Capital info SOLUTIONS
Training • Consulting • Development

salesforce.com PARTNER

INDEX

CAPITAL INFO SOLUTIONS

Shymala plaza, Behind HUDA Maitrivanam, Ameerpet, Hyderabad,
8686864286, 040-66028688.

Email:capitalinfosol@gmail.com

www.capitalinfosol.com

"Only Capital Info Solutions Students Can Answer These Questions"

SALESFORCE$_{CRM}$

InrerviewQuestions

By**SatishMyla**

1) Visualforce Basic Questions :

1. What is view state in visual force ?
2. Which api used to design visual force page?
3. What is the difference between actionSupport and actionFunction
4. What is the actionRegion?
5. What is difference between insert and include?
6. How do you use static resource in VF page?
7. What is remote action?
8. How many records we can print a pageBlock
9. What is the difference between related List ,enhanced List ,detail
10. What is the difference between controller and extension?
11. What is Ajax? Have you used it ?if so tell the scenario ?
12. What is Jquery ? Where you have used ?
13. What is S-Controls ?
14. What is the use Static Resource in Visual force?
15. Can I pass parameters from VF page to apex method?
16. How do you refer to current page id
17. Tell me something $Action

18.How do you embed Google map in visual force?
19.How do you pass the parameters from page to page ?
20.What are custom components?
21.How do you make a VF page available for Salesforce1 .
22.How to use Sforce connection ?
23.What is custom component ?
24.How to inplementautoLookup and query from VF pages ?

2) Apex fundamentals

1. What is Apex ?
2. What API is used in the apex?
3. What are the access modifiers in the apex?
4. What is the difference between With Sharing and Without Sharing ?
5. What is a constructor?
6. What is the use of the static variables?
7. What are reference variables in apex?
8. What are Sobjects?
9. What is the difference between List and Set?
10. What is Map in apex?
11. Can we have duplicate Keys in Map
12. How many objects we can store in list ?
13. What are setter and getter methods?
14. How do you refer to current page id in apex?
15. How to do you invoke standard actions in apex class?
16. What is page reference?
17. How do you pass the parameters from on apex class to another to another?
18. What is virtual class?
19. What is interface?
20. What is abstract class?
21. What is overloading?
22. What is overriding?

23.When we invoke with sharing method in without sharing class .Now method is Executed as?

24.Will the inner class inherits the sharing properties of outer class?

25.Base class is declared as With Sharing and Derived class is declared as without Sharing what will happen?

26.Can I have constructor with parameters in apex?

27.Dereferncing a Null pointer value error?

28.Variable is not available?

29.Too many Records :10001

3) Batch Apex and Schedule apex questions.

1. What are the Soql limitations in apex?
2. What are transaction limits in apex?
3. What is the need of batch apex?
4. What is Database.Batchable interface?
5. Define the methodsin Batchable interface?
6. What is purpose of Start method in batch apex?
7. What is the Database.QueryLocator ?
8. What is the Iterable<Sobject> ?
9. How to define the custom Iterable?
10. What is the use of execute method?
11. How many times execute method is called?
12. What is scope of execute method?
13. Can we call callouts from batch apex?
14. Can we call another batch apex from batch apex?
15. How many callouts we can call in batch apex?
16. If you get Callouts Governing limits error how do you rectify?
17. Batch is synchronous or Asynchronous operations?
18. How to synchronize the batch apex?
19. How do you call batch apex from the batch apex in earlier versions of API 26.0?
20. What all the general errors/ exception we will get in executing batch apex?
21. What is the maximum size of the batch and minimum size of the batch?
22. Tell some of the scenario's that you have developed using batch apex?
23. What is Database.BatchableContext ?
24. How to track the details of the current running Batch using BatchableContext?
25. How many batch jobs can be added to queue?
26. What is Database.Statefull interface
27. What is Database.AllowCallouts
28. Why should we call Database.execute() and future methods in Test.startTest() and Test.StopTest()
29. What is ASyncApexJob object?
30. When a BatchApexWorker record is created?

4) Schedule Apex :

1. What is Schedule apex?
2. How many ways we can schedule the batch apex?
3. What is Schedulable interface?
4. What is the order of execution?
5. How can we schedule the batch apex?

6. How many schedule jobs we can schedule at a time?
7. What is cronTrigger?
8. How to identify the jobs next schedule?
9. What is the difference between Synchronous and Asynchronous jobs
10. How many future calls we can make in aday?
11. What is the difference between manual schedule and apex schedule?
12. What is the best scenario that you have designed using schedule apex?
13. How to invoke asynchronous callouts in schedule apex?
14. Can we call synchronous callouts in Schedule apex?
15. What are the problems that you have encountered while using schedule apex?

5) Triggers:

1. What is trigger?
2. What are differenttypes of Triggers in sfdc?
3. What areTrigger Context variable?
4. What is the difference between Trigger. New and Trigger.NewMap
5. What is the difference betweenTrigger.New and Trigger.old
6. What is the difference between and Trigger.New and Trigger.Old in update Triggers
7. Can we call the Batchapex from the trigger?
8. What are the problems you have encountered when calling batch apex from the trigger.
9. Can we call the callouts from triggers?
10. What are the problems that you encountered while calling apex callouts in triggers.
11. What is the recursive triggers?
12. What is bulkifying triggers?
13. What is the use of future methods in triggers?
14. What is the order of executing the trigger apex?
15. What is trigger handler?
16. How do you avoid recursive triggers?
17. How many triggers we can define ona object?
18. Can we define two triggers with with same event on single object?
19. Tell me some scenarios' where you have written the triggers?
20. What is the best scenario that you have developed on triggers?
21. How many time workflow filed update will be called in triggers?
22. When are the rollup summary fields are calculated ?

6. WebServices:

1. What is the difference between HTTP1.0 and HTTP 2.0
2. Difference between REST API and SOAP API?

8. In how many ways we can deploy the code from sandbox to production?
9. What all the problems that you have when we are deploying ?
10. What should you use command based deployment

8.Flow and Plugins

1.What are visualflows ?
2.How to define custom wizards?
3.How to integrate visualflows to visualforce page ?
4.What is process.plugin
5.How to call webservices from Flows
6.What are the updates on flows in winter 15

9.Custom Setting

1.What are custom settings ?
2.What are different types of custom setting ?
3.When to use Custom Setting and Custom Labels
4.What are global actions ?

10.Mailing Concepts

1.What is the difference between SingleEmailMessage and MassEmail Message
2.How to pass the VF page as PDF in Email
3.what is TargerObject Id
4.What is the difference between whoId and what Id

3. What do you mean by statefull?
4. How many types of WSDL we have in salesforce?
5. What is the difference between Enterprise ,Partner WSDL?
6. What is difference between Webservice and Callouts?
7. What is the use of Metadata API?
8. How to fetch data from another Salesforce instance using API?
9. What the annotations in salesforce?
10. In How many ways we get the session id using SOAP API?
11. Can we use Http callouts in SOAP API?
12. How to pass the session in REST API?
13. How many ways we can provide security in REST API?
14. What is diffenectbetween OAuth 1.0 and OAuth2.0 ?
15. How many ways we can specify the Request Header in REST API to getaccesstoken?
16. What types of response you have taken from the external server?
17. Can we call any resource from external server using url?
18. What is future annotation ?
19. Did you work with heroku?
20. By default webservices are synchronus/asynchronous ?
21. Have you worked in Siebel to Salesforce data Migration?
22. How to implement Salesforce to Salesforce connection?
23. How to call Apex method from a Custom Button?
24. Have you worked in integrating Google chart?
25. What is Meta Data API
26. What is the use of Tooling API
27. What is the use of Apex WSDL
28. How to create a new Custom Object usingwebservices?
29. How to create a new Custom Fields /Workflows using webservices?
30. How to connect Microsoft excel to Salesforce ?
31. Data migration using third party tools like Informatica cloud ?
32. Did you use any email campaign services ?

7. Deployment and Change Sets :

1. What are the diffent types of changesets ?
2. How to create a sandbox ?
3. How many times we can refresh a(configuration /develope/full)
4. Which things we can not pass using the change sets to production?
5. Can we move the record types using the change sets to production?
6. Can we move approval using change sets to production?
7. Can we move role /users/profiles using change set to production?

Apex

Apex is a strongly typed object oriented programming language.

→ It allows the developers to execute flows and transaction Control Statements.

→ Apex enables Developers to add business logic to most system events like button clicks related record updates and visualforce pages.

→ Apex language

i) Integrated :- It provides built in support for DML calls.

ii) Inline salesforce object query language :-

iii) Easy to use,

iv) Easy to test.

v) Version

vi) Multi tenant aware

When should i use Apex :-

→ To create email service.

→ Create webservices.

→ Perform complex validation over multiple objects.

→ To create complex business processes that are not supported by work flow.

→ Create custom transaction logic.

→ Attach custom logic to another operation.

How does Apex Work :-

All Apex programs runs entirely ON-demand on Force.com Platform.

When the developer writes and saves Apex code to the platform

→ First the platform application servers compiles the code into abstract set of instructions that can be understood by Apex runtime interpreter.

→ the compile code is stored to metadata.

→ when the end users triggers the execution of Apex by clicking button of visualforce page the application servers retrieves the compiled instructions from the metadata and send them to run time interpretes before returning the result.

Object oriented programming (OOPs):-

oop (object oriented programming) is a methodology that provides a way of modularizing a program by creating partitioned memory area for both data and methods that can be used as template for creating copies of such modules (objects) on demand.

unlike procedural programming, here in the oop programming model, programs are organised arround objects and data rather than action and logic.

The main oops principles are

Encapsulation
Inheritance
polymorphism.

Encapsulation :- the wrapping up of data and methods together is called encapsulation. for example, if we take a class, we write the variables and methods inside the class. Thus, class is binding them together. So class is an example for encapsulation.

Inheritance :- It creates new classes from existing classes, so that the new classes will acquire all the features of the existing classes is called Inheritance.

A good example for Inheritance in nature is parents producing the children and children inheriting the qualities of the parents.

polymorphism :- polymorphism represents one form in multiple forms. In programming, we can use a single variable to refer to objects of different types and then, using that variable we call the methods of different objects. Then a method call can perform different tasks depending on the type of the object.

Apex fundamentals :-

Data type :-

→ Data type in the Apex tells about what type of data can be stored.

→ What is the range of the data that can be stored.

i) premitive data types

ii) collections

iii) Enums

i) premitive data types :-

These are the data types which are predefined by the Apex.

→ A premitive data types such as an Integers, Double, Long, Date, Date Time, String, ID & Boolean.

→ All premitive date types are passed by value, not by reference.

→ All Apex variables, whether they are class member variables, are initialized to null. Make sure that we initialize variables to appropriate values before using them.

Apex premitive datatypes include :-

Boolean :- A value that can only be assigned true, false or null.

Eg :- Boolean isActive = false;

Date :- A value that indicates a particular day. Date values contain no information about time. Date values must always be created with a system static method.

Eg :- Date myDate = Date.newInstance(2013, 05, 15);

output is 2012-05-15 00:00:00

Time and DateTime :- These are date types associated with dates and times along with Date data type. the time data types stores times (hours, minutes, second and milliseconds). the Date data types stores dates (year month and day). The Datetime data type stores both dates and times.

Each of these classes has a newInstance method with which we can construct particular date and time values.

Eg :- Time t1 = newInstance(19, 20, 1, 20);

o/p is 19:20:01

→ We can also create dates and times from the current clock.

```
Date my = Datetime.now();
Date t = Date.today();
```

→ The date and time classes also have instance methods for Converting from one format to another.

Eg:- `Time t2 = Datetime.now().time();`

→ We can also manipulate the values by using a range of instance methods.

Eg:- `Date t3 = Date.today(0);`

`Date Next = t3.addDays(30);`

We will get something like this as the output.

2013-05-15 00:00:00

2013-06-16 00:00:00

Integer, Long, Double and Decimal :- To store numeric values in variables, declare variables with one of the numeric data types. Integer, Long, Double and Decimal.

Integer :- A 32-bit number that doesn't include a decimal point. Integers have a minimum value of -2,147,483,648 and a maximum value of 2,147,483,647.

Eg:- `Integer i=1;`

Class:-

Class is a collection of datamembers and methods.

Eg:-
```
class Student
{
 Integer no;
 String name;
public void getDetails()
 {
 System.debug('rollno' +no);
 System.debug('name' +name);
    }
}
```

→ these are datamembers of the class.

→ This is the method of the class.

Eg2:-
```
class Employee
{
  Integer exp;
  String department;

  void show()
  {
    //Write the logic
  }
}
```

→ Variables / datamembers of the class.

→ Method of the class.

→ TO define an Apex class specify the following.

i) Access modifiers:-

→ You must use one of the access modifiers for top level class. (public & global)

→ you do not have to use access modifiers in the declaration of inner classes.

ii) optional definition modifiers such as virtual, abstract.

iii) Required : the keyword class followed by the class name.

iv) optional extensions : AND/OR implementation.

Syntax:-

```
private | public | global [virtual | abstract | with sharing |
                                                (none)]
class ClassName [implements InterfaceNameList |
(none)] [extends ClassName | (none)]
{
    // the body of the class.
}
```

Access modifiers :-

1) private :- If you declare a class a private it is only known to the block in which it is declared.
   → By default all the inner classes are private.

2) public :- If you declare class as a public, this class is visible throughout your application and you can access the application anywhere.

3) global :- If you declare a class a global this apex class is visible to all the apex applications in the application or outside the application.

   NOTE :-
   If a method of a class (inner) is declare as global then the top level class also must be declared as global.

4) With Sharing :- If you declare a class as a with sharing, sharing rules given to the current users will be taken into the consideration and the users can access & perform the operations based on the permissions given to him on objects & fields. (field level security, sharing rules).

5) Without Sharing :- If you declare a class as a without sharing then this apex class runs in system mode which means apex code has access to all the objects and fields irrespective of current users sharing rules, field level security, object permissions.

NOTE:-

1) If the class is not declared as With sharing or without sharing then by the class is by default taken as

2) Both inner classes and outer classes can be declared as Withsharing.

3) If innerclass is declared as Withsharing and toplevel class is declared as Without Sharing. then by default entire context will run in Withsharing Context.

4) If a class is not declared as With/without sharing and if this class is called by another class in which Sharing is enforced then both the classes run with Withsharing.

5) Outerclass is declared as With Sharing and inner class is declared as Without Sharing then inner class runs in Without Sharing context only. (Inner classes don't take the ~~properties~~ (sharing properties) from outer class).

6) Virtual :- If a class is declared with keyword virtual then this class can be extended (Inherited) & this class methode can be overridden by using a class called overridden.

7) Abstract :- this class contains Abstract methods.

Eg1:-

```
public class outerclass
{
    //code
    class innerclass
    {
        // Innerclass code
        {
    }
```

Eg2:-

```
public with sharing class sharingclass
{
    //code
    }
```

Eg3:-

```
public without sharing class nosharing
{
    //code
    }
```

Eg4 :-

```
public With sharing class outer
{
    //outer class code
    without sharing class inner
    {
      // Inner class code
        }
}
```

In the above code outer class runs with current users sharing rules. But inner class runs with System context.

Eg5 :-

```
public Without sharing class outer
{
    //outer class code
    With sharing class inner
    {
        //Inner class code
        }
}
```

In this both inner and outer classes runs with current users permissions.

Class Variables :-

The variables in the class should specify the following properties when they are defined.

i) optional: modifiers such as public or final as well as static.

ii) Required : the data type of the variable, such as String or Boolean.

iii) optional : the value of the variable

iv) optional : the name of the variable

Syntax :-

[public | private | protected | global | final] [static]
data_type variable_name

Eg:-

```
private static final Integer MY_INT;
private final Integer i = 1;
```

Class Methods :-

To define a method, specify the following.

i) optional : Modifiers, such as public or protected.

ii) Required : The data type of the value returned by the method, such as String & Integer. use void if the method does not return a value.

iii) Required : A list of input parameters for the method, seperated by commas, each preceded by its data type, and enclosed in parantheses ( ). If

there are no parameters, use a set of empty parentheses.
A method can only have 32 input parameters.

iv) Required : The body of the method, enclosed in braces { }.
All the code for the method, including any local variable declarations, is contained here.

Syntax:-

```
(public|private|protected|global) [override] [static]
    data_type method_name (input parameters)
{
    // the body of the method.
}
```

Eg1:-

```
public static Integer getInt()
{
  return MY_INT;
}
```

Eg2:-

```
public class Example
{
    public Integer show(Integer age)
    {
      System.debug('My age is ' +age);
    }
}
```

Object :-

Object is a instance of the class. This has both State and behaviour.

→ Memory for the datamembers are allocated only when you create a object.

Syntax:-

```
classname objectname = new classname();
```

- classname → This the name of class for which we are creating an object.
- objectname → It's a reference variable
- new → It is a keyword which we are allocating the memory
- classname() → Constructor.

Eg:-

```
class Example
{
    //code
};
Example e = new example();
```

Constructor :-

Constructor is a special method which have the following properties.

i) Method name will be same as class.

ii) Accessspecifier will be public.

iii) This method will invoked only once that is at the time of creating an object.

iv) This is used to instantiate the data membess of the class.

Eg:-

```
public class TestObject
{
  //the no argument constructor
   public TestObject()
   {
      //code
   }
}
```

There are 3 types of constructors.

1) Default Constructor
2) Non-parameterized Constructor
3) parameterized Constructor

1) Default Constructor:-

If an Apex class have doesn't Contain any constructor then Apex Compiler by default creates a dummy constructor on the name of class when we create an object for the class.

Eg:-

```
public class Example
{

}

Example e = new Example();
```

In the above example the apex class doesn't contain any Constructor. so when we create object for Example class the Apex Compiler creates a default Constructor.

Eg:-

```
public example()
{
}
```

2) Non-parameterized constructor & parameterized Constructor :-

It is a constructor that doesn't have any parameters, or constructor that has parameters.

Eg:-

```
public class Example
{
    Integer rno;
    String Name;
    public Example(Integer x, String myname)
    {
        rno = x;
        name = myname;
    }
    public Example()
    {
        //code r.no=10;
        name=sam;
    }
}
```

→ parameterized Constructor

→ this is non-parameterized Constructor

Write a Apex program to demonstrate usage of Constructor.

1) open developer console under by clicking the name on the salesforce page.
2) click File & select Apex class.
3) Enter the class name.
4) Write the Apex class.

Eg:-

```
Public class Employee
{
  String EmployeeName;
  Integer EmployeeNO;
  Public Employee()
  {
   EmployeeName = 'Hari';
   EmployeeNO = 10;
  }
  Public void Show()
  {
    System.debug('EmployeeName is '+EmployeeName);
    System.debug('EmployeeNO is ' + EmployeeNO);
  }
}
```

5) open the anonymous block.

```
Employee e1 = new Employee();
Employee e2 = new Employee();
  e1.show();
  e2.show();
```

this will give an output of Employee Name is Hari and EmployeeNO is 10.

EmployeeName is Hari.
EmployeeNo 10

usage of Apex program with within visualforce page :-

1) When you want to use call Apex class in visualforce page we have to declare in the following format.

```
<Apex:page Controller = "class name" >
```

Whenever we call a visualforce page in which controller attribute is defined it will first create an object for the apex class which is defined in Controller.

2) When object is created for the Apex class first it invokes the Constructor.

Referring to the apex class members in visualforce :-

When you want to refer apex class variables in the visualforce page we need to use getters & setters methods.

get()

```
public class Example
{
  String name;
}
```

get method :-

When visualforce page want to get the value of a variable defined in the Apex. It will invoke get method of that variable.

Eg:-

```
<apex:outputlabel > {!myname} </apex:outputlabel>
```

↓

this is a variable defined in apex class.

In the above statement visualforce page is trying to use myname variable which is declared in Apex class. so it is invoke automatically getMyname( ) method in the apex class and this method will return the value of that.

```
public class Example
{
    String name ;
public void set(String name)
    {
    this.name = name;
    }
public String getName()
{
  return name;
}
}
```

→ setter method. this will take the value from the visualforce page and stores to the Apex variable name.

→ getter method. this method will return a value to a visualforce page whenever a name variable is called.

Eg:-

```
public class Example1
{
  Integer no;
  public void set(String Integer no)
    {
      this.no = no;
    }
  public Integer getNo()
  {
    return no;
  }
}
```

1) Write an example for getter method using visualforce and apex class:-

Example class:-

```
public class Example
{
    String name;

  public String getName()
   {
     return 'Sam';
    }
}
```

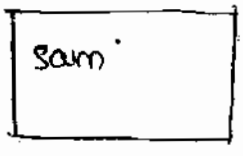
Example page:-

```
<apex:page controller="Example">
<apex:outputlabel>{!name}</apex:outputlabel>
</apex:page>
```

→ In the above program when the page is loaded first it creates an object for the example class.

→ When outputlabel calls label in the visu {!name} in the VF page it invokes getName() method in the controller class. which will return "Sam".

output:-



2) Apex class Example :-

① When visualforce page is loaded it creates an object for the Example class.

```
public class Example
 {
   String name;
 public Example()
 {
   name = 'Hari';
 }
 public voiString getName()
 {
    return name;
 }
}
```

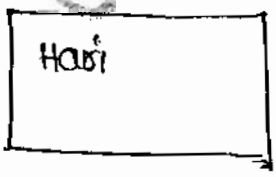
② At the time of creating the object it first calls the constructor and assign name = 'Hari'.

③ When visualforce page called {!name} it invokes getName() in the apex class.

Example page :-

```
<apex:page controller="Example">
<apex:outputlabel> Your name is {!name} </apex:outputlabel>
</apex:page>
```

Output :-



2) Writing the values into Apex variables from visualforce page :-

This is called read/write operation on the variable.

Eg:- {!age}

```
public void setAge(Integer age)
    {
      this.age = age;
    }
```

Eg:- {!name}

```
public void setName(String name)
    {
      this.name = name;
    }
```

Write an apex class to return the value to visualforce page :-

Apex class :-

```
public class Example1
{
  public Integer age;
  public Example1()
    {
      age = 10;
    }
```

```
public Integer getAge()
    {
     return age;
    }
Public String getName()
    {
     return 'samkumar';
    }
}
```

Vf page:-

```
<apex:page controller="Example1">
  <apex:outputLabel> {!age} </apex:outputLabel>
  <apex:outputLabel> {!name} </apex:outputLabel>
</apex:page>
```

Write an apex class to demonstrate setter method i.e, Passing the values and saving the values to apex variables:-

Apex class:-

```
public class Example1
    {
     public String name;
     Public String getName()
        {
         return name;
        }
```

```
Public void setName(String name)
        {
        this.name=name;
          }
        }
```

Vf page :-

```
<apex:page controller="Example1">
  <apex:form>
    <apex:outputlabel> Enter Name </apex:outputlabel>
    <apex:inputText value="{!name}"/>
  <apex:commandButton value="click" reRender="one"/>
  <apex:outputlabel id="one"> your name is {!name}
                              </apex:outputlabel>
  </apex:form>
</apex:page>
```

→ We can also define setter and getter methods in a single line.

```
Public Integer{set;get;}
```

How to call the apex methods in a Visualforce page :-

```
public class Demo
  {
  public pageReference show()
    {
    return null; // when we give return null it will come
      }          back to the same page.
  }
```

```
<apex:commandButton value="click" action="{!show}"/>
```

→ When we click on the "click" button it will invoke pageReference show() method.

→ pageReference is the return type of every method that we have called from visualforce page.

```
Public class Example1
  {
    public String name;
    public String getName()
      {
       return name;
        }
```

```
public void setName(String name)
    {
      this.name = name;
    }
public pageReference show()
    {
      name = 'this is my name' +name;
      return null;
    }
}
```

```
<apex:page Controller="Example1">
 <apex:form>
   <apex:outputLabel> Enter Name </apex:outputLabel>
   <apex:inputText value="{!name}"/>
   <apex:CommandButton value="click" rerender="one"
                       action="{!show}"/>
   <apex:outputLabel id="one">{!name}</apex:outputLabel>

 </apex:form>
</apex:page>
```

Eg:- Simple apex class to perform addition and subtraction based on the button you have clicked.

```
public class Example1
{
 public Integer bvalue { get; set; }
 public Integer avalue { get; set; }
 public Integer result { get; set; }
 public String operation { get; set; }

 public pageReference subb()
    {
    result = avalue - bvalue;
    Operation = 'SUBTRACTION';
    return null;
    }

 public pageReference subb()
    {
    result = avalue + bvalue;
    Operation = 'ADDITION';
    return null;
    }
}
```

```
<apex:page controller="Example1">
  <apex:form>
  <apex:pageBlock title="calculator">
   <apex:pageBlockSection columns="1" title="Simple operations"
                          collapsible="false">
  <apex:pageBlockSectionItem>
   <apex:outputLabel> Enter A value </apex:outputlabel>
  <apex:inputText value="{!avalue}"/>
  </apex:pageBlockSectionItem>
  <apex:pageBlockSectionItem>
    <apex:outputlabel> Enter B value </apex:outputlabel>
     <apex:inputText value="{!bvalue}"/>
  </apex:pageBlockSectionItem>
  <apex:pageBlockSectionItem>
    <apex:outputLabel> you have performed {!operation}
          of {!avalue} and {!bvalue} and the result is
              {!result} </apex:outputlabel>
    </apex:pageBlockSectionItem>
  </apex:pageblockSection>
  </apex:pageBlock>
 </apex:form>
</apex:page>
```

Array :-

Array is a collection of similar elements, where the memory is allocated sequently.

```
DataType[] arrayname = new DataType[size]; //This is called
                                    dynamic declaration.
DataType[] array name = new DataType[] { value1, value2 }; //
                                    Static declaration

Integer[] marks = new Integer[]{10,20,30};

Account a1 = new Account(name='Sam');
Account a2 = new Account(name='ram');
Account[] acc = new Account[]{a1,a2};
String[] sl = new String[]{'ram','sam','ram'};
String[] sl = new String[4];
  sl[0] = 'kumar';
  sl[1] = 'Ravi';
```

Q:- Write a program to display array of Strings in pageBlockTable

Ans:-

```
Public class ArrayExample {
```

```
Public String[] myval {set; get;}

Public String name {get; set;}

public ArrayExample()
    {
    name = 'pravad';
    myval = new String[] {'sam', 'ram', 'kiran'};
    }
}
```

```
<apex:page Controller = "ArrayExample">
  <apex:form>
   <apex:pageBlock>
    <apex:pageBlockTable value = "{! myval }" var = "a">
      <apex: column value = "{! a }"/>
    </apex:pageBlockTable>
   <apex:outputLabel> {! name } </apex:outputLabel>
  </apex:pageBlock>
  </apex:form>
</apex:page>
```

Q:- Write a program to display array of account records.

Ans:-

```
public class ArrayExample {
public Account[] myVal{set;get;}
public ArrayExample()
    {
Account a1=new Account(name='sashi',
                       Industry='Banking');
Account a2=new Account(name='Ravi', Industry='Banking');
Account a3=new Account(name='praveen', Industry='Banking');
  myval=new Account[]{a1,a2,a3};
    }
}
```

```
<apex:page controller="ArrayExample">
<apex:form>
 <apex:pageBlock>
  <apex:pageBlockTable value="{!myval}" var="a">
  <apex:column value="{!a.name}"/>
  <apex:column value="{!a.industry}"/>
 </apex:pageBlockTable>
```

```
<apex:outputLabel>{!name} </apex:outputLabel>
</apex:pageBlock>
</apex:form>
</apex:page>
```

Picklist using selectoption from Apex :-

1)
```
<apex:selectList size="1">
    <apex:selectoption itemLabel="Java" itemValue="SCJP"/>
    <apex:selectoption itemLabel="SF" itemValue="SDFC"/>
</apex:selectList>
```

In the apex program we can create same selectoption using a object selectoption.

```
Selectoption op1 = new selectoption(itemvalue, itemlabel)
```

```
public class SelectExample {
  public selectoption[] myoptions {set; get;}
  public SelectExample()
      {
        selectoption op3 = new selectOption('null', '-None-');
        Selectoption op1 = new Selectoption('one', 'Jan');
```

```
selectoption op2 = new selectoption('two', 'feb');
myoptions = new selectoption[]{op3,op1,op2};
    }
}
```

```
<apex:page controller="SelectExample">
 <apex:form>
  <apex:selectlist size="1">
   <apex:selectoptions value="{!myoptions}">
                    </apex:selectoptions>
  </apex:selectlist>
 <apex:selectlist size="1">
   <apex:selectoption itemlabel="Java" itemvalue="Java">
                                    </apex:selectoption>
   <apex:selectoption itemlabel="SFDC" itemvalue="SF">
                                        </apex:selectoption>
 </apex:selectlist>
 </apex:form>
</apex:page>
```

Collections :-

Difference between Array and collections.

Array	collections.
1. Array is a collection of homogeneous (similar) elements.	1. It is a collection of homogeneous & hetrogeneous elements.
2. Arrays can not grow and shrink dynamically.	2. It can grow and shrink dynamically.
3. Arrays can be accessed faster and less memory.	3. collections are slow compared to arrays & consume more memory.

List :-

→ List is an interface.

→ A list is an ordered collection of elements that are distinguished by their indices. ie.

→ List elements can be of any datatype - primitive types, collections, objects, user-defined types and built-in apex types.

Index 0	Index 1	Index 2	Index 3	Index 4
Green	Blue	yellow	Red	Black

→ Insertion order is preserved.

→ can grow dynamically at run time.

→ Duplicate values are allowed.

→ Null values are accepted.

Methods in List class :-

add(Object) :- Adds an element to the end of the list.

add (Integer, Object) :- Inserts an element into the list at the specified index position.

addAll (List) :- Adds all of the elements in the specified list to the list that calls the method. Both lists must be of the same type.

addAll(Set) :- Add all of the elements in specified Set to the list that calls the method. The Set and the list must be of the same type.

clear() :- Removes all elements from a list, Consequently setting the list's length to zero.

clone() :- Makes a duplicate copy of a list.

deepClone (Boolean, Boolean, Boolean) :- Makes a duplicate copy of a list of sObject records, including the sObject records

themselves.

equals(List) :- Compares this list with the Specified list and returns true if both lists are equal. otherwise returns false.

get(Integer) :- Returns the list element Stored at the specified index.

getsObjectType() :- Returns the token of the sObject type that makes up a list of sObjects.

hashCode() :- Returns the hashcode corresponding to this list and its contents.

isEmpty() :- Returns true if the list has zero elements.

iterator() :- Returns an instance of an iterator for this list.

remove(Integer) :- Removes the list element Stored at the specified index, returning the element that was removed.

set(Integer, object) :- sets the specified value for the element at the given index.

size() :- Returns the number of elements in the list.

sort() :- sorts the items in the list in ascending order.

Eg:- List<String> str = new List<String>();

String s1='sam';

String s2 = 'Ram';

String s3 = 'Ravi';

str.add(s1);

Str.add(s2);

Str.add(1,s3);

List<String> finalist = new List<String>();

Finallist.addAll(str);

String x = Str.get(1); // Ravi

→ Write a program to demonstrate the list.

Public class ListExample {

public List<String> result{set;get;}

public ListExample()

{

result = new List<String>();

result.add('sam');

result.add('ram');

result.add('hari');

result.add(1,'kumar');

}

}

```
<apex:page controller="ListExample">
    <apex:pageblock>
     <apex:pageblocktable value="{!result}" var="a">
        <apex:column value="{!a}"/>
     </apex:pageblocktable>
    </apex:pageblock>
</apex:page>
```

List of Objects Demo :-

```
Public class ListExample{
   public list<Account> result {set; get;}
    Public ListExample()
      {
    Account a1 = new Account(name='sam' Industry='Banking');
    Account a2 = new Account( name='ram' Industry='Energy');
        result = new list<Account>();
         result.add(a1);
         result.add(a2);
        }
}
```

```
<apex:page Controller = "ListExample">
  <apex:pageBlock>
    <apex:pageBlockTable value = "{!result}" var="a">
    <apex:column value = "{!a.name}"/>
  </apex:pageBlockTable>
  </apex:pageBlock>
</apex:page>
```

Create a list of Apex class objects:-

1. Student Apex:-

```
global class Student
    {
    public String name {get; set;}
    public Integer age{get; set;}
    public Student(String name, Integer age)
        {
        this.name = name;
        this.age = age;
            }
    }
```

2. ListExample Apex :-

```
public class ListExample
{
public List<Student> result {set; get;}
public ListExample()
{
result = new List<Student>();
Student S1 = new Student('sam', 20);
Student S2 = new Student('Ram', 40);
Student S3 = new Student('praveen', 40);
result.add(S1);
result.add(S2);
result.add(S3);
}
}
```

ListExample Visualforce :-

```
<apex:page Controller = "ListExample">
<apex:pageblock>
<apex:pageblockTable value = "{!result}" var = "a">
<apex:column value = "{!a.name}"/>
```

```
<apex:column value="{!a.age}"/>
</apex:pageBlockTable>
</apex:pageBlock>
</apex:page>
```

Write an apex program to generate list of selectoptions:-

```
public class ListSelect
{
public List String myval{get;set;}
public List<selectoption> myoptions;
public List<selectoption> getMyoptions()
{
return myoptions;
}
public ListSelect()
{
myoptions = new List<selectoption>();
selectoption s = new selectoption('null','-None-');
selectoption s1 = new selectoption('one','Jan');
myoption.add(s);
myoptions.add(s1);
```

```
myoptions.add(new selectoption('two','Feb');
myoptions.add(new selectoption('three','Mar');
        }
    }

<apex:page Controller="ListSelect">
  <apex:form>
   <apex:selectList size="1" value="{!myval}">
   <apex:selectoptions value="{!myoptions}"/>
    <apex:actionSupport event="onchange" reRender="one"/>
  </apex:selectList>
  <apex:outputLabel id="one"> You have Selected {!myval}
                    </apex:outputLabel>
  </apex:form>
  </apex:page>
```

Set :-

It is a unordered collection of elements where elements can be of any data type. primitive types, collections, sobjects, user-defined types, and built-in Apex types.

1. set don't allow the duplicate values.
2. Insertion order is not preserved in the set
3. It grow dynamically at run time.

Eg :-

```
Set<String> names = new Set<String>();
Set<Account> acc = new Set<Account>();
Set<Customers__c> mycustomers = new Set<Customers__c>();
```

public void add(Object) // this method will add elements to the set.

```
Set<String> names = new Set<String>();
names.add('one');
names.add('two');
names.add('one');
```

NOTE :- set will not allow duplicates, but if we insert it will not raise any error it will not takes value.

public void addAll(List) // this method will add all elements to the set.

```
List<String> mylist =new List<String>();
    mylist.add('one');
    mylist.add('two');
set<String> mynames = new Set<String>();
  mynames.add('sam');
  mynames.addAll(mylist);
```

public Integer size() // This method will return no. of elements in the set.

```
set<String> names =new set<String>();
    names.add('one');
    names.add('two');
    names.add('one');
Integer mySize = names.size(); // this will return 3 and
                                  Stored to my size variable.
```

Public void remove(Integer index) // This method will remove the elements at the specified index.

```
names.remove(1);
```

Example:- write an apex class to demonstrate set.

```
public class SetExample
{
public Set<String> names{get;set;}
public SetExample()
 {
names = new Set<String>();
 names.add('one');
 names.add('two');
 names.add('sam');
 names.add('one');
 names.add('one');
    }
}
```

vf page:-

```
<apex:page Controllles = "setExample">
  <apex:pageBlock>
  <apex:pageBlockTable value = "{!names}" var = "a">
  <apex:column value ="{!a}"/>
  </apex:pageBlockTable>
   </apex:pageBlock>
</apex:page>
```

→ Write an apex class to add list of elements to set

```
public class SetExample
    {
public set<String> names{get;set;}
 public SetExample()
  {
List<String> mylist = new List<String>();
  mylist.add('sashi');
  mylist.add('sashi');
  mylist.add('rakesh');
  mylist.add('rakesh');
  names = new Set<String>();
  names.add('one');
  names.add('two');
  names.add('one');
  names.add('one');
  names.add(mylist);
    }
}
```

public void RetainAll(List) // this will keep only the values that are existing in list and set.

Remove rest of the values.

```
public class SetExample
 {
public Set<String> names{get;set;}
 public boolean test{get;set;}
  public SetExample()
    {
 List<String> mylist = new List<String>();
  mylist.add('sashi');
  mylist.add('rakesh');
  names = new Set<String>();
    names.add('sashi');
    names.add('ram');
    names.add('kumar');
    test =names.retainAll(mylist);
    }
}
```

Visualforce page :-

```
<apex:page Controller="sctExample">
  <apex:pageBlock>
  <apex:outputLabel> {!test} </apex:outputLabel>
   <apex:dataList value="{!names}" var="a"> {!a}
   </apex:dataList>
  </apex:pageBlock>
  </apex:page>
```

Map :-

A map is a collection of key-value pairs where each unique key maps to a single value. Keys and values can be any datatype - premitive types, collections, sObjects, user-defined types, and built-in Apex types.

For Eg, the following table represents a map of Countries and currencies.

Country (key)	'United States'	'Japan'	'France'	'England'	'India'
Currency (value)	Dollar	'Yen'	'Euro'	'pound'	'Rupee'

clear() :- Remove all of the key-value mappings from the map.

clone() :- Makes a duplicate copy of the map.

containsKey(Object) :- Returns true if the map contains a mapping for the specified key.

deepClone() :- Makes a duplicate copy of a map, including sObject records if this is a map with sObject record values.

equals(Map) :- Compares this map with the specified map & returns true if both maps are equal. Otherwise returns false.

get(Object) :- Returns the value to which the specified key is mapped, or null if the map contains no value for this key.

getSObjectType() :- Returns the token of the sObject type that makes up the map values.

hashCode() :- Returns the hashCode corresponding to this map.

isEmpty() :- Returns true if the map has zero key-value pairs.

keySet() :- Returns a set that contains all of the keys in the map.

put(Object, Object) :- Associates the specified value with the specified key in the map.

putAll(map) :- copies all of the mappings from the specified map to the original map.

putAll(sObject[]) :- Adds the list of sObject records to a map declared as Map<ID, sObject> & Map<String, sObject>

remove(key) :- Removes the mapping for the specified key from the map, if present, and returns the corresponding value.

size() :- Returns the no. of key-value pairs in the map.

values() :- Returns a list that contains all of the values in the map in arbitary order.

Set of key: key can not be duplicate

List of values: It allows duplicates.

```
Map<Integer,String> my = new Map<Integer,String>();
    set<Integer> keys = list <String> values
     void put (key,value)
       m.put (1,'sam');
       m.put (2,'Ram');
       m.put (3,'kumar');
     {1 → 'sam', 2 → 'Ram', 3 → 'kumar'}
```

→ Object get(key) when we give key it will return values.

```
String x = m.get(1) => sam
String y = m.get(3) => kumar.
```

→ Set<Object> keySet():- If we use this method we will get set of keys.

```
Set<Integer> s = m.keySet(); => {1,2,3}
```

→ List<Object> values():- It will return list of values in the map.

```
List<String> val = m.values(); => {'sam','ram','kumar'}
```

```
public List<String> myresult {get; set;}

  Map<String, String> mymap;

 public MapExample()
     {
   myresult = new List<String>();
   mymap = new Map<String, String>();
    mymap.put ('Hyd', 'satish');
    mymap. put ('warangal', 'Reddy');
    mymap.put ('kasimnagar', 'chaitu');
         }

public pagereference show() {
  myresult.clear();
  set<String> keys = mymap.keyset();
   myresult.addAll (keys);
     return null;
      }

Public pagereference display()
   {
    myresult = mymap.values();
     return null;
      }
   }
```

```
Map<String, List<String>> mybranches = new Map<String,
                                        List<String>>();

List<String> hyd = new List<String>();
    Hyd.add('SR');
    Hyd.add('Jb');
   Mybranches.put('Hyd', Hyd);

List<String> war = new List<String>();
     Hyd.add('Kazi');
     Hyd.add('Bhalampali');
     Mybranches.put('War', war);

    Mybranches.put('War', war);

 List<String> adi = new List<String>();
     Adi.add('Manchiyal');
    Adi.add('godavari');
    Mybranches.add('adi',adi);

 Set<String> keys = Mybranches.keyset();
     For(String s: keys)
         {
  selectoption op = new selectoption(s,s);
         }
  List<String> val = mybranches.get(city);
```

```
Public class DependExample{

    public pageReference show(){

      List<String> bran = mybrances.get(mycity);
        branch.clear();

      for(String x:bran)
        {
        selectoption opl = new Selectoption(x,x);
         branch.add(opl);
           }
        return null;
        }
```

Apex class:-

```
Public String mycity{get;set;}

    Map<String, List<String>> mybranches = new Map<String,
                                          List<string>>();
    Public List<Selectoption> city{set;get;}
    Public List<Selectoption> branch{set;get;}

     Public DependExample()
       {
    List<String> hyd = new List<String>();
      hyd.add('srnagar');
      hyd.add('lbnagar');
```

```
List<String> bang = new List<String>();
    bang.add('Ecity');
    bang.add('Marthali');
  mybranches.put('Hyd', hyd);
  mybranches.put('Bang', bang);
  set<String> keys = mybranches.keySet();
  city = new List<Selectoption>();
  branch = new List<Selectoption>();
   city.add(new Selectoption('null', 'None'));
  List<String> my = new List<String>();
   my.add('None');
   for(String a: keys)
      {
    Selectoption op1 = new Selectoption(a,a);
      city.add(op1);
      }
   }
  }
```

Vf page :-

```
<apex:page Controller="DependExample">
  <apex:form>
```

```
<apex:selectList value="{!mycity}" size="1">
 <apex:selectOptions value="{!city}"></apex:selectOptions>
<apex:actionSupport event="onchange" action="{!show}"
                    rerender="one"/>
 </apex:selectList>
 <apex:selectList size="1" id="one">
  <apex:selectOptions value="{!branch}">
  </apex:selectOptions>
 </apex:selectList>
 </apex:form>
</apex:page>
```

Example :-

Apex class :-

```
public class ListsMapsControllers {
 public Map<String, String> inputFields {get; set;}
    public ListsMapsControllers() {
    inputFields = new Map<String, String>() {
    'firstName' => 'Jonny', 'lastName' => 'Appleseed', 'age' =>
                    '42' };
 }
```

```
public pageReference SubmitFieldData() {
        doSomethingInterestingWithInput();
        return null;
    }
public void doSomethingInterestingWithInput()
    {
    inputFields.put('age', (Integer.valueOf(inputFields.get('age'))
                                +10).format());
        }
    }
```

```
<apex:page Controller = "ListMapsController">
  <apex:outputpanel id = "box" layout = "block">
    <apex:pageMessage/>
  <apex:form>
    <apex:repeat value = "{!inputFields}" var = "fieldkey">
      <apex:outputText value = "{!fieldkey}"/>
  <apex:inputText value = "{!inputfields[fieldkey]}"/><br/>
    </apex:repeat>
  <apex:CommandButton action = "{!submitfieldData}"
      value = "Submit" id = "button" rerender = "box"/>
```

```
</apex:form>
</apex:outputpanel>
</apex:page>
```

SOQL Queries :-

Salesforce object Query language is used to query the records from the database.com based on the requirement.

There are 2 types of SOQL Statements.

1) Static SOQL
2) Dynamic SOQL.

1) **Static SOQL** :- The static SOQL statement is written in [ ] (array brackets).

→ This statements are similar to LINQ.

Eg:-
```
String searchfor = 'Jones';
Contact[] Contacts = [select testfield__c, firstName,
                      lastName from Contact where
                      lastname = :searchfor];
```

2) **Dynamic SOQL** :- It is used to refer to the creation of a SOQL String at run time with Apex code.

→ Dynamic SOQL enables you to create more flexible application.

→ To create Dynamic SOQL query at run time use Database.Query() method in one of the following ways.

→ Return a single sObject when the query returns a single record.

sObject s = Database.query(String_limit-1);

→ return a list of sObjects when the query returns more than a single record.

Examples :-

Eg1:-

```
String myTestString = 'TestName';
List<sObject> L = Database.Query('SELECT Id FROM
                  MYCustomObject__C WHERE Name=:myTestString');
```

Eg2:-

```
String resolvedField1 = myVariable.field1__c;
List<sObject> L = Database.Query('SELECT Id From MyCustomObject__C
                  WHERE field1__C = ' + resolvedField1);
```

Syntax of SOQL query:-

```
SELECT field1, field2.....
FROM ObjectType
[WHERE condition]
```

Examples:-

```
SELECT Id, Name
List<Account> acc = [select Id, name from Account];
List<Account> acc = [select id, name from Account where
                     annual_revenue < 10000];
```

Q) Write a query to fetch Customers Name, Balance from customers object where balance is more than 100000.

```
List<Customers_C> Customerss = [select Id, customer_Name_C,
                                Balance_C from Customer_C
                                where Balance >= _C > 100000];
```

Q) Write a query to fetch TID, type_C from Transaction if mode is 'cash'.

```
List<Transaction__C> tra = [select Name, type__C from
                            Transaction__C where
                            Mode_C = 'cash'];
```

Querying Multiselect picklist values :-

When you want to write a query with a condition based on multiselect picklist in which meltiple item values are selected then we can use the following operators to write the condition.

NOTE:- First letus create proof field with datatype multiselect with the values

i) passport
ii) Aadhaar
iii) Voter Id
iv) PAN card

Q) write a query to fetch all the customers names & account types who has submitted PAN card as a proof.

```
List<customer__c> customers = [select id, customer_name__c,
                               Account_Type__c from customer__c where
                               proof__c = 'pan card'];
```

Q) Write a query to fetch customers name, balance from customers object whose proof is pan card or Aadhar card.

NOTE:- when you want to verify whether the value what we have selected is in the list or not by using INCLUDES.

```
List<Customer__c> Customers = [select Customer_Name__c,
                               Balance__c from Customer__c where
                               proof__c INCLUDES ('Aadharcard','Pancard')]
```

Q) Write a query to fetch the list of Ids & customer names from customer object where proof is Aadhar card or passport and voter id.

When you want to verify whether we have selected more than one value then use ';'

```
proof__c =: 'passport;voterid'

List<Customer__c> customers = [select Customer_Name__c,
                               Balance__c from customer__c where
                               Proof__c INCLUDES ('Aadharcard','passport;voterid')];
```

LIKE :-

Expression is true if the value spin the specified field name matches the characters of the text string in the specified value.

→ The LIKE operator in SOQL and SOSL is similar to the LIKE operator in SQL.

It provides a mechanism for matching partial text strings and includes support for wildcards.

→ the % and _ wildcards are supported for the LIKE operator.

→ The % wildcard matches zero or more characters.

→ The _ wildcard matches exactly one character.

→ The text string in the specified value must be enclosed in single quotes.

→ The LIKE operator is supported for string fields only.

→ The LIKE operator performs a case insensitive match, unlike case sensitive match in SOQL.

Eg:- SELECT AccountId, FirstName, LastName
FROM Contact
WHERE LastName LIKE 'app1_%'

IN :-

If the value equals any one of the specified values in a WHERE clause.

Eg:- SELECT Name FROM Account
WHERE BillingState IN ('California', 'NewYork')

NOT

NOT IN :-

If the value does not equal any of the specified values in a WHERE clause. For Eg

Eg:- SELECT Name FROM Account

WHERE Billingstate NOT IN ('California', 'NewYork')

Subquery :-

The query with in a query is called subquery.

Eg:- Write a query to return list of Accounts that do not have any open opportunities.

```
SELECT Id
FROM Account
WHERE Id NOT IN
(
    SELECT AccountId  --> Account Id is a lookup field
    FROM Opportunity      in the opportunity.
    WHERE Isclosed = false
)
```

Eg2:- SEL

Eg2:- SELECT Id

FROM Opportunity

WHERE AccountId NOT IN

(

SELECT AccountId

FROM Contact

WHERE LeadSource = 'Web'

)

NOTE:- Whenever we create a master-detail & lookup field id of the master record is stored in this field.

Date formats :-

When you want to use the date formats in query, it should be any one of the below formats.

Format	Format Syntax	Example
Date only	YYYY-MM-DD	1991-01-01
Date, time & time zone offset	YYYY-MM-DDThh:mm:ss+hh:mm ss+hh Eg:- 1991-01-01T23:01:01+01:00	
	YYYY-MM-DDThh:mm:ss+hh:mm	

LIMIT :-

Use LIMIT to specify the maximum numbers of rows to return.

Syntax :-

```
SELECT fieldList
    FROM objectType
    [WHERE conditionExpression]
     LIMIT number_of_rows
```

Eg :-

```
SELECT Name
    FROM Account
    WHERE Industry = 'Media' LIMIT 25
```

NOTE :- You can't use a LIMIT clause in a query that uses an aggregate function, but does not use a GROUP BY clause. for example, the following query is invalid.

```
SELECT MAX(createdDate)
 FROM Account LIMIT 1
```

OFFSET :-

Use OFFSET to specify the starting row offset into the result set returned by your query.

—> using OFFSET is helpful for paging into large result sets, in scenarios where you need to quickly jump to a particular subset of the entire results.

Syntax:-

```
SELECT fieldList
FROM objectType
[WHERE conditionExpression]
ORDER BY fieldOrderByList
LIMIT number_of_rows_to_return
OFFSET number_of_rows_to_skip
```

Eg:-

```
SELECT Name
FROM Merchandise__c
WHERE price__c > 5.0
ORDER BY Name
LIMIT 100
OFFSET 10
```

GROUP BY:-

With API version 18.0 and later, you can use GROUP BY with aggregate functions, such as SUM() & MAX(), to summarize the data and enable you to rollup query results rather than having to process the individual records in your code.

Syntax:-

```
[GROUP BY fieldGroupByList]
```

Eg1:-

```
SELECT LeadSource FROM Lead.
```

Eg2:-

```
SELECT Leadsource, COUNT(Name)
FROM Lead
GROUP By Leadsource.
```

Eg3:-

```
SELECT LeadSource
FROM Lead
GROUPB BY LeadSource.
```

NOTE:- You must use a GROUP By clause if your query uses a LIMIT clause and an aggregated function.

Eg:-

```
SELECT Name, Max(CreatedDate)
FROM Account
GROUP By Name
LIMIT 5
```

GROUP BY ROLLUP :- this allows the query to calculate subtotals so you don't have to maintain that logic in your code.

Syntax:-

```
[GROUP BY ROLLUP (fieldName1, fieldName2, fieldName3])]
```

Eg1:- Rolls the results up by one field

```
SELECT leadSource, COUNT(Name) cnt
FROM Lead
GROUP BY ROLLUP(LeadSource)
```

Eg2:- Rolls the results up by two fields

```
SELECT Status, LeadSource, COUNT(Name) cnt
FROM Lead
GROUP BY ROLLUP(Status, LeadSource)
```

GROUP BY CUBE :- this is particular useful if you need to compile cross-tabular reports of your data. use GROUP BY CUBE with aggregate functions, such as SUM() and COUNT(fieldName)

Syntax:-

```
[GROUP BY CUBE (fieldName1, fieldName2, fieldName3])]
```

Eg:- Returns subtotals of accounts for combination of Type & Billing city.

```
SELECT Type, BillingCountry,
  GROUPING(Type) grpType, GROUPING(BillingCountry) grpCty,
  COUNT(Id) accts
  FROM Account
  GROUP BY CUBE (Type, BillingCountry)
  ORDER BY GROUPING(Type), GROUPING(BillingCountry)
```

HAVING :-

With API version 18.0 and later, you can use a HAVING clause with a GROUP BY clause to filter the results returned by aggregate functions, such as SUM().

Syntax:-

[HAVING havingConditionExpression]

Eg1:- Determine how many leads are associated with each leadsource.

```
SELECT LeadSource, COUNT(Name)
FROM Lead
GROUP BY LeadSource
```

Eg2:- Generate more than 100 Leads

```
SELECT LeadSource, COUNT(Name)
FROM Lead
GROUP BY LeadSource
HAVING COUNT(Name) > 100
```

Eg3:- Returns Accounts with duplicate names.

```
SELECT Name, Count(Id)
FROM Account
GROUP BY name
HAVING Count(Id) > 1
```

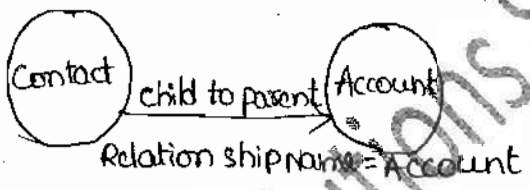
Relationship queries :-

When you want to write a queries based on parent child relation ship will be using relationship queries on standard & custom objects.

→ There are two types of relationship queries.

1) child to parent relationship
2) parent to child relationship.

1) child to parent relationship :-



→ In the above diagram Contact is a child of Account object.

→ In the child to parent relationship relationship name will be parent object which means foriegn key is ~~an~~ ~~are~~ in Account object.

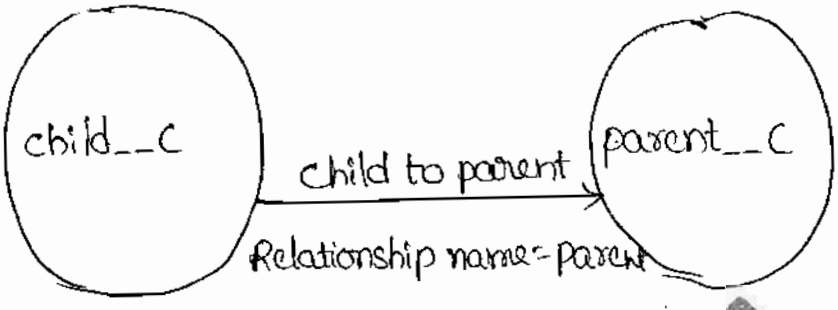
Eg:-

```
SELECT Contact.FirstName, Contact.Account.Name from Contact
```

Eg2:- Write a query to fetch list of contacts and Accountnames from Contact objects whose Account.Industry = 'Media'.

```
SELECT Id, Name, Account.Name FROM Contact
WHERE Account.Industry = 'Media'.
```

Child to parent on custom objects :-

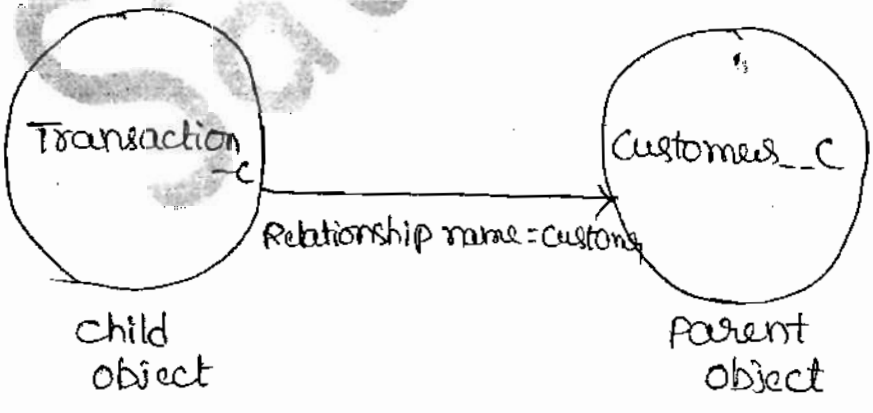


NOTE :- When we use relationship name for the custom objects in SOQL query we should append objectname__r to the relationship name.

Eg1:-

```
List<child__c> ch = [Select Id, Name, parent__r.FirstName,
                     Parent__r.LastName__c from child__c
                     where age__c<25];
```

Eg2:-



NOTE :- When you are writing a query from child to parent relationship always ~~parent object name will be~~ Mas relation name will be most master-detail field name(or) lookup field name.

Q) Write a query to fetch list of transactions along with Account type and customers names whose transaction type is deposit.

NOTE:- Masters-detail field in the transaction is customers details__c

```
List<transactions__c> tx = [select Id, customersDetails__r.
                            CName__c, customersDetails__r.
                            AccountType__c, name from
                            Transaction__c];
```

Write a apex program to display first five contact details from Contact object along with Account Name and industry.

Apex class :-

```
public class childparent {
  public List<Contact> cont { get; set; }
  public pageReference show()
    {
  cont = [select id, Name, Account.Name, Account.Industry
                from Contact Limit 5];
      return null;
       }
}
```

Vf page :-

```
<apex:page Controller = "childparent">
  <apex:form>
    <apex:pageBlock rendered = "{!NOT(ISNULL(Cont))}">
    <apex:pageBlockTable value = "{!cont}" var = "a">
      <apex:column value = "{!a.Name}"/>
      <apex:column value = "{!a.Account.Name}"/>
      <apex:column value = "{!a.Account.Industry}"/>
    </apex:pageBlockTable>
    </apex:pageBlock>
    <apex:CommandButton value = "click" action = "{!show}"/>
  </apex:form>
</apex:page>
```

Apex-I

Long:- A 64 bit numbers that doesn't include a decimal point longs have a minimum value of -263 and a maximum value of 263-1.

Eg:- Long l = 2147483648L;

Double:- A 64-bit numbers that includes a decimal point. Doubles have a minimum value of -263 and a maximum value of 263-1.

Eg:- Doubb d = 3.14159;

Decimal:- A numbers that includes a decimal point. Decimal is an arbitary precision number. Currency fields are automatically assigned the type decimal.

Eg:- Decimal dec = 19.23;

Null variables:- If we declare a variable and don't initialize it with a value, it will be null. Null means the absence of a value. We can also assign a null to any variable declared with a Prermitive type.

Both of these statements result in a variable set to null.

Boolean x = null;

Decimal d;

String:- Strings are set of characters and are enclosed in a single quotes. They store text values such as a name & an address

Eg:- Date d1 = Date today();

String s = String value of (d1);

The o/p of above example should be today's date. 2013-06-16.

SObject Types:-

An SObject, can be a generic SObject or be a specific SObjects, such as an Account, Contact, or MyCustom__c.

→ SObjects (short for "salesforce objects") are standard & custom objects that stores record data in the Force.com database. There is also an SObject data type in Apex that is the programmatic representation of these SObjects and their data in code.

→ Developers refer to SObjects and their fields by their API names.

Eg:- Account a = new Account();

MyCustomObject__c cO = new MyCustomObject__c();

↓

API name of the custom object.

→ the following example creates an invoice Statement with some initial values for the description__c fields and assigns it to variables of type Invoice-statement__c, which is an SObject type also.

Eg:- Invoice-Statement__c inv = new Invoice-Statement__c (Description_c = 'TestInvoice', Status_c = 'pending')

→ SObject variables are initialized to null, but can be assigned a valid object reference with the new operator.

Q:- write a Apex program to display the transaction details along with customers Name, Account type, city using transaction object.

Apex class:-

```
public class childparent
  {
public List<Transaction__c> tran{get;set;}

  public pageReference show()
      {
    tran=[select Type__c, cus customer_Details__r.
          Account_Type__c, Customer_Details__r.Customer_Name__c,
          Customer_Details__r.city__c from Transaction__c];
        return null;
      }
  }
```

Vf page:-

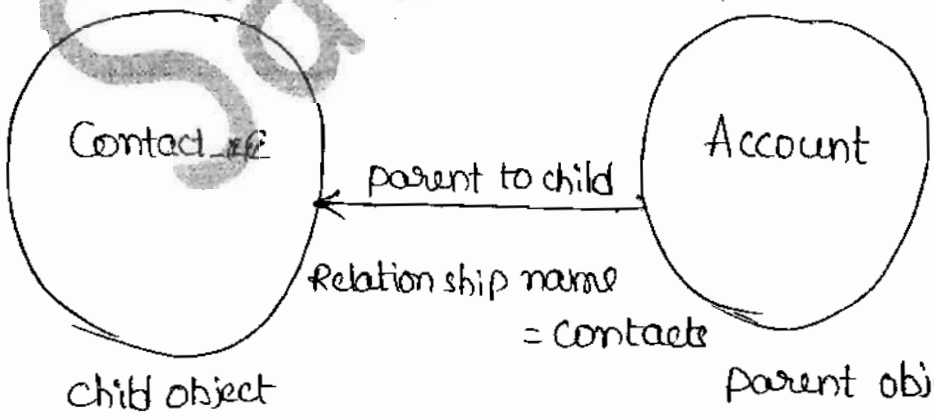
```
<apex:page Controller="childparent">
 <apex:form>
   <apex:CommandButton value="click" action="{!show}"/>
    <apex:pageBlock>
     <apex:pageBlockTable value="{!tran}" var="a">
     <apex:column value="{!a.Customer_Details_r.
                          Customer_Name_c}"/>
```

```
<apex:column value = "{!a.customes_Details__r.Account_Type__c}"/>
<apex:column value = "{!a.customes_Details__r.city__c}"/>
</apex:pageBlockTable>
</apex:pageBlock>
</apex:form>
</apex:page>
```

Parent to child relationship on standard objects :-

When you are trying to write a query to refer the child object records from the parent object then the child object name could be the relationship name.

→ the child object name should be referred from with plural name.



syntax/-

```
SELECT Account.Name, (SELECT Contact.FirstName,
        Contact.LastName FROM Account.Contacts)
FROM Account
```

Eg1 :-

```
SELECT Name,
    (
    SELECT LastName
    FROM Contacts
    )
FROM Account
```

Eg2 :-

```
SELECT Amount, Id, Name,
    (
     SELECT Quantity, Listprice,
    pricebookEntry.unitprice, pricebookEntry.Name
     FROM OpportunityLineItems
    )
FROM opportunity
```

Eg3 :-

```
SELECT Amount, Id, Name, (SELECT Quantity, Listprice,
pricebookEntry.unitprice, pricebookEntry.Name,
PriceBookEntry.product2.Family FROM OpportunityLineItems)
 FROM opportunity
```

Q:- Write an apex program to display list of accounts along with their corresponding Contact details.

Apex class :-

```
public class parentchild
    {
Public List<Account> acc{get;set;}

  Public pageReference show()
       {
        acc=[select Name, Industry, (select AssistantName,
              Email from Contacts) from Account];
        return null;
       }
    }
```

Vf page :-

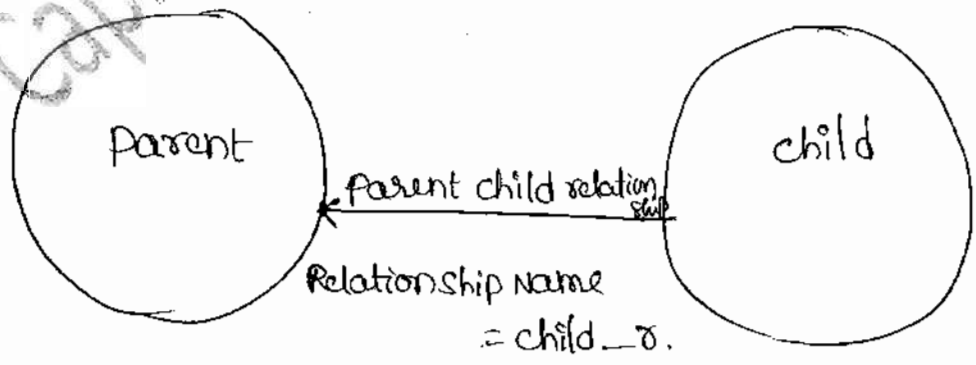
```
<apex:page Controller="parentchild">
  <apex:form>
  <apex:commandButton value="click" action="{!show}"/>
<apex:datalist value="{!acc}" var="a">
```

```
<apex:outputLabel >Account Name:{!a.name} :::
                Account.Industry{!a.industry}</apex:outputLabel>
  <apex:dataList value="{!a.contacts}" var="b">
      {!b.Email}---{!b.AssistantName}
    </apex:dataList>
  </apex:dataList>
 </apex:form>
</apex:page>
```

Q:- 1

Parent to child relationship on custom objects :-

When you are writing a query to refer custom child object fields from the parent child relationship then we have to append child object plural name __r.

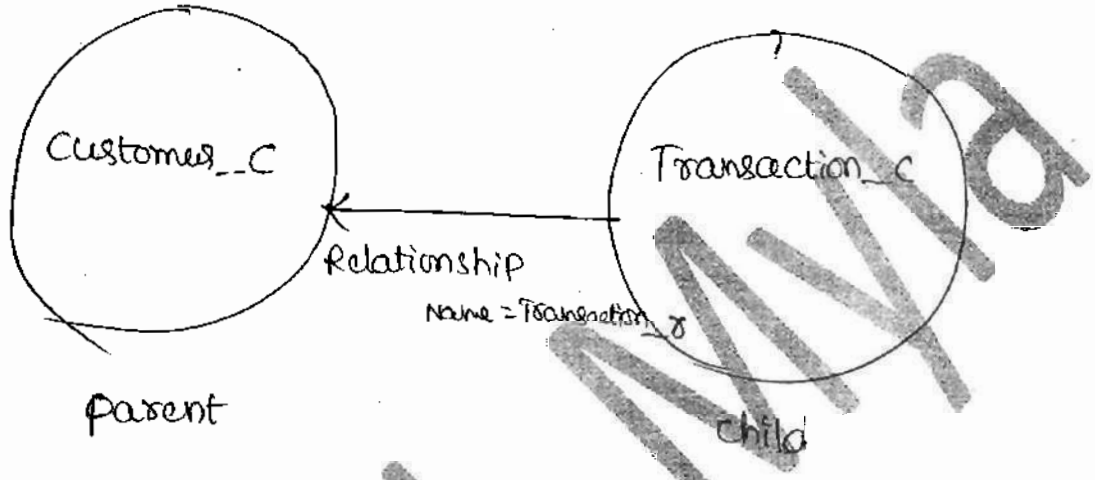


Eg:-

SELECT customer__Name, AccountType, (

[SELECT parentName__c, Age__c, (select child_Name__c,

Dob__c from childs__r) from parent__c]



Q:) Write an Apex program to print customer details along with corresponding transaction details of the customers whose account type is saving.

Apex class:

```
public class parentchild
  {
public  list<customer__c> acc {get;set;}.
   public pageReference show()
        {
     acc=[select customer_name__c, Account_Type__c,
```

```
(select Type__c, Amount__c from Transactions__r)
                from Customers__c)];
        return null;
    }
}
```

```
<apex:page Controller="parentchild">
 <apex:form>
  <apex:commandButton value="Click" action="{!show}"/>
  <apex:pageBlock>
   <apex:pageBlockTable value="{!acc}" var="a">
 <apex:column value="{!a.Customer_Name__c}"/>
 <apex:column value="{!a.Account_Type__c}"/>
 <apex:column>
    <apex:pageBlockTable value="{!a.Transactions__r}"
                         var="b">
    <apex:column>{!b.Type__c}</apex:column>
    <apex:column>{!b.Amount__c}</apex:column>
    </apex:pageBlockTable>
 </apex:column>
</apex:pageBlockTable>
</apex:pageBlock>
```

```
</apex:form>
</apex:page>
```

SOSL Statements :-

SOSL Statements evaluate to a list of SObjects, where each list contains the search results for a particular sObject type. The result lists are always returned in the same order as they were specified in the SOSL query.

→ If a SOSL query does not return any records for a specified sObject type, the search results include an empty list for that sObject.

→ For example, you can return a list of accounts, contacts, opportunities and leads that begin with the phrase map.

```
List<List<sObject>> searchList = [FIND 'map*' IN ALL FIELDS
RETURNING Account (Id, Name), Contact, Opportunity, Lead];
```

NOTE:-

The syntax of the FIND clause in Apex differs from the syntax of the FIND clause in the SOAP API.

→ In Apex, the value of the FIND clause is demarcated with single quotes. For Eg.

```
FIND 'map*' IN ALL FIELDS RETURNING Account (Id, Name),
Contact, Opportunity, Lead.
```

→ In the Force.com API, the value of the FIND clause is demarcated with single ~~quotes~~ braces. {

For Eg:-

FIND {map*} IN ALL FIELDS RETURNING Account (Id, Name), Contact, Opportunity, Lead.

From searchList, you can create arrays for each object returned.

Account [] accounts = ((List<Account>) searchList[0]);

Contact [] contacts = ((List<Contact>) searchList[1]);

Opportunity [] opportunities = ((List<Opportunity>) searchList[2]);

Lead [] leads = ((List<Lead>) searchList[3]);

Dynamic SOSL :-

Dynamic SOSL refers to the creation of a SOSL string at runtime with Apex code.

→ Dynamic SOSL enables you to create more flexible applications. For example, you can create a search based on input from an end user, or update records with varying field names.

→ To create a dynamic SOSL query at runtime, use the search query method. For example

```
List<List<sObject>> myQuery =
        search.query(SOSL_search_String);
```

The following example excercises a simple SOSL query string.

```
String searchquery = 'FIND\'Edge*\' IN ALL FIELDS RETURNING
    Account(id,name), Contact, Lead';
List<List<sObject>> searchList = search.query(searchquery);
```

→ Dynamic SOSL statements evaluate to a list of lists of sObjects, where each list contains the search results for a particular sObject type. the result lists are always returned in the same order as they were specified in the dynamic SOSL query.

→ the search query method can be used where ever an inline SOSL query can be used, such as in regular assignment statements and for loops.

SOSL example :-

```
Public with Sharing class DefevenceDemoControlles
    {
 public List<opposturnity> optyList{get; set;}
 public List<Lead> leadList {get; set;}
 public List<Contact> conList {get; set;}
 public List<account> acclist{get; set;}
   public DefevenceDemoControlles()
     {
       }
 public void soslpemo_method()
     {
  OptyList = New List<opportenity>();
  leadList = New List<Lead>();
  ConList = New List<Contact>();
  acclist = New List<account>();
List<List<Sobject>> SearchList = [FIND 'test' IN ALL FIELDS
          RETURNING Account (Id, Name, type),
Contact (name, email), Opportunity (name, StageName),
     Lead (Company, name, Status)];
```

```
        acclist = ((List<Account>)searchList[0]);
        conList = ((List<Contact>)searchList[1]);
        optyList = ((List<opportunity>)searchList[2]);
        leadList = ((List<Lead>)searchList[3];
            }
        }
```

Vf page :-

```
<apex:page controller="DefaultDemoController">
 <apex:form>
  <apex:CommandButton value="show records using SOSL"
                      action="{!soslDemo_method}"/>
  <apex:pageBlock title="Accounts">
 <apex:pageBlockTable value="{!acclist}" var="acc">
    <apex:column value="{!acc.name}"/>
    <apex:column value="{!acc.Type}"/>
 </apex:pageBlockTable>
</apex:pageBlock>
<apex:pageBlock title="contacts">
```

```
<apex:pageBlockTable value="{!conList}" var="con">
    <apex:column value="{!con.name}"/>
    <apex:column value="{!con.email}"/>
</apex:pageBlockTable>
</apex:pageBlock>
<apex:pageBlock title="Leads">
    <apex:pageBlockTable value="{!leadList}" var="lead">
        <apex:column value="{!lead.name}"/>
        <apex:column value="{!lead.Company}"/>
    </apex:pageBlockTable>
</apex:pageBlock>
<apex:pageBlock title="Opportunities">
<apex:pageBlockTable value="{!optyList}" var="opty">
    <apex:column value="{!opty.name}"/>
    <apex:column value="{!opty.StageName}"/>
</apex:pageBlockTable>
</apex:pageBlock>
</apex:form>
</apex:page>
```

Email programming :-

When you coant to send a email from the Salesforce to the external system & reciewing a mail from the external system to the salesforce then we use Email services. There are 2 types of emails what we have

1) Inbound email messaging
2) outbound email messaging

outbound email :- This is used to send an email to external system using apex code. There are 2 types of outbound Emails.

1) Single Email Message
2) Mass Email Message.

NOTE :- Messaging name space provides classes and methods for salesforce outbound and inbound email feenctionality.

1) SingleEmailMessage :-

This is a instantiates an email object used to sending a single email message.

Syntax :-

```
Messaging.SingleEmailMessage mail = new Messaging.SingleEmail-
                                    Message();
```

This class Contains methods of sending single email message. When we want to send a message first we have to create the object of SingleEmailMessage.

The list of methods available in the SingleEmailMessage are

⇒ SetBccAddresses(String[])

this method will set Bcc Address to whom the mail should be sent. we can set upto 25 email address.

Eg:- String[] toBccaddress = new String[]{'a1@gmail.com', 'a2@gmail.com'};

myemail.setBccAddresses(toBccaddress);

⇒ SetCcAddresses(String[])

this method will set Cc Address to whom the mail should be sent. we can set upto 25 email address.

Eg:- String[] toCcaddress = new String[]{'c1@gmail.com'}

myemail.setCcAddresses(toCcaddress);

⇒ SetToAddresses(String[])

this method will set the address, we can set upto 100 addresses.

Eg:- String[] toaddress = new String[]{'abc@gmail.com', 'xyz@gmail.com'};

myemail.setToAddresses(toaddress);

⟹ setSubject(String)

this method will set the Subject of the mail.

Eg:- myemail.setSubject('Test Email');

⟹ setplainTextBody(String)

this method will set the TextBody of the mail.

Eg:- myemail.setplainTextBody('Hi sam');

⟹ setHtmlBody(String)

this method will set the main body of the mail.

Eg:- myemail.setHtmlBody('<h1> this is the link for Registration </h1>');

Example :-

Apex class :-

```
Public class mailcg
  {
Public pageReference show()
    {
  }
Messaging.SingleEmailMessage msg = new Messaging.
                                   SingleEmailMessage();
String[] toaddress = new String[]{'abc@gmail.com',
                                  'xyz@gmail.com'};
msg.setToAddresses(toaddress);
String[] tobccaddress = new String[]{'a1@gmail.com',
                                     'a2@gmail.com'};
msg.setBccAddresses(tobccaddress);
String[] toccaddress = new String[]{'c1@gmail.com'};
msg.setCCAddresses(toccaddress);
msg.setSubject('Test');
msg.setplainTextBody('Hai Ram');
```

```
msg.setHtmlBody('<h1> This is the link for Registration </h1>');
Messaging.sendEmail(new Messaging.SingleEmailMessage[]
                                        {msg});
return null;
    }
}
```

Vf page :-

```
<apex:page controller="mailcg">
 <apex:form>
  <apex:commandButton value="click" action="{!show}"/>
  </apex:form>
</apex:page>
```

⇒ In the above program when we want to send an Email we use the following syntax.

Public Messaging.sendEmailResult[] sendEmail (Messaging.Email[] emails, Boolean allOrNothing)

This method sends list of email objects instantiated with either SingleEmailMessage or MassEmailMessage.

→ In the method the boolean parameter is optional. When you give the boolean value is false, if any one of email is failed it stops processing all other emails, if boolean value is true if any one of the email is failed, it discarded only that mail and the rest of the mails processed asusually.

SendEmailMessage :-

Syntax :-

public Messaging.SendEmailResult[] sendEmailMessage(List<ID> emailMessageIds, Boolean allOrNothing)

This method sends 10 draft email messages as specified by the email message Id's.

MassEmailMessage :-

We can send a mass email message to a recipient list that consists of Contacts, leads, person accounts, or users you can view in Salesforce.

Messaging.MassEmailMessage :-

This class has all the methods defined in the Email class and also Email base class methods.

Methods :-

public void setDescription(String description) :- This will give the description about the mail.

public void setTargetObjectIds(ID[] targetobjectIds) :- We can add upto 250 IDs per email. If you specify a value for the target object Ids field optionally specify a whatID's as well.

public void setWhatIds (ID[] whatIds) :- The values can be any one of the following, Contract, case, opportunity and product.

Syntax :-

```
Messaging.MassEmailMessage mail = new Messaging.MassEmail
                                  Message();
```

Example :-

Apex class :-

```
public class testemail
  {
  private final List<Id> Contactids;
  public List<Contact> Con;
```

```
Public testemail(Apexpages.StandardController controller)
    {
    con = [select Id from Contact limit 250];

  for(Integer i=0; i<250; i++)
    {
      contactids.add(con[i].Id);
        }
    }
  public void sendEmail()
    {
Messaging.MassEmailMessage mail = new Messaging.MassEmail
                                        Message();
  mail.setTargetObjectIds(contactids);
  mail.setTe
Messaging.sendEmail(new Messaging.MassEmailMessage[] {mail});
      }
  }
```

Vf page :-

```
<apex:page StandardController="Contact" extensions="testemail">
  <apex:form>
```

```
<apex:CommandButton value="sendEmail" action="{!sendEmail}"/>
</apex:form>
</apex:page>
```

pdf EmailFileAttachment :-

Messaging.EmailFileAttachment :-

EmailFileAttachment is used in single email message to specify attachments passed in as a part of request in other terms sending file attachments in the email.

Methods :-

Public void setFileName(String name) :- This method will set the name of the file which we attach.

public void setBody(Blob, attachment) :- this method will set the attachments, when we want to set the attachments in the Blob format, we need to use the Blob methods.

Public Static Blob topdf(String) :- this method cast the specified String into pdf format.

Public Static Blob ValueOf(String) :- This method cast the specified String into Blob format.

public String toString(Blob b) :- If we give a Blob value it is Converted into String.

public void setContentType(String) :- This will specify the what type of attachment Content we have added.

public void setInline(Boolean b) :- When you give true it doesn't require any action when the Content is open.

To attach a pdf file in email we have to write the code like below.

```
pdf.getparameters().put('id',(String)acc.id);
    pdf.setRedirect(true);
    Blob b = pdf.getContent();
Messaging.EmailFileAttachment myem = new
                    Messaging.EmailFileAttachment();
Messaging.SingleEmailMessage em = new Messaging.SingleEmail
                                    Message();
    em.setFileAttachments(new Messaging.EmailpileAttachment[]
                                    {myem});
```

Create a visualforce page with in a attachment pdf

```
<apex:page StandardController = "Account" renderAs = "pdf">
 <apex:pageBlock>
  <apex:pageBlockSection>
  <apex:outputfield value = "{!account.Name}"/>
   <apex:outputfield value = "{!account.Industry}"/>
  </apex:pageBlockSection>
 </apex:pageBlock>
</apex:page>
```

Write a program to add visualforce page as a attachment to mail in pdf format.

Apex class :-

```
public class MyAttachment
      {
   Account acc;
public String myname {set; get;}
 public MyAttachment()
      {
  acc = [select id, name from Account where
                name = :myname limit 1];
      }
```

```
public pageReference Show()
    {
Messaging.SingleEmailMessage mymail = new Messaging.
                                    SingleEmailMessage();
Messaging.EmailFileAttachment fmail=new Messaging.
                                    EmailFileAttachment();
PageReference pdf = Page.attachmentpdf;
pdf.getparameters().put('id',(String)accid);
  Pdf.setRedirect(true);
  Blob b= pdf.getConnec
      Blob b = pdf.getContent();
  fmail.setFileName('attachment.pdf');
  fmail.setBody(b);
  mymail.setSubject('Month Bill');
String toadd = new String[]{'abc@gmail.com'};
mymail.setToAddresses(toadd);
mymail.setBody('plz find the attachment');
mymail.setFileAttachments(new Messaging.EmailFile
                                    Attachment[]{mail});
Messaging.sendEmail(new Messaging.Email[]{mymail});
  return null;
    }
}
```

Vf page:-

```
<apex:page Controller="MyAttachment">
 <apex:form>
 <apex:inputText value="{!myname}"/>
<apex:commandButton action="{!show}" value="click"/>
 </apex:form>
</apex:page>
```

Sending the documents through mail:-

public void SetDocument Attachments(IDC[]):- This method will send the documents that we got in the Salesforce in the email using messaging concept.

Step1:- Create a document in Salesforce

1. Go to Document object click on new document.
2. Enter the name, description of the document.
3. Select External file, choose the document file & save.

Step2:- In Apex program write a query to fetch the id of the document where document name = 'Satish'.

Attatch this document to singleEmailMessage

```
Document d = [select id from Document where name='satish'];
Messaging.SingleEmailMessage mail = new SingleEmailMessage();
mail.setDocumentAttachments(new ID[]{d.id});
```

Example :-

Apex class :-

```
public class mailcg
{
public pageReference show()
{
Document d = [select id from Document where name='satish'];
Messaging.SingleEmailMessage msg = new Messaging.SingleEmail
                                   Message();
msg.setDocumentAttachments(new ID[]{d.id});
msg.set
String[] krish = new String[]{'abc@gmail.com'};
msg.setToAddresses(krish);
msg.setSubject('Hai');
```

```
msg.setplainTextBody('Hello');
msg.setHtmlBody('<h1>How r you</h1>');
Messa
Messaging.sendEmail(new Messaging.singleEmailMessage[] {msg});
return null;
    }
}
```

VF page :-

```
<apex:page Controller="mailcg">
 <apex:form>
  <apex:CommandButton value="click" action="{!show}"/>
 </apex:form>
</apex:page>
```

Inbound Email Service :-

Inbound email service is nothing but this will recieve a mail from external system to Salesforce and the apex class will process the email and attachments and perform requested operation.

→ To perform this operation we got some classes defined in messaging name space.

1) Messaging. InboundEmail class :-

This class object represents email recieved by the Salesforce. This class has following properties.

```
public String textAttachments
public String subject
public String replyTo
public String toAddress
public String plainTextBody
public String messageId
public String inReplyTo { get; set; }
```

This inReplyTo field of the incoming email, identifies the email (s) emails to which this one is rep (parent emails) contains the parent email (s) email message.

public String HtmlBodyIsTruncated {get;set;}

This indicates the whether the html body is truncated or not.

public String CcAddress

public String fromAddress

public String fromName

ii) Messaging.InboundEnvelope :-

The object of this class stores the envelope information (i.e. From adress & To address) associated with inbound email. this class has got two properties. they are

public String fromAddress

public String toAddress

Eg:- Messaging.InboundEnvelope em = new

Messaging.InboundEnvelope();

em.fromAddress = 'xyz@gmail.com';

em.toAddress = 'abc@gmail.com';

Headers :-

A list of RFC 2822 headers. In RFC 2822 includes following data.

* Recieved from
* Custom headers
* Message Id

Inbound Email Header class we have 2 attributes.

the object of this class refers to the headers contents of RFC 2822 format. Apart from that it also has 2 attributes. They are

Name & value.

"iii) Messaging. InboundEmailResult Class":-

the object of this class is used to return the result of the email service.

NOTE:- If the object is null the result is achieved to be successfull. This class has 2 properties.

1) public boolean Success:- this attribute indicates whether the email recieved was successfully processed or not. Incase if the email was not successfully processed, salesforce rejects the inbound email and sends reply to the original setup.

2) public String message:- This message stores the message written by the salesforce in the reply mail.

→ Any Apexclass which coant to process inbound email should to implement an interface Messaging.InboundEmailHandler. This interface has a method.

```
globalMessaging.InboundEmailResult handleInboundEmail
                (Messaging.InboundEmail email,
                 Messaging.InboundEnvelope envelope)
```

→ In the above method first parameter email object stores the email recieved from email object to the salesforce.

→ the second parameter stores the to & from address.

Eg1:- Write a program to handle the inbound email and create a new task based on the address form which it has recieve.

```
global class CreateTaskEmailExample implements
                Messaging.InboundEmailHandler{
  global Messaging.InboundEmailResult handleInboundEmail
    (Messaging.inboundEmail email, Messaging.
                inboundEnvelope envelope)
  {
Messaging.InboundEmailResult result = new
                Messaging.InboundEmailResult();
```

```
String ename = email.fromName;
String edescription = email.plainTextBody;
String eindustry = email.Subject;
Account a = new Account(name = ename, description = edescription,
                        industry = eindustry);
    insert a;
    }
}
```

Eg2:-

```
global class createTaskEmailExample implements
        Messaging.InboundEmailHandler {
global Messaging.InboundEmailResult handleInboundEmail
    (Messaging.inboundEmail email, Messaging.inbound
                                    Envelope envelope)
    {
Messaging.InboundEmailResult result =
        new Messaging.InboundEmailResult();
  try {
Contact U con = [SELECT Id, Name, Email FROM Contact
                 WHERE Email = :email.fromAddress limit 1];
```

```
Task t = new Task( Description = email.plainTextBody,
        priority = 'Normal', status = 'Inbound Email',
            subject = email.subject, IsReminderSet = true,
        ReminderDateTime = System.now() + 1,
        WhoId = vCon.Id
                );
insert t;
System.debug('New Task object : '+newTask);
        }
catch (QueryException e)
    {
    System.debug('query Issue : '+e);
    }
}
}
```

Email Service :-

Email Service is an automated process that use Apex classes to process the contents, headers, Attachments of Inbound Email.

NOTE:- Visualforce email templates can not be used for mass emails.

i) We can associate each email service with one of more salesforce generated email addresses to which the users can send messages for processing.

Navigation:-

Setup → Build → Develop → Email Service → click on new Email service.

→ Give the service name, now select the Apex class to which the mail should be forwarded to process the mail and the class should implement InboundEmail interface.

→ Select which type of attachment we have to select list of address/domains whenever we want to recieve the mails.

→ Convert text to Binary fields. Click on Active & save.

Steps to create Email service:-

1) Go to setup → develop → Emailservices

2) click the new email service button.

3) on the Email service page, fill the form like below.

a) Email Service Name : Sbx_EmailAWSResponseHandler

b) Apex class : Sbx_EmailAWS Response Header.

c) Accept Attachments : None

d) Advanced Email security settings : (unchecked)

e) Accept Email From : (leave blank)

f) Convert Text Attachments to Binary Attachments (unchecked)

g) Active : (checked)

h) Over Email Rate Limit Action : Discard message

i) Deactivated Email Address Action : Discard message

j) Deactivated Email Service Action : Discard message

k) Unauthenticated Sender Action : Discard message

l) Enable Error Routing : (unchecked)

m) Route Error Emails to This Email Address : (leave blank)

4) Click Save and New Email Address button.

a) on the Email service Address page, fill in the form as follows.

b) Email Address : sbx_ emailawsresponsehandler

c) Active : (checked)

d) Context User : (choose the user that will have full access

to the soap box mailer app, and to all related services)

4) Accept Email From: (leave blank)

5) Click save.

6) Copy the full email address provided by Salesforce, which will be utilized in the next step.

DML operations :-

The operations are

Insert
update
Delete
Upsert

There are two ways to perform DML operations.

1) By using DML Statements

Eg:- List<Account> acclist = new List<Account>();

acclist.add(new Account(Name='Acme1'));

acclist.add(new Account(Name='Acme2'));

insert acclist;

2) By using Database class

Eg:- List<Account> acclist = new List<Account>();

acclist.add(new Account(Name='Acme1'));

acclist.add(new Account(Name='Acme2'));

Database.SaveResult[] sr = Database.insert(acclist, false);

→ there is one difference between above two operations.

In the Database class method you can specify whether to allow partial processing of the records if any errors are

encountered.

By passing +

By parsing the boolean value as a parameters to Database.insert.

→ If you give the parameters as true if any error occurs it doesn't allow the operation to continue.

→ If you specify false the remining DML operations can still succeed, where as insert in DML if any one of the record fails the total operation is discarded.

Eg:-

```
Public pageReference show()
    {
List<Account> acc = new List<Account>();
Account a1= new Account (Name ='Today4', Industry = 'Banking');
Account a2 = new Account(Industry = 'Banking');
Account a3 = new Account(Name = 'Today4',
                         Industry = 'Banking');
    acc.add(a1);
    acc.add(a2);
    acc.add(a3);
Database.SaveResult[] r = Database.insert(acc,true);
  return null;
    }
```

→ In the above program when we give Database.insert(acc, true), if any error occurs in any one of the records a1, a2, a3 the entire operation of insert is rolledback.

→ When you give Database.insert(acc, false) for above Statements, if any error occurs in any one of the records a1, a2, a3 only that record is terminated and Status is saved to SaveResult class rest of the operations are processed normally.

SOQL DML and loops:-

SOQL for loops iterate over all of the SObject records returned by a SOQL query. The syntax of a SOQL for loop is either.

```
for(variable : [soql_query])
    {
      code_block
    }
(or)
for(variable_list : [soql_query])
    {
    code_block
    }
```

Eg1:-

```
String s='Bened';
for(Account a:[SELECT Id,Name from Account Where
                                    Name =:s])
{
//your code
}
```

Eg2:- Write a code to fetch list of accounts where account name is 'siebel' and update them by oracle.

```
List<Account> accs = [SELECT Id, Name FROM Account
                      WHERE Name = 'Siebel'];
//Loop through the list and update the Name field
for(Account a:accs)
{
a.Name = 'oracle';
}
//update the database
update accs;
```

Eg3:- Write a code a fetch list Contacts for a given account & update Contact email with account email id.

```
Account acc = [select email,(select id, email__c from contacts)
               from Account where name='Sam' limit 1];
```

```
List<contact> con = acc.contacts;
List<contact> myContacts = new List<Contact>();
  for(contact x : con)
     {
     x.email = acc.emailid__c;
      myContacts.add(x);
       }
  update myContacts;
```

Eg4:- Write a code to fetch all the transaction records which are created yesterday and delete them.

```
List<Transaction__c> tran = [select id from Transaction__c
                    where createddate = YESTERDAY];
    delete tran;
```

Eg5:- Write a code to create new student records by fetching the data from college object where branch is CSE.

```
List<Student__c> Std = new List<Student>();
List<college__c> colg = Database.query('select name, Branch__c,
                    Year__c from college__c where branch__c='CSE');
    for(college__c : colg)
     {
    Student__c S = new Student__c();
       S.name = c.name;
```

```
S.Branch__c = c.branch__c;
    Std.add(S);
    }
  insert Std;
```

How to count numbers of DML statements in a transaction:-

Eg1:-

```
Account a=new Account(Name='Sam', Industry='Banking');
 insert a;
Account a1 = new Account(Name='Sam', Industry='Banking');
 insert a1;
```

Eg2:-

```
void show()
    {
  Account a = new Account(Name='Sam', Industry='Banking');
   insert a;
  Account x=[select id,name from Account limit 1];
     x.name='Sam';
    update x;
 Account y=[select id,name from Account limit 1];
     delete y;
    }
```

Eg3:-
```
void show()
    {
for(integer i=1; i<=100; i++)
    {
Account a = new Account(Name='Sam', Industry='Banking');
insert a;
    }
}
```

NOTE:- In a single transaction we can not make more than 150 DML statements.

Eg4:-
```
void show()
    {
for(integer i=1; i<=160; i++)
    {
Account a=new Account(Name='Sam', Industry='Banking');
insert a;
    }
}
```

In the above transaction show we made 160 DML Statements. So we get an errors like below.

System.LimitException: Too many DML Statements: 151
Errors is in expression '{!show}' in page dmlStatementCount

Bulkify the DML operation:-

When we are performing DML operations on any record we have governor limits of 150 records for every statement.

→ To support the large no. of DML statements in a single transaction without bypassing the governor limits. to meet this requirement we are Bulkifying the operation by adding all the records to a list and calling the DML on the list.

NOTE:- If you call DML operation on the list object it is considered as single DML statement. this operation is called bulkifying operation.

Governor limit:- In a single transaction we can perform DML operations on maximum 10,000 rows. If you perform more than 10,000 rows it gives an exception like below.

System.LimitException: Too many rows 10001.

Eg:- List<Account> acc = new List<Account>();

for(Integer i=1; i<=4000; i++)

{

Account a = new Account(Name = 'ellcustoday',
Industry = 'Banking');

acc.add(a);

}

```
insert acc;
List<Customer__c> acc1 = new List<Customer__c>();
for(Integer i=1; i<=6002; i++)
{
Customer__c a = new Custom__c(Customer_Name__c=
                                'customer');
    acc1.add(a);
}
insert acc1;
return null;
}
```

Interface in Apex :-

Interface is a collection of unimplemented methods. This will specify the signature of the method, types of inputs that we pass the method specifies what type is given as an output.

NOTE :- Generally the interface methods we give it as global

Syntax :- public interface Exam

```
{
Account show(String s);
}
```

All the methods in the interface are public and abstract methods. Interface also contains final static datamembers

→ The class which is implementing this interface should define all the methods in the interface.

Eg:- intesface test

```
{
public String show()
 void disp();
}
global class Example implements test
{
Public String show()
{
return 'sam';
}
public void disp()
{
Account a = new Account (Name = 'sam');
}
}
```

→ If you don't define any methods in the interface it giving an error. we can create an object for class.

```
class Test
  {
  }
Test t = new Test();
```

→ interface Demo

```
{
void show();
}
```

For interface we can not create an object

Demo d = new Demo(); is not allowed. Coz interface Contains only unimplemented methods and static methods.

→ class Exam implements Demo

```
{
public void show()
  {
  }
}
```

Demo d = e; is allowed. Coz 'e' is the object of the class which is implementing the demo interface.

NOTE:- Interface reference variable can store the object of the class which has successfully implemented interface.

In the above scenario

Demo d=t; is not allowed. Bcoz test class is not implemented interface demo.

Interface Iterator<AnyType> :-

This is the interface defined by the salesforce in apex. It has 2 methods.

1. Boolean hasNext()
2. AnyType next()

1. Boolean hasNext() :- this method returns true if there is another item in the collection being traverse else returns false.

2. AnyType next() :- Returns the next item in the collection.

Both the methods given above should be declared as a global & public.

NOTE :- We can use only custom iterators in while loop, it is not supported for loop.

Write a program to create Custom iterator :-

```
class satish implements Iterator<Account>
{
```

```
public List<Account> acc;
  public Integer i;
 public satish()
    {
 acc = [select name from Account];
     i=0;
     }
  public boolean hasNext()
       {
     if(i>acc.size())
      return false;
     else
      return true;
       }
public Account next()
       {
     if(i==0)
   return null;
     i++;
     return a[i-1];
       }
     }
```

NOTE :-

```
satish S = new satish();
Iterator<Account> it = S;
```

Since satish class as implemented iterator interface we can store the object of the satish class in Iterator it.

Interface Iterable :-

Iterable is a interface in which we have a method.

```
Iterator<class> iterator()
```

Eg:

```
class Bened implements Iterable<Account>
{
public Iterator<Account> iterator()
{
Satish S = new Satish(); // satish class is implementing interface Iterator, so we can store this object in interface iterator.
Iterator<Account> it = S;
return it;
}
}
```

Eg2:- class Example

{

```
List <Account> acc;
 public Example ( )
     {
acc = new List <Account> ();
     }
 Public pageReference Check ( )
     {
  Bened b = new Bened ();
Iterator <Account> it = b. iterator ();
 // satish s = b. iterator ();
  while (it. hasNext ())
     {
Account a = (Account) it. next ();
    acc. add (a);
    return null;
     }
   }
 }
```

Eg3:

```
global class satishIterator implements Iterator<Account>
    {
  public List<Account> acc;
    public Integer i;
   Public satishIterator()
          {
acc = [select name from Account Limit 5];
     i=0;
    }
 Public boolean hasNext()
       {
      if(i>acc.size())
       return false;
       else
       return true;
        }
   Public Account next()
      {
        if(i==5)
        return null;
         i++;
      return acc[i-1];
          }
     }
```

Eg4:-

```
public class BeredIterable implements Iterable<Account>
    {
    public Iterator<Account> iterator()
        {
        return new SatishIterator();
        }
    }
```

Eg5:-

```
public class ExampleIterator
    {
    public List<Account> acc;
    public List<Account> getAcc()
        {
        return acc;
        }
    public Example Iterator()
        {
        acc = new List<Account>();
        }
    Public pageReference Show()
        {
```

```
Iterable<Account> my = new BenedIterable();
Iterator<Account> test = my.iterator();
while(test.next()!=null)
    {
Account a=test.next();
  acc.add(a);
    }
return null;
    }
}
```

vf page:-

```
<apex:page Controller="ExampleIterator">
  <apex:form>
<apex:commandButton value="click" action="{!show}"/>
  <apex:pageBlock>
  <apex:pageBlockTable value="{!acc}" var="a">
<apex:column value="{!a}"/>
</apex:pageBlockTable>
  </apex:pageBlock>
</apex:form>
</apex:page>
```

Database.QueryLocator Class:-

Database.QueryLocator class stores the record set return by the database.getQueryLocator.

Methods:-

There are 2 methods in the QueryLocator class.

⇒ getQuery():-

Syn:- public String getQuery()

This method returns the query used to instantiate the Database.QueryLocator object.

This is very much useful when testing the start method.

Eg:- Database.QueryLocator dq = Database.getQueryLocator
([select name from Account]);

String str = dq.getQuery();

// str = 'select name from Account';

⇒ iterator():-

Syn:- public Database.QueryLocatorIterator iterator()

This will return new instance of queryLocator Iterable.

Eg:- List

Eg:-

```
List<Account> acc = new List<Account>();
Database.QueryLocator dq = Database.getQueryLocator();
Database.QueryLocatorIterable iq = dq.iterator();
while(iq.hasNext())
{
Account a = (Account) iq.next();
acc.add(a);
}
```

BatchApex :-

Batch Apex allows you to define a single job that can be broken up into managable chunks, where every chunk can be processed seperately.

Eg:- If you need to make a field update of every record of Account object in your Organization, then we have governing limits that would restrict us from achieving the above task.

Reasion :- In a single transaction we can process only 10,000 records. Now, in the above case if we have more than 10,000 records in the Organization then we can not perform this field update.

BatchApex :- In the BatchApex it will fetch all the records on which you want to perform the field update and divide them into list of 200 records and on every 200 records operation is performed seperately.

→ this would help us to execute on more than 10,000 records as, it won't perform an operation on all the records in a single transaction instead it dividing them into no. of

sub tasks where each sub task may contain the records upto 2000.

Database. Batchable interface :-

To use the BatchApex concept the Apex class should implement Database. Batchable interface.

→ Database. Batchable interface consists of 3 methods, that must be implemented.

1. Start() method
2. execute method.
3. finish method.

1. Start method :- Start method is automatically called at the beginning of the batch apex job.

→ This method will collect the records & objects on which the operation should be performed.

→ This records are broken down into sub tasks and given to execute method.

Syntax :-

```
global (Database.QueryLocator | Iterable<sObject>)
        start(Database.BatchableContext bc) { }
```

→ the return type of the Start method can be

i) Database.QueryLocator

(or)

ii) Iterable<sObject>

i) Database.QueryLocator :-

use Database.QueryLocator as return type to the start method when you are fetching the records using a simple select query.

NOTE :- The governing limit says

i) Total number of records retrieved by SOQL Queries 50,000.

ii) Total number of records retrieved by Database.getQueryLocator 10,000.

But, in the Batch Apex the governing limits for SOQL query are bypassed and we can fetch upto 50 million records using Database.QueryLocator.

Example :-

```
global Database.QueryLocator Start(Database.BatchableContext bc)
{
    return Database.getQueryLocator('select id, name from Account');
}
```

Eg2:- Write a start method to fetch all customers Name, AccountType from customer object whose AccountType is 'Saving'.

```
global Database.QueryLocator Start(Database.BatchableContext
                                                    bc)
{
String sql = 'select id, Customer_name__c, Account_Type__c
        from Customer__c where Account_Type__c
                = '+'\'saving\'';
}
```

ii) Iterable :- use iterable as a return type when you want to use Complex logic & scope to fetch the records for the batch job.

Eg:- global class MyTest implements Iterable<Account>

```
{
global Iterable<Account> start(Database.BatchableContext
                                                    bc)
{
return new MyTest(); //This should return object of the
                        class that has implemented
                        iterable interface.
}
```

2) execute method :-

The records which are fecthed from the Start method are diuided into batches of 200 records each. Now every batch of 200 records are seperately pass to the execute method seperately and the operation what we want to perform on these records are wel written in this execute method.

Syntax :-

global void execute (Database.BatchableContext BC, List<p>){ }

this method takes two parameters.

→ A reference to the Database.BatchableContext object.

→ A list of sobjects such as List<sObject>, or a list of parameterized types.

i.e, Set of 200 records which are passed to the execute method are stored in this list.

3) finish method :-

When the Start method has fetched 1000 records on which we want to perform the operations they are diuided into 5 batches/Groups of size 200.

}

→ on every batch of the group the execute method is called.

→ which means execute method is called 5 times.

→ After executing every group/batch the governing limits are reset for the execute method.

→ once all the batches are executed then finish method will be called to send email notification & for post execution work.

NOTE:-

Syntax:-

global void finish (Database.BatchableContext BC){ }

NOTE:- All the 3 methods of the Batchable interface refers to Database.BatchableContext object. Where this object is used to track the progress of the batch job.

Example1:- Write a BatchApex program to update Account Name with in the Account object with Sufix Mr. before the AccountName.

Example:- Write a batch apex program to update Account name in the Account object with suffix 'Mr.' before the Account Name.

```
global class AccountBatch implements Database.Batchable<Sobject>
{
global Database.QueryLocator Start(Database.BatchableContext
                                   bc)
{
String query = 'select id, name from Account';
return Database.getQueryLocator(Query);
}
global void execute(Database.BatchableContext bc,
                    List<Account> scope)
{
List<Account> acc = new List<Account>();
for(Account a:acc)
{
a.name = 'Mr.'+a.name;
acc.add(a);
}
update acc;
}
```

```
global void finish(Database.BatchableContext bc)
    {
    }
}
```

Example 2:- Write a batch apex program to phone no. in the Contact object with the phone no. in the corresponding Account object. Where Contact is a child of Account.

```
global class contactupdate implements Database.Batchable<sObject>
{
global Database.QueryLocator start(Database.BatchableContext bc)
    {
    String query = 'select id, phone, Account.phone from
                                                Contanct';
    return Database.getQueryLocator(query);
    }
global void execute(Database.BatchableContext bc,
                        List<Contact> scope)
    {
List<Contact> con = new List<Contact>();
    for(Contact c:con)
    {
```

```
c.phone = c.Account.phone;
    Con.add(c);
    }
update con;
    }
global void finish(Database.BatchableContext bc)
    {
    }
}
```

How to invoke batchapex job (or) how to execute the batch apex job programmatically.

We can use Database.executeBatch() method to programmatically begin the batch job.

Syntax:-

Public Static ID executeBatch(sObject className)

Public Static ID executeBatch(sObject className, Integer scope)

→ the above two methods are static methods of database class. We can use any one of the method to execute

the batch job.

NOTE:- the class name what we are passing to the Database.executeBatch() method should be object of the class which has implemented Database.Batchable interface.

Orders of execution of Batch Apex job when we invoke through Database.executeBatch():-

Step1:- create a object for the class which has implemented Database.Batchable interface.

Step2:- pass this object which you have created in the first step as a parameter to the Database.Batchable interface.

Step3:- When database.executeBatch() method is called it will add the batch job to the queue.

Step4:- once the resource is available in the queue automatically start() method will be invoked and it will collect all the records or which will we need to perform the operation.

Step5:- NO the records that are fetched from the Start method are divided into small group/batch of size given in the

Database.executer method.

NOTE:- In case if you don't specify the size by default it takes 200

Step6:- on every batch of records execute method will be invoked.

Step7:- once all the batches are executed, then finish method will be called.

NOTE:- 1) Database.execute method is going to return id of the batch job. using which we can monitor & abort the operation.

2) All the asynchronous jobs are stored to 'AsyncApexJob' object, from this we can monitor, no of jobs processed for our asynchronous job and status of the job.

Example:-

```
ID batchprocessid = Database.executeBatch(reassign);

AsyncApexJob aa = [select id, status, JobItemsprocessed
                   from AsyncApexJob where
                   ID =: batchprocessid];
```

Scenario :-

1. Create a visualforce page with Input text button.
2. When we click on the Replace button in the custom visualforce page.
3. Fetch All the records in the Customer Object with the Customer name matching with name given in the visualforce page.
4. update their Account Type as 'Saving' account.

Step1 : create a batch Apex class to perform update of the Account Type.

```
global class CustomerBatch implements Database.Batchable
                                        <SObject>
{
public String myname;
global CustomerBatch(String myname)
{
this.myname = myname;
}
global Database.QueryLocator Start(Database.BatchableContext
                                        BC)
{
```

```
return Database.getQueryLocator('select id, Account_Type__c
        from Account where name = '+myname);
    }
global void execute(Database.BatchableContext bc,
                    List<Customer__c> scope)
    {
List<Customer__c> cust = new List<Customer__c>();
for(Customer__c c: cust)
    {
c.Account_Type__c = 'saving';
cust.add(c);
    }
update cust;
}
global void finish(Database.BatchableContext bc)
    {
Messaging.singleEmailMessage myemail = new
                    Messaging.singleEmailMessage();
String[] toadd = new String[] {'abc@gmail.com'};
```

```
myemail.setToAddresses(toadd);
myemail.setSubject('Batchprocessed');
myemail.setplainTextBody('Batch Completed successfully');
Messaging.sendEmail(new Messaging.Email() {myemail};
        }
    }
```

Step2:- Create Apex class to invoke the batch apex job.

```
public class TestMybatch
    {
    public String cname{set;get;}
    Public pageReference show()
        {
        CustomerBatch mybatch = new CustomerBatch(cname);
    ID id = Database.executeBatch(mybatch, 400);
        System.debug('My job id' + id);
        }
    }
```

Step3:- Create a visualforce page to call the TestMybatch class

```
<apex:page Controller = "TestMyBatch">
  <apex:form>
   <apex:outputLabel> Enter Name </apex:outputLabel>
   <apex:inputText value = "{!myname}"/>
  <apex:commandButton value = "Click" action = "{!show}"/>
  </apex:form>
</apex:page>
```

Database. Stateful :-

Each execution of batch apex job is considered as discrete transaction. Which means when you have a batch job with 1000 records executed with optional scope of 200 records then

If 5 batch Batch1 ⟶ 1-200
            Batch2 ⟶ 201-400
            Batch3 ⟶ 401-600

⇒ When we call execute on batch1 to summarize the value

```
Integer sum=0;
public void execute(Database.BatchableContext bc, List<Account> scope)
    {
      for(Account a: scope)
      sum = sum+a.AnnualRevenue;
      }
```

→ First Batch of 1-200 records call the execute method : with 200 records.

Before calling execute() : sum = 0

After calling execute() : sum = 30000 (assume it)

→ When the execute() is called on batch 2 of records 201-400 then

Initial value of sum again set to zero.

Before calling execute() on Batch1 : 1-200 records : sum = 0;

After calling execute() on Batch1 : 1-200 records : sum = 30000

Before calling execute() on Batch2 : 201-400 records : sum = 30000

When we call execute() on Batch2 : 201-400 records.

First sum is set = 0 again : sum = 0.

then again fresh summary value is calculated again.

→ Which means the state of the batch is forwarded from one execute() to another execute().

→ If you specify Database.Stateful in the class definition, you can maintain state across these transactions. This is useful for counting or summarizing records as they are processed. For eg. Suppose your job processed opportunity records. you could define a method in execute to aggregate

totals of the Opportunity amounts as they were processed.

```
global class SummarizeAccountTotal implements
                Database.Batchable<Sobject>, Database.Stateful
{
global final String query;
global integer Summary;
global SummarizeAccountTotal(String q)
    {
     Query = q;
     Summary = 0;
    }
global Database.QueryLocator start(Database.BatchableContext Bc)
{
return Database.getQueryLocator(query);
}
global void execute(Database.BatchableContext Bc, List<Sobject>
                                                    Scope)
{
  for(Sobject s:Scope)
  {
  Summary = Integer.valueOf(s.get('total__c'))+Summary;
    }
}
```

```
global void finish(Database.BatchableContext BC)
    {
    }
}
```

Governing limits :-

1. only one batch apex job's Start method can run at a time in an organization.
2. up to 5 queued & active batch jobs are allowed for apex.
3. the maximum number of batch apex method executions per a 24-hour period is 250,000.
4. the batch Apexstart method can have upto 15 query cursors open at a time per user.
5. A maximum of 50 million records can be returned in the Database.QueryLocator object.
6. the Start, execute, and finish methods can implement upto 10 callouts each.

NOTE :- If we have 1000 records with scope of 200 records then they are divided into 5 batches.

So execute() method is called 5 times. Which means in every execute() we call 10 callouts. so in this scenario we call

start () ⟶ 10 callouts

execute() ⟶ 10* 5 = 50 callouts

finish() ⟶ 10 callouts

70 callouts in entire operations

Limitations :-

⟶ Methods declared as future aren't allowed in the classes that implement Database.Batchable interface.

⟶ Methods declared as future can't be called from Batch Apex class.

⟶ For sharing & recaluculation, we recommend that the execute method delete and then re-create all Apex managed sharing for the records in the batch.

⟶ For each 10,000 AsyncApexJob records, Apex creates one additional AsyncApexJob record of type BatchApexWorker for internal use.

Apex scheduler :-

It will invokes the Apex class to run at specific time.

Anybody who want to schedule their class they have to implement schedulable interface.

Schedulable interface :-

The class that implements this interface can be scheduled to run at different intervals. This interface has several method they are

```
public void execute(SchedulableContext sc)
```

Eg:-
```
public class Myschedule implements schedulable
{
public void execute(SchedulableContext sc)
{
Account a = new Account(Name = 'Faraz');
insert a;
}
}
```

→ The scheduler will run in systemcontext, which means all the classes are executed whether the user has permission or not.

→ We can monitor or stop the execution of scheduled apex

job using salesforce user interface from setup.

Navigation :-

Setup → Monitoring → Jobs → Scheduled Jobs

→ The execute() method must be declared as public & global.

→ using System.schedule.

System.schedule :-

once you are implemented schedulable interface use System.schedulable method to execute the class

System.Schedule() method takes 3 parameters.

1) Name of the job.

2) Expression that is used to represent time and date of the operation.

3) the object of the class which you want to execute.

→ Expression is written in the form of 'seconds, minutes, hours, day of the month, month, day of the week, optional years.'

'seconds	Min	HOURS	DAY-MONTH	Month	DAY-week	optional years'
0-60	0-60	0-24	1-31	1-12	1-7	

Special characters :-

? :- Specifies no specific value. This is only available for day of the month and day of the week.

Eg1 :- Write the expression to schedule an operation 10th of August at 12:30 PM.

Ans :- '0 30 12 10 8 ?'

'0 30 12 10 AUG ?'

'0 30 12 10 AUG ? 2013'.

Eg2 :- Write an expression to schedule an operation on Mon Monday 12:30.

Ans :- '0 30 12 ? 1 MON'

'0 30 12 ? 1 2'.

***** :- Specifies all the values.

Eg :- Write the expression to schedule an operation on every day of the AUG at 12:30 PM.

'0 30 12 * AUG ?'

'0 30 12 * 8 ?'

Eg :- Expression to schedule on every hour on 10th AUG.

'0 30 * 10 8 ?'.

L :- Specifies the end of the range. this is available only day of the month & day of the week.

Eg:- Write an expression to schedule the operation on last friday of the March at 10:20.

'0 20 10 ? 3 6L'.

W :- specifies nearest weekday of the given day. this is available for only day of the month.

Eg:- If we specifies 20W and 20th is a saturday so the class runs on 19th. If i give 1W and 1st is a saturday then it runs on monday not on previous month.

Eg:- Write an expression to schedule an operation on nearest weekday of March 20th at 10:20.

'0 20 10 20W 3 6L'

This is to specify the last woking day of last week day of last of month.

:- This specifies nth day of the month.

Week_Day # Day_Month

2#2 it will run monday of every 2nd month.

1 :- JAN, MAR means JAN-MAR.

→ If you want to schedule any operation we have to create an object for the class which has implemented for schedulable interface.

```
Class MySchedule implements Schedulable
    {
    public void execute(SchedulableContext sc)
        {
        }
    }
MySchedule my = new MySchedule();
String str = '0 0 10 * 3 2';
System.schedule('MYJOB', str, my);
                  ↓      ↓    ↓
                Name   Time  Object.
                       format
```

System.scheduleBatch() :-

System.scheduleBatch() is used to run a schedule a batch job only one for a future time. This method has got a 3 parameters.

param1 :- Instance of a class that implements Database.Batchable interface.

param2 :- Job name.

param3 :- Time interval after which the job should start execute.

param4 :- It's an optional parameters which will defines the no.of that processed at a time.

The System.ScheduleBatch() returns the scheduled job Id (CronTrigger id).

→ We can use this job id to abort the job.

Example :- you have to first implement the Schedulable interface for the class the specify the Schedule using Schedule Apex page or system.schedule method.

Apex class :-

```
global class purge implements Schedulable
{
    global void execute (SchedulableContext sc)
    {
    }
List<Credit_Card__c> creditcard = new List<Credit_Card__c>();
for
```

```
for(credit_card__c cc:[select Id, Name from Credit_card__c
                       where Age__c >2])
  {
  creditCard.add(cc);
    cc.CW__c = '0000';
    }
  Database.update(creditCard);
  }
}
```

Eg:- If we want to schedule

i) create an object for the class which has implemented the schedulable interface.

ii) create the time frame.

iii) Invoke the System.schedule method with job name, Schedule object, timeframe.

```
global class Mybatch implements
```

i) global class MyBatch implements Database.Batchable <Sobject>

```
{
global Database.QueryLocator Start(Database.BatchableContext bc)
{
return Database.getQueryLocator('select Id,name from Account');
}
global void execute(Database.BatchableContext bc,
                    List <Sobject> scope)
{
List <Account> acc = new List <Account>();
for(Sobject x:scope)
{
Account a = (Account) x;
a.name = 'Mr' + a.name;
acc.add(a);
}
update acc;
}
global void finish(Database.BatchableContext bc)
{
}
}
```

ii) global class MySchedule implements Schedulable

```
{
    global void execute(SchedulableContext sc)
    {
        MyBatch mb = new MyBatch();
        Database.execute(mb);
    }
}
```

iii) global class TestSchedule

```
{
    public pageReference show()
    {
        String timeframe = '0 10 8 10 * ?';
        MySchedule ms = new MySchedule();
        System.Schedule('MyJob', timeframe, ms);
    }
}
```

Vf page :-

```
<apex:page Controller = 'TestSchedule'>
  <apex:form>
<apex:CommandButton value="Click" action="{! show}"/>
</apex:form>
</apex:page>
```

Schedulable Apex limitations :-

→ We can schedule only 100 jobs at a time.

→ Max. no. of apex schedule jobs in 24 hours is 2,50,000 numbers of jobs.

→ Synchronous Web service callouts are not supported in Schedulable Apex.

Real Time Senario's and UseCases

Real Time Senaraio 1:

Database.Stateful Interface

I have batch apex code updating two fields in Opportunity line Item(OLI). While I run the code for complete batch the values in the field are updating correctly. However, If I spilt the batch apex in two or many, I am facing the issue of the last batch value getting updated in Opportunity line item instead of complete one.

for ex:- Consider 3 record of value 50 each, I am expecting the field in OLI to be updated by 150 and it is updating rightly while I run the batch for Database.executeBatch(job,3); where as if I split the same as Database.executeBatch(job,2); then the field is getting updated by 50 instead of 150.

```
global class SummarizeAccountTotal implements Database.Batchable<sObject>, Database.State
ful{

  global final String query;
  global Map<Id, Account> accountmap;

  global SummarizeAccountTotal(){
  accountmap = new Map<Id, Account> ();
  }

  global Database.QueryLocator start(Database.BatchableContext BC){
    return Database.getQueryLocator(query);
  }

  global void execute(Database.BatchableContext BC, List<sObject> scope){

  List<Opportunity> ops = (List<Opportunity>)scope;
```

```
for (Opportunity o : ops) {

if (accountmap.containskey(o.AccountId)) {
Account a = accountmap.get(o.AccountId);
a.Test_Amount__c += o.Amount;
accountmap.put(o.AccountId, a);
}
else{
accountmap.put(o.AccountId, new Account (Id = o.AccountId, Test_Amount__c = o.Amount))

}

}
}

global void finish(Database.BatchableContext BC){

try {
update accountmap.values();
}
catch (Exception Ex) {
system.debug(Ex);
}

}
}
```

Allow Callout interface Senario with example

I had to integrate a client's Salesforce and NetSuite System. The requirement was to implement a web-service that synchronises the financial-data between two systems everyday. In dev terms, the client's Salesforce has to make a GET request to the NetSuite System, parse the returned JSON response and save the data in respective Salesforce fields. When I looked into this, I found there was an Apex Governor Limit of 10 HTTP callouts in any context, i.e., you cannot make more than 10 HTTP request in one session. For the current requirement, I had to make around 3500 calls a day.

So after a lot of digging, I figured out that the workaround to this problem is using Batch Apex and implementing HTTP requests inside the same. Batch Apex was introduced to empower the developers to be able to build complex, long-running processes on the Force.com platform. By inserting the HTTP request in the same and it works like a background web-service that runs asynchronously and needs no supervision. Sounds sweet, right?? So let's get started.

To use batch Apex, you must write an Apex class that implements the Salesforce-provided interface *Database.Batchable*, and then invoke the class programmatically. In order to make the callouts, you also have to include *Database.AllowsCallouts* in class definition.

Let us first create an Apex class the
implements *Database.Batchable* and *Database.AllowCallouts*.

```
global class BatchSync implements Database.Batchable<sObject>,  Database.AllowsCallouts {

public String query = 'Select ID, Name from Account';
global Database.QueryLocator start(Database.BatchableContext BC) {
  return Database.getQueryLocator(query);
}

   global void execute(Database.BatchableContext BC, List<Account> records) {
     String endpoint;

     for ( integer i = 0; i< records.size(); i++ ){
      try {
       HttpRequest req = new HttpRequest();
       HttpResponse res = new HttpResponse();
       Http http = new Http();
```

```
        // Set values to Params

        endpoint = 'Your endpoint';

        req.setHeader('Authorization', header);
        req.setHeader('Content-Type', 'application/json');
        req.setEndpoint(endpoint);
        req.setMethod('POST');
        req.setBody('Information you wanna send');
        req.setCompressed(true); // This is imp according to SF, but please check if
                        // the webservice accepts the info. Mine did not :P
                        // Had to set it to false

        if (!Test.isRunningTest()) {
          res = http.send(req);
          String sJson = res.getBody();
          System.debug('Str:' + res.getBody());
        }
        // now do what u want to with response.
        }
        catch (Exception e) {
          System.debug('Error:' + e.getMessage() + 'LN:' + e.getLineNumber() );
        }
      }
    }

    global void finish(Database.BatchableContext BC){
      //you can also do some after-the-job-finishes work here
    }
}
```

You can query any valid SF sObject in *Database.QueryLocator*. In this example, you can use the ***records*** Account object to use any data-field from Accounts and even perform DML operations.

Now, to execute the above Batch Job programatically, you need to use *Database.executeBatch* method. The *Database.executeBatch* method takes two parameters:

1. An instance of a class that implements the *Database.executeBatch* interface.
2. The *Database.executeBatch* method takes an optional parameter *scope*. This parameter specifies the number of records that should be passed into the *execute*method. Use this parameter when you have many operations for each record being passed in and are running into governor limits.

Test Class:

```
@isTest
public with sharing class TestBatchSync {

   static testmethod void m1() {

      List<Account> accns = new List<Account>();
      for(integer i = 0; i<10; i++){
         Account a = new Account(Name='testAccount'+'i' );
         accns.add(a);
       }

       Test.StartTest();
       insert accns;
       BatchSync sync = new BatchSync();
       sync.query =  'Select ID, Name from Account';
       ID batchprocessid = Database.executeBatch(sync);
       Test.StopTest();
   }

}

@isTest
private class TestScheduleSync {
   static testmethod void testSync() {

      Test.StartTest();
      List<Account> accns = new List<Account>();
       for(integer i = 0; i<10; i++){
         Account a = new Account(Name='testAccount'+'i' );
         accns.add(a);
       }

       String CRON_EXP = '0 0 23 * * ?';

       insert accns;
       scheduleDailySync sync = new scheduleDailySync();

       // Schedule the test job
       String jobId = System.schedule('testBasicScheduledApex', CRON_EXP,sync);
```

```
        // Get the information from the CronTrigger API object
            CronTrigger ct = [SELECT Id, CronExpression, TimesTriggered, NextFireTime
FROM CronTrigger WHERE id = :jobId];

        // Verify the expressions are the same
        System.assertEquals(CRON_EXP, ct.CronExpression);
            System.assertEquals(0, ct.TimesTriggered);
            System.assertEquals('2022-09-03 00:00:00', String.valueOf(ct.NextFireTime));

        Test.StopTest();
    }

}
```

Real Time scenario where you can use Batch apex:

Business Use Case 1 :

We use batch apex for bulk processing. Let's say you want to update thousands to accounts on a regular basis or (even if just one time) and real time update is not a MUST HAVE requirement. You know how you are bound by governor limits for all the retrievals , updates , inserts.

Batch class has much wider governor limits . Its ar async operation and you can run a nightly job and process all these records without hitting limits.

Examples :
I used batch class for a scenario where I had to send email to all contract owners 15 days before paymem due date. There were thousands of contracts nad at one time there could be many contract owners meeting this criteria. Also, apart from sending email ,I had to update a flag on contract. So, I schedules a daily nightly batch job for this.

Business Use Case 2:

An organization's sales team is using Salesforce CRM primarily to manage their Leads, Contacts, and opportunities. This is akin to a retail setting whereby there are multiple physical locations and each location has its own sales team and sales manager (think car dealership). In this setting, Leads (and/or Contacts) enter through multiple channels such as internet marketing, phone calls, walking through the front door, etc and are assigned to a sales rep based on a round-robin format. That sales rep becomes the owner of the Lead, but only has protection for a certain period of time before the Lead is redistributed.

The protected period of time is based on the sales rep's last activity to the Lead/Contact. In other words, if the rep is not actively working the Lead/Contact then they lose protection, and if that Lead/Contact ends up buying a product then the rep will not be credited for the commission.

The question then became, how can we automate this process of removing Lead protection based on the sales rep's activity? One of the great features of working with the Force.com platform is that there are always multiple ways to solve these types of problems. And these are good problems to solve because the solutions free up people's time and help advance business. In this situation, the basic design for this scenario is that when the number of days since the last activity is greater than 5 days, revert the ownership of the Lead/Contact from the sales rep to the sales manager. This lets the sales manager redistribute the Lead/Contact to another sales rep.

The first step was to create a new formula field that returns a Number and call it Days_Since_Last_Activity. The formula is this: Today() - LastActivityDate. At this point, you might think about creating a Workflow Rule that says when Days_Since_Last_Activity is greater or equal to 5 then Update the Owner field. However, a Workflow Rule is only fired under three circumstances: a)Only when a record is created; b) Every time a record is created or edited or; c) When a record is edited and did not previously meet the criteria. This means that the workflow rule won't get fired until the record gets edited, and we want the ownership to change immediately upon the passing of the protected period regardless of whether or not someone edits the record.

The solution in this case is to use Batch Apex to query the database for all Leads/Contacts (we use Contacts in the example code) that have crossed the protected period, i.e. the Days_Since_Last_Activity fields is greater than 5. Then, we reassign ownership to the correct sales manager based on which physical location this Contact is associated with. Next, we create a new Task associated with this Contact so that the Days_Since_Last_Activity gets reset. Lastly, we schedule this Batch Apex to run each night so that the ownership is being recalculated on a daily basis.

Below are the Apex and Test Classes that make this work.

```
global class UpdateAllContacts implements Database.Batchable<sObject>{

   //This is the query that is passed to the execute method.  It queries all of the Contacts who
have passed
   //the protected period.
```

```
  String query = 'Select Id, Club_Location__C, OwnerId FROM Contact WHERE
Days_Since_Last_Activity__c > 5';

  global database.queryLocator start(Database.BatchableContext BC) {
    return database.getQueryLocator(query);

  } //close start method

  global void execute(Database.BatchableContext BC, list <Contact> scope) {

    List <Task> taskList = new List<Task>();

    // Iterate through the whole query of Contacts and transfer ownership based on the location.
    // This example only has two locations.
    // Create a Task that's associated with each Contact.  This resets the Days Since Last
Activity formula field.
    for(Contact c : scope) {
      if(c.Location__c == 'Location 1') {
        c.OwnerId = 'XXXXXXXXXXXXXXXXX';
        Task tsk = new Task();
        tsk.WhoId = c.Id;
        tsk.ActivityDate = System.today();
        tsk.Status = 'Completed';
        tsk.Subject = 'Ownership Transferred';

        taskList.add(tsk);

      } //close if statement
      else {
        c.OwnerId='XXXXXXXXXXXXXXXXX';
        Task tsk = new Task();
        tsk.WhoId = c.Id;
        tsk.ActivityDate = System.today();
        tsk.Status = 'Completed';
        tsk.Subject = 'Ownership Transferred';

        taskList.add(tsk);
      } //close else
    }  close for-loop

    try {
      insert taskList;
    } catch (system.dmlexception e) {
      System.debug('Tasks not inserted: ' + e);
    }
```

```
    try {
      update scope;
    } catch (system.dmlexception e) {
      System.debug('Scope not updated: ' + e);
    }

  } //close execute method

  global void finish(Database.BatchableContext BC) {

    AsyncApexJob a = [Select Id, Status, NumberOfErrors, JobItemsProcessed,
      TotalJobItems, CreatedBy.Email
      from AsyncApexJob where Id =
      :BC.getJobId()];

    // Create and send an email with the results of the batch.
   Messaging.SingleEmailMessage mail = new Messaging.SingleEmailMessage();

    mail.setToAddresses(new String[] {a.CreatedBy.Email});
    mail.setReplyTo('batch@mycompany.com');
    mail.setSenderDisplayName('Batch Processing');
    mail.setSubject('Contact Update ' + a.Status);
    mail.setPlainTextBody('The batch apex job processed ' + a.TotalJobItems +
    ' batches with ' + a.NumberofErrors + ' failures.');

    Messaging.sendEmail(new Messaging.SingleEmailMessage[] { mail });

  } //close finish method
} //close class
```

Apex Class Used to Schedule the Batch Apex Class:

```
global class ScheduleUpdateContacts implements Schedulable
{

  global void execute(SchedulableContext SC) {
    UpdateAllContacts uac = new UpdateAllContacts();
    database.executebatch(uac);
  } //close execute method
} //clo
```

@isTest

```
private class UpdateAllContactsTest {

   static testMethod void TestUpdateAllContacts() {

      // User Id's for the two sales managers.
      String aId = 'XXXXXXXXXXXXXXX';
      String bId = 'XXXXXXXXXXXXXXX';

      List <Contact> contacts = new List <Contact>();
      List <Task> tasks = new List <Task>();

      Test.StartTest();

      // Create 50 Contacts and assign them to the sales manager of the opposite location.
      for (integer i=0; i<50; i++) {
         Contact c = new Contact(FirstName='Test',
                  LastName='Contact'+ i,
                  Location__c = 'Location 1',
                  OwnerId = bId);
         contacts.add(c);

      } //close for-loop

      // Create 50 more Contacts and assign them to the sales manager of the opposite location.
      for (integer i=0; i<50; i++) {
         Contact c = new Contact(FirstName='Test',
                  LastName='Contact' + i,
                  Location__c = 'Location 2',
                  OwnerId = aId);
         contacts.add(c);

      } //close for-loop

      insert contacts;

      List <Contact> cont = [Select ID, FirstName from Contact Where FirstName='Test' limit
200];

      // Create a Task for each Contact that was just inserted.  Set the date of the task so that it
will set the
         Last Activity Date to a date older than your protection period.
      for (Integer i=0; i<100; i++) {
         Task tsk = new Task();
         tsk.WhoId = cont.get(i).Id;
         tsk.ActivityDate = System.today() - 15;
```

```
        tsk.Status = 'Completed';
        tsk.Subject = 'Test Subject';

        tasks.add(tsk);

      } // close for-loop

      try {
        insert tasks;
      } catch (System.DMLexception e) {
        System.debug('Task List not inserted: ' + e);
      }

      // Call the Batch Apex method.
      UpdateAllContacts uac = new UpdateAllContacts();
      ID batchprocessid = Database.executeBatch(uac);
      Test.StopTest();

      AsyncApexJob async = [Select Id, Status, NumberOfErrors, JobItemsProcessed,
TotalJobItems from AsyncApexJob where Id = :batchprocessid];
      System.debug('Final results are ' + async);

      System.AssertEquals(async.NumberOfErrors, 0);
       System.AssertEquals([Select count() from Contact Where OwnerId=:aId AND
FirstName='Test'], 50);
      System.AssertEquals([Select count() from Contact Where OwnerId=:bId AND
FirstName='Test'], 50);
      System.AssertEquals([Select count() from Task Where Subject = 'Test Subject'], 100);

    } //close testmethod

} //close Class

global class TestScheduledApexFromTestMethod implements Schedulable
{
// This test runs a scheduled job at midnight Sept. 3rd. 2022
 public static String CRON_EXP = '0 0 0 3 9 ? 2022';

  global void execute(SchedulableContext ctx)
{
    CronTrigger ct = [SELECT Id, CronExpression, TimesTriggered,NextFireTime
                      FROM CronTrigger WHERE Id = :ctx.getTriggerId()];

    System.assertEquals(CRON_EXP, ct.CronExpression);
    System.assertEquals(0, ct.TimesTriggered);
```

```
    System.assertEquals('2022-09-03 00:00:00', String.valueOf(ct.NextFireTime));

    Account a = [SELECT Id, Name FROM Account WHERE Name =
                                        'testScheduledApexFromTestMethod'];

    a.name = 'testScheduledApexFromTestMethodUpdated';
    update a;
  }
}
```

Note : you must guarantee that the trigger won't add more scheduled classes than the 100 that are allowed.

Test Class to for schedule Apex:

```
@istest
class TestClass {

  static testmethod void test() {
  Test.startTest();

    Account a = new Account();
    a.Name = 'testScheduledApexFromTestMethod';
    insert a;

    // Schedule the test job
    String jobId = System.schedule('testBasicScheduledApex',
    TestScheduledApexFromTestMethod.CRON_EXP,
      new TestScheduledApexFromTestMethod());

    // Get the information from the CronTrigger API object
    CronTrigger ct = [SELECT Id, CronExpression, TimesTriggered,
     NextFireTime
      FROM CronTrigger WHERE id = :jobId];

    // Verify the expressions are the same
    System.assertEquals(TestScheduledApexFromTestMethod.CRON_EXP,ct.CronExpression);
      // Verify the job has not run
    System.assertEquals(0, ct.TimesTriggered);
    // Verify the next time the job will run
    System.assertEquals('2022-09-03 00:00:00',
      String.valueOf(ct.NextFireTime));
    System.assertNotEquals('testScheduledApexFromTestMethodUpdated',
                                [SELECT id, name FROM account WHERE id = :a.id].name);
  Test.stopTest();
```

```
    System.assertEquals('testScheduledApexFromTestMethodUpdated',
                        [SELECT Id, Name FROM Account WHERE Id = :a.Id].Name);

  }
}
```

To schedule an Apex class to run at regular intervals, first write an Apex class that implements the Salesforce-provided interface Schedulable.

The scheduler runs as system: all classes are executed, whether the user has permission to execute the class or not. For more information on setting class permissions, see "Apex Class Security Overview" in the Salesforce online help.

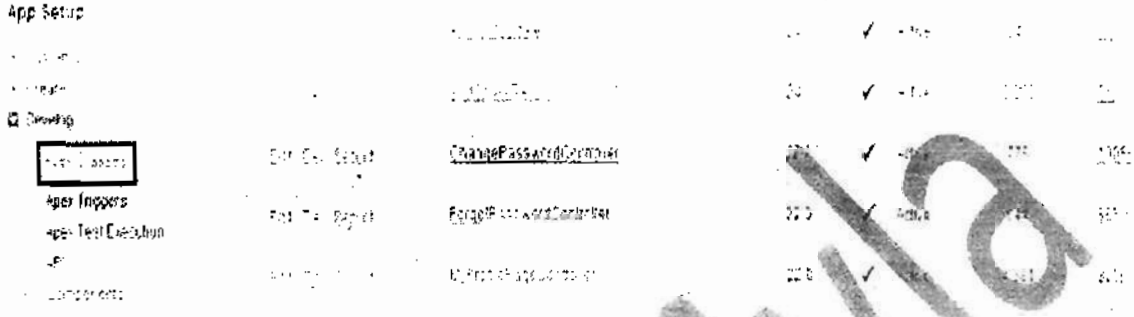
To monitor or stop the execution of a scheduled Apex job using the Salesforce user interface, go to **Setup -->Monitoring --> Scheduled Jobs**. For more information, see "Monitoring Scheduled Jobs" in the Salesforce online help.

The Schedulable interface contains one method that must be implemented, execute.
global void execute(SchedulableContext sc){}

Example:

```
global class scheduledAccountUpdate implements Schedulable
{
  global void execute(SchedulableContext SC)
  {
    AccountUpdate acc = new AccountUpdate('Description','Updated Account');
  }
}
```

To get details about AccountUpdate Class go to http://www.infallibletechie.com/2012/05/batch-apex.html
Go to Setup à Develop à Apex Classes and then click 'Schedule Apex' button.


App Setup
Apex Triggers
Apex Test Execution

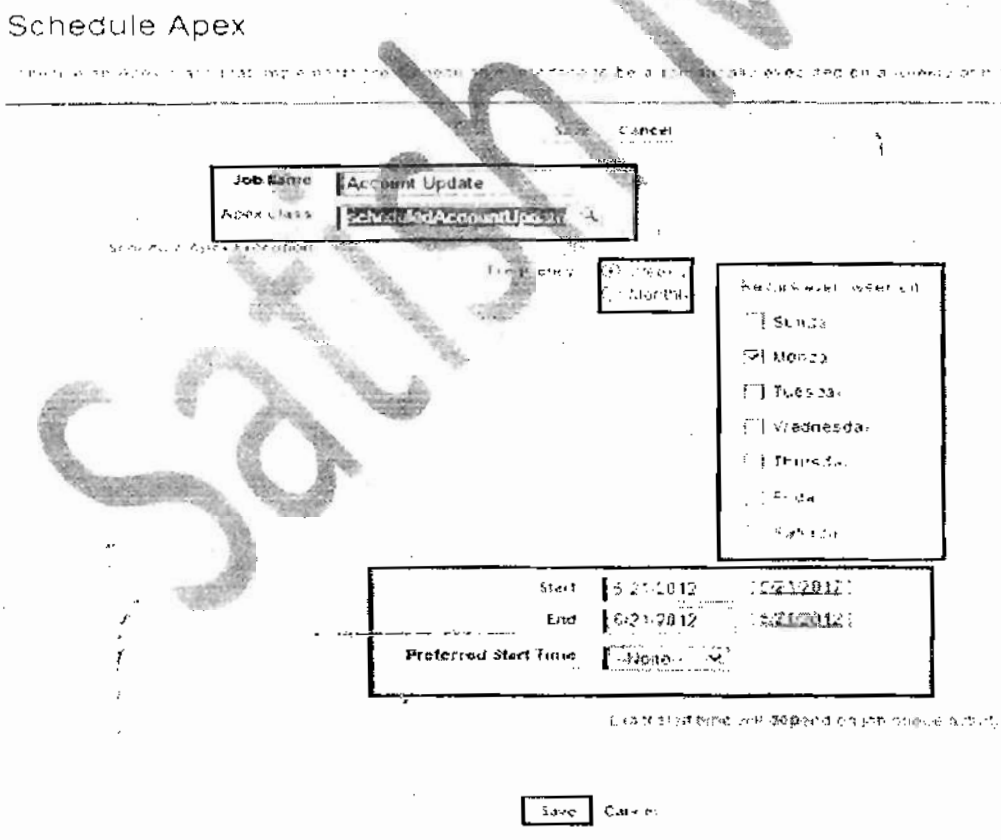

Schedule Apex
Cancel
Job Name
Account Update
Apex class
Monday
Start
End
Preferred Start Time
Save

Figure clearly explains how to schedule Batch Apex. The frequency may be set to Weekly or Monthly. Start and End dates are important. Preferred Start Time is also important. The job will be sent to Apex Job queue. The preferred Start Time won't be the exact time because it depends upon the job queue. The job will be executed after other jobs in the job queue have been executed.

Once you've successfully saved your Apex Classes go to Setup –> Develop –> Apex Classes –> Schedule Apex. Use the class above that implements the *Schedulable* interface and select the frequency that you want the class to run.

Batch Apex Governor Limits

Keep in mind the following governor limits for batch Apex:

- Up to five queued or active batch jobs are allowed for Apex.
- The maximum number of batch Apex method executions per a 24-hour period is 250,000 or the number of user licenses in your organization multiplied by 200, whichever is greater.
- The batch Apexstart method can have up to 15 query cursors open at a time per user.
- A maximum of 50 million records can be returned in the Database.QueryLocator object. If more than 50 million records are returned, the batch job is immediately terminated and marked as Failed.
- If the start method of the batch class returns a QueryLocator, the optional scope parameter of Database.executeBatch can have a maximum value of 2,000.
- of Database.executeBatch, Salesforce chunks the records returned by the startmethod into batches of 200, and then passes each batch to the execute method. Apex governor limits are reset for each execution of execute.
- The start, execute, and finish methods can implement up to 10 callouts each.
- Only one batch Apex job's start method can run at a time in an organization. Batch jobs that haven't started yet remain in the queue until they're started. Note that this limit doesn't cause any batch job to fail and execute methods of batch Apex jobs still run in parallel if more than one job is running.

Schedule Apex Exception – "No Apex Classes Found"?

So I ran across this issue a couple of months ago and forgot to blog about it for my reference and anyone else that runs across this problem. So here's the issue, I have an Apex class scheduled in production that runs another Apex class. I've made some changes to this second class and need to deploy it to production. Before you do that, for Force.com platform requires you to delete the scheduled job for that class since they are related.

No problem. So I deleted the scheduled class, pushed my new code to production (passes all tests! w00t!) and went back to schedule this class to run and saw this error:

Continued in next page...

Frequently asked interview questions with real time scenarios given in the next pages

Testing :-

→ If you want to validate whether the application what we have created is working as expected.

→ there are two ways of testing an application.

1) using salesforce user interface.
2) Testing bulk functionality. (upto 200 records can be passed through your code)

→ Before you deploy your code & package to the Force.com App exchange the following conditions should be satisfied.

i) Atleast 75% of your Apex Code must be covered by unit test.
ii) All the test cases will be successfull.

NOTE :- When you deploy any code production will be executed Test methods and the test classes are not part of Apex Code coverage.

→ Every triggers must have some test case.

→ All classes & triggers should compile successfully.

What are the factors that needed to be tested in Apex programming:-

1) Single Action :- this is to test a single record, produces the correct expected result.

2) Bulk actions :- Any Apex code, whether a triggers, a class & an extension, may be invoked for 1 to 200 records. You must test not only the single record case, but the bulk cases as well.

3) positive behaviour :- Test to verify that the expected behaviour occurs through every expected permutation, that is, that the users filled out every thing correctly and did not go past the limits.

4) Negative behaviour :- there are likely limits to your applications, such as not being able to add a future date, not being able to specify a negative amount & so on.

5) Restricted users.

Apex unit test :-

→ unit test are the class methods that verify whether a particular piece of code is working properly & not.

→ unit test method takes no arguments & commits no data to the database, will not send any emails.

Testmethod keyword :-

→ When you want to create a test method then should be prefixed with keyword testmethod.

→ All the test methods are static.

Eg:- public class myclass

```
{
    Static testMethod void myTest()
    {
    // Add test method logic using System.assert(),
                    System.assertEquals()
    // and system.assertNotEquals() here
    }
}
```

NOTE:- Test methods can not be defined in Apex trigger.

@isTest Annotation :-

If you define any method as @isTest then the method is terminated test method just like what we have defined.

→ If any class is defined with annotation @isTest then that class is defined as test class.

Eg1:- @isTest
private class myclass
{
Static testMethod void myTest()
{
//code_block
}
}

This is the same test class as in previous example but it defines the test method with the isTest annotation instead.

Eg2:- @isTest
private class myclass
{
@isTest Static void myTest()
{
//code_block
}
}

NOTE:- class defined with the isTest annotation don't count against your Organization limit of 8MB for all apex code.

Eg:-

```
@isTest
private class MyTestclass
    {
    //Methods for testing
@isTest Static void test1()
    {
     //Implement test code
       }
@isTest Static void test2()
      {
      //Implement test code
        }
      }
```

→ Test classes must be defined as isTest with access specifier as Public/private.

→ By default access level is private.

→ Methods of the test class can be called only from the running method.

→ A test method can not be invoked in non-test method.

Eg:-

Apex class (TV Remote Control):-

```
public class TvRemoteControl
    {
    //volume to be modified
      Integer volume;
  //constant for maximum volume value
   Static final Integer MAX_VOLUME=50;
   //constructor
  public TvRemoteControl(Integer v)
     {
    //set initial value for volume
        volume =4;
         }
public Integer increaseVolume(Integer amount)
        {
        volume +=amount;
    if(volume > MAX_volume)
          {
          volume = MAX_VOLUME;
             }
        return volume;
           }
Public Integer decreaseVolume(Integer amount)
        {
         volume -= amount;
```

```
    if(volume<0)
      {
      volume=0;
       }
     return volume;
       }
public static String getMenuOptions()
   {
   return 'AUDIO SETTING - VIDEO SETTINGS';
       }
   }
```

Test class :- (TVRemoteControlTest);

```
@isTest
class TVRemoteControlTest
  {
  @isTest static void testVolumeIncrease()
     {
  TVRemoteControl rc = new TVRemoteControl(10);
  Integer newVolume = rc.increaseVolume(15);
  System.assertEquals(25, newVolume);
       }
  @isTest static void testVolumeDecrease()
     {
  TVRemoteControl rc = new TVRemoteControl(20);
```

```
Integer newVolume = rc.decreaseVolume(15);
System.assertEquals(5, newVolume);
    }
@isTest static void testVolumeDecreaseUnderMin()
    {
TVRemoteControl rc = new TVRemoteControl(10);
Integer newVolume = rc.decreaseVolume(100);
System.assertEquals(0, newVolume);
    }
@isTest static void testGetMenuOptions()
    {
//Static method call. No need to create a class instance
String menu = TVRemoteControl.getMenuOptions();
System.assertNotEquals(null, menu);
System.assertNotEquals('', menu);
    }
}
```

NOTE:- From API 28.0 test methods no longer be reside in the non-test class.

Accessing private members of the class in test class :-

→ private members of the class can't be accessed outside the class. This will create a problem in checking private members.

→ If you want to check the private members of the class add annotation @TestVisible to before the private/Protected members of the class.

→ When you keep the annotation @TestVisible before the private members those are visible with in a test class.

Eg:-

Apex class (visible sample class):

```
public class visibleSampleclass
    {
     //private members variables
@TestVisible private Integer recordNumber =0;
@TestVisible private String areacode ='(415)';
 //public member variable
 public Integer maxrecords =1000;
  //private inner class
  @TestVisible class Employee
      {
```

```
String fullName;
String phone;
//Constructor
@TestVisible Employee (String s, String ph)
    {
    fullname = s;
     phone = ph;
      }
    }
//private method
@TestVisible private String privateMethod (Employee e)
    {
System.debug ('I am private.');
  recordNumber++;
String phone = areaCode + ' ' + e.phone;
String s = e.fullName + '\'s phone number is ' + phone;
System.debug(s);
  return s;
    }
//public method
public void publicMethod()
      {
       maxRecords++;
```

```
System.debug ('I am public');
    }
@IsTestVisible private class MyException extends VisibleSampleclass()
  {
```

Testclass (visible sample class Test):-

```
//Test class fo visible sample class
@isTest
private class visibleSampleclassTest
  {
//this test method can access private membess of anothes class
//that are annotated with @IsTestVisible
Static testmethod void test1()
  {
visibleSampleclass Sample = new VisibleSampleclass();
//Access private inner class
VisibleSample class. Employee emp = new VisibleSampleclass
                                    Employee ('Joe Smith', '555-1212');
//call private method.
String s = Sample.privateMethod (emp);
//
```

```
//verify result
System.assert(
    s.Contains('(510)') &&
    s.Contains('Joe Smith') &&
    s.Contains('555-1212'));
    }
Static testmethod void test2()
    {
     try
      {
   throw new VisibleSampleClass.MyException('thrown from a test');
        }
  Catch(VisibleSampleClass.MyException e)
      {
       //Handle Exception
        }
      }
Static testmethod void test3()
    {
     //Access public method
    // NO @TestVisible is used
VisibleSampleClass Sample = new VisibleSampleClass();
   Sample.publicMethod();
      }
}
```

Batch Apex example with testcase :-

Batch Apex class :-

```
global class Batchclass implements Database.Batchable<sObject>
    {
  public String query = 'select id from Account';
global Database.QueryLocator Start(Database.BatchableContext Bc)
      {
      return Database.getQueryLocator(query);
       }
global void execute(Database.BatchableContext Bc, List<SObject> scope)
    {
    for(Account a : (List<Account> scope)
        {
          System.debug(a);
         }
      }
  global void finish(Database.BatchableContext BC)
     {
      System.debug('finished');
      }
}
```

Test case :-

```
Private class TestBatchclass
{
Static testMethod void testBatchClass()
{
Batchclass bc = new BatchClass();
bc.query = 'Select Id from Account limit 200';
Test.startTest();
Database.executeBatch(bc,200);
Test.stopTest();
}
}
```

Schedule Apex example with testcase :-

Batch class :-

```
global class customersBatch implements Database.Batchable<Sobject>
    {
 global final String Query;
 global final String Entity;
 global final String Field;
 global final String value;
 global customersBatch (String q, String e, String f, String v) {
       query=q; Entity =e; Field=f; value =v;
       }
global Database.QueryLocator Start (Database.BatchableContext Bc)
    {
     return Database.getQueryLocator(query);
      }
global void execute (Database.BatchableContext Bc, List<Sobject> scope)
    {
     for(Sobject S: scope)
       {
      S.put(Field, value);
         }
     update scope;
      }
 global void finish (Database.BatchableContext Bc) {
         }
    }
```

```
global class CustomersSchedule implements Schedulable
    {
global void execute(SchedulableContext sc)
      {
      CustomersBatch cb = new CustomersBatch();
      Database.executeBatch(cb);
      }
    }
```

Test case :-

```
@isTest
private class scheduleTest {
 Static testMethod void myUnitTest()
      {
      Test.startTest();
 String CRON_EXP = '0 00 1 1 ? 2025';
 String jobId = System.schedule('testScheduledApex',
                CRON_Exp, new CustomersportalRevokeSchedulable());
 CronTriggers ct = [select id, CronExpression, TimesTriggered,
                NextFireTime from CronTrigger where id=:jobId];
  System.assertEquals(CRON_EXP, ct.cronExpression);
 System.assertEquals(0, ct.TimesTriggered);
 System.assertEquals('2025-01-01 00:00:00', string.valueOf
                                        (ct.NextFireTime));
      Test.stopTest();
      }
}
```

Triggers :-

Trigger is a Apex code that executes before & after on the following types of DML operations.

→ Insert
→ Update
→ Delete
→ Merge
→ Upsert
→ Undelete

Triggers are divided into 2 types

1) Before Triggers
2) After Triggers.

1) Before Triggers :-

Before triggers can be used to update & validate values of a record before they are saved to the database.

2) After Triggers :-

After triggers can be used to access field values of the records that are stored in the database and use this values to make changes in other records.

Syntax :-

```
trigger triggerName on objectName (trigger_events)
{
code_block
}
```

Where trigger_events can be comma seperated list of events.

Types of events in the triggers :-

1) Before insert
2) Before update
3) Before delete
4) After insert
5) After update
6) After delete
7) After undelete

Eg :-

```
trigger Sample on Account (before insert, after delete)
{
//code-block
}
```

NOTE:- Triggers can only contains keywords applicable to an inner class.

-> you do not have to commit the data manually, it automatically saves into database.

Trigger.New :-

Trigger.New is a context variable which contains list of new records which has caused the trigger to fire.

-> Trigger.New can be used in the trigger events.

1. Before insert
2. Before update
3. After insert
4. After update
5. After undelete

NOTE:- There is no concept called Trigger.New in delete operations.

Create a table customers

CID	Name	Age	phone
111	aaa	23	1234
222	bbb	34	3455
333	ccc	45	9876

Trigger.New in before insert :-

We have an object with three records.

CID	Name	Age	phone
111	aaa	28	1234
222	bbb	34	3455
333	ccc	45	9876

In this object if we are trying to insert new records, into Customer object.

444	ddd	34	1234
555	eee	23	3456

then the new records which we are trying to insert are stored in Trigger.New in before insert event. Which means

List<Customer__c> cus = Trigger.New ; // Ip 4

444	ddd	34	1234
555	eee	23	3456

These records are stored into Trigger.New.

NOTE :- The records which are stored in the Trigger.New we can directly perform changes in before insert.

Eg:- for(customers__c c:Trigger.New)

```
{
    c.Age__c=30;
}
```

NOTE :- Before insert event will occur before the new records are inserted into the database. So we can not retrieve the new records using DML operations in before triggers. i.e,

When we have customers table with following records.

CID	Name	Age	phone
111	aaa	23	1234
222	bbb	34	3455
333	ccc	45	9876

When we are trying to perform insert these two new records

444	ddd	34	1234
555	eee	23	3456

If there is any before insert triggers and we have written any DML in it.

trigger example on customer__c (before insert)

{

List<customer__c> my = [select cid__c, name, Age__c,
Phone__c from customer__c];

// this query will only fetch three records

111	aaa	23	1234
222	bbb	34	3455
333	ccc	45	9876

as remaining two records are not yet inserted.

}

Before Insert :-

These triggers will fire when we are trying to insert a new records into a specified object.

→ operations which we have written in trigger will be implemented before new records are saved to the database.

→ In before insert, Trigger.New stores the list of new records which we are trying to insert.

Before Insert :-

These triggers will be fired when we are trying to insert a new records into a specified object.

→ operations which we have written in triggers will be implemented before new records are saved to the database.

→ In before insert, Triggers.New stores the list of new records which we are trying to insert.

Scenario1 :-

When we are trying to insert new record into object. If there is any record existing with same account name it should Prevent duplicate record.

```
trigger accountinsert on Account (before insert)
{
    for(Account a : Triggers.New)
    {
        List<Account> mynew = [select id, Name from Account
                               where name = :a.name];
        if (mynew.size()>0)
        {
            a.Name.addError('Account with name is existing');
        }
    }
}
```

Testcase :-

```
@isTest
public class AccountInsert
{
public static testMethod void testinsert()
{
String addError;
String myname = 'madhuri';
Account a2=new Account(name = myname);
List<Account> x =[select name from Account where
                                name= myname];
if (x.size()<1)
{
System.assertEquals(0, x.size());
insert a2;
}
else
{
addError = 'Exisisting';
}
System.assertEquals('Exisisting', addError);
}
}
```

Scenario2 :-

Write a trigger to prefix Account name with 'Mr' when ever new record is inserted.

Trigger :-

```
trigger accountprefix on Account (before insert)
    {
      for(Account a: Trigger.New)
        {
        a.Name = 'Mr' +a.name;
          }
      }
```

Testcase :-

```
@isTest
Public class AccountInsert
    {
Public Static testMethod void testinsert()
      {
    Account a =new Account(name = 'sam');
        a.name = 'Mr.' +a.name;
        insert a;
      }
}
```

Scenario3 :-

When ever a new record is ~~inserted~~ created into account object. Before this new record is inserted into Account, delete all the Contacts records with this account name.

Triggers :-

```
trigger contactDeletion on Account (before insert)
    {
  List<String> myname = new List<String>();
    for(Account a : Trigger.New)
      {
      myname.add(a.name);
      }
List<Contact> myContacts = [select id, name from Contact
                            where name in :mynames];
  delete myContacts;
  }
```

Testcase:-

```
@isTest
public class AccountInsert
{
public static testMethod void testdeletion()
{
String myname = 'sam';
Account a = new Account(name = myname);
Contact con = new Contact(lastname = 'sam');
insert con;
Contact c = [select id, name from Contact where Name = :myname
                                                limit 1];
if(c!=null)
{
System.assertEquals(c.name, a.name);
delete c;
}
insert a;
}
```

After insert :-

→ This trigger will be fired after new records are successfully saved to the database.

→ We can use Trigger.New to refer to the list of new records which we have inserted.

→ On Trigger.New we can only perform read only operations.

→ On the new list of records we can perform DML operations.

NOTE :- On any records that are successfully saved to database. If we want to perform any changes on those records we have perform DML operations.

Eg:

CID	Name	Age	phone
111	aaa	23	1234
222	bbb	84	3455
333	ccc	45	9876
4¢			

When we insert a new records

444	ddd	32	3456
555	eee	56	7655

After triggers will be performed after Commiting the new records into database. which means

CID	Name	Age	Phone
111	aaa	23	1234
222	bbb	34	3455
333	ccc	45	9876
444	ddd	32	3456
555	eee	56	7655

After saving this records then after insert trigger will be called, so operation written in trigger will be performed after records are successfully inserted.

Scenario1 :- (Excercise)

When ~~ever a new~~ record ~~is~~ customer record is successfully ~~created. then update all the customers~~

When ever a new transaction is performed successfully then update the customer object balance field based on

if Transaction type = deposit, Balance = balance + amount;
= withdraw Balance = balance - amount;

NOTE:- Customer and Transaction has Lookup detail relation.

Scenario :-

When ever a new Contact is created for a account update the Corresponding account phone field with the new Contact phone field.

Triggers :-

```
trigger updatephone on contact (after insert)
    {
    List<Account> acc = new List<Account>();
    for(Contact c: Triggers.New)
        {
    Account a=[ select id, phone from Account where
                                id=:c.AccountId ];
        a.phone = c.phone;
        acc.add(a);
        }
    update acc;
    }
```

Testcase:-

```
@isTest
public class Testphone
    {
Static testmethod void updatephone()
    {
Account a = new Account (name = 'saro', phone = '123');
insert a;
Account my = [select id from Account where name = 'sam'];
Contact c = new Contact (lastname = 'kumar',
                         accountid = a.id, phone = '456');
 insert c;
Account acc = [select id, phone from Account where
                    id =: c.accountid limit 1];
   acc.phone = c.phone;
    update acc;
System.assertEquals(acc.phone, c.phone);
    }
}
```

update events in salesforce :-

There are two update events.

1. Before update
2. After update

Trigger.old and Trigger.New are in update events in salesforce.

→ We have a customer object with the following records

Customer

CID	Name	Age	phone
111	aaa	23	3455
222	bbb	34	2344
333	ccc	27	9876
444	ddd	56	2346

In the above table when we are toying to update the records.

333	ccc	27	9876	to	333 ccc 29 7654
444	ddd	78	2346	to	444 ddd 35 1234

Trigger.New :-

Trigger.New will store the set of records on which new values on which you are performing update.

i.e, Trigger.New will have

333	ccc	29	7654
444	ddd	35	1234

these two records will store into Trigger.New

Triggers.old :-

Triggers.old will store the set of records on which we are performing update. This store the records with old values.

i.e, in the above case Triggers.old will have

333	ccc	27	9876
444	ddd	78	2346

Event : Before update :-

When ever we are trying to update any records in the object. The operations which need to be performed before saving the changes to database are written in before update.

Eg:-

Customers

CID	Name	Age	phone
111	aaa	23	3455
222	bbb	34	2344
333	ccc	27	9876
444	ddd	56	2346

Step1 :-

update these records

333 ccc 29 7654

444 eee 35 1234

Step2 :-

Before update operations are performed

Step3 :-

111	aaa	23	3455
222	bbb	34	2344
333	ccc	29	7654
444	ddd	35	1234

→ When we modify the value of a record and click on update.

→ Before triggers will be called on object and all the operation written it will be performed.

↔ Records are updated with new values in the database.

NOTE:- If we want to make any changes in the values of new record we can directly perform using Trigger.NEW in before trigger.

→ We can not perform any changes in the records that are in Trigger.NEW using DML operations as they are not at commited in before update.

After update :-

the operations written in the afterupdate triggers will be fired when the changes that we have done are saved to the database.

Eg:- Customer

CID	Name	Age	phone
111	aaa	23	3455
222	bbb	34	2344
333	ccc	27	9876
444	ddd	56	2346

Step1:-

Update these records

333 ccc 29 7654

444 eee 35 1234

CID	Name	Age	phone
111	aaa	23	3455
222	bbb	34	2344
333	ccc	29	7654
444	ddd	35	1234

Operations written in after update performed now on new set of data.

→ When we make some changes on the records of the object and click on update.

→ First before triggers operations are performed on the object.

→ All completing before update operations all values in Triggers.New are updated to object.

→ Once records Triggers.New are updated to object then After triggers operations are performed.

NOTE:- In the after update triggers operation we can only read the data from triggers.

→ If we want to perform any changes on the records in after update triggers we have to write DML Statements.

Scenario1:-

When ever Customer record is updated, before updating the record create new record in test object with old values of customer record.

Trigger:-

```
trigger Customerupdate on Customer__c (before update)
{
List<Test__c> test = new List<Test__c>();
for(Customer__c x: Trigger.old)
{
Test__c t = new Test__c();
t.name = x.name;
t.salary__c = x.salary__c;
t.phone__c = x.phone__c;
test.add(t);
}
insert test;
}
```

Testcase:-

```
{
```

Testcase :-

```
@isTest
public class insertCustomer
{
Static testMethod void testCustomer()
    {
  Customer__c X = new Customer__c();
   X.name = 'satya';
   X.salary__c = 1000;
   X.phone__c = '123';
   insert X;
  Test__c t = new Test__c();
   t.name = x.name;
   t.phone__c = X.phone__c;
   t.salary__c = x.salary__c;
   insert t;
 System.assertEquals (t.name, x.name);
 System.assertEquals(t.salary__c, x.salary__c);
   X.name = 'satya k';
   x.salary__c = 30000;
   update x;
    }
}
```

Scenario 2:-

Trigger:-

```
trigger Customerupdate on customer__c (before update)
    {
List<Test__c> test = new List<Test__c>();
    for(customer__c x : Trigger.New)
      {
    Test__c t=new Test__c();
      t.name=x.name;
      t.salary__c=x.salary__c;
      t.phone__c=x.phone__c;
       test.add(t);
      }
   insert test;
  }
```

Testcase:-

```
@isTest
public class insertCustomer
  {
```

```
static testmethod void testCustomer()
    {
  Customer__c X = new Customer__c();
       X.name = 'satya';
       X.salary__c = 1000;
       X.phone__c = '123';
       insert X;

      X.name = 'satya k';
      X.salary__c = 30000;
       update X;
   Test__c t = new Test__c();
  System.assertEquals (t.name, X.name);
  System.assertEquals (t.salary__c, X.salary__c);
       t.name = X.name;
       t.phone__c = X.phone__c;
       insert t;
     }
}
```

Scenario3:-

To update the owner of a case based on the values selected within a pick list and populate the owner field with the createdBy field data. When we have selected any

FieldName = Status

pick list values =

Priced - (Initial)

Priced - (Re-priced)

Price file loaded

```
trigger CaseTrigger on case(before update)
{
  for(Case c : Trigger.new())
  {
  if(c.Status == 'priced - (Initial)'||c.status == 'priced -
                (Repriced)'||c.status == 'price file
                                    Loaded' {
    c.OwnerId = c.createdById;
    }
  }
}
```

Scenario4 :-

Write a triggers that will prevent a user from creating a lead that already exisist as a Contact. We'll use the lead/contact's email address to detect duplicates.

Lead is created & updated.

1. Lead has an email address
2. Try to find a matching Contact based on email address (using SOQL!)
3. If a match is found, give the user an error.
4. If a match is not found, do nothing.

```
trigger FindDupes on Lead (before insert, before update)
  {
 for(Lead mylead : Trigger.New) {
   if (mylead.Email != null)
     {
   List<Contact> dupes = [select id from Contact where
                          Email = :mylead.Email];
   if (dupes != null && dupes.Size() > 0) {
   String errorMessage = 'Duplicate Contact found!';
   errorMessage += 'Record ID is' + dupes[0].Id;
    mylead.addError(errorMessage);
     }
   }
}
```

Delete events in Salesforce triggers :-

There are two types of delete events.

1. Before delete
2. After delete

Trigger.old in before & after delete :- this will store the list of records which are trying to delete.

→ on this records we can only perform read only operations.

Customer

CID	Name	Age	phone
111	aaa	23	3455
222	bbb	34	2344
333	ccc	27	9876
444	ddd	56	2346

→ If we are trying to delete 333-ccc-27-9876
444-ddd-56-2346

then Trigger.old will contain these two records.

Scenario1:- When we are tying to delete Customer record delete all the corresponding child records from Test object. where Customer is lookup field in the loan object.

Triggers:-

```
trigger Customerupdate on Customer__c(after delete)
    {
List<Test__c> test = [select id from Test__c where
                      CDetails__c in : Trigger.old];
    delete test;
    }
```

Test case:-

```
@isTest
public class TestExam
    {
    static testmethod void custest()
        {
    Customer__c c = new Customer__c(name = 'satya', Salary__c = 1000);
    insert c;
    Test__c t = new Test__c(name = 'chandu', CDetails__c = c.id);
```

```
    insert t;
List<Test__c> test = [select id from Test__c where
                                    cpetails__c =:c.id];
    if (test.size()>0)
        {
      delete test;
       }
    delete c;
    }
}
```

NOTE :- there is no concept called trigger.New in after delete, before delete triggers.

Event: After undelete :-

When we are undelete the records from the recycle bin this operations written in after undelete will be performed.

Triggers.New :- The records which we have undeleted are stored in Triggers.New.

Scenario1 :-

when we create the opportunity with probability =50% then the opportunity owner will be automatically added to AccountTeam of the associated account for the opportunity.

Trigger :-

```
trigger addToAccountTeam on opportunity (after insert, after update)
    {
List<AccountShare> acc_share = new list<AccountShare>();
List<AccountTeamMember> acc_Team = new list<AccountTeamMember>
                                                    ();
  for(opportunity opp : Trigger.New)
    {
     if(opp.probability ==50)
       {
AccountTeamMember team=new AccountTeamMember();
   team.accountid = opp.accountid;
     team.userId = opp.ownerid;
AccountShare share = new AccountShare();
  Share.accountid = opp.accountid;
  Share.accountAccessLevel = 'Read/write';
  Share.opportunityAccessLevel = 'Read only';
```

```
share.caseAccessLevel = 'Read only';
    acc_share.add(share);
    acc_Team.add(tTeam);
        }
    }
if(acc_Team!=null)
    {
    insert acc_Team;
    }
if(acc_share!=null && acc_share.size()>0)
    {
    insert acc_share;
    }
}
```

Scenario02 :- Invoking the Apex class from the triggers.

When every new customers record is created/updated then incometax should be calculated based on salary and should be updated to field ittax (currency field).

Triggers:-

```
trigger CustomersUpdate on Customers__c (before insert,
                                         before update)
{
for(Customers__c cust : Triggers.New)
```

```
    {
    IncomeTax it = new IncomeTax();
    cust.Ittax__c = it.caluculateTax(cust.salary__c);
        }
    }
```

Testcase :-

```
@isTest
 Public class IncomeTest
  {
  Static testMethod void testIncome()
    {
  Customer__c  c=new Customer__c(name='sam', salary__c=30000);
   IncomeTax t=new IncomeTax();
   c.Ittax__c = t.calculateTax(c.salary__c);
    insert c;
  Customer__c  cust = [Select id, salary__c, Ittax__c from
                       Customer__c Where id=:c.id];
   cust.ittax__c = t.caluculateTax(cust.salary__c);
    update cust;
    }
}
```

Scenario 1:-

Apex class :-

```
public class IncomeTestax
{
public Decimal tax{set; get;}
public Decimal caluculateTax(Decimal amount)
{
  if(amount >500000)
  {
    tax = amount * 0.20;
  }
  else
  {
    tax = amount * 0.10;
  }
return tax;
}
}
```

Scenario3 :-

Before we insert a new record in the customer object calculate the 'ittax field value based on salary field value and then insert.

->If we delete any of the exisisting customer record then first Create new test object record with customer record values then delete customer record.

Apex class :-

```
global class TriggerExample
    {
global static void calculateTax (Customer_c[] cust)
      {
   for (Customer_c c:cust)
       {
       if(c.salary__c<50000)
           {
           c.ittax__c =c.salary_c*0.10;
           }
        else
          {
        c.ittax__c =c.salary__c*0.20;
          }
       }
```

```
}
global static void createTest(Customer__c[] cust)
    {
  List<Test__c> test = new List<Test__c>();
  for(Customer__c c:cust)
      {
    Test__c t = new Test__c(name=c.name, Salary__c =
                                        c.Salary__c);
        test.add(t);
        }
  insert test;
    }
}
```

Triggers:-

```
trigger CustomerUpdate on Customer__c (before insert,
                                       before delete)
    {
  if(Trigger.isBefore && Trigger.isInsert)
      {
    TriggerExample.CalculateTax(Trigger.New);
        }
```

```
else
  {
  if (Triggers.isBefore && Triggers.isDelete)
      {
      TriggersExample.createTest(triggers.old);
       }
    }
```

Test class :-

```
@isTest
public class TriggersExampleTest
   {
Static testmethod void testTriggersExample()
    {
Customers__c C=new Customers__c(name = 'satya', salary__c=40000);
  List<Customers__c> cust = new List<Customers__c>();
   cust.add(C);
   TriggersExample.calculateTax(cust);
    insert cust;
  Test__c t=new Test__c(name=C.name, salary__c = C.salary_c);
      insert t;
      delete c;
     }
  }
```

Usage of Trigger.NewMap and Trigger.OldMap :-

When ~~every~~ we try

Scenario :-

When ever we try to update the phone of account record then update the related Contact phone no. with the new ~~Contact~~ Account phone no before account record is updated.

When we delete the account record then delete the corresponding Contact records.

```
trigger Contactupdate on Account (before update, after delete)
  {
  if (trigger.isBefore && Trigger.isUpdate)
    {
    Map <Id, Account> mymap = Trigger.newMap;
    List <Contact> con = new List <Contact> ();
List <Contact> cons = [select id, phone, account id from Contact
                       where accountid in : mymap.keyset ()];
    for (Contact c : cons)
      {
    c.phone = mymap.get (c.accountid).phone;
      Con.add (c);
        }
```

```
update con;
  }
if (Trigger.isAfter && Trigger.isDelete)
   {
 Map<ID,Account> deleteacc = Trigger.oldMap;
List<Contact> mycontact =[select Id from Contact where
                    Accountid in: deleteacc.keyset()];
  delete mycontact;
     }
}
```

Recursive triggers :-

→ you want to write a trigger that creates a new record as part of its processing logic. However, that record may then cause another trigger to fire, which in turn causes another to fire, and so on. you don't know how to stop that recursion.

→ use a static variable in an Apex class to avoid an infinite loop. Static variables are local to the context of a web request (or test method during a call to runTests()), so all triggers that fire as a result of a user's action have access to it.

Scenario :-

Suppose there is a scenario where in one trigger perform update operation, which results in invocation of second trigger and the update operation in second trigger acts as triggering criteria for trigger one.

Class :-

```
public class utility
{
public static boolean isFutureUpdate;
}
```

Triggers :-

```
trigger updateSomething on Account (after insert, after update)
    {
/* this trigger performs its logic when the call is not from
 @future */
  if (utility.isfutureupdate != true)
      {
   Set<Id> idsToprocess = new Set<Id>();
   for (Account acct : trigger.new)
       {
    if (acct.NumberOfEmployees > 500)
        {
        idsToprocess.add(acct.Id);
        }
       }
   /* sending Ids to @future method for processing */
futureMethods.processLargeAccounts (idsToprocess);
      }
  }
```

Class :-

```
public class FutureMethods
    {
    @future
public static void processLargeAccounts(Set<Id> acctIDs)
        {
    List<Account> acctsToUpdate = new List<Account>();
    /* isFutureUpdate is set to true to avoid recursion */
    Utility.isFutureUpdate = true;
    Update acctsToUpdate;
        }
    }
```