```
////////////////////////////////  salesforce exclusive Asynchronous
apex  ///////////////////////////////////

/////////////////////////////////// BATCH
APEX ////////////////////////////////////////////////////////////////
/*
1.We can perform DML operation like insert,update,delete on only 10k records at
once because SF has 10k as governor limit.So we use batch apex
as it can process upto 50 million records in background {future & queuable only has
double limit of normal soql & dml limit , like normal soql
limit is 100 & dml is 150 so for future and queable it will be 200 & 300 as it
comes in asynchronous apex but batch apex can process upto 50
million records in background}. Default batchsize is 200 , minimum 1 & max batch
size is 2000
2.we can only have 5 batch jobs running at a time
3.Future methods cannot be called from batch apex
4.We can chain jobs in batch apex like queuable (eg:- Database.executebatch(new
secondBatch() 200); -> write this in first batch job finish method )
*/

//BATCH APEX SYNTAX
/*
start() -> querys all the records to process
execute() -> process batch of records from start
finish() -> any post processing logic like sending email
*/
Public class TestDemoBatch implements Database.batchable<Sobject>{
    Public Database.QueryLocator start(Database.BatchableContext bc) {
        return Database.getQueryLocator('SELECT Id from Account'); //it will not
hit exception even if query returns 1 million records , normal limit is 50k but
getquerylocator doesnot have any limit
    }
    Public void execute(Database.BatchableContext bc, List < sobject >
subListFromStartMethod){
    // Logic to process all records
    for (Account acc: subListFromStartMethod ) {
        acc.name = 'Batch updated-' + acc.Name;
    }
        UPDATE subListFromstartMethod;
}
    Public void finish(Database.BatchableContext bc){
    // send mail
    Database.executebatch(new secondBatch() 200); // for chaining

}
}

//To call batch apex
Database.executeBatch(new TestDemoBatch(), 200);



/////////////////////////////////// SCHEDULE
APEX ////////////////////////////////////////////////////////////////
/* U WILL LEARN
1. WHAT IS SCHEDULED APEX?
2. HOW TO SCHEDULE FROM UI AND
FROM CRON EXPRESSION
3. SCENARIOS TO USE SCHEDULE APEX
```

```
4. HOW TO MONITOR SCHEDULED APEX */

//We can schedule a batch class , queuable class or any block of code to run in
schedule apex (eg:- run batch daily at 1 AM)
//SYNTAX
public class MyClass implements Schedulable {
    public void execute(SchedulableContext context) {
        //CODE THAT WILL NEEDS TO RUN IN SCHEDULE
    }
}

//eg :- 1
public class ScheduleApexDemo implements Schedulable {
    public void execute(schedulableContext sc) {
        /Any code written inside this method can be schedule
        List < Account > accs=[SELECT id, name FROM Account WHERE CREATEDDATE =
Today];
        for (Account acc: accs) {
            acc.name - 'Created today' - acc.Name;
        }
    UPDATE accs;
    }
}

//eg :- 2 (CALLING BATCH CLASS FROM SCHEDULE APEX)
public class SchedulcApexDemo implements Schedulable {
    public void execute(SchedulableContext sc) {
        Database.executeBatch(new TestDemoBatch()); //OR  Database.executeBatch(new
TestDemoBatch() , 100); --> WHERE TestDemoBatch IS BATCH CLASS NAME AND 100 IS
BATCH SIZE

    }
}

//eg :- 3 (CALLING Queueable CLASS FROM SCHEDULE APEX)
public class QueueableScheduleClass implements Schedulable {
    public void execute(SchedulableContext sc) {
        System.enqueueJob(new QueueableClassDemo());
    }
}

/*
Scheduling can be done in 2 ways
1. From User interface.
2. Programatically using System.schedule method.
*/

//1. From User interface - SETUP -> CLASSES -> SCHEDULEAPEX -> GIVE JOBNAME ->
SELECT CLASS -> SELECT FREQUENCY




////////////////////////////////  salesforce exclusive apex
triggers ////////////////////////////////////

/*RecordId is not generated in beforeInsert trigger as record is not saved in
database, RecordId is generated for afterInsert trigger*/
```

```
/* BEST PRACTISES FOR WRITTING TRIGGERS
One Trigger per Object (As order of execution of multiple triggers on same object
is not guarenteed)
Bulkify your Trigger. Code should work if records are
inserted/Updated/Deleted/undeleted in bulk
Avoid DML Statements/ SOQL queries in For Loop (Else it hits Governor limits)
Do not write logics in Trigger, Use Trigger Handler Instead
Avoid Hardcoding lds
Prevent Recursive Triggers
Use Comments to make your trigger readable
*/

/*
EVENTS & CONTEXT VARIABLES
Before Insert - Trigger.new
After Insert - Trigger.new , Trigger.newMap
Before update - Trigger.new , Trigger.newMap , Trigger.old , Trigger.oldMap
After update - Trigger.new , Trigger.newMap , Trigger.old , Trigger.oldMap
Before delete - Trigger.old , Trigger.oldMap
After delete - Trigger.old , Trigger.oldMap
After undelete - Trigger.new , Trigger.newMap
*/

trigger AccountTrigger on Account(Before Insert, after Insert, Before update, after
update, Before delete, After delete, after undelete){
    //CONTEXT VARIABLES (values which developer needs to write logic)
    //CONTEXT Variable 1: Trigger.new -> List of records that are got
inserted/updated
    //CONTEXT Variable 2: Trigger.isbefore -> returns true if trigger is running on
before event
    //CONTEXT Variable 3: Trigger.isInsert -> returns true if trigger is called
when user has done insert operation
    //CONTEXT Variable 4: Trigger.isafter -> returns true if trigger is called
after the record is inserted/updated
    //CONTEXT Variable 5: Trigger.newMap -> returns the list of records that are
inserted/updated with latest values in map format
    //CONTEXT Variable 6: Trigger.oldMap -> returns the list of records that are
inserted/updated with old/prior values in map format
    //CONTEXT Variable 7: Trigger.old -> Returns the List of records that are
inserted/updated with old/prior values
    //CONTEXT Variable 8: Trigger.isUpdate -> Returns true if trigger is called
when record is updated

    /*1. BEFORE INSERT
    senario 1) While user creating an account, if user provides Billing address but
not Shipping address, write a logic to  populate shipping address with billing
address
    senario 2) While user creating an account, if Annual revenue provided by user
is less than 1000, then write a logic to throw an error to user.*/
    If(Trigger.isBefore && Trigger.isInsert){
        for (account accRec: Trigger.new) {
            //senario 1
            if (accRec.ShippingCity == nul1)
                accRec.ShippingCity = accRec.Billingcity;
            if (accRec.ShippingCountry == null)
                accRec.Shippingcountry = accRec.BillingCountry;
            if (accRec.shippingState == nul1)
                actRec.shippingState = accRec.BillingState;
            if (accRec.ShippingStreet == null)
```

```
                accRec.Shippingstreet = accRec.billingstreet;
            if (accRec.ShippingPostalCode == null)
                accRec.ShippingPostalCode = accRec.BillingPostalCode;

            //senario 2
            if (accRec.AnnualRevenue < 1000)
                accRec.addError('Annual Revenue cannot ba less than 1000');
        }
    }
    //NEVER USE INSERT/UPDATE DML STATEMENT IN BEFORE EVENTS , they r automatically
taken care of

    /*2. AFTER INSERT
    When user created an account, write a logic to create contact with same name
and associate account & contact*/
    //AFTER INSERT LOGIC TO BE WRITTEN IN THIS BELOW BLOCK
    if (Trigger.inAfter && Trigger.isInsert) {
        List < Contact > conListToInsert = new List < Contact > ();
        for (Account accRec: Trigger.new) {
    Contact con = new Contact();
            con.LastName = accRec.Name
            con.AccountId = accRec.Id;
            conListToInsert.add(con);
        }
        if (conListToInsert.size() > 0)
    INSERT conListToInsert;
    }

    /*3. BEFORE UPDATE
    When user updates account record, if user changes account name, throw an error
'Account name once created cannot be modified' */
    //BEFORE UPDATE LOGIC 10 BE WRITTERN IN THIS BELOW BLOCK
    IF(Trigger.isBefore && Trigger.isupdate){
        System.debug('New Values');
        System.debug(Trigger.new);
        System.debug(Tigger.newmap); //Id, Recordwithnewvalues
        System.debug('0ld values');
        System.debug(Trigger .0ld);
        System.debug(Trigger.oldMap); //1d, recordwdtholdvalues
        for (Account accRecNew: Trigger.new) {
    Account accRecOld = Trigger.oldMap.get(accRecNew.Id);
            if (accRecNew.Name - accRecOld.Name)
                accRecNew.addError('Account Name cannot be modified/ changed once
it is created');
        }
    }

    /*4. AFTER UPDATE
    On Account record update, if mailing address is changed, update all its child
contacts mail address field same as account mailing address */
    //AFTER UPDATE LOGIC TO BE WRITTEN IN THIS BELOW BLOCK
    If(Trigger.isAfter && Trigger.isupdate){
        Set < Id > accIdWhichGotBillingAddressChanged = new Set < Id > ();
        for (Account accRecNew: Trigger.New) {
    Account accRecOld = Trigger.OldMap.get(accRecNew.Id);
            if (accRecNew.BillingStreet != accRecOld.BillingStreet) {
                accIdWhichGotBillingAddressChanged.add(accRecNew.Id);
            }
        }
```

```
        //This set accIdWhichGotBillingAddressChanged will have accountIds which
got billing address changed
        List < Account > accswithContacts =[SELECT Id, Name, billingcity,
billingstreet, billingstate, billingcountry, (SELECT Id, Name FROM Contacts) FROM
Account WHERE Id IN: accIdWhichGotBillingAddressChanged];
        List < Contact > contsListToupdate = new List < Contact > ();
        for (Account acc: accsWithContacts) {
            List < Contact > consOfTheLoopedAccount = acc.contacts;
            for (Contact con: consOfTheLoopedAccount)
            {
                con.mailingstreet = acc.billingstreet;
                con.MailingCity = acc.BillingCity;
                con.Mailingstate = acc.BillingState;
                con.Mailingcountry = acc.Billingcountry;
                contslistToUpdate.add(con);
            }
        }
        If(contsListToupdate.size() > 0){
        UPDATE contsListToupdate;
        }

        /*5. BEFORE DELETE
        An active account should not be deleted.*/
        //BEFORE DELETE LOGIC TO BE WRITTEN IN THIS BELOW BLOCK
        If(Trigger.isBefore && Trigger.isDelete){
            //Trigger.new  & trigger.newmap r not available in Delete cperation
(new and newmap)
            //Trigger.old & Trigger.oldmap are available in Delete operatian
            for (Account accold: Trigger.old) {
                if (accold.Active_c == "Yes")
                    accold.addError('You cannot delete an active account');

            }
        }

        /*6. AFTER DELETE
        When ever account is deleted, send an email to account owner.*/
        //AFTER DELETE LOGIC IS WRITTEN IN THIS BELOW BLOCK
        If(Trigger.isafter && Trigger.isDelete){
            //Sending email to user who deletes the records
            //Trigger.new is not available in Delete operation (new and newmap)
            //Trigger.old & oldmap are available in Delete cperatian
            List < Messaging.SingleEmailMessage > emailObjs = new List <
Messaging.SingleEmailMessage > ();
            for (Account accold: Trigger.o1d) {
                Messaging.SinglEmailMessage emailObj = new
Messaging.SingleEmailMessage();
                List < String > emailaddress = new List < String > ();
                emailAddress.add(Userinfo.getUserEmail());
                emailobj.toaddresses(emailAddress);
                emailobj.setSubject('Account has been sucessfully deleted' +
accold.Name);
                emailobj.setPlainTextBody('Hello. . no body written here. .please
refer subject');
                emailObjs.add(emailobj);
            }
            Messaging.sendEmail(emailObjs);
        }
    }
```

```
    /*7. AFTER UNDELETE
Send an email to account owner when account is being restored from Recycle bin*/
    //AFTER UNDELETE LOGIC IS WRITTEN IN THIS BELOW BLOCK
    If(Trigger.isafter && Trigger.isundelete){
        //Sending email to user who restores the records
        //Trigger.new/newmap is available in unDelete operation (new and newmap)
        //Trigger.old & oldmap is notavailable in unDelete cperatian
        List < Messaging.SingleEmailMessage > emailObjs = new List <
Messaging.SingleEmailMessage > ();
        for (Account accold: Trigger.new) {
            Messaging.SinglEmailMessage emailObj = new
Messaging.SingleEmailMessage();
            List < String > emailaddress = new List < String > ();
            emailAddress.add(Userinfo.getUserEmail());
            emailobj.toaddresses(emailAddress);
            emailobj.setSubject('Account has been sucessfully restored' +
accold.Name);
            emailobj.setPlainTextBody('Hello. . no body written here. .please refer
subject');
            emailObjs.add(emailobj);
        }
        Messaging.sendEmail(emailObjs);
    }

}

/*8. RECRUSSIVE TRIGGER ON CONTACT WITH HANDLER CLASS*/
trigger ContactTrigger on Contact(after insert){
    if (Trigger.isAfter && Trigger.isInsert && !ContactTriggerHandler.isTriggerRan)
{
        ContactTriggerHandler.isTriggerRan = true;
        ContactTriggerHandler.createDuplicateContact(Trigger.new);
    }
}

public class ContactTriggerHandler {
    public static Boolean isTrigagerRan = false;
    public static void createDuplicateContact(List<Contact> newconsList) {
    List < Contact > dupConsToInsert = new List < Contact > ();
    for (Contact con: newconsList) {
    Contact con1 = new Contact();
        con1.lastname = 'Duplicate contact';
        dupConsToInsert.add(con1);
    }
    INSERT dupConsToInsert;
}
    }
```

```
////////////////////////////////  sanjay gupta apex
triggers ///////////////////////////////////
/*dml is required in after and in before dml is not required*/

/*  1)  Before Insert
If Account Industry is not null and having value as 'Media' then
populate Rating as Hot. */

trigger AccountTrigger on Account(before insert, after insert) {
    if (Trigger.isInsert) {
        if (Trigger.isBefore) {
            AccountTriggerHandler.beforeInsert(Trigger.New);
        } else if (Trigger.isAfter) {
            AccountTriggerHandler.afterInsert(Trigger.New);
        }
    }
}
////////////////////

public class Account TriggerHandler {
public static void beforeInsert(List < Account > newList) {
        for (Account acc : newList) {
            if (acc.Industry != null && acc.Industry == 'Media') {
                acc.Rating = 'Hot'; I
            }
        }
    }
}

/* 2) After Insert
Create related Opportunity when Account is created. */

public static void createRelated0pp(List < Account > newList){
    List < 0pportunity > oppToBeInserted = new ListcOpportunity > ();
    for (Account acc: newlist) {
    Opportunity opp = new Opportunity();
        opp.Name = acc.Name;
        opp.AccountId = acc.Id;
        opp.StageName = 'Prospecting';
        opp.CloseDate = System.today():
        oppToBeInserted.add(opp);
    }
    if (!oppToBeInserted.isEmpty()) {
insert oppToBeInserted;
    }
}

/*
```

```
Before Update
If account phone is updated then put update message in description.*/

if (Trigger.isUpdate) {
    if (Trigger.isBefore) {
        AccountTriggerHandler.updatePhoneDescription(Trigger.New, Trigger.oldMap);
    } else if (Trigger.isAfter) {
        AccountTriggerHandler.updateRelatedOppPhone(Trigger.New, Trigger.oldMap);
    }
}
//////////////////

public static void updatePhoneDescription(List < Account > newList, Map < Id,
Account > oldMap) {
    for (Account acc: nenlList) {
        if (oldMap != null && acc.Phone != oldMap.get(acc.Id).Phone) {
            acc.Description = "Phone is modified on Account";
        }
    }
}

/* After Update
If Account phone is updated then populate that on all related
opportunities. */

public static void updateRelatedOppPhone(List < Account > newlist, Map < Id,
Account > oldMap){
    Map < Id, Account > accIdToAccountMap = new Map < Id, Account > ();
    List < opportunity > oppToBeUpdated = new List < opportunity > ();
    for (Account acc newList) {
        if (oldMap != null && acc.Phone != oldMap.get(acc.Id).Phone) {
            accIdToAccountMap - put(acc.Id, acc);
        }
    }
    for (opportunity opp: [SELECT Id, Phone FROM Opportunity wHERE AccountId IN :
accIdToAccountMap.keySet()]) {
        Opportunity oppor = new Opportunity();
        if (accIdToAccountMap.containsKey(opp.AccountId)) {
            oppor.id opp.id;
            oppor.Account_Phone__c = accIdToAccountMap.get(opp.AccountId).Phone;
            oppToBeUpdated.add(oppor);
        }
    }
    if (loppToBeUpdated.isEmpty()) {
update oppToBeUpdated;
    }
}

/* Before Delete
Employee record cannot be deleted if active is true.*/

trigger EmployeeTrigger on Employee_c(before delete, after delete)
if (Trigger.isDelete) {
    if (Trigger.isbefore) {
        EmployeeTriggerHandler.checkEmployeeStatus(Trigser.old);
else if (Trigger.isAfter) {
            EmployeeTriggerHandler.updateLeftEmpCountOnAcc(Trigger.old);
        }
    }
```

```
}
//////////////////

public class EmployeeTriggerHandler {
    public static void checkEmployeeStatus(List <Employee__ c> oldList) {
    for (Employee_c emp : oldlist)
    if (emp.Active__c == true) {
        emp.addError('Active Employee cannot be removed ');
    }
}
}

/*After Delete
When Employee record is deleted then update Left Employ
Count on Account.*/

public static void updateLeftEmpCountOnAcc(List < Employee_c > oldList) {
    Set Id > accIds = new Set < Id > ();
    ListAccount > accToBeUpdated = new List < Account > ();
    Map 1d, Account > accIdToAccMap;
    List Employee_c > empList = new List < Employee_c > ();
    Map Id, Decimal > accIdToTotalCount = new Map < Id, Decimal > ();
    for (Employee_c emp : oldList) {
        if (emp.Account _c I = 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.1stmpty()) {
        for (Employee__c emp : empList) {
            if (accIdToAccMap.containsKey(emp.Account c)) {
                if (accIdTotalcount.containskey(emp.Account_c)) {
            Decimal count = accIdTotalcount.get(emp.ACcount_c) + 1;
                    accIdTotalcount.put(emp.Account_c, count);
                } else {
                    accIdToTotalCount.put(emp.Account_c, accIdToAccMap
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 (laccToBeUpdated.isEmpty()) {
        update accToßeUpdated;
    }
}
```