MVC

Visualforce

Apex

Exception handling & Testing

MVC Architecture

**Controller here**

1. StandardController

**model**

**view**

1. Custome Controller(Apex )

Visual Pages/Components Standard Objects

Standard pages : Salesforce Pages custom objects

Model: What schema and data does salesforce uses to represent the system completely. In salesforce, we can say that sObjects are the model as every entity in salesforce is mapped to some sObject.

View: How the schema and data is represented. Visualforce is used to present the data to users.

Controller: How the interface actions. Controllers are used to perform the actions whenever users interact with visual force.

In SFDC

**1.** **Visual Force pages, Page Layouts, Tabs comes under View Layer of Model View controller .**

**2. Workflows, Apex Classes, Triggers comes under Controller part in Model View controller .**

**3. Objects, Fields, Relationships comes under Model Layer of Model View Controller .**

Data Layer—sObject –standard and custom

Business Layer--

Presentation layer

Force.com platform provide IDE, Compiler,Runtime Enviroment

Declarative & Programmatic:

Declarative Programmatic

Visualforce Pages

Visualforce components

Page Layout

Record types

User Interface

Apex Trigger, Apex Controller, Apex Classes

Bussiness

Workflow,validation,approval process

model

Data Model

Metadata API, Rest API, Bulk API

Object,field,relationship

Web Service calls

Force.com/Apex/Mypage

Application Logic/Apex

MyPage

Visual Force

Client

Where to use visual force?

* Email templates
* Mobile interfaces
* Generate pdfs
* Embedded in page layout—adding apex to force.com
* Custom tabs

Visual force is a platform that will allow developer to create a custom application that can be easily hosted on salesforce platform.

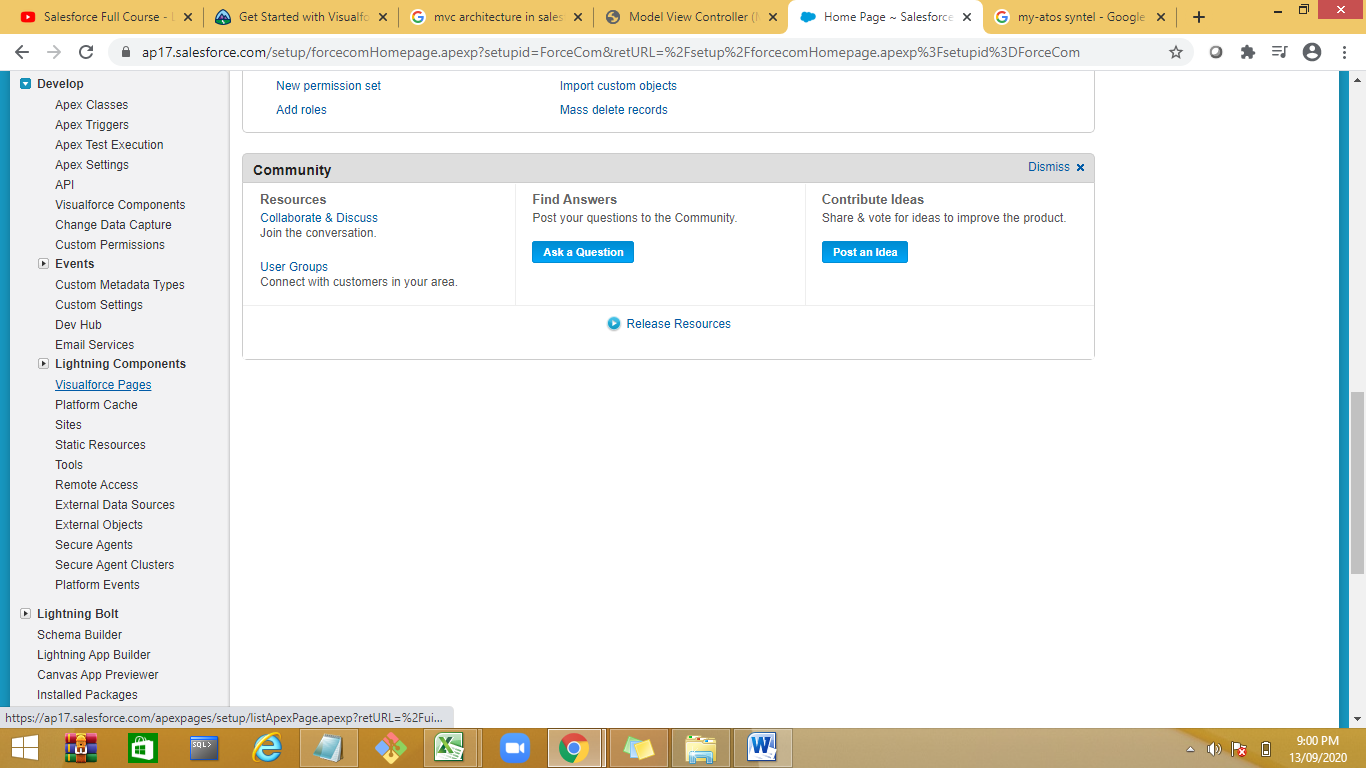
Visualforce page is made up of 2 components

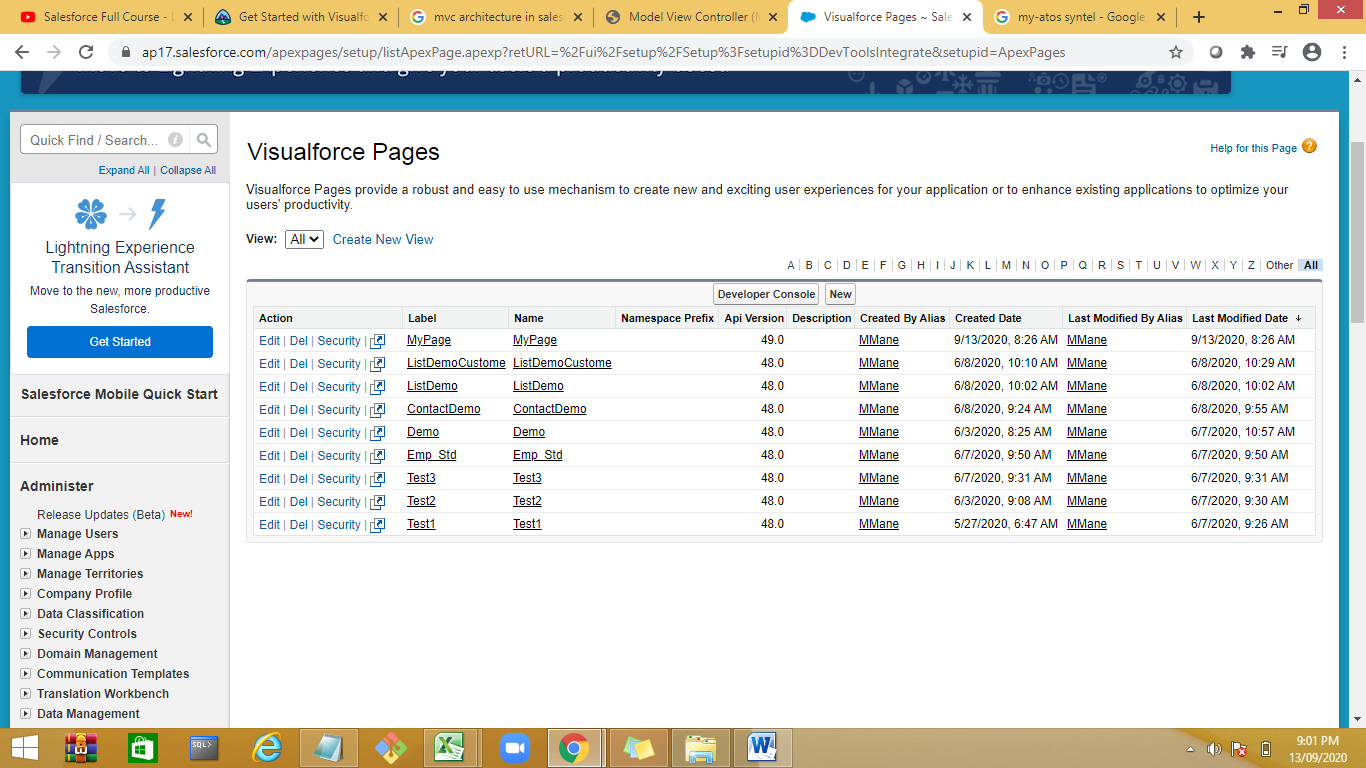
1.Visualforce markup-visualforce tags,html,javascript,any other web enabled code.use to define the user interface

2.visualforce controller—set of instructions to perform action when user interact with components.

Demo

Create own visualforce page:

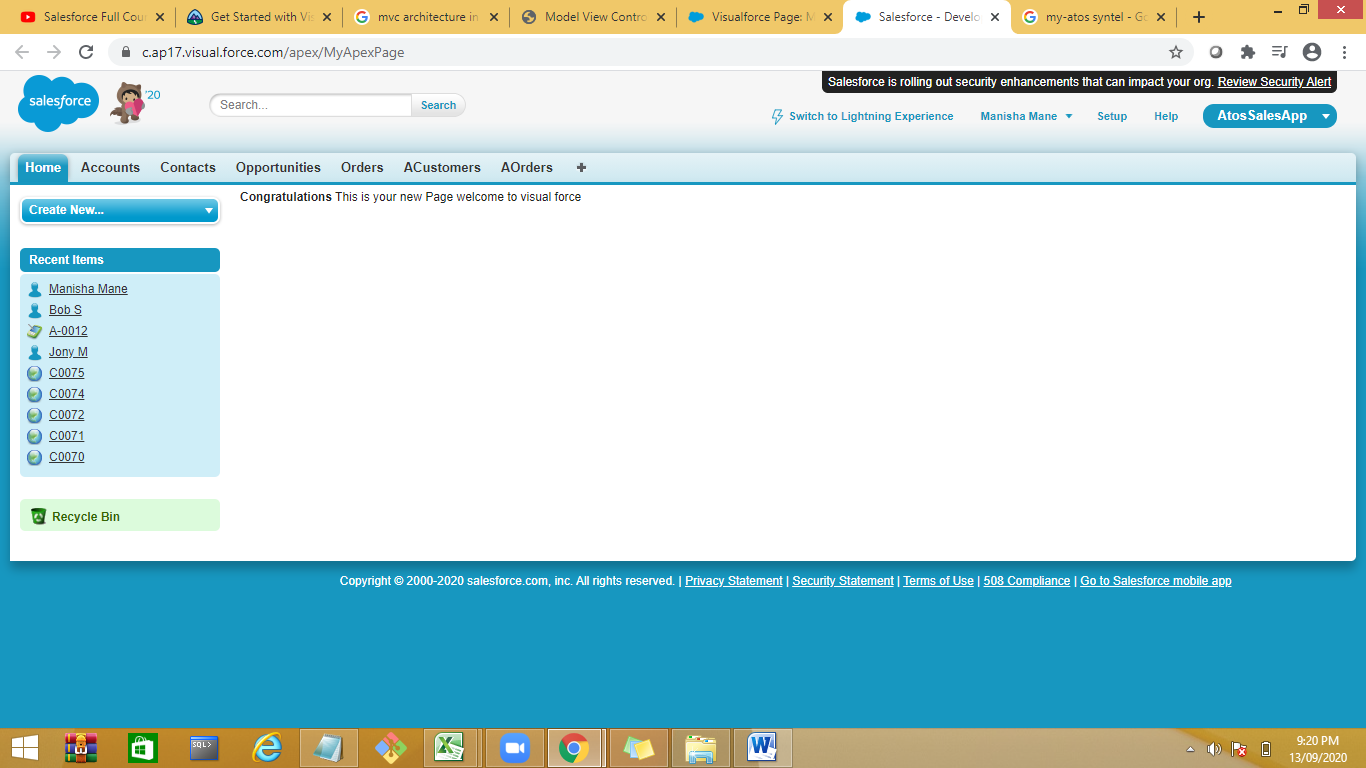




Create a new Page

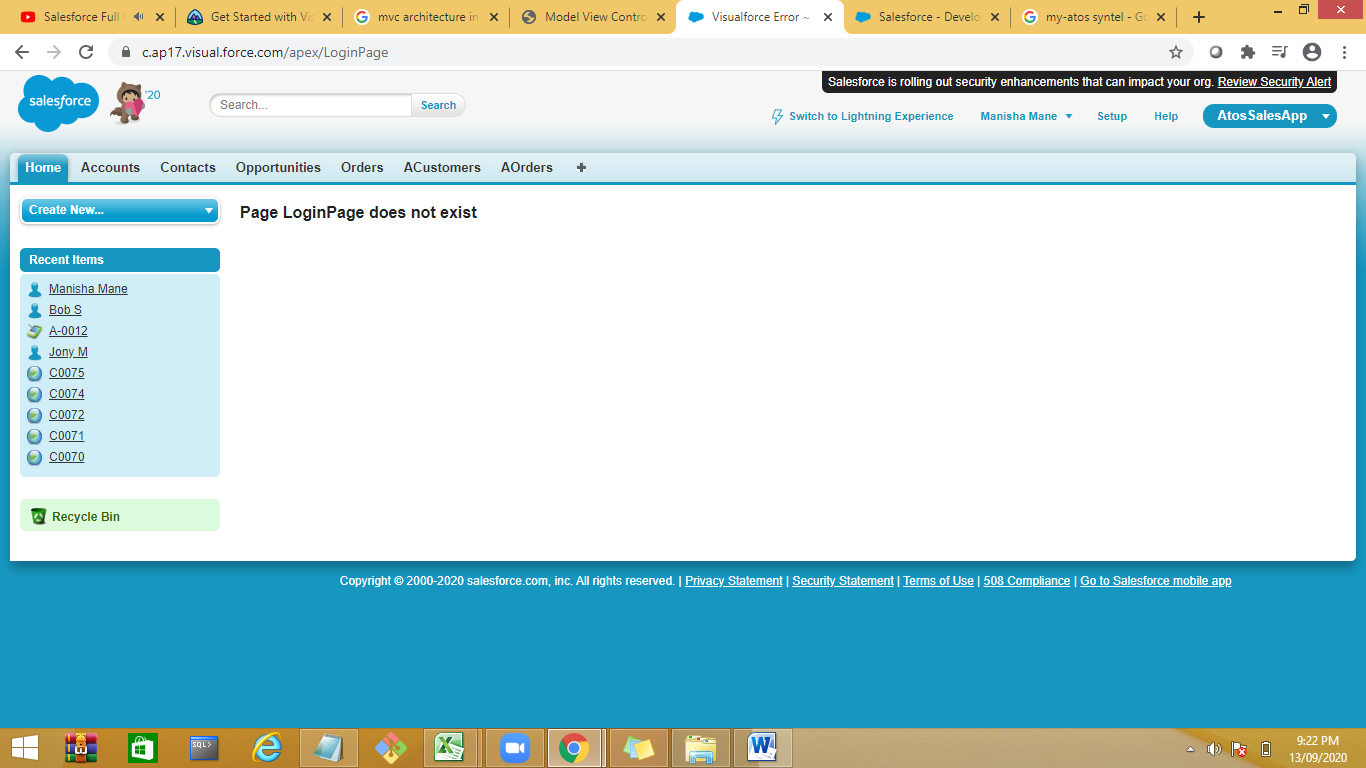


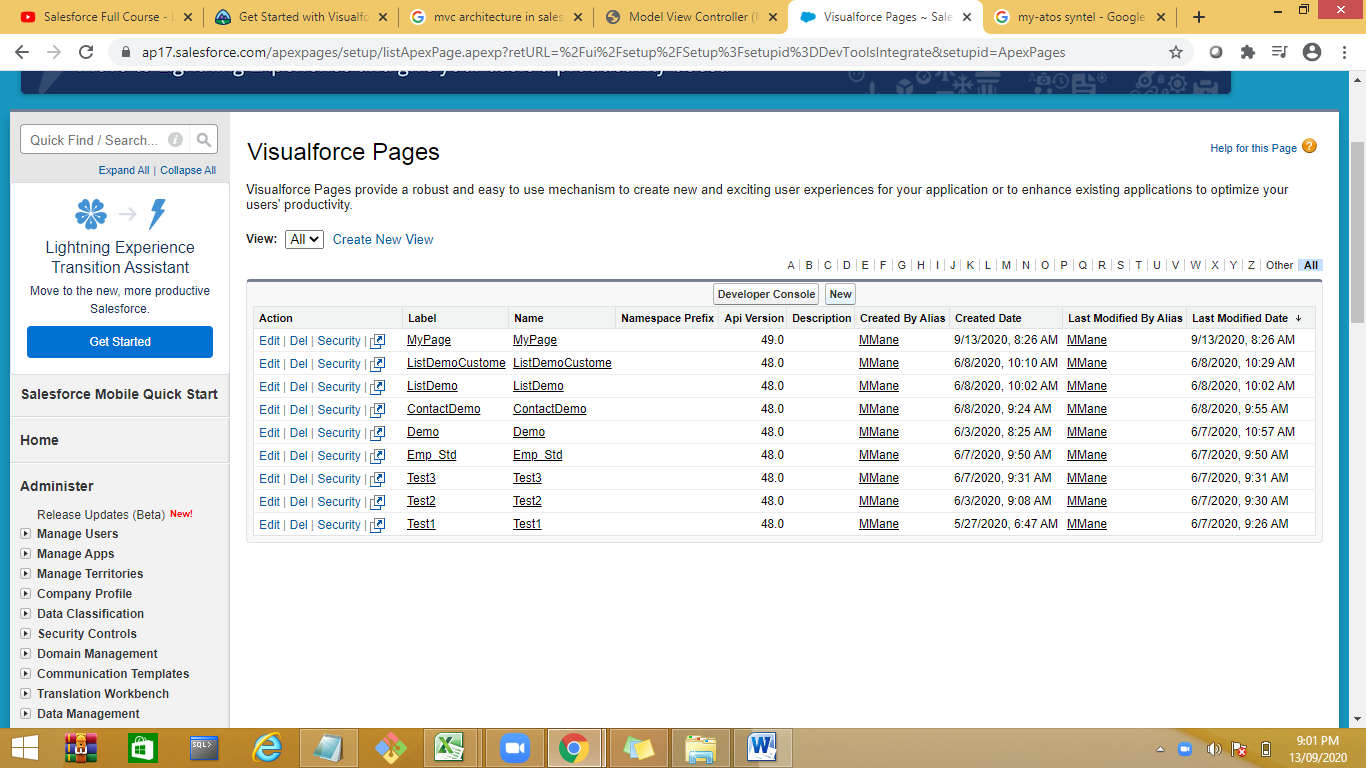
Use preview button to preview the page.



Inline editor

Initially will give an error as page doesnit exist.





Creating a Visualforce Page

Go to the link **developer console → File → New → Visualforce page**. The new window opens asking for a page name. Let us now call it **HelloworldPage**. Let us now write the code as shown in the following diagram

Example 1:

<apex:page >

<h1>

This is Login page

</h1>

<apex:pageBlock title="Salesforce ">

<apex:pageBlockSection title="Cloud List">

This are some of services available in Saleforce cloud!!!

<ul>

<li>Sales Cloud</li>

<li>App Cloud</li>

<li>Service Cloud</li>

</ul>

</apex:pageBlockSection>

</apex:pageBlock>

</apex:page>

--------------------------------------------------------------

Exam[ple 2

<apex:page >

<h1>

welcome {! $User.FirstName}

</h1>

</apex:page>

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

Example 3

<apex:page >

<h1>

welcome {! $User.FirstName} {!$User.LastName}

</h1>

<apex:form >

<apex:pageBlock >

<apex:pageBlockSection >

Enter Name : <apex:inputText id="name" required="true"/>

</apex:pageBlockSection>

<apex:pageBlockSection >

Enter Age :<apex:inputText id="age" />

</apex:pageBlockSection>

<apex:pageBlockSection >

Enter Password: <apex:inputSecret id="password" />

</apex:pageBlockSection><apex:pageBlockButtons >

<apex:commandButton action=**"{!save}"** value=**"Save"**/> </apex:pageBlockButtons>

</apex:pageBlock> </apex:form> </apex:page>

**Login Demo with Controller**

<apex:page controller="LoginSignUpController">

<apex:form >

<apex:pageBlock >

<apex:pageBlockSection >

<apex:inputField id="name" value="{!student.User\_Name\_\_c}" required="true"/>

</apex:pageBlockSection>

<apex:pageBlockSection >

<apex:inputField id="age" value="{!student.Roll\_No\_\_c}"/>

</apex:pageBlockSection>

<apex:pageBlockSection >

<apex:pageBlockSectionItem >

<apex:outputLabel value="password" for="pass"/>

<apex:inputSecret id="pass" value="{!student.Password\_\_c}" required="true"/>

</apex:pageBlockSectionItem>

</apex:pageBlockSection>

<apex:pageBlockSection >

<apex:inputField id="gender" value="{!student.mobile\_\_c}"/>

</apex:pageBlockSection>

<apex:pageBlockSection >

<apex:inputField id="gender" value="{!student.EmailId\_\_c}"/>

</apex:pageBlockSection>

<apex:pageBlockSection >

<apex:inputField id="gender" value="{!student.Gender\_\_c}"/>

</apex:pageBlockSection>

<apex:pageBlockSection >

<apex:inputField id="birthday" value="{!student.Birthday\_\_c}"/>

</apex:pageBlockSection>

<apex:pageBlockSection >

<apex:inputField id="birthday" value="{!student.Department\_\_c}"/>

</apex:pageBlockSection>

<apex:commandButton value="signup" action="{!signUp}"/>

</apex:pageBlock>

</apex:form>

</apex:page>

public class LoginSignUpController {

public Student\_\_c student{get;set;} //Assume it to be ur custom user

public LoginSignUpController()

{

student=new Student\_\_c();

}

public PageReference login()

{

if( (student.shivamindtree\_\_EmailId\_\_c == null) || (student.shivamindtree\_\_Password\_\_c == null))

{

return null;

}

List<Student\_\_c> students= [select Id,shivamindtree\_\_EmailId\_\_c,shivamindtree\_\_Password\_\_c from Student\_\_c];

for(Student\_\_c loginStudent:students)

{

if((loginStudent.shivamindtree\_\_EmailId\_\_c == student.shivamindtree\_\_EmailId\_\_c) && (loginStudent.shivamindtree\_\_Password\_\_c == student.shivamindtree\_\_Password\_\_c))

{

PageReference page = new PageReference('/apex/UserStart');

page.getParameters().put('studentId', loginStudent.Id);

page.setRedirect(true);

return page;

}

}

return null;

}

public PageReference signUp()

{

insert student;

PageReference pageRef = new PageReference(ApexPages.currentPage().getUrl());

pageRef.setRedirect(true);

return pageRef;

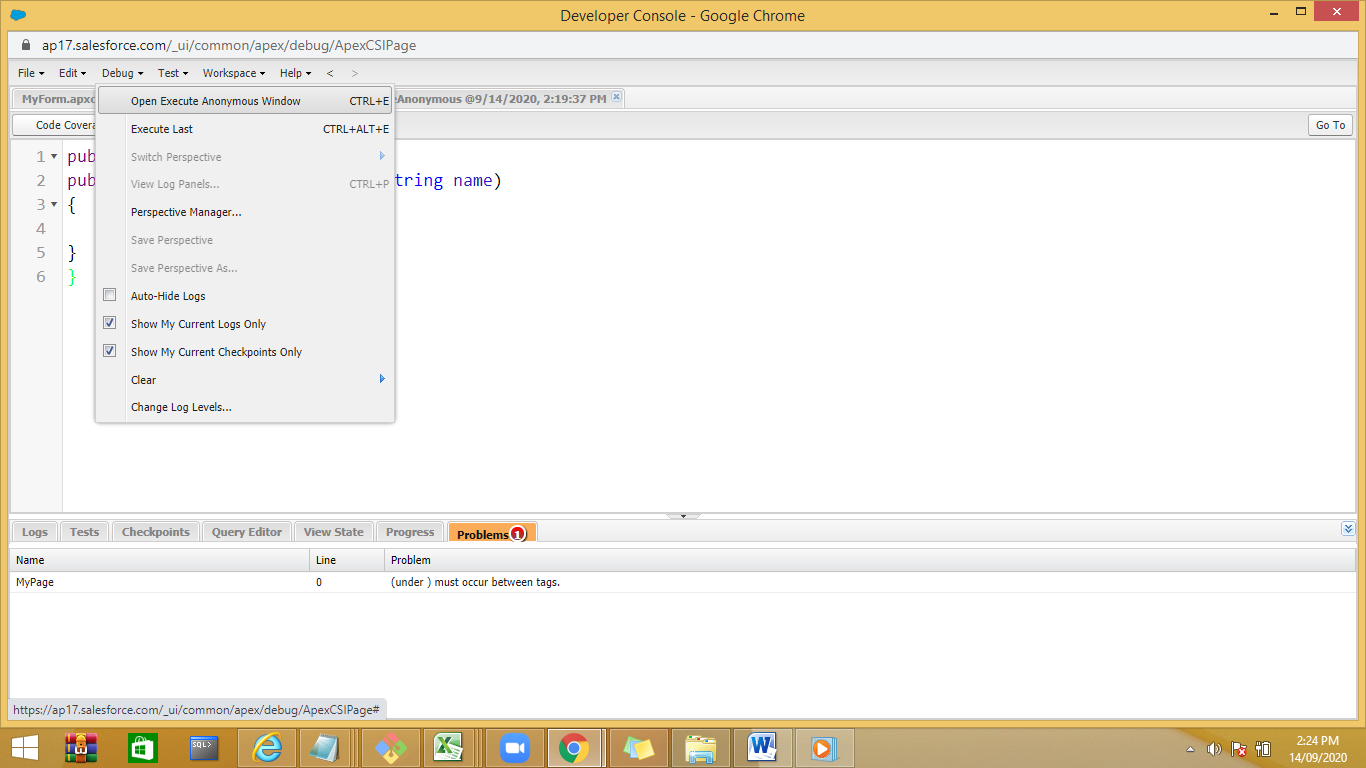
}

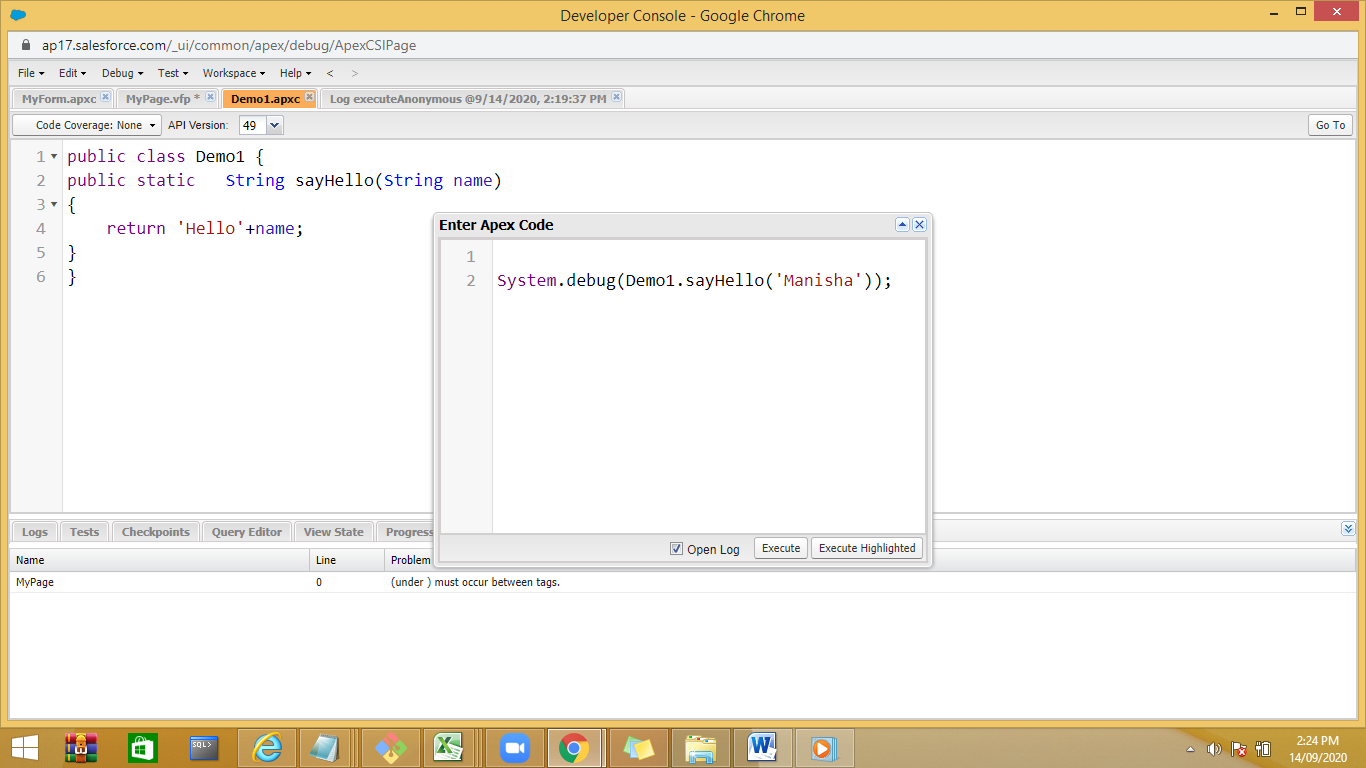
}

------------------------------------------------------------------------------------

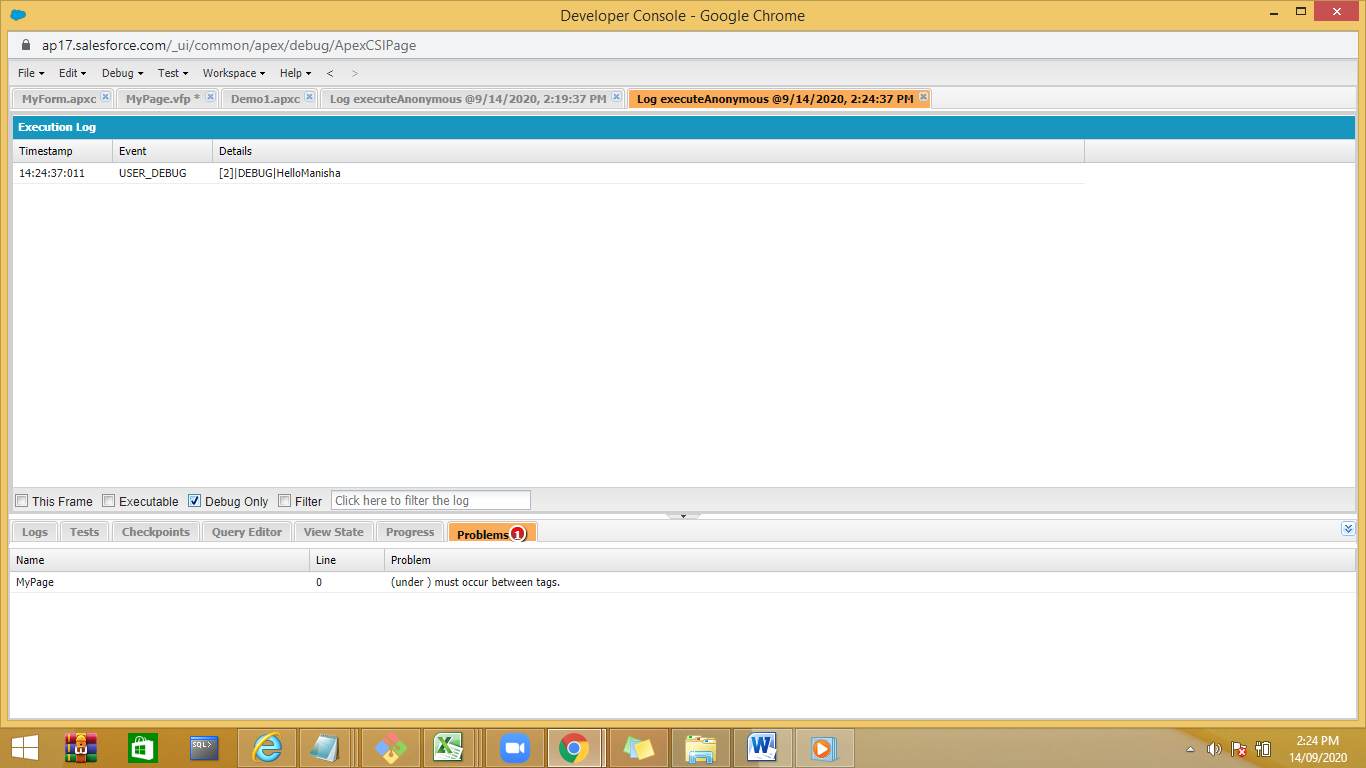
Open developer

Debug🡪





In logs select Debug Only



----------------------------------------------------------------------------------------

**Controller :Standard controller, Custome controller –defines the action . we can use standard controller for both standard object and custom object .**

**Binding UI element with Controller**

Demo Standard Controller

<apex:page standardController="Account">

<h1>

Sample Page 1

</h1>

<apex:form >

Enter name :

<apex:inputText value="{!Account.Name}"/>

<br/>

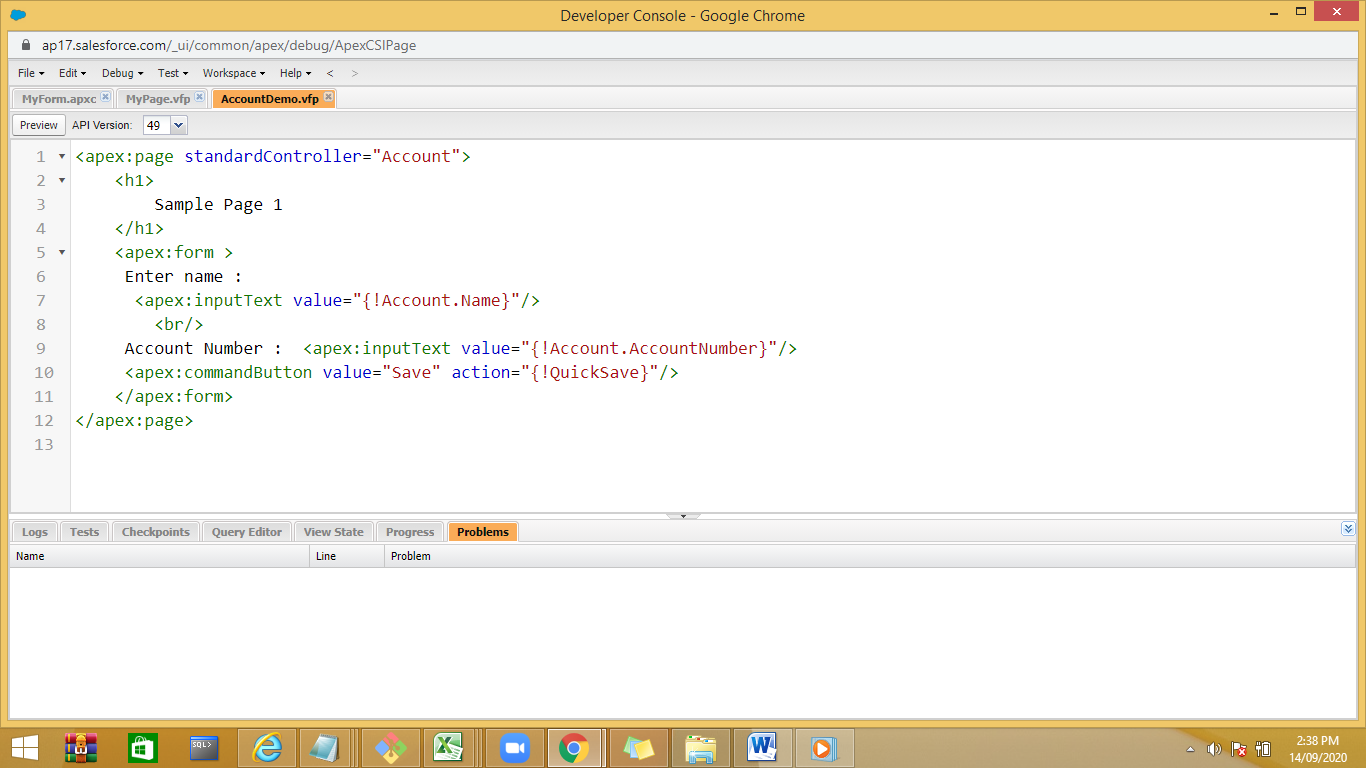
Account Number : <apex:inputText value="{!Account.AccountNumber}"/>

<apex:commandButton value="Save" action="{!QuickSave}"/>

</apex:form>

</apex:page>

Preview the page:



<apex:page standardController="ACustomer\_\_c">

<h1>

Sample Page 1

</h1>

<apex:form >

Enter name :<apex:inputText value="{!ACustomer\_\_c.CName\_\_c}"/><br/>

Account Email : <apex:inputText value="{!ACustomer\_\_c.CEmail\_\_c}"/> <br/>

Location :<apex:inputText value="{!ACustomer\_\_c.Location\_\_c}"/>

<apex:commandButton value="Save" action="{!QuickSave}"/>

</apex:form>

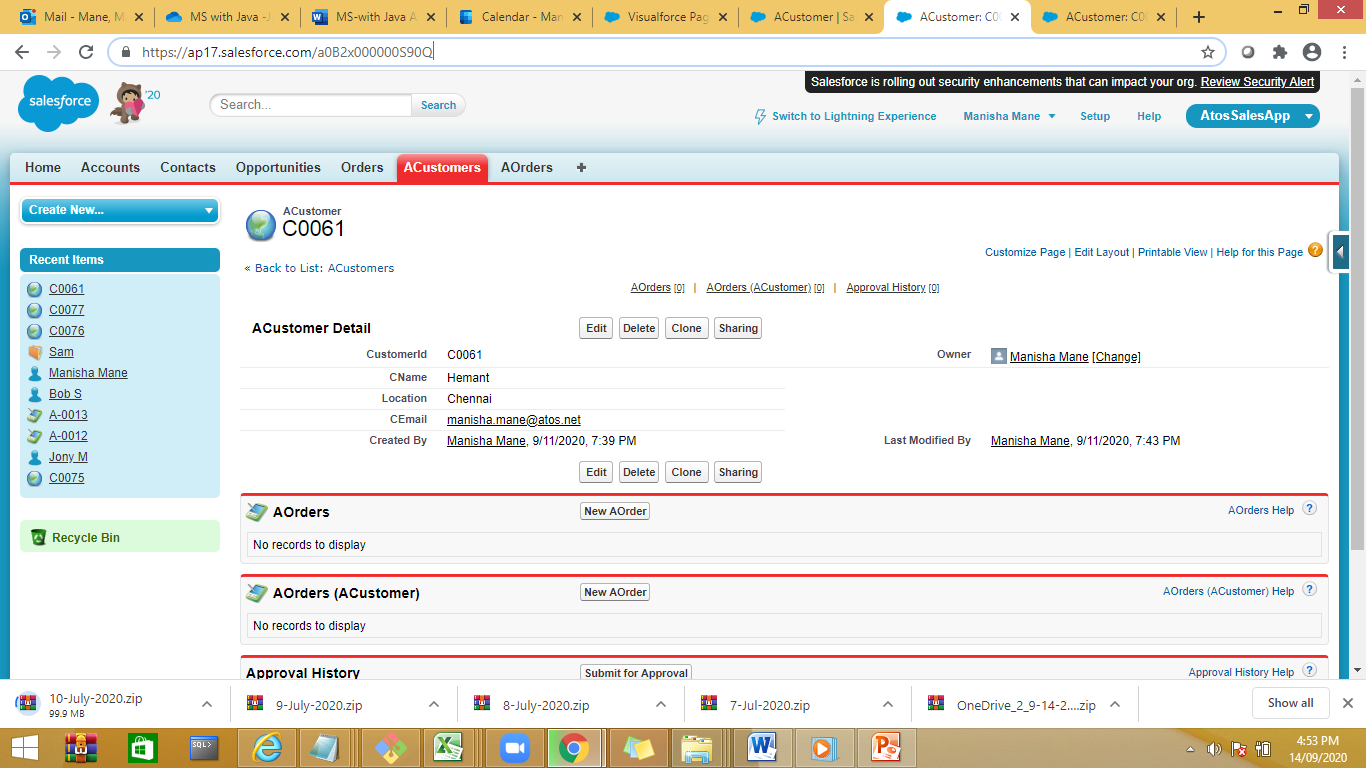
</apex:page>

Actions will be same for standard and custom controllers

**Passing id type of record**

Id –15 digit in developer org

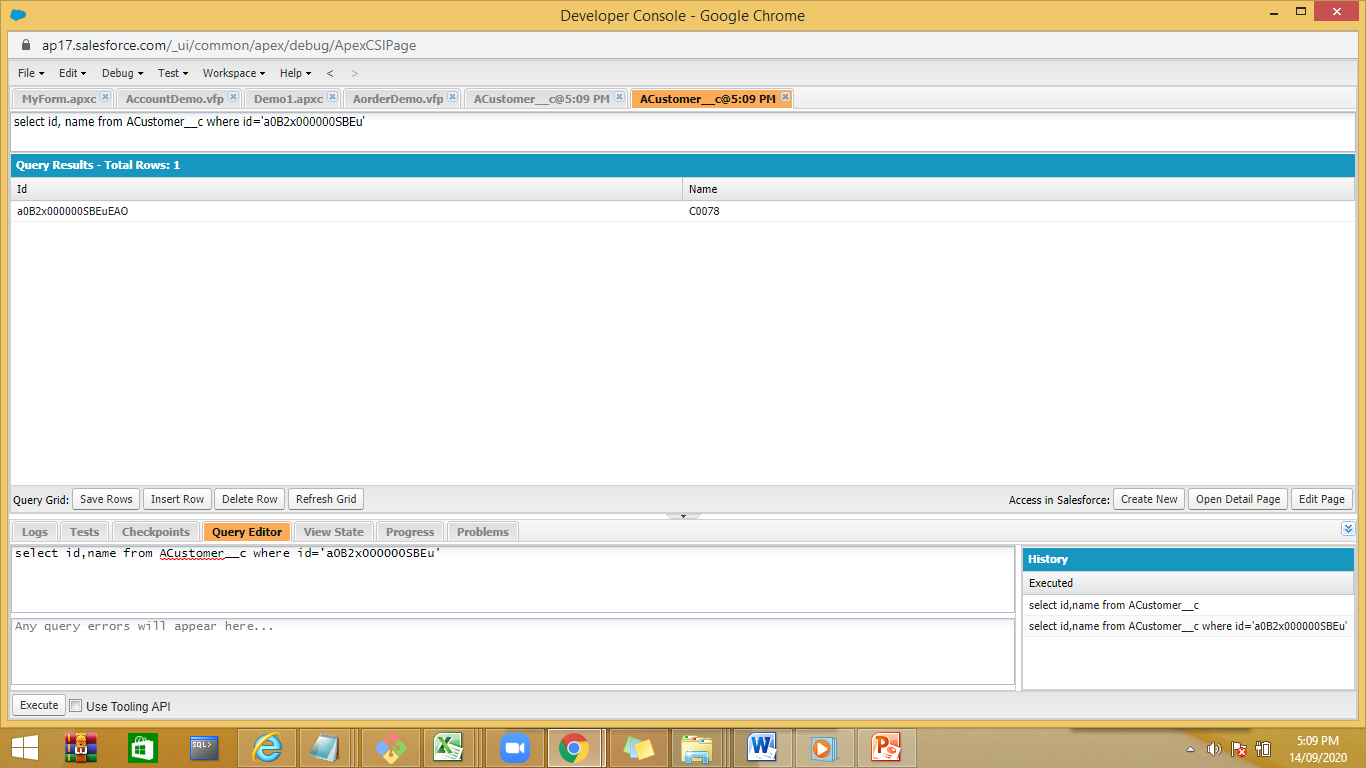
In production environment it becomes 18 digit. So don’t hardcode the id in code



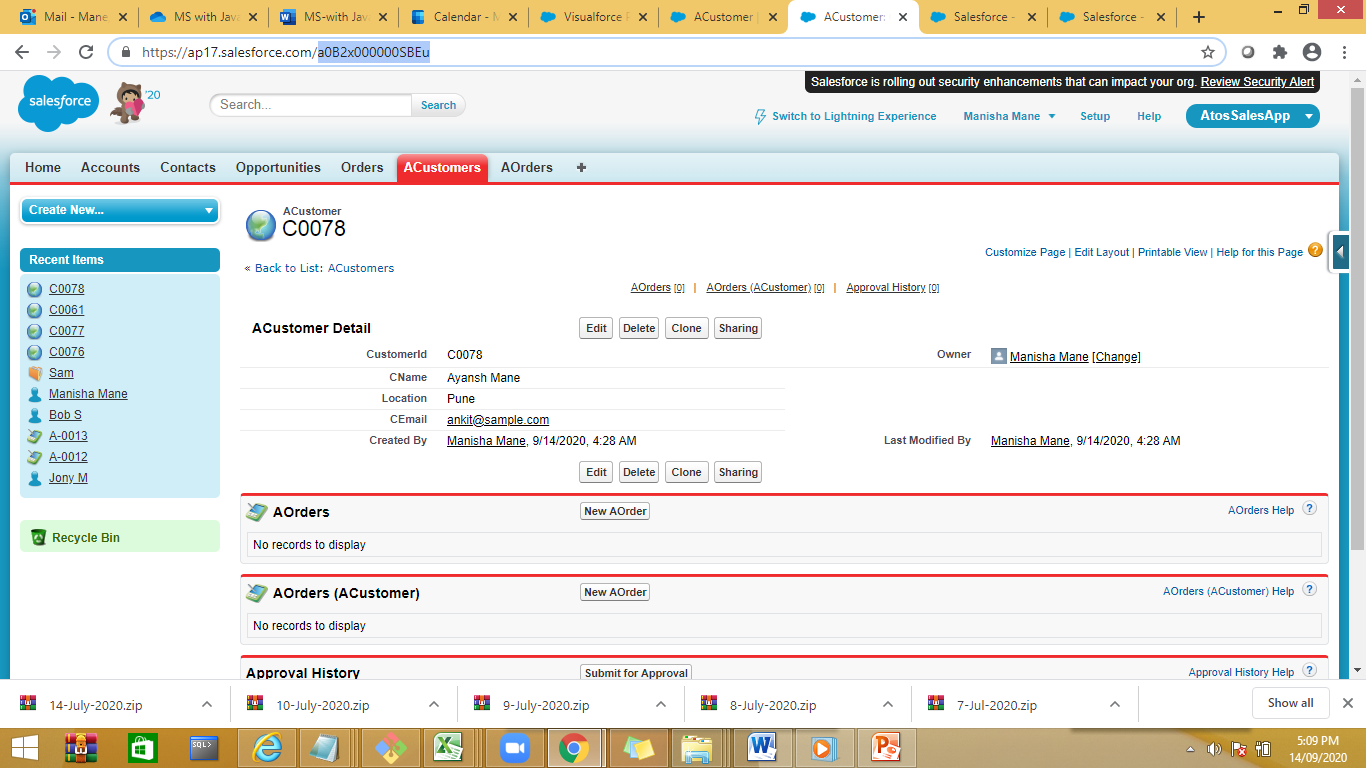
Observ Url , the id is 15 digits

Demo

Add new record check the id



Or



**Custom Controller**

Create a Apex class as controller

public class MyAccountControllerDemo {

public String actName{get;set;}

public String actNumber{get;set;}

public String msg{get;set;}

public String actPhone{get;set;}

public PageReference saveAccount(){

try

{

Account ob = new Account();

ob.Name = actName ;

ob.AccountNumber = actNumber;

ob.Phone=actPhone;

insert ob;

msg=' Record inserted successfully';

return null;

}catch(Exception e){

msg = 'Record creation failed '+ e.getMessage();

return null;

}

}

public PageReference deleteAccount(){

List<Account> actList = [select id , name from account where name = :actName];

Integer noOfRecords = 0;

if(!actList.isEmpty())

noOfRecords = actList.size();

delete actList;

msg = noOfRecords + ' records deleted ';

return null;

}

}

Create a view Visualforce page

<apex:page controller="MyAccountControllerDemo" >

<h1>

Custom Account Controller

</h1>

<apex:form>

Enter Name : <apex:inputText value="{!actName}"/><br/>

Enter Phone Number :<apex:inputText value="{!actPhone}"/><br/>

Enter Account Number<apex:inputText value="{!actNumber}"/><br/>

<apex:commandButton value="Save" action="{!saveAccount}"/>

<apex:commandButton value="Delete" action="{!deleteAccount}"/>

</apex:form>

</apex:page>

Preview the page add new account and cross check with Account Object .

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



<apex:page controller="MyEmpController">

<apex:form>

Enter name <apex:inputText value="{!e1.Name}"/><br/>

Enter Address<apex:inputText value="{!e1.Address\_\_c}"/><br/>

<apex:commandButton value="save" action="{!createEmp}"/>

</apex:form>

<h2>

<apex:outputText value="{!msg}"></apex:outputText>

</h2>

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

**Example**

<apex:page controller="MyPage5Controller">

<apex:outputText value="{!country}"></apex:outputText>

<apex:form>

<apex:selectRadio value="{!country}">

<apex:selectOption itemLabel="India" itemValue="IN"/>

<apex:selectOption itemLabel="USA" itemValue="US"/>

<apex:selectOption itemLabel="Japan" itemValue="JP"/>

<apex:selectOption itemLabel="UK" itemValue="UK"/>

<apex:selectOption itemLabel="Pakistan" itemValue="PK"/>

</apex:selectRadio>

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

</apex:form>

</apex:page>

public class MyPage5Controller {

public String msg{get;set;}

public String country{get;set;}

public MyPage5Controller(){

}

public PageReference showSelectedCountry(){

msg='Selected country is '+ country;

return null;

}

}

**Example –Radio Button,Checkbox**

<apex:page controller="MyPage5Controller">

<apex:outputText value="{!msg}"></apex:outputText>

<apex:form >

<apex:selectRadio value="{!country}">

<apex:selectOption itemLabel="India" itemValue="IN"/>

<apex:selectOption itemLabel="USA" itemValue="US"/>

<apex:selectOption itemLabel="Japan" itemValue="JP"/>

<apex:selectOption itemLabel="UK" itemValue="UK"/>

<apex:selectOption itemLabel="Pakistan" itemValue="PK"/>

</apex:selectRadio>

<p/>

<apex:selectRadio value="{!ovalue}">

<apex:selectOptions value="{!options}"></apex:selectOptions>

</apex:selectRadio>

<apex:outputText value="{!ovalue}"></apex:outputText>

Choose Account:

<br/>

<apex:selectCheckboxes value="{!selectedAccounts}">

<apex:selectOptions value="{!aoptions}"/>

</apex:selectCheckboxes><br/>

<apex:commandButton value="Show" action="{!showSelectedCountry}"/>

</apex:form>

<apex:outputText value="{!amsg}"></apex:outputText>

</apex:page>

public class MyPage5Controller {

public String country{get;set;}

public String ovalue{get;set;}

Public List<String> selectedAccounts{get;set;}

public String msg{get;set;}

public String amsg{get;set;}

public List<SelectOption> options{get;set;}

public List<SelectOption> aoptions{get;set;}

public list<String> clist = new List<String>();

public MyPage5Controller(){

selectedAccounts = new List<String>();

clist.add('Mumbai');

clist.add('Pune');

clist.add('Delhi');

clist.add('Chennai');

options = new List<SelectOption>();

aoptions = new List<SelectOption>();

List<Account> actList = [select name from Account limit 6];

for(String s : clist){

options.add(new SelectOption(s,s));

}

for(Account act : actList){

aoptions.add(new SelectOption(act.Name , act.name));

}

}

public PageReference showSelectedCountry(){

msg='Selected country is '+ country;

amsg='Selected accounts ';

for(String s : selectedAccounts){

amsg = amsg+ s + ' ';

}

return null;

}

}

------------------------------------

**DataTable element**

Example 1

<apex:page standardController="Account" recordSetVar="actList">

<apex:dataTable value="{!actList}" var="actRecord">

<apex:column >{!actRecord.Name}</apex:column>

<apex:column >{!actRecord.Id}</apex:column>

<apex:column >{!actRecord.AccountNumber}</apex:column>

</apex:dataTable>

</apex:page>

Example 2

public class MyController6 {

public List<Account> actRec{get;set;}

public MyController6()

{

actRec=[ select id,name,accountnumber from account];

}

}

**PDF Demo**

<apex:page Controller="MyController6" renderAs="pdf">

<apex:dataTable value="{!actRec}" var="actRecord">

<apex:column >{!actRecord.Name}</apex:column>

<apex:column >{!actRecord.Id}</apex:column>

<apex:column >{!actRecord.AccountNumber}</apex:column>

</apex:dataTable>

**Excel Demo**

<apex:page Controller="MyController6" contentType="application/vnd.ms-excel#Account.xls">

<apex:dataTable value="{!actRec}" var="actRecord">

<apex:column >{!actRecord.Name}</apex:column>

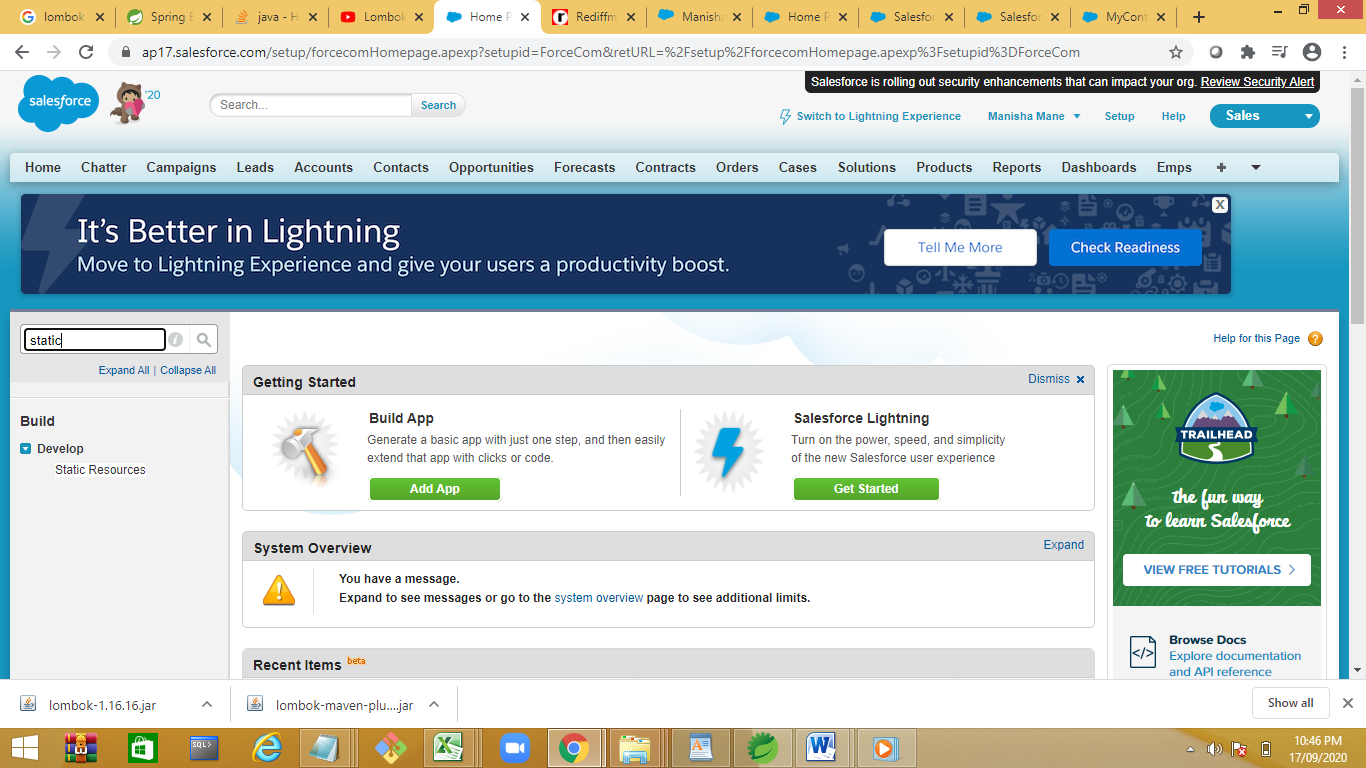
<apex:column >{!actRecord.Id}</apex:column>

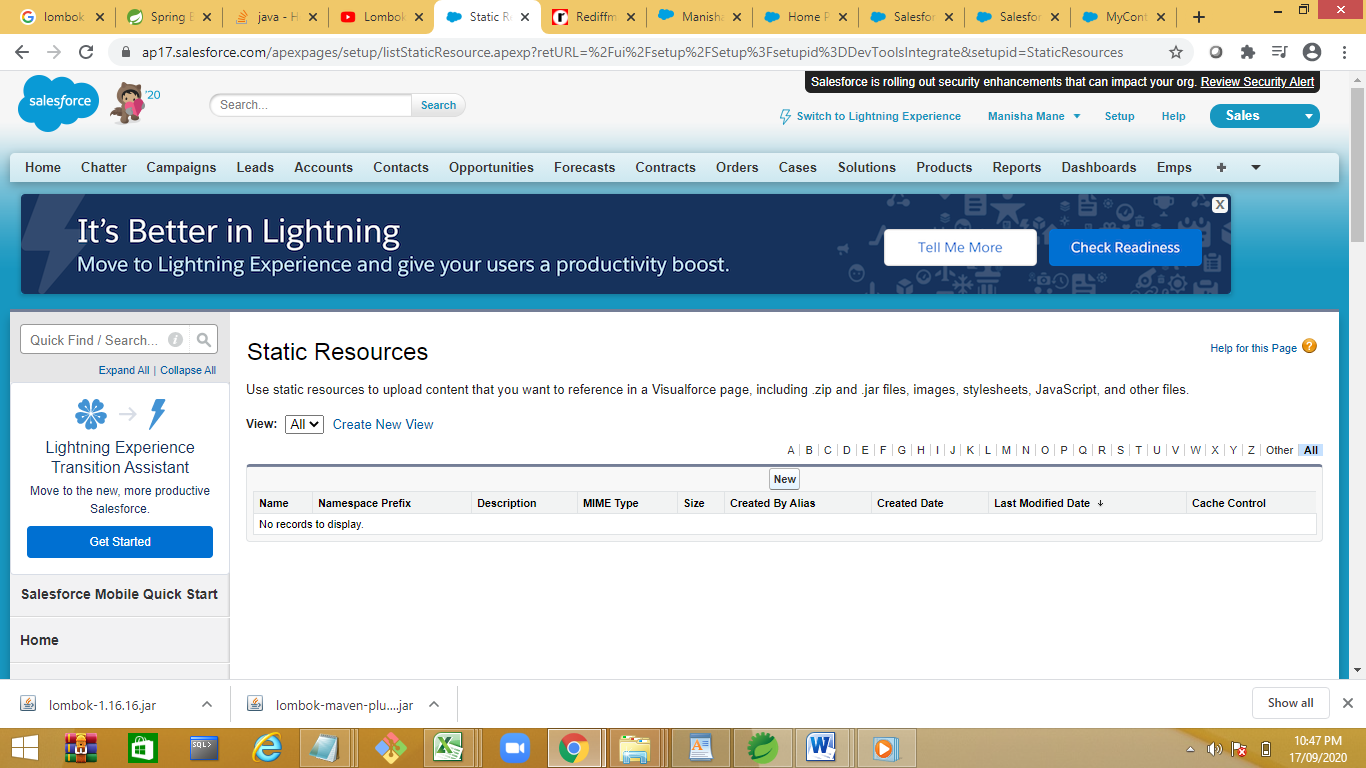
<apex:column >{!actRecord.AccountNumber}</apex:column>

</apex:dataTable>

</apex:page>

**Static resource, Image demo:need to upload iamges as static resources**





<apex:page Controller="MyController6" >

<apex:image url="{!$Resource.MyImage1}" width="50" height="50"/>

<apex:dataTable value="{!actRec}" var="actRecord">

<apex:column >{!actRecord.Name}</apex:column>

<apex:column >{!actRecord.Id}</apex:column>

<apex:column >{!actRecord.AccountNumber}</apex:column>

</apex:dataTable>

</apex:page>

**Calling JavaScript function in visual page**

MyPage1.js

function helloWorld() {

alert ("This is a warning message!");

document.write ("This is a warning message!");

}

<apex:page Controller="MyController6" >

<apex:includeScript value="{!$Resource.MyPage1}"/>

<apex:image url="{!$Resource.MyImage1}" width="50" height="50"/>

<apex:dataTable value="{!actRec}" var="actRecord">

<apex:column >{!actRecord.Name}</apex:column>

<apex:column >{!actRecord.Id}</apex:column>

<apex:column >{!actRecord.AccountNumber}</apex:column>

</apex:dataTable>

</apex:page>

**ForDemo**

public class ForDemo {

public void display()

{

List<Account> accList=[select id,name from Account Limit 10];

for(Account ob:accList)

{

System.debug('Name '+ob.Name+'Id='+ob.Id);

}

for(Account ob:[select id,name from Account Limit 10])

{

System.debug('Name '+ob.Name+'Id='+ob.Id);

}

}

}

<apex:page controller="JavaScriptDemo" >

<script>

function func()

{

alert('function calling');

}

function func1()

{

// String x=document.getElementById('txtAccount').value;

alert('Hello function calling!!!!!!!!!!');

}

</script>

<apex:form>

<apex:outputText value="{!callfunc1}" escape="false"></apex:outputText>

<apex:outputText value="{!callfunc2}" escape="false"></apex:outputText>

First Name :<input type='text' id='txtAccount' value=''/>

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

<apex:commandButton value="show1" action="{!show1}"/>

</apex:form>

</apex:page>

public class JavaScriptDemo {

public string callfunc1{get;set;}

public string callfunc2{get;set;}

public string fname{get;set;}

public JavaScriptDemo()

{

System.debug('Inside JS Demo' );

}

public PageReference show()

{

callfunc1='<script> func(); </script>';

return null;

}

public PageReference show1()

{

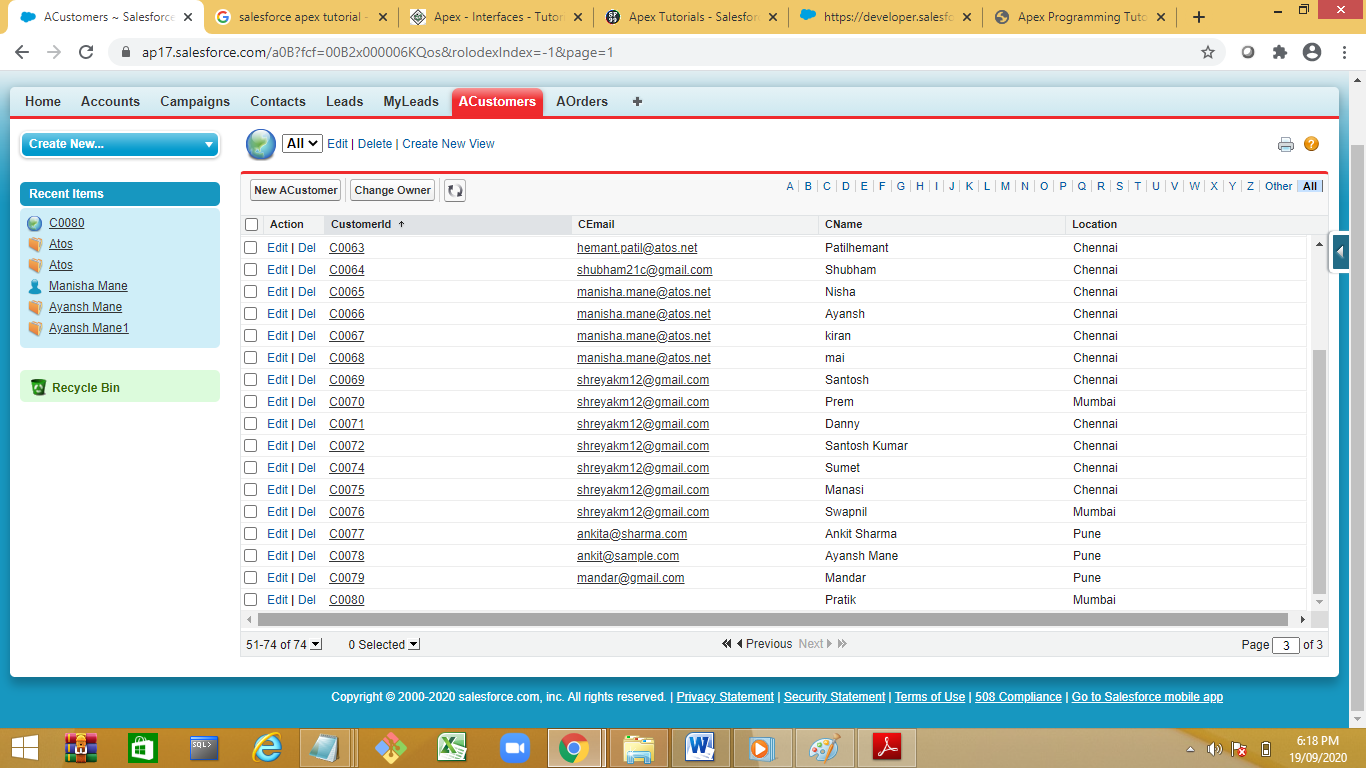
callfunc2='<script> func1(); </script>';

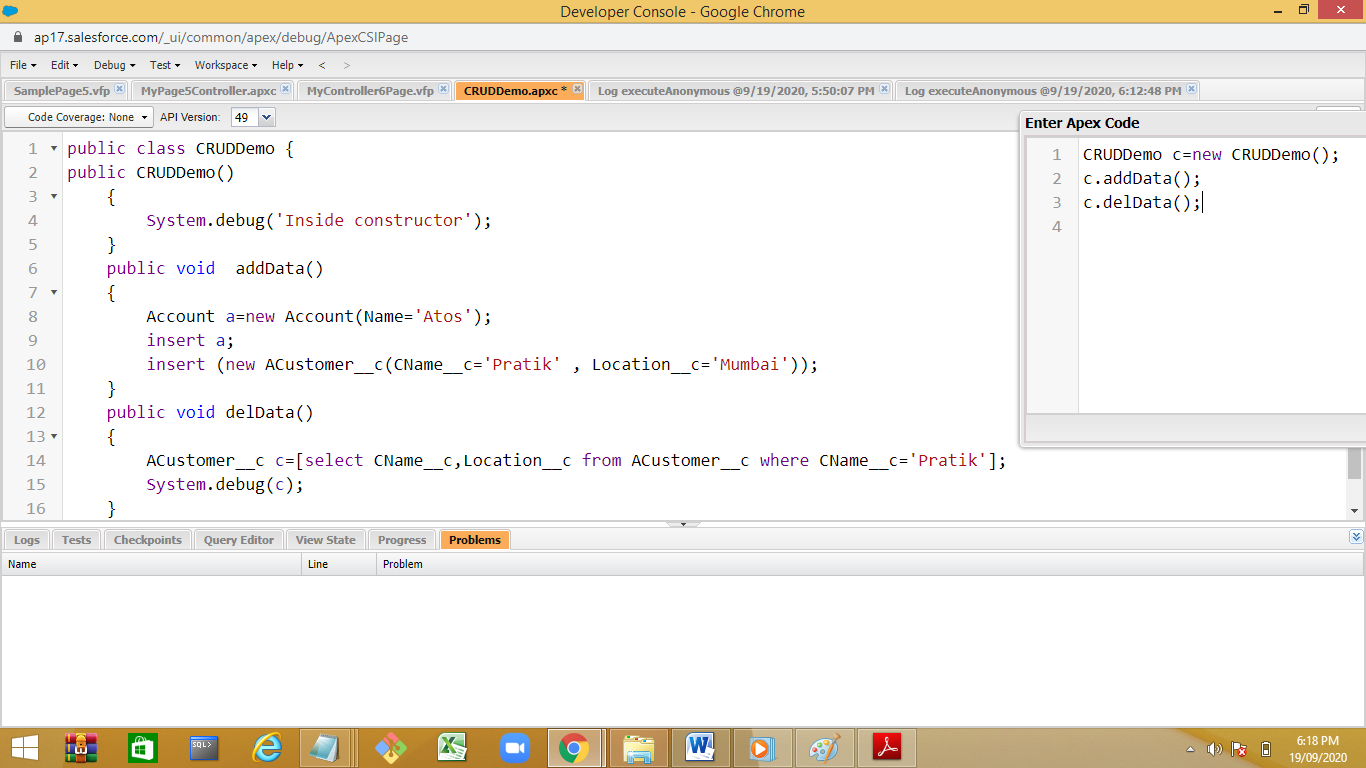
return null;

}

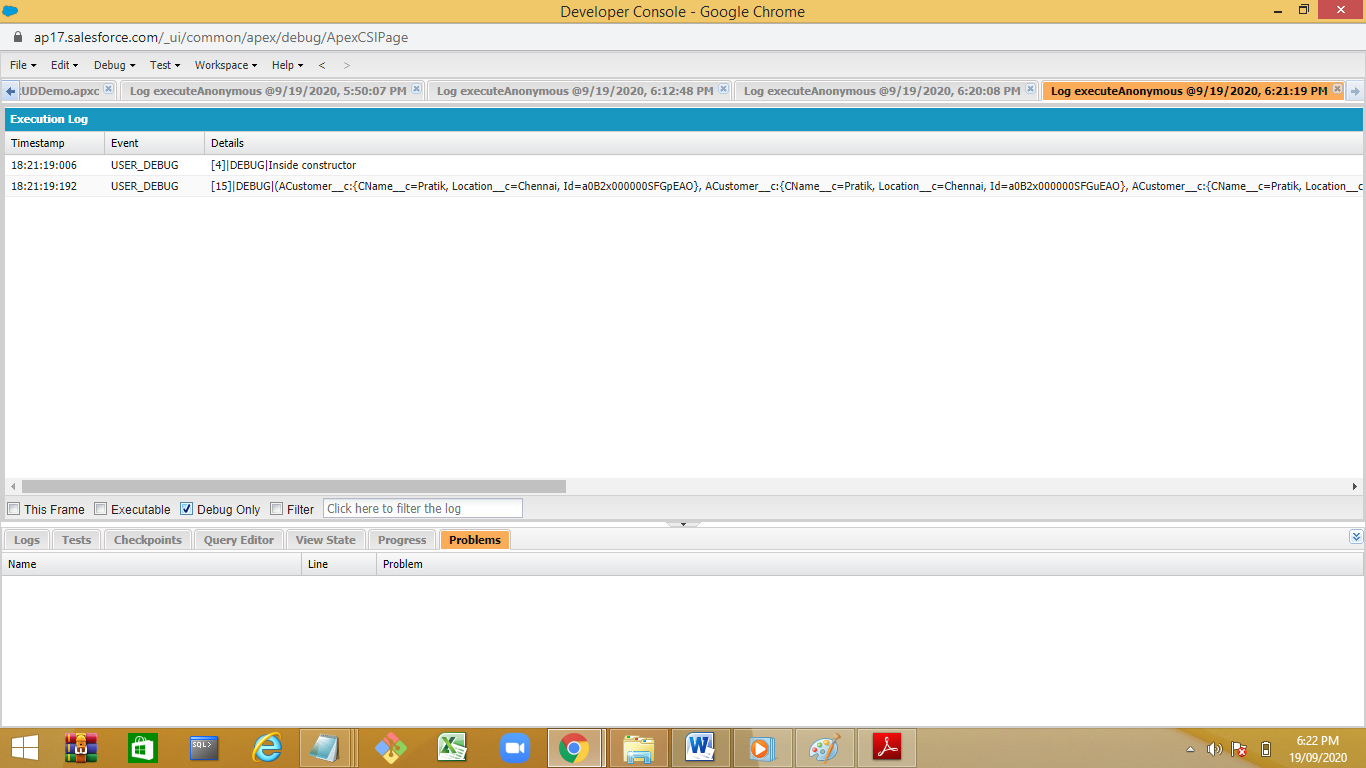
}

CRUD operation:





Added multiple entries for Customer ‘Pratik’

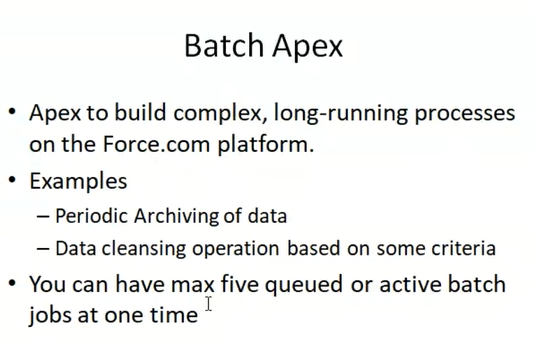


**Batch Apex:**

are backgroung processes.

## Batch Apex

Batch Apex is used to run large jobs (think thousands or millions of records!) that would exceed normal processing limits. Using Batch Apex, you can process records asynchronously in batches (hence the name, “Batch Apex”) to stay within platform limits. If you have a lot of records to process, for example, data cleansing or archiving, Batch Apex is probably your best solution.



To write a Batch Apex class, your class must implement the Database.Batchable interface and include the following three methods:

**start**

Used to collect the records or objects to be passed to the interface method execute for processing. This method is called once at the beginning of a Batch Apex job and returns either a Database.QueryLocator object or an Iterable that contains the records or objects passed to the job.

**execute**

Performs the actual processing for each chunk or “batch” of data passed to the method. The default batch size is 200 records. Batches of records are not guaranteed to execute in the order they are received from the start method.

**finish**

Used to execute post-processing operations (for example, sending an email) and is called once after all batches are processed.

https://trailhead.salesforce.com/en/content/learn/modules/asynchronous\_apex/async\_apex\_batch

## Batch Apex Syntax

Demo:

global class BatchApexExample implements Database.Batchable<sObject> {

global Database.QueryLocator start(Database.BatchableContext BC) {

// collect the batches of records or objects to be passed to execute

String query = 'SELECT Id, Name FROM Account';

return Database.getQueryLocator(query);

}

global void execute(Database.BatchableContext BC, List<Account> accList) {

// process each batch of records default size is 200

for(Account acc : accList) {

// Update the Account Name

acc.Name = acc.Name + 'Atos Syntel Customer';

}

try {

// Update the Account Record

update accList;

} catch(Exception e) {

System.debug(e);

}

}

global void finish(Database.BatchableContext BC) {

// execute any post-processing operations like sending email

}

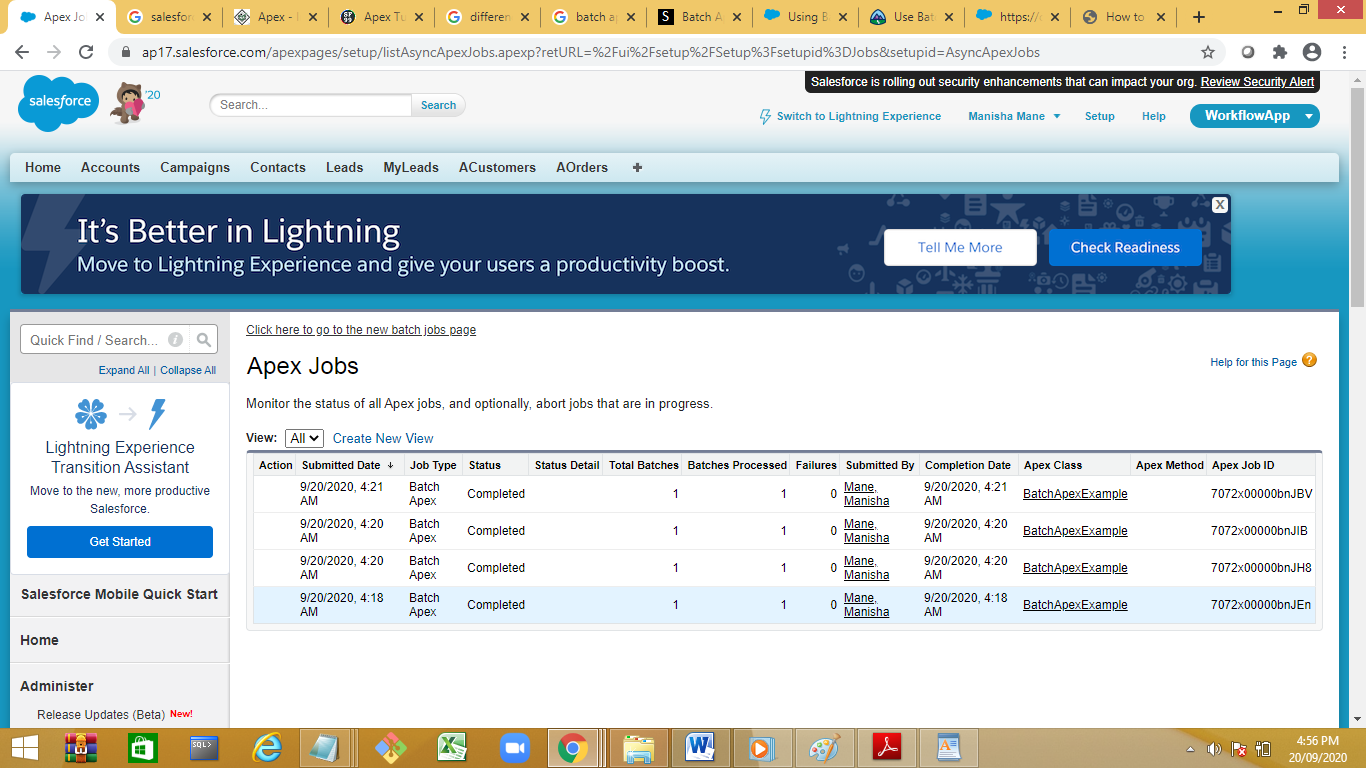
}

Execution

BatchApexExample b=new BatchApexExample();

Database.executeBatch(b);

To check the status of batch:



## Invoking a Batch Class

To invoke a batch class, simply instantiate it and then call Database.executeBatch with the instance:

|  |  |
| --- | --- |
| 1  2 | MyBatchClass myBatchObject = new MyBatchClass();  Id batchId = Database.executeBatch(myBatchObject); |

You can also optionally pass a second scope parameter to specify the number of records that should be passed into the execute method for each batch. Pro tip: you might want to limit this batch size if you are running into governor limits.

|  |  |
| --- | --- |
| 1 | Id batchId = Database.executeBatch(myBatchObject, 100); |

## Scheduling Batch Apex

can also use the Schedulable interface with batch Apex classes. The following example implements the Schedulable interface for a batch Apex class called batchable:

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | global class scheduledBatchable implements Schedulable {     global void execute(SchedulableContext sc) {        BatchApexExample b = new BatchApexExample();        Database.executeBatch(b);     }  }  Demo :schedule the batch for particular timings:    Schedule Apex |

## Test class for batch Apex

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20 | @isTest  private class BatchApexExampleTest {      static testmethod void test() {          // Create test accounts to be updated by batch      Account[] accList = new List();      for (Integer i=0;i<400;i++) {          Account ac = new Account(Name = 'Account ' + i);          accList.add(ac);      }      insert accList;            Test.startTest();              BatchApexExample b = new BatchApexExample();          Database.executeBatch(b);          Test.stopTest();          // Verify accounts updated      Account[] accUpdatedList = [SELECT Id, Name FROM Account];      System.assert(accUpdatedList[0].Name.Contains('sfdcpoint'));      }  } |

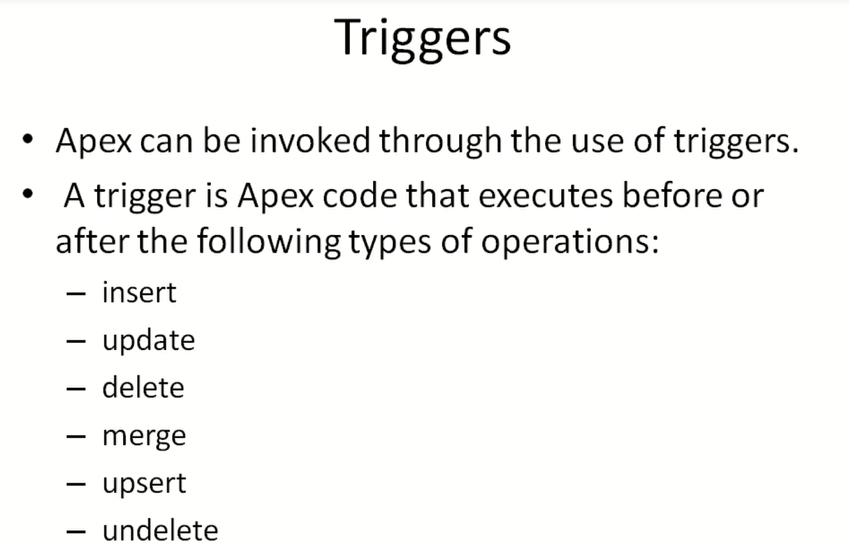
**Apex Jobs:**

**Default jobs are 5 , we can increase the number depend on the license.**

**ApexTriggers:**

### ****What is Triggers in Salesforce?****

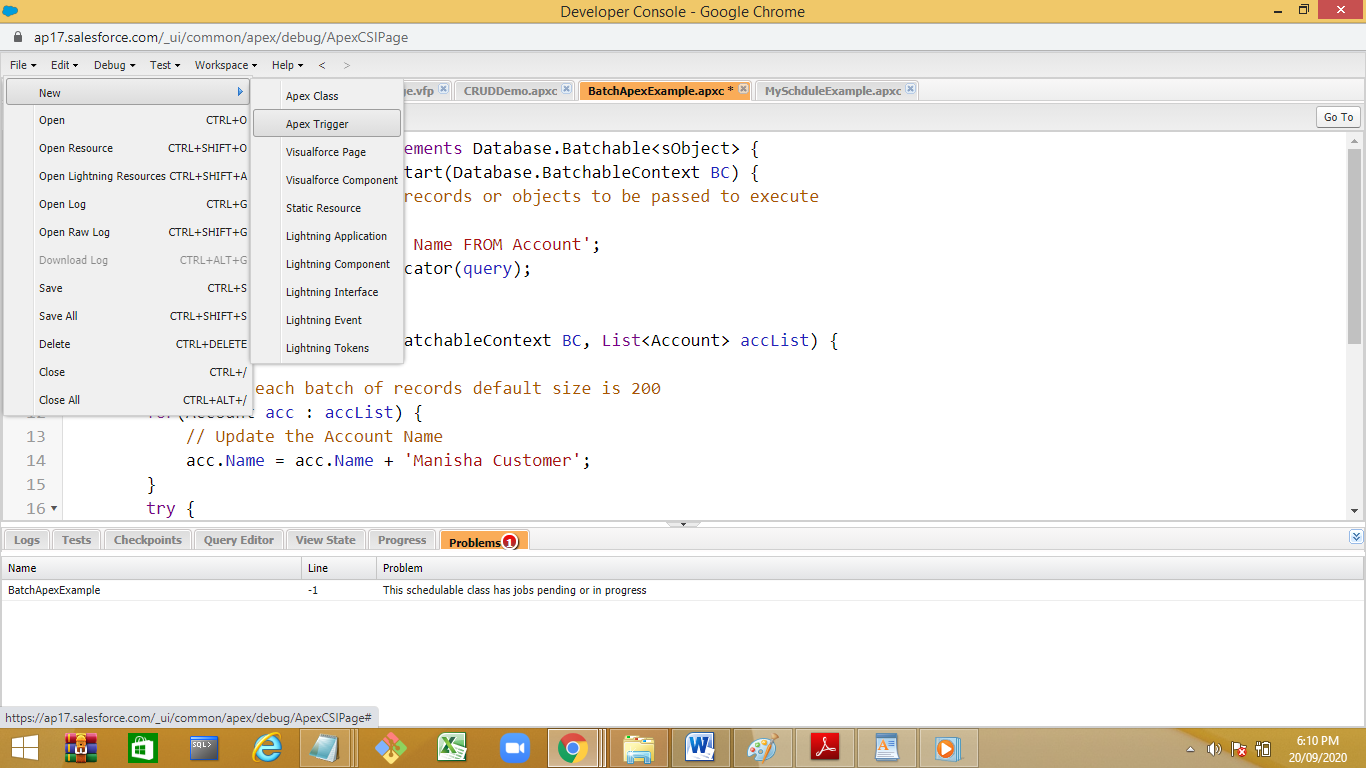
A **trigger is** an **Apex script** that executes before or after data manipulation language (**DML**) events occur. Apex triggers enable you to perform custom actions before or after events to record in Salesforce, such as insertions, updates, or deletions. Just like database systems support triggers, Apex provides trigger support for managing records.

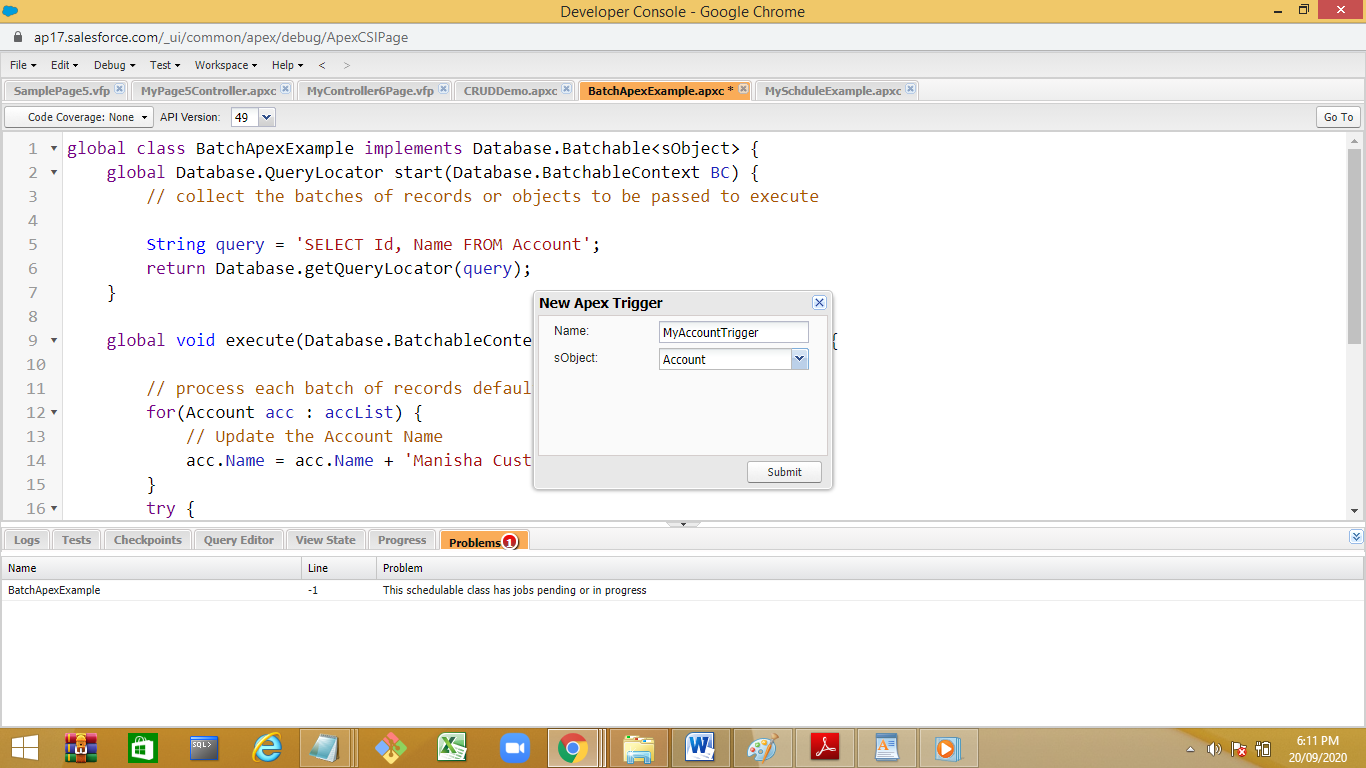


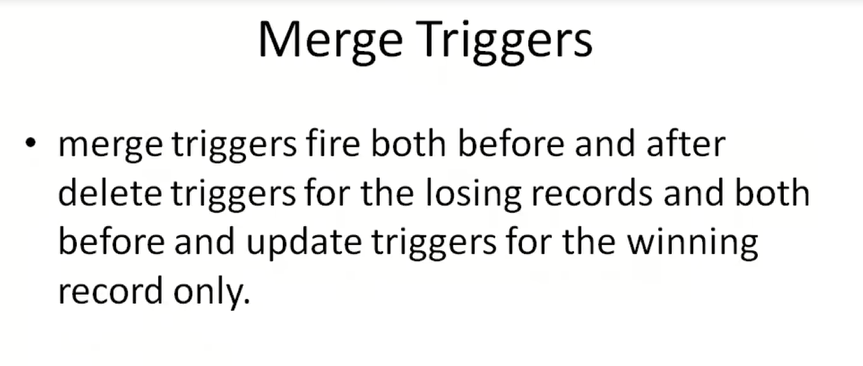
Upsert –update and insert

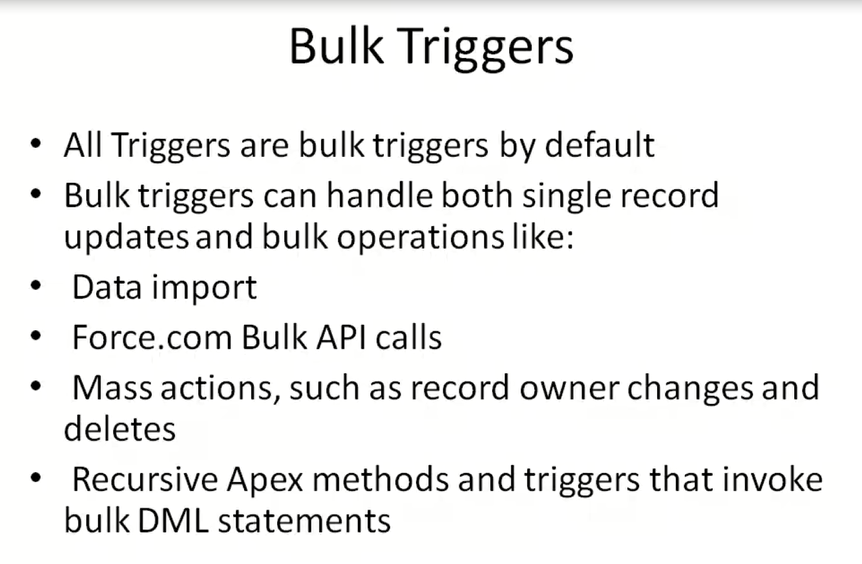
Merge--merging of 2 or more records defaut and update default merging is done on Id field.

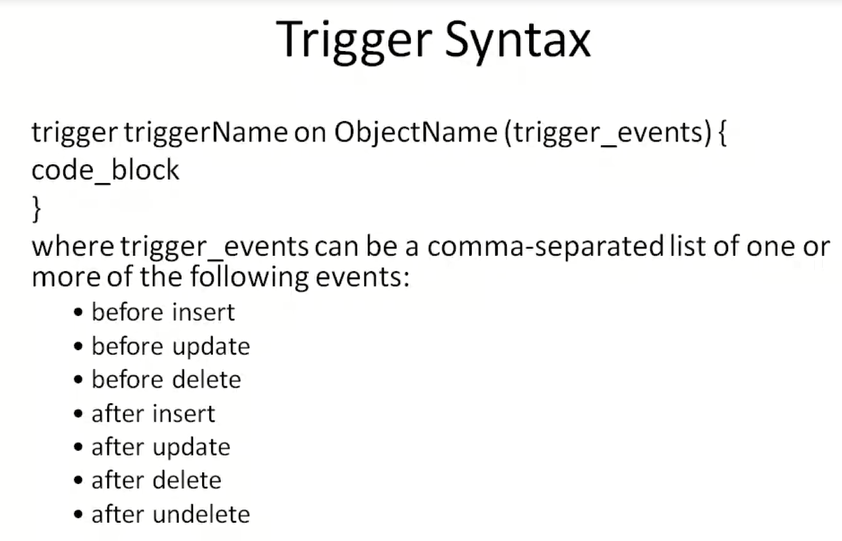
Trigger Demo:

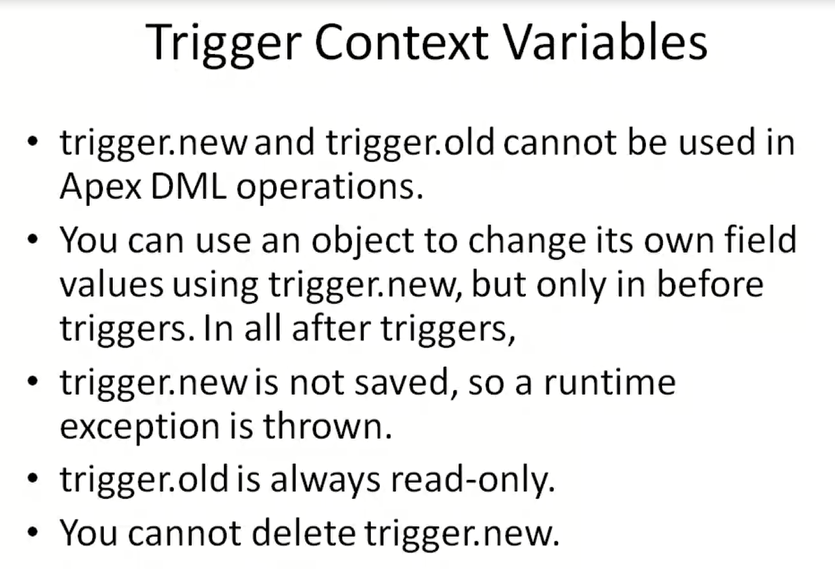












### ****What are the considerations while implementing the Triggers?****

**Consider the following before implementing the triggers.**

* Upsert trigger fires on 4 different events :- before(insert, update), after (insert, update)
* Merge trigger are fired on both events on delete
* Field history is updated after the trigger has successfully finished processing data.
* Any callout should be asynchronous so that trigger does not have to wait for the response.
* A trigger cannot have a static keyword in its code.
* If a trigger completes successfully the changes are committed to the database and if it fails the transaction is rolled back.

### ****What are context variables in triggers?****

All triggers define implicit variables that allow developers to access run-time context. These variables are contained in the **System.Trigger** class.

**Here is list of context variables in triggers**

* **isExecuting**: Returns true if the current context for the Apex code is a trigger, not a Visualforce page, a Web service, or an executeanonymous() API call.
* **isInsert**: Returns true if this trigger was fired due to an insert operation, from the Salesforce user interface, Apex, or the API.
* **isUpdate**: Returns true if this trigger was fired due to an update operation, from the Salesforce user interface, Apex, or the API.
* **isDelete**: Returns true if this trigger was fired due to a delete operation, from the Salesforce user interface, Apex, or the API.
* **isBefore**: Returns true if this trigger was fired before any record was saved.
* **isAfter**: Returns true if this trigger was fired after all records were saved.
* **isUndelete**: Returns true if this trigger was fired after a record is recovered from the Recycle Bin (that is, after an undelete operation from the Salesforce user interface, Apex, or the API.)
* **new**: Returns a list of the new versions of the sObject records. This sObject list is only available in insert, update, and undelete triggers, and the records can only be modified in before triggers.
* **newMap**: A map of IDs to the new versions of the sObject records. This map is only available in before update, after insert, after update, and after undelete triggers.
* **old** : Returns a list of the old versions of the sObject records. This sObject list is only available in update and delete triggers.
* **oldMap**: A map of IDs to the old versions of the sObject records. This map is only available in update and delete triggers.
* **size**: The total number of records in a trigger invocation, both old and new.

**Row level trigger**

**Ex delete command will delete 10 record,then trigger will excute for each deleted rows is called as row level trigger.**

**statement level trigger:trigger statement will execute once.trigger code will execute only once .**

insert trigger is only applicable for before insert

trigger MyAccountTrigger on Account (before insert) {

List <Account> oldRec=(List<Account>) Trigger.Old;

List <Account> newRec=(List<Account>) Trigger.new;

for(Account ob:newRec)

{

System.debug('inserting '+ob.Name);

// insert ob;

}

/\* for(Account ob:oldRec)

{

System.debug('inserting '+ob.Name);

// insert ob;

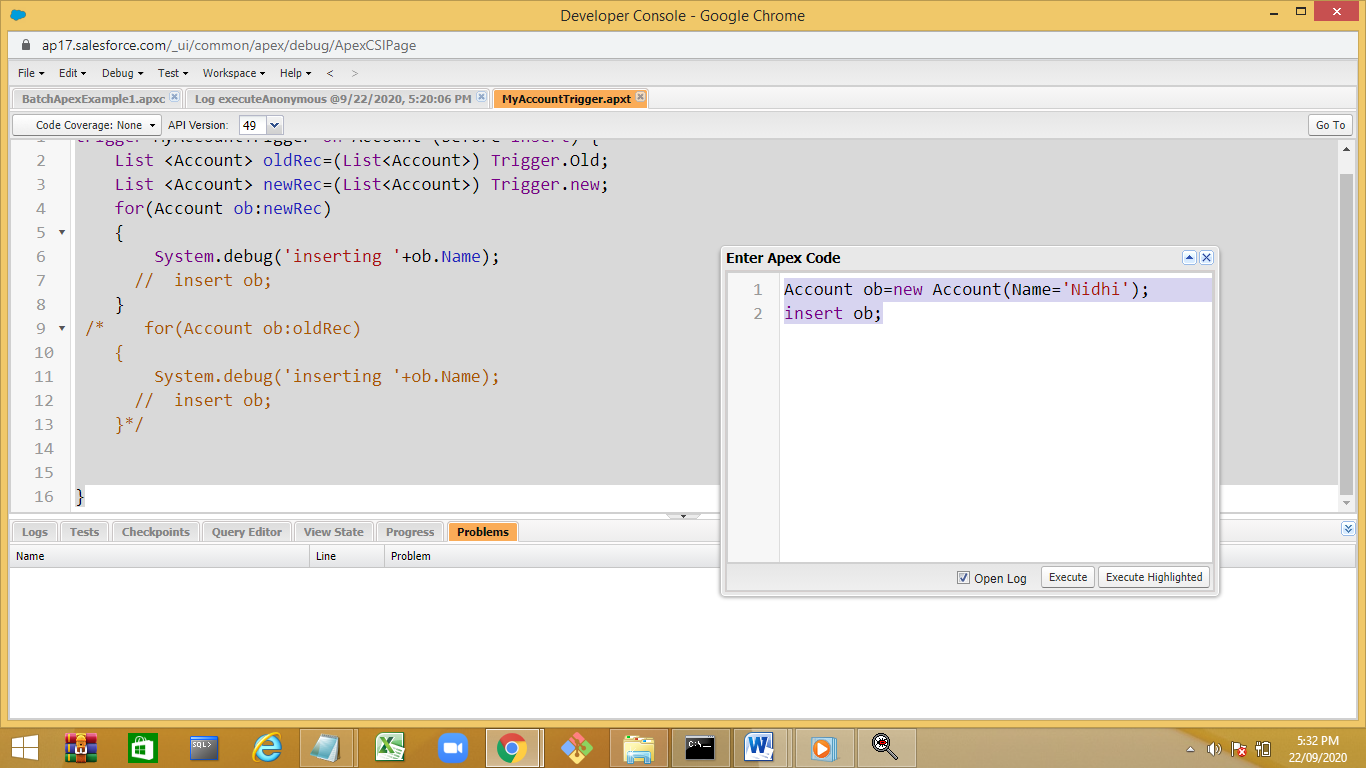
}\*/

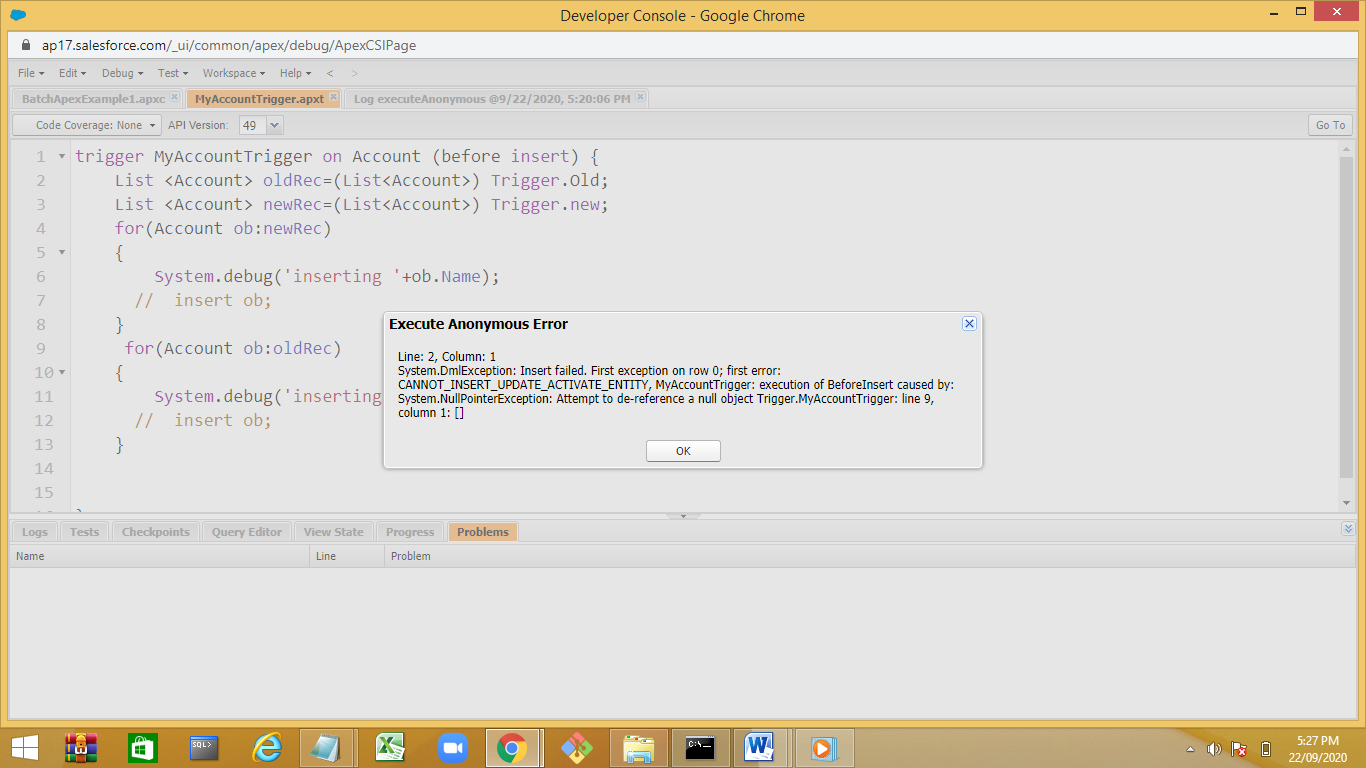
}

Execution :

Account ob=new Account(Name='Nidhi');

insert ob;





Update Trigger:

trigger MyAccountTrigger on Account (before update,after update) {

List <Account> oldRec=(List<Account>) Trigger.Old;

List <Account> newRec=(List<Account>) Trigger.new;

for(Account ob:newRec)

{

System.debug('updating'+ob.Name);

}

for(Account ob:oldRec)

{

System.debug('Updating '+ob.Name);

// insert ob;

}

}

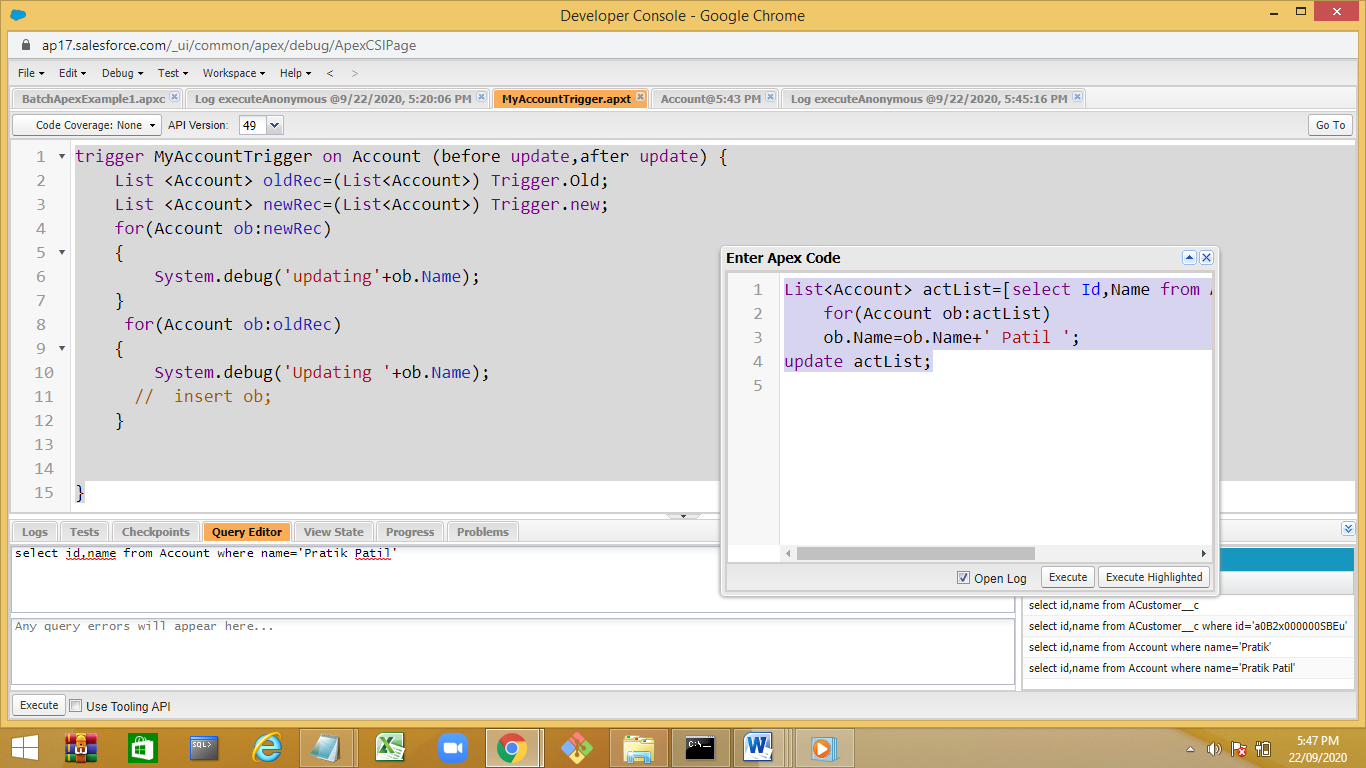
Execution:

List<Account> actList=[select Id,Name from Account where Name='Pratik'];

for(Account ob:actList)

ob.Name=ob.Name+' Patil ';

update actList;



trigger MyAccountTrigger on Account (after update) {

List <Account> oldRec=(List<Account>) Trigger.Old;

List <Account> newRec=(List<Account>) Trigger.new;

for(Account ob:newRec)

{

System.debug('updating'+ob.Name);

}

for(Account ob:oldRec)

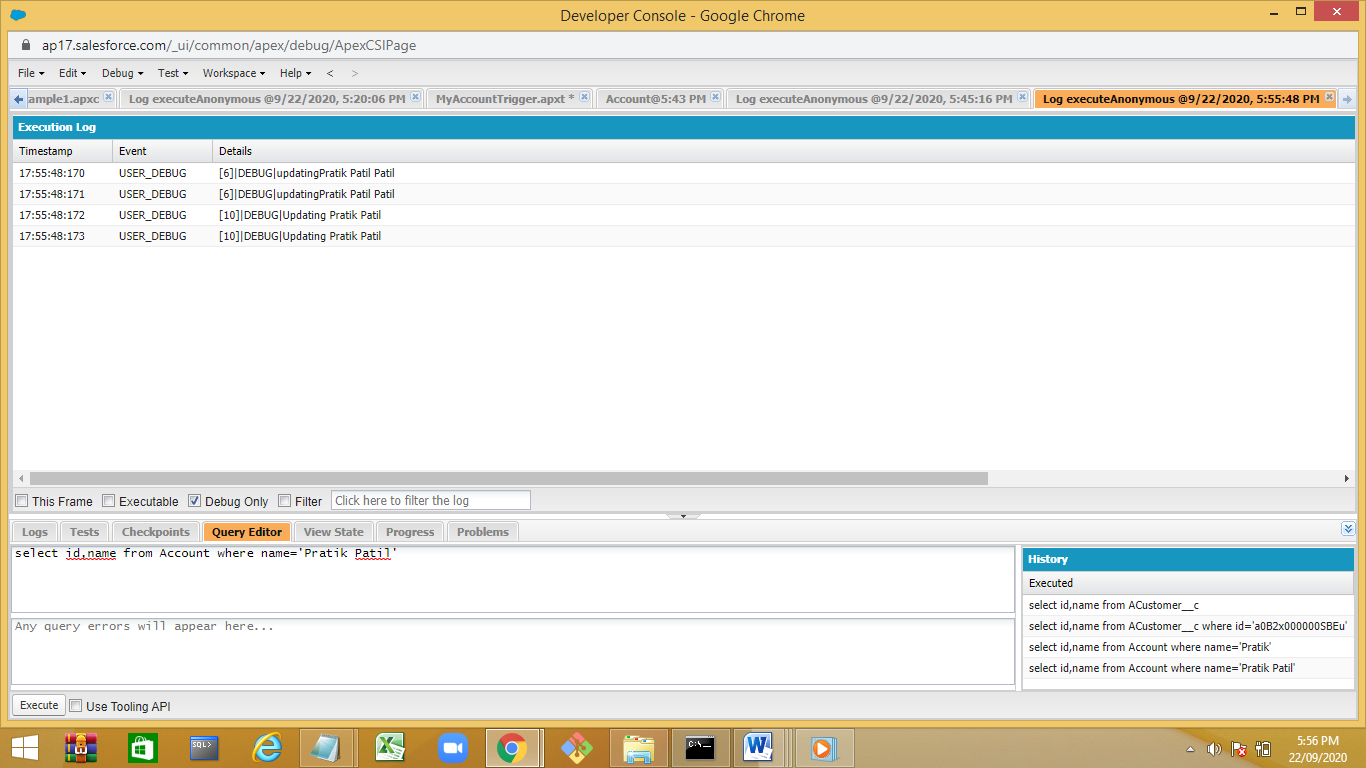
{

System.debug('Updating '+ob.Name);

// insert ob;

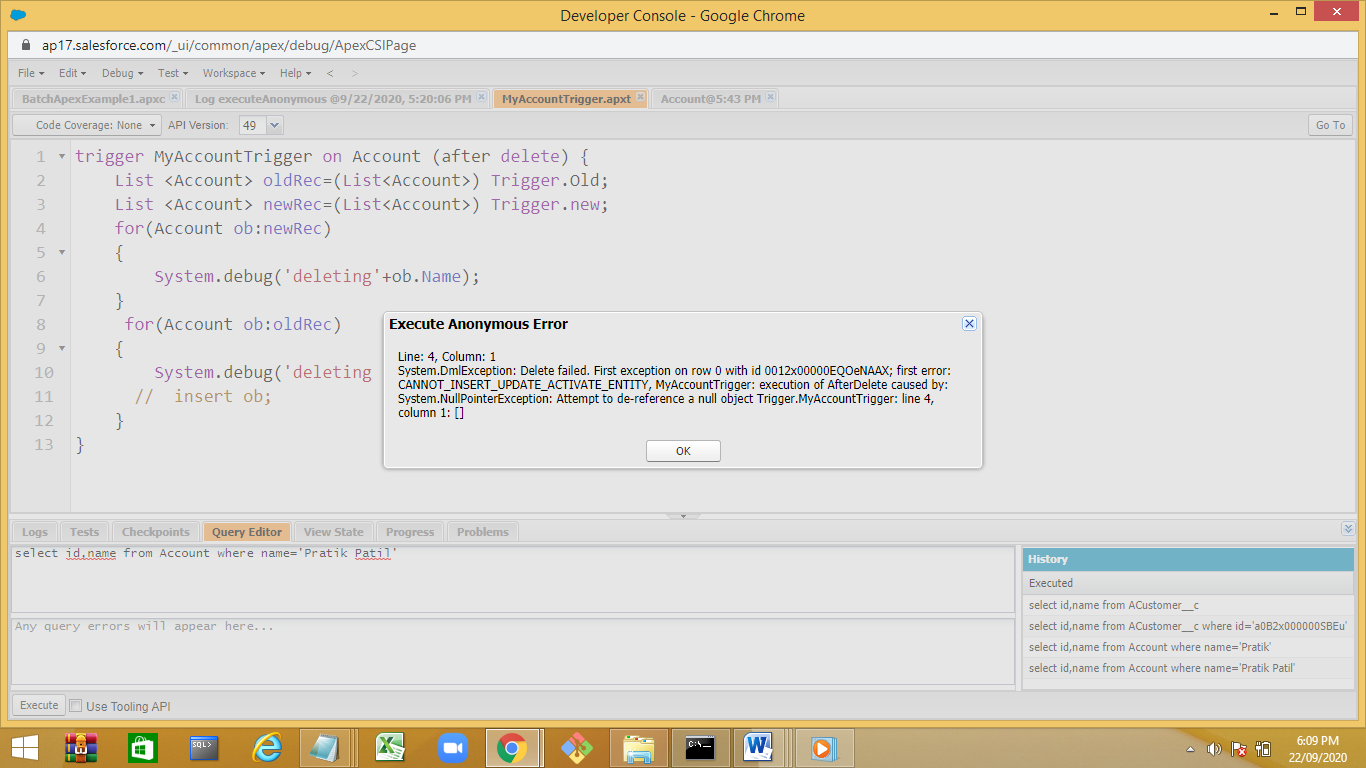
}

}



Delete trigger: (can not access new values)

Error with after delete we can not get the new values .



trigger MyAccountTrigger on Account (before delete) {

List <Account> oldRec=(List<Account>) Trigger.Old;

List <Account> newRec=(List<Account>) Trigger.new;

/\*for(Account ob:newRec)

{

System.debug('deleting'+ob.Name);

}\*/

for(Account ob:oldRec)

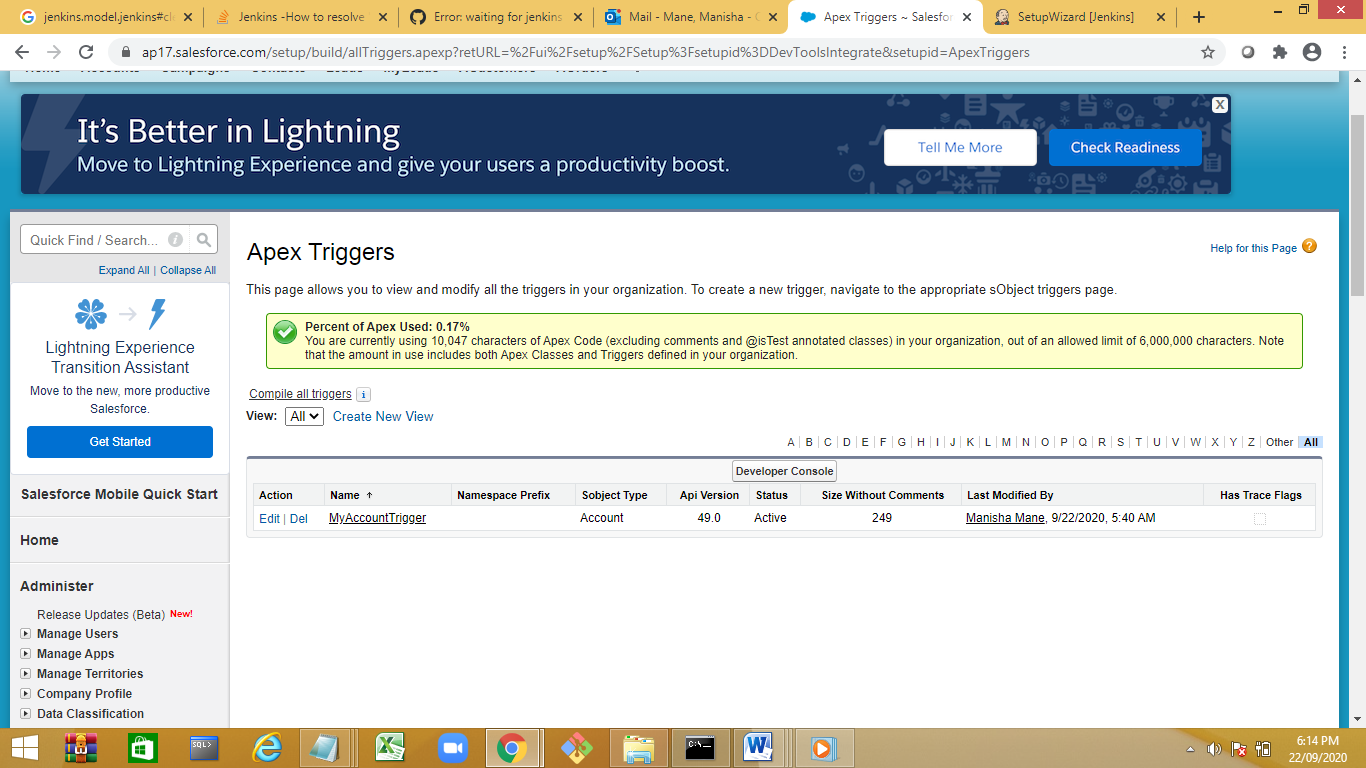
{

System.debug('deleting '+ob.Name);

// insert ob;

}

}



Create a new trigger:

trigger MyAccountTrigger2 on Account (before update) {

List <Account> oldRec=(List<Account>) Trigger.Old;

List <Account> newRec=(List<Account>) Trigger.new;

for(Integer i=0;i<oldRec.size();i++)

{

if(oldRec.get(i).name=='Nidhi')

{

newRec.get(i).name=oldRec.get(i).name;

System.debug('can not update the record');

}

}

}

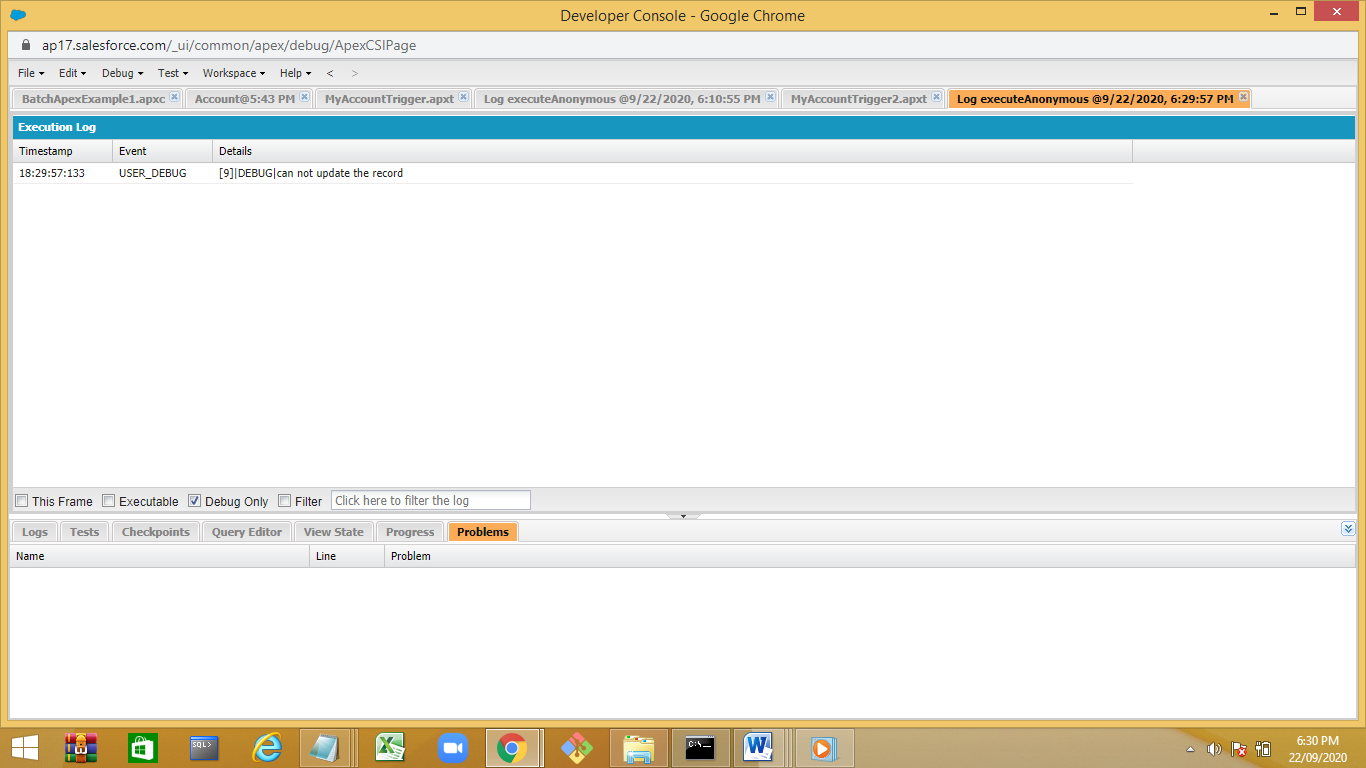
Execution:

Account acc=[select Id,Name from Account where Name='Nidhi' LIMIT 1];

acc.Name='Manisha K';

update acc;

Output



Demo:

trigger MyAccountTrigger2 on Account (before update,after update) {

if(Trigger.isBefore)

{

List <Account> oldRec=(List<Account>) Trigger.Old;

List <Account> newRec=(List<Account>) Trigger.new;

for(Integer i=0;i<oldRec.size();i++)

{

if(oldRec.get(i).name=='Nidhi')

{

newRec.get(i).name=oldRec.get(i).name;

System.debug('can not update the record');

}

}

}

else

{

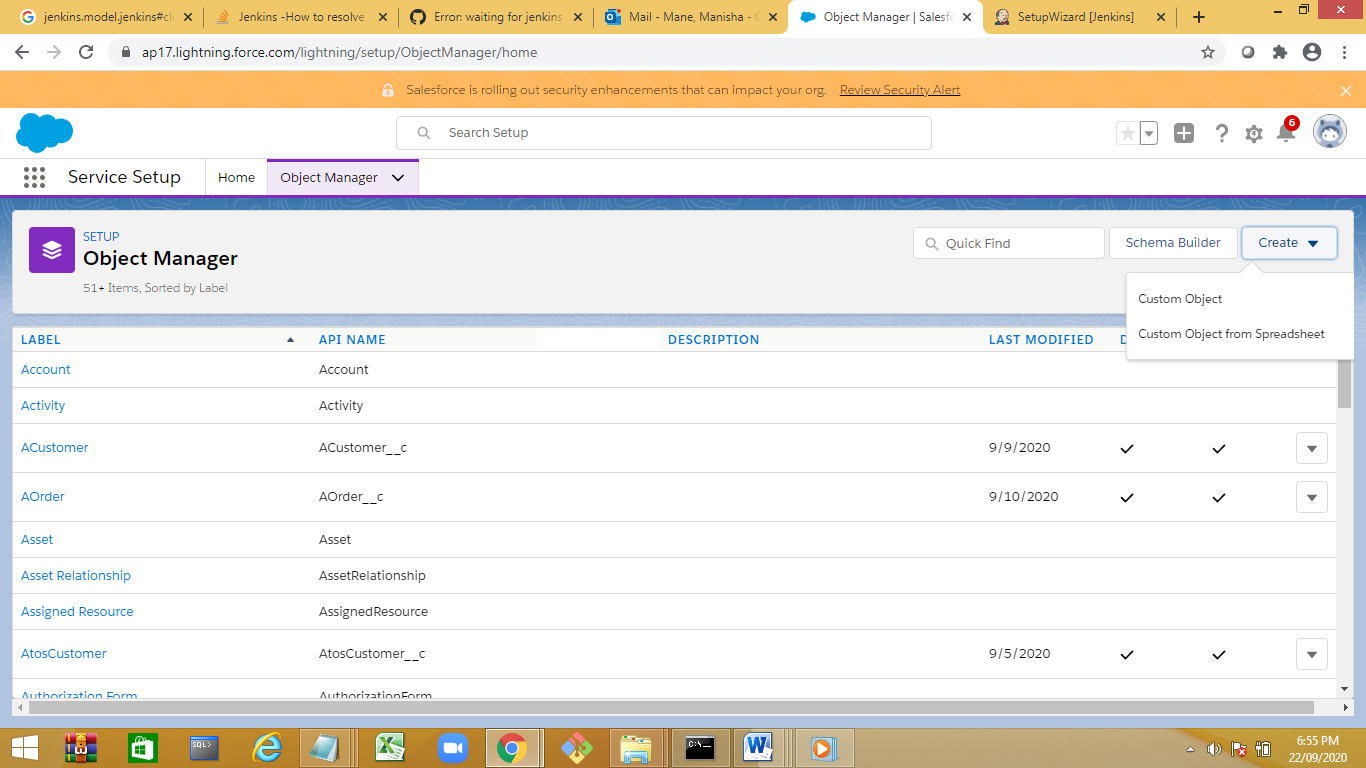
System.debug('record is updated ');

}

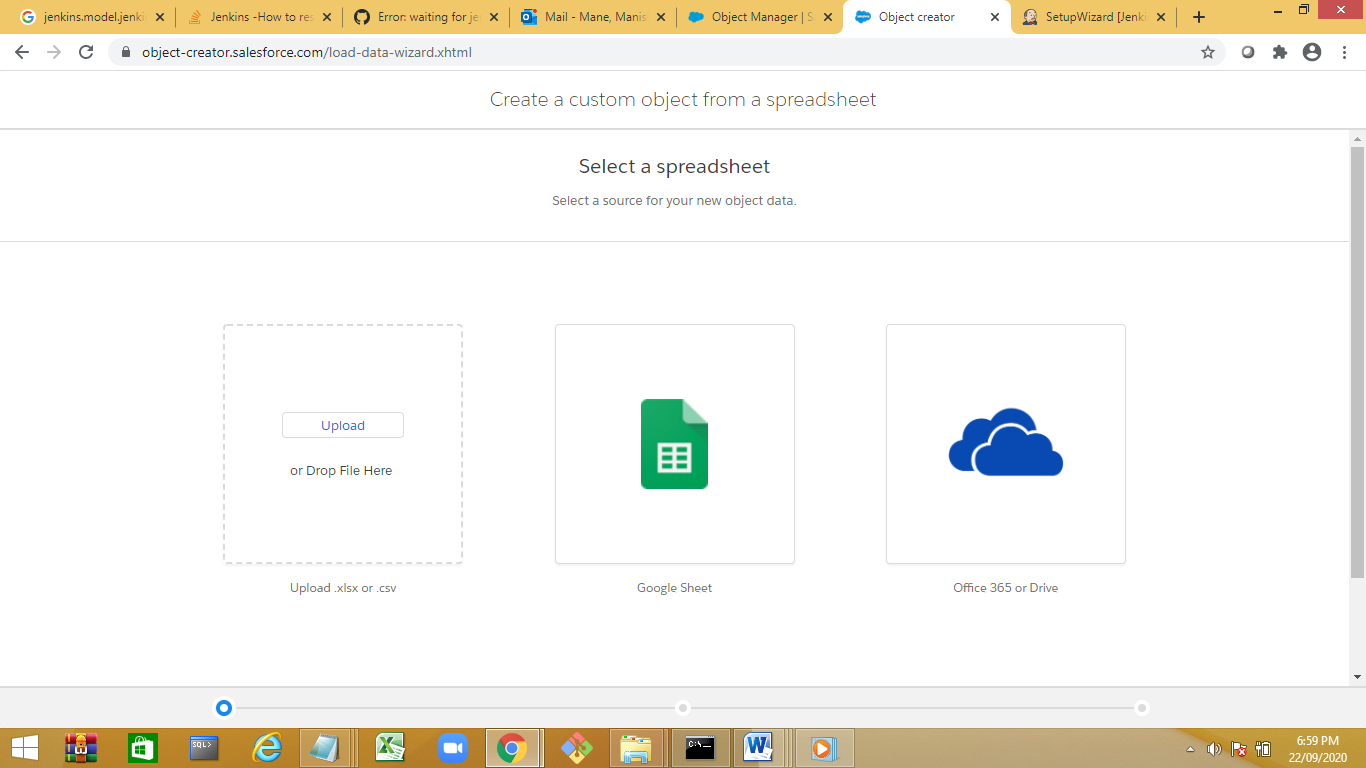
}

Custom App –

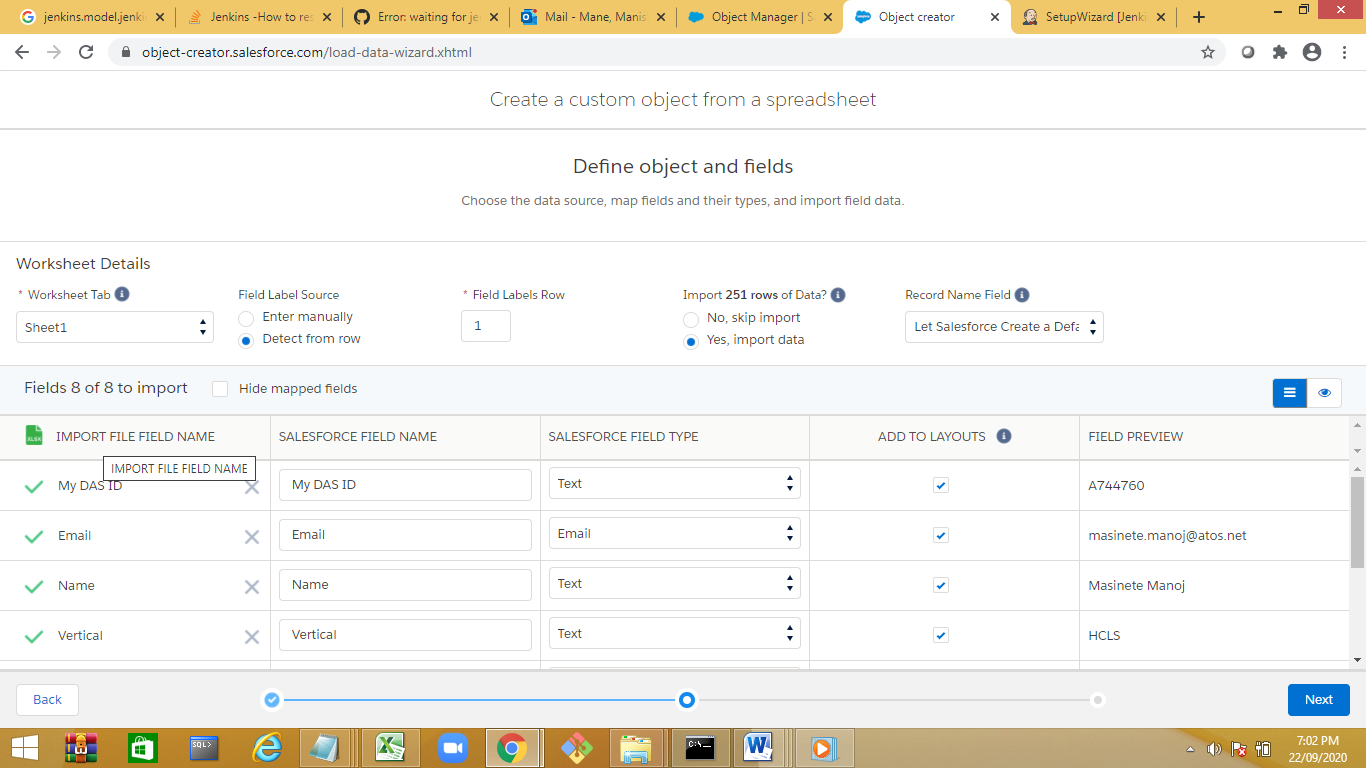
Creating custom object from spread sheet:



Login with salesforce



Upload the excel file



Follow the wizard,

