left_Employee_Count__c=accIdToTotalCount.get(accId);
        accToBeUpdated.add(acc);
      }
      if(!accToBeUpdated.isEmpty()){
        update  accToBeUpdated;
      }
    }
}
```

31. When the Employee record is inserted, deleted and undeleted then update 'Present Employee Count' on related Account. [Parent-Child SOQL]

Solution:

//Same code will work for insert, delete and undelete

```
trigger EmployeeTrigger on Employee__c(after insert,after delete,after undelete) {
    if(Trigger.isInsert){
        if(Trigger.isAfter){
            EmployeeTriggerHandler.updatePresentEmpCount(Trigger.New);
        }
    }
    if(Trigger.isDelete){
        if(Trigger.isAfter){
            EmployeeTriggerHandler.updatePresentEmpCount(Trigger.New);
        }
    }

    if(Trigger.isUnDelete){
        if(Trigger.isAfter){
            EmployeeTriggerHandler.updatePresentEmpCount(Trigger.New);
        }
    }
}

public class EmployeeTriggerHandler{

public static void updatePresentEmpCount(List<Employee__c> empList){
    List<Account> accList= new List<Account>();
    Set<Id> idSet= new Set<Id>();
    for(Employee__c emp:empList){
        if(emp.Account__c!=null){
```

```
                idSet.add(emp.Account__c);
            }
        }

        for(Account acc:[SELECT Id, Name,(SELECT Id FROM Employees__r)
             FROM Account WHERE Id IN:idSet]){
            acc.Present_Employee_Count__c=acc.Employees__r.size();
            accList.add(acc);
        }

        if(!accList.isEmpty()){
            update accList;
        }
}
```

32. Upon contact creation an email should be sent to email populated on Contact with specified template.

Solution:

```
trigger ContactTrigger on Contact (after insert) {
    if(Trigger.isInsert){
        if(Trigger.isAfter){
            ContactTriggerHandler.sendEmailToContact(Trigger.new);
        }
    }
}
public class ContactTriggerHandler{
        public static void sendEmailToContact(List<Contact> conList){
                List<Messaging.Email> emailList=new List<Messaging.Email>();
                for(Contact con:conList){
                        if(con.Email!=null){
                                Messaging.SingleEmailMessage emailMsg= new
                                        Messaging.SingleEmailMessage();
                                String[] toAddress= new String[]{con.Email};
                                emailMsg.setToAddresses(toAddress);
                                String emailSubject='Welcome '+con.FirstName;
                                emailMsg.setSubject(emailSubject);
                                String disName='Sanjay Gupta';
                                emailMsg.setSenderDisplayName(disName);
                                String content= 'Hi '+con.FirstName+ ',<br/><br/>'+
                                'Welcome to SalesForce EcoSystem! <br/><br/>'+
                                'Happy learning! <br/><br/>'+
                                'Thank you! <br/><br/>';
                                emailMsg.setHtmlBody(content);
                                emailList.add(emailMsg);
                        }
```

```
            }
            Messaging.sendEmail(emailList);
        }
}
```

33. Create two record types named as “Partner Case” and “Customer Case” on Case Object. On creation of Case, as per the record type populate the total number of Partner Case or Customer Case on Account object. Create Custom Fields on Account to have total numbers.

Solution:

```
trigger CaseTrigger on Case (after insert) {
    if(Trigger.isInsert){
        if(Trigger.isAfter){
            CaseTriggerHandler.countCases(Trigger.new);
        }
    }
}
public class CaseTriggerHandler{
    public static void countCases(List<Case> cList){
        List<Account> accList= new List<Account>();
        Set<Id> idSet= new Set<Id>();
        Id partnerCaseRecordTypeId = [Select Id From RecordType Where
            DeveloperName = 'Partner_Case'].Id;//1.way to do with
            DeveloperName which is API name
        Id customerCaseRecordTypeId=[Select Id From RecordType Where
            Name = 'Customer Case'].Id;//2.way to do with Name
        for(Case c:cList){
            if(c.AccountId!=null){
                idSet.add(c.AccountId);
            }
        }
        for(Account acc:[SELECT
            Id,Total_Case__c,Customer_Case__c,Partner_Case__c,(SELE
            CT Id,RecordTypeId  FROM Cases) FROM Account WHERE Id
            IN:idSet]){
```

```
                    decimal countPartner=0;
                    decimal countCustomer=0;
                    for(Case c:acc.Cases){
                        if(c.RecordTypeId==partnerCaseRecordTypeId){
                            countPartner++;
                        }else if(c.RecordTypeId==customerCaseRecordTypeId){
                            countCustomer++;
                        }
                    }
                    acc.Customer_Case__c=countCustomer;
                    acc.Partner_Case__c=countPartner;
                    acc.Total_Case__c=acc.Customer_Case__c+acc.Partner_Case
                        __c;
                    accList.add(acc);
                }
                if(!accList.isEmpty()){
                update accList;
                }
        }
}
```

34. When any Opportunity is created with amount populated or Opportunity Amount is updated then populate total Amount on Account Level for all related opportunities in Annual Revenue Field. If opportunity is deleted or undeleted then update Amount on Account as well. (Hint: rollup summary)

Solution:

```
//Call this method for insert, update, delete and undelete event
public static void populateAmountOnAccount(List<Opportunity>
        oppList,Map<Id,Opportunity> oldMap){
        Set<Id> accIds = new Set<Id>();
        for(Opportunity opp : oppList){
                if(oldMap != null){
                        if(opp.AccountId!=null&&opp.Amount!=null&&opp.Amount!=old
                        Map.get(opp.Id).Amount){
                                accIds.add(opp.AccountId);
                        }
                }else{
                        if(opp.AccountId != null && opp.Amount != null){
                                accIds.add(opp.AccountId);
                        }
                }
      }

        List<Account> accList=[SELECT Id,AnnualRevenue,(SELECT Id, Amount
                FROM Opportunities) FROM Account WHERE Id IN: accIds];
        if(!accList.isEmpty()){
                for(Account acc:accList){
                        Decimal total=0;
                        for(Opportunity opp:acc.Opportunities){
                                total=total+opp.Amount;
                        }
```

```
                    acc.AnnualRevenue=total;
                }
            }
            if(!accList.isEmpty()){
                update accList;
            }
}
```

35. Database class and addError( ) method in trigger.

Solution:

```
trigger AccountTrigger on Account (after insert) {
    if(Trigger.isInsert){
        if(Trigger.isAfter){
            AccountTriggerHandler.createOpp(Trigger.new);
        }
    }
}

public class AccountTriggerHandler{

    public static void createOpp(List<Account> accList){
        List<Opportunity> oppList= new List<Opportunity>();
        for(Account acc:accList){
            Opportunity opp= new Opportunity();
            opp.Name=acc.Name;
            opp.AccountId=acc.Id;
            opp.StageName='Prospecting';
            // opp.CloseDate=System.today();
            oppList.add(opp);
        }

        Database.SaveResult[] srList= DataBase.insert(oppList,false);
        for(Integer i=0;i<srList.size();i++){
            if(!srList[i].isSuccess()){
                String errors='';
                for(DataBase.Error err:srList[i].getErrors()){
                    errors=errors+err.getMessage();
```

```
                        }
                        accList[i].addError(errors);
                    }
                }
            }
        }
```

36. Prevent Recursion in Apex Trigger.

Solution: https://www.youtube.com/watch?v=9R4y3CENJcs

37. When an opportunity is updated to closed won or closed lost then update description having value 'Opportunity is Closed Won' or 'Opportunity is Closed Lost' accordingly. (Hint: Recursion)

Solution:

```
public class preventRecursion {
    public static Boolean firstCall=false;
}

trigger OpportunityTrigger on Opportunity(after update){
    if(Trigger.isUpdate){
        if(Trigger.isAfter){
            if(!preventRecursion.firstCall){
                preventRecursion.firstCall=true;
                OpportunityTriggerHandler.updateStage(Trigger.New,Trig
            ger.oldMap);
        }
    }
}
public class OpportunityTriggerHandler{
    public static void updateStageRecursion(List<Opportunity>
    oppList,Map<Id,Opportunity>oldMap){
        List<Opportunity> oppToBeUpdated= new List<Opportunity>();
        for(Opportunity opp: oppList){
            if(opp.StageName=='Closed Won'||opp.StageName=='Closed
            Lost'){
                Opportunity o= new Opportunity(id=opp.Id);
```

```
                        if(opp.StageName=='Closed Won'){
                            o.Description='Opportunity is Closed Won';
                        }else if(opp.StageName=='Closed Lost'){
                            o.Description='Opportunity is Closed Lost';
                        }
                    oppToBeUpdated.add(o);
                    }
                }
                if(!oppToBeUpdated.isEmpty()){
                    update oppToBeUpdated;
                }
        }
}
```

38. Write a trigger, if the owner of an account is changed then the owner for the related contacts should also be updated. [Without Map]

Solution:

```
trigger AccountTrigger on Account (after update) {
    if(Trigger.isUpdate){
        if(Trigger.isAfter){
            AccountTriggerHandler.updateOwnerOfRelatedContact(Trigger.
                new, Trigger.oldMap);
        }
    }
}

public class AccountTriggerHandler{

    public static void updateOwnerOfRelatedContact(List<Account>
        accList,Map<Id,Account>oldMap){
        List<Contact> conList= new List<Contact>();
        Set<Id>idSet= new Set<Id>();
        for(Account acc:accList){
            if(acc.OwnerId!=oldMap.get(acc.Id).OwnerId){
                idSet.add(acc.Id);
            }
        }

        for(Account acc:[SELECT Id,OwnerId, (SELECT OwnerId FROM
            Contacts) FROM Account WHERE Id IN:idSet]){
            if(acc.Contacts!=null){
                for(Contact c:acc.Contacts){
                    c.OwnerId=acc.OwnerId;
                    conList.add(c);
```

```
                        }
                    }
                }
                if(!conList.isEmpty()){
                    update conList;
                }
            }
        }
```

39. Write a trigger, if the owner of an account is changed then the owner for the related contacts should also be updated. [Using Map]

Solution:

```
trigger AccountTrigger on Account (after update) {
    if(Trigger.isUpdate){
        if(Trigger.isAfter){
            AccountTriggerHandler.updateOwnerOfRelatedContact(Trigger.
                new, Trigger.oldMap);
        }
    }
}

public class AccountTriggerHandler{
    public static void updateOwnerOfRelatedContact(List<Account>
    accList,Map<Id,Account>oldMap){

        List<Contact> conList= new List<Contact>();
        Map<Id,Account> accToAccountMap= new Map<Id,Account>();
        for(Account acc:accList){
            if(acc.OwnerId!=oldMap.get(acc.Id).OwnerId){
                accToAccountMap.put(acc.Id,acc);
            }
        }

        for(Contact con:[SELECT AccountId,OwnerId FROM Contact
        WHERE AccountId IN:accToAccountMap.keySet()]){
            con.OwnerId=accToAccountMap.get(con.AccountId).OwnerId;
            conList.add(con);
        }
```

```
            if(!conList.isEmpty()){
                update conList;
            }
        }
}
```

40. Whenever a new User having profile “System Administrator” is inserted and is Active, add the user to the public group “Admins”. Create a public group named Admins.

Solution:

```
trigger UserTrigger on User (after insert) {
    if(Trigger.isInsert){
        if(Trigger.isAfter){
            UserTriggerHandler.addUserToGroup(Trigger.new);
        }
    }
}

public class UserTriggerHandler{

    public static void addUserToGroup(List<User> usList){

        Id systemAdminId= [SELECT Id FROM Profile WHERE
            Name='System Administrator'].Id;

        Id groupId=[SELECT Id FROM Group WHERE Name='Admins'].Id;
        List<GroupMember> groupList = new List<GroupMember>();

        for(User us : usList){
            if(us.ProfileId == systemAdminId && us.IsActive){
                GroupMember grp= new GroupMember();
                grp.GroupId=groupId;
                grp.UserOrGroupId=us.Id;
                groupList.add(grp);
            }
        }
```

```
            if(!groupList.isEmpty()){
                insert groupList;
            }
        }
}
```

41. Write a trigger on contact to prevent duplicate records based on Contact Email.

Solution:

```
public static void preventDuplicateEmail(List<Contact> conList,Map<Id,Contact>
oldMap){
        Set<String> emailSet= new Set<String>();
        for(Contact con:conList){
                if(oldMap==null && con.Email!=null){
                        emailSet.add(con.Email);
                }else{
                        if(con.Email!=null && con.Email!=oldMap.get(con.Id).Email){
                                emailSet.add(con.Email);
                        }
                }
        }

        List<Contact> existingContactList= new List<Contact>([SELECT Id,Email
FROM Contact WHERE Email IN:emailSet]);
        Set<String> emailListForExisting= new Set<String>();
        if(!existingContactList.isempty()){
                for(Contact con:existingContactList){
                        emailListForExisting.add(con.Email);
                }
       }
       for(Contact con:conList){
                if(emailListForExisting.contains(con.Email)){
                con.addError('Duplicate email');
                }
        }
}
```

42. Set OWD as Private for Account. Once an Account record is created, it should be automatically shared with any one user who belongs to Standard User profile.

Solution:

```
trigger AccountTrigger on Account (after insert) {
    if(Trigger.isInsert){
        if(Trigger.isAfter){
            AccountTriggerHandler.shareAccWithStdUser(Trigger.new);
        }
    }
}

public class AccountTriggerHandler{

    public static void shareAccWithStdUser(List<Account> accList){

        Id standartUserId = [SELECT Id FROM Profile WHERE
        Name='Standard User'].Id;

        List<User> listOfUserId = [SELECT Id FROM User WHERE
        ProfileId= :standartUserId AND IsActive=True LIMIT 1];

        List<AccountShare> accShareList=new List<AccountShare>();

        for(Account acc:accList){
            AccountShare aShare= new AccountShare();
            aShare.UserOrGroupId=listOfUserId[0].Id;
            aShare.AccountId=acc.Id;
            aShare.RowCause='Manual';
            aShare.AccountAccessLevel='Edit';
```

```
                aShare.OpportunityAccessLevel='Edit';
                accShareList.add(aShare);
            }
            if(!accShareList.isEmpty()){
                insert accShareList;
            }
        }
}
```

43. Demo Trigger.isExecuting Context Variable.

Solution:

```
public class AccountHandler{
	public Boolean handleAccount(List<Account> accList){

		System.debug('Trigger is executing : ' + Trigger.isExecuting);

		if(Trigger.isExecuting){
			//do whatever you want to do as part of the trigger invocation
		}
		else{
			//do whatever you want to do if the call originated from a
			different context, such as from the controller.
		}
		return Trigger.isExecuting;
	}
}
```

Note:

- **If you find any bug in any of the code snippets then please share in the comments.**
- **A problem can be solved through more than one way. If you know any other way then please share it as well in comments**