**PRACTICAL NO : 09A A65\_PRASANNA ANJANKAR**

**Aim :** Write an apex code to perform DML operatios on standard or custom objects created by user.

**Theory :**

You can perform DML operations using the Apex DML statements or the methods of the Database class. For lead conversion, use the convert Lead method of the Database class. There is no DML counterpart for it.

**Apex DML Statements**

Use Data Manipulation Language (DML) statements to insert, update, merge, delete, and restore data in Salesforce. The following Apex DML statements are available:

**Insert Statement :**

The insert DML operation adds one or more sObjects, such as individual accounts or contacts, to your organization’s data. insert is analogous to the INSERT statement in SQL.

Syntax :

*insert sObject*

*insert sObject[]*

**Update Statement:**

The update DML operation modifies one or more existing sObject records, such as individual accounts or contacts, in your organization’s data. update is analogous to the UPDATE statement in SQL.

Syntax:

*update sObject*

*update sObject[]*

**Upsert Statement:**

The upsert DML operation creates new records and updates sObject records within a single statement, using a specified field to determine the presence of existing objects, or the ID field if no field is specified.

Syntax:

upsert sObject​​[opt\_field]

upsert sObject[]​​[opt\_field]

**Delete Statement:**

The delete DML operation deletes one or more existing sObject records, such as individual accounts or contacts, from your organization’s data. delete is analogous to the delete() statement in the SOAP API.

Syntax:

delete sObject

delete sObject[]

**Undelete Statement:**

The undelete DML operation restores one or more existing sObject records, such as individual accounts or contacts, from your organization’s Recycle Bin. undelete is analogous to the UNDELETE statement in SQL.

Syntax:

undelete sObject | ID

undelete sObject[] | ID[]

**Merge Statement****:**

The merge statement merges up to three records of the same sObject type into one of the records, deleting the others, and re-parenting any related records.

Syntax:

merge sObject sObject

merge sObject sObject[]

merge sObject ID

merge sObject ID[]

**Code and Srceenshots of Output :**

1. **Insert a book Record :**

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

<apex:form >

<apex:pageBlock >

<apex:messages />

<apex:pageBlockSection title="Create new Book" collapsible="false">

Hello,{!$User.FirstName}

</apex:pageBlockSection>

<apex:pageBlockSection title="Book details" collapsible="false">

<apex:panelGrid columns="4">

<apex:outputText value="Name of Book" />

<apex:inputField value="{!Book\_\_c.Name}" />

<apex:outputText value="Author"/>

<apex:inputField value="{!Book\_\_c.Author\_Name\_\_c}" />

<apex:outputText value="Price" />

<apex:inputField value="{!Book\_\_c.Price\_\_c}" />

<apex:outputText value="Quantity" />

<apex:inputField value="{!Book\_\_c.Quantity\_\_c}"/>

<apex:outputText value="Publication"/>

<apex:inputField value="{!Book\_\_c.Publication\_\_c}"/>

<apex:outputText value="Availability"/>

<apex:inputField value="{!Book\_\_c.Availability\_\_c}"/>

</apex:panelGrid>

</apex:pageBlockSection>

<apex:pageBlockButtons >

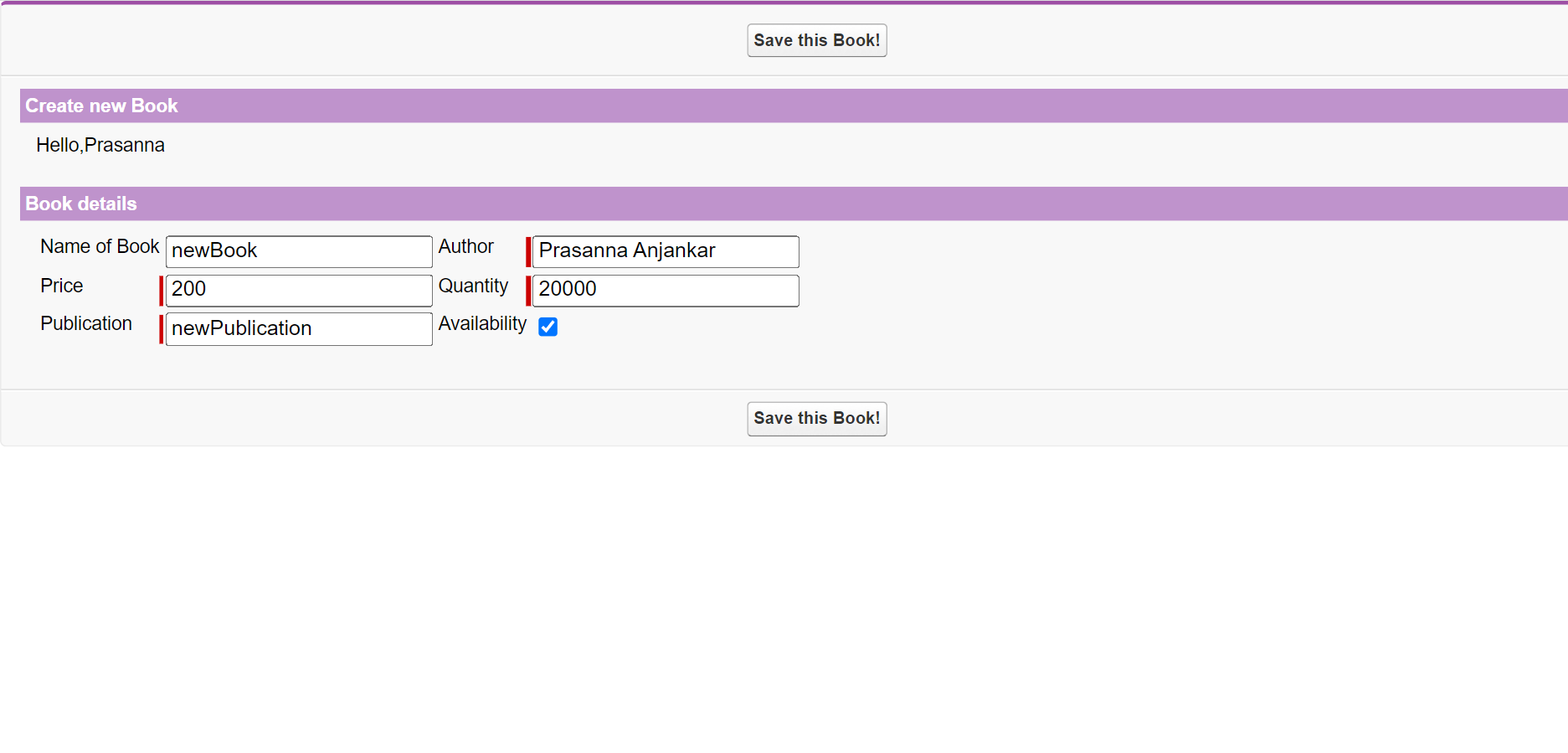
<apex:commandButton action="{!save}" value="Save this Book!"/>

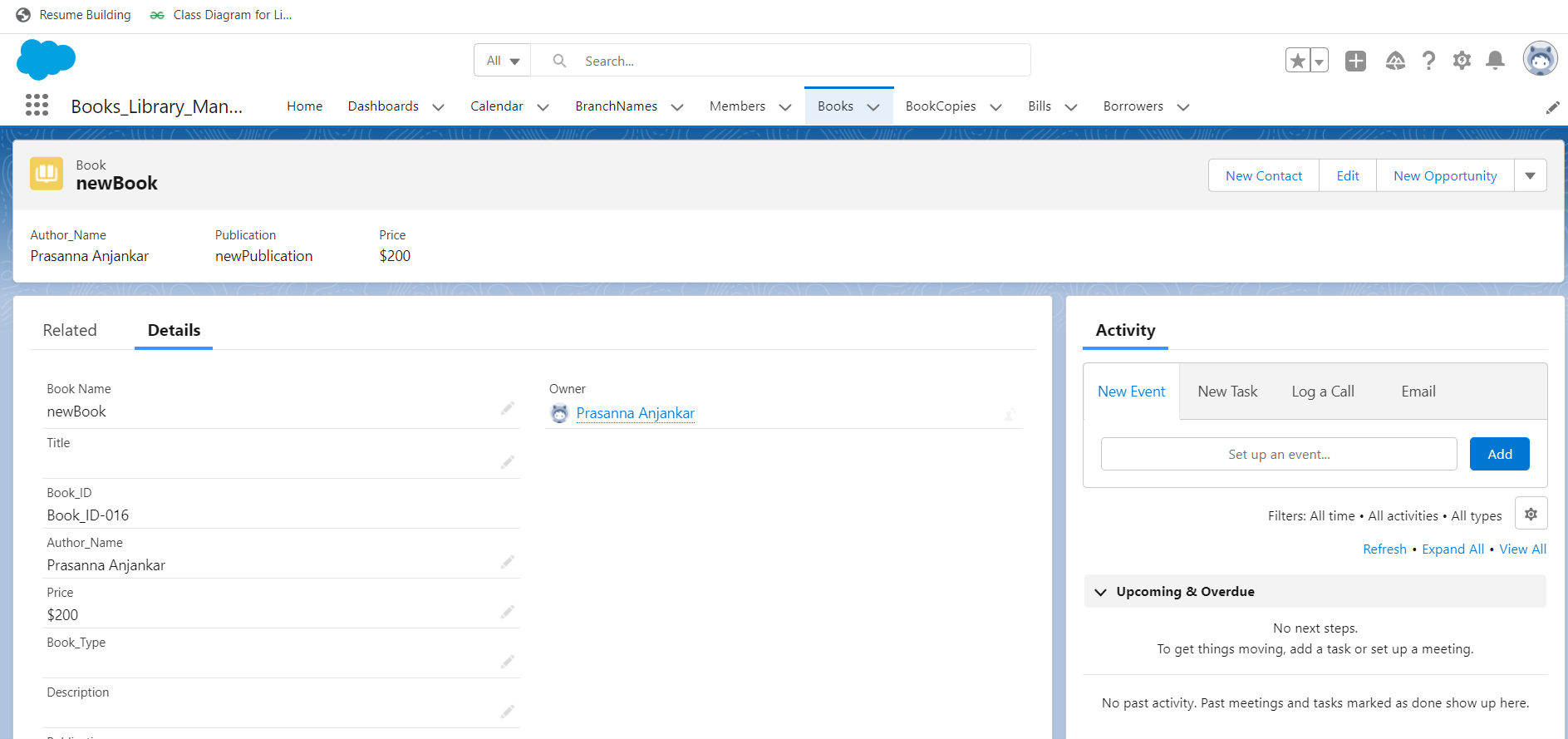
</apex:pageBlockButtons>

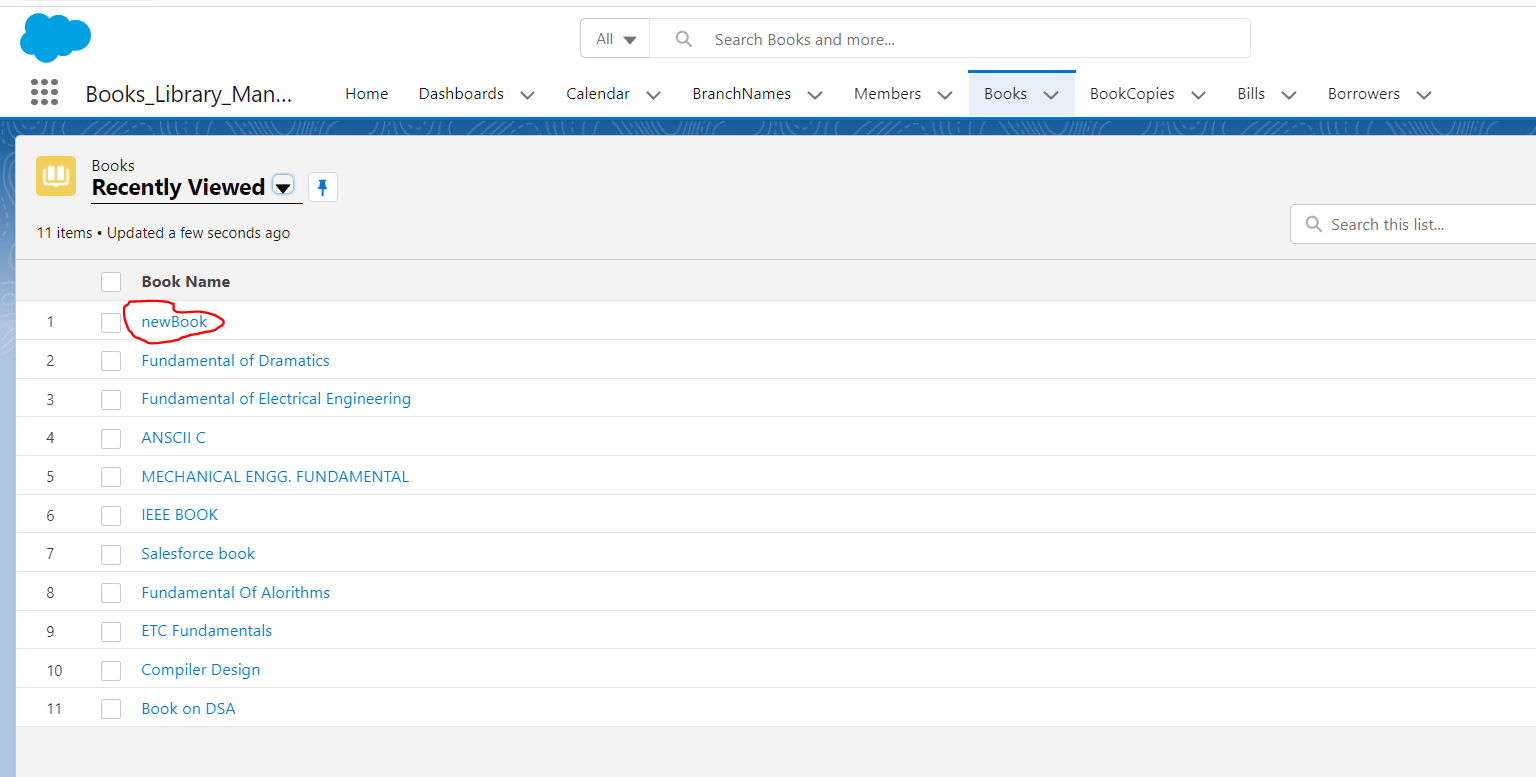
</apex:pageBlock>

</apex:form>

</apex:page>







**2. New member addition:**

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

<apex:form >

<apex:pageBlock >

<apex:messages />

<apex:pageBlockSection title="Create new Member" collapsible="false">

Hello,{!$User.FirstName}

</apex:pageBlockSection>

<apex:pageBlockSection title="Member details" collapsible="false">

<apex:panelGrid columns="4">

<apex:outputText value="Name of Member" />

<apex:inputField value="{!Member\_\_c.Name}"/>

<apex:outputText value="Phone"/>

<apex:inputField value="{!Member\_\_c.Phone\_No\_\_c}" />

<apex:outputText value="Email" />

<apex:inputField value="{!Member\_\_c.Email\_\_c}" />

<apex:outputText value="ID" />

<apex:inputField value="{!Member\_\_c.Member\_ID\_\_c}" />

<apex:outputText value="Country" />

<apex:inputField value="{!Member\_\_c.Country\_\_c}" />

<apex:outputText value="State" />

<apex:inputField value="{!Member\_\_c.State\_\_c}" />

<apex:outputText value="City" />

<apex:inputField value="{!Member\_\_c.City\_\_c}" />

<apex:outputText value="Branch" />

<apex:inputField value="{!Member\_\_c.BranchName\_\_c}" />

</apex:panelGrid>

</apex:pageBlockSection>

<apex:pageBlockButtons >

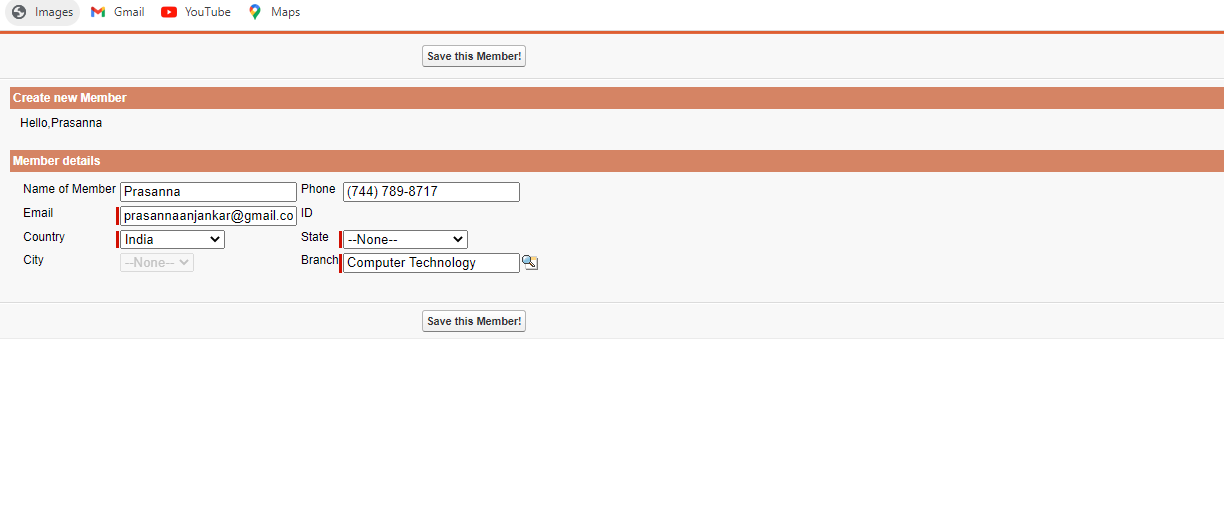
<apex:commandButton action="{!save}" value="Save this Member!"/>

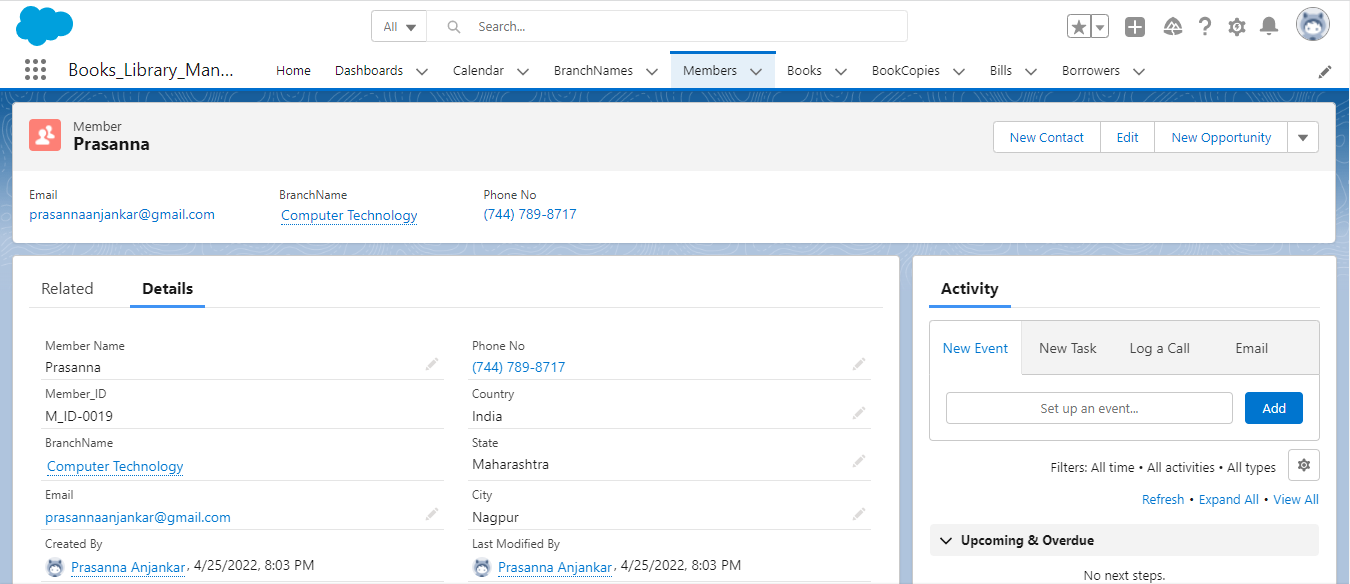
</apex:pageBlockButtons>

</apex:pageBlock>

</apex:form>

</apex:page>





**3. Members List:**

<apex:page standardController="Member\_\_c"

recordSetVar="mems">

<apex:dataTable value="{!mems}" var="m" border="1" cellspacing="2px">

<apex:column >

<apex:facet name="header"> Name</apex:facet>

<apex:outputText value="{!m.Name}"/>

</apex:column>

<apex:column >

<apex:facet name="header"> Phone</apex:facet>

<apex:outputText value="{!m.Phone\_No\_\_c}"/>

</apex:column>

<apex:column >

<apex:facet name="header"> E-mail id</apex:facet>

<apex:outputText value="{!m.Email\_\_c}"/>

</apex:column>

<apex:column >

<apex:facet name="header"> Country</apex:facet>

<apex:outputText value="{!m.Country\_\_c}"/>

</apex:column>

<apex:column >

<apex:facet name="header"> State</apex:facet>

<apex:outputText value="{!m.State\_\_c}"/>

</apex:column>

<apex:column >

<apex:facet name="header"> Member ID</apex:facet>

<apex:outputText value="{!m.Member\_ID\_\_c}"/>

</apex:column>

</apex:dataTable>

<apex:form >

<apex:commandLink action="{!previous}" value="previous"/>

<br/>

<apex:commandLink action="{!next}" value="next"/>

</apex:form>

</apex:page>



**4. Books List:**

<apex:page standardController="Book\_\_c" recordSetVar="booksCollections" >

<apex:pageBlock title="Books Here!">

<apex:pageBlockTable value="{!booksCollections}" var="b">

<apex:column headerValue="Book Name" value="{!b.Name}"/>

<apex:column value="{!b.Quantity\_\_c}"/>

<apex:column value="{!b.Price\_\_c}"/>

<apex:column value="{!b.Name}"/>

</apex:pageBlockTable>

<apex:form >

<apex:selectList value="{!filterid}" size="1">

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

</apex:selectList>

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

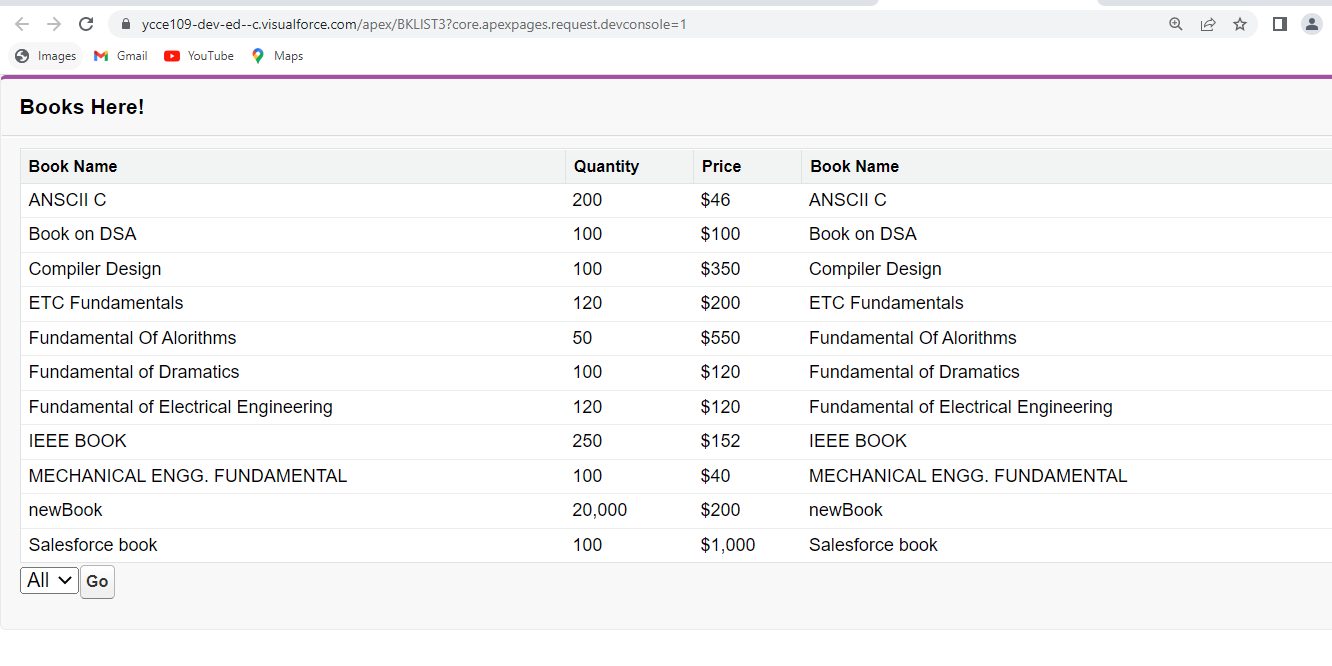
<br/>

<p align="center" style="font-size:14pt;font-style:bold"/>

</apex:form>

</apex:pageBlock>

</apex:page>



1. **Working with Trigger:**

trigger Book\_Trigger on Book\_\_c (before insert, before update , before delete)

{

//after insert, after delet, after update

//after undelete

LIST<FeedItem> posts = new LIST<FeedItem>();

if(Trigger.isInsert)

{

for(Book\_\_c b : trigger.new)

{

FeedItem post = new FeedItem();

post.ParentId = UserInfo.getUserId();

post.Body = b.Name\_of\_Book\_\_c + b.Author\_\_c + b.Availability\_\_c + b.B\_ID\_\_c + b.CreatedById + b.Publication\_\_c + 'Book Created';

posts.add(post);

}

insert posts;

}

else if (Trigger.isUpdate)

{

if(trigger.isBefore)

{

for(Book\_\_c b : trigger.old)

{

FeedItem post = new FeedItem();

post.ParentId = UserInfo.getUserId();

post.Body = b.Name\_of\_Book\_\_c + 'Book Updated';

posts.add(post);

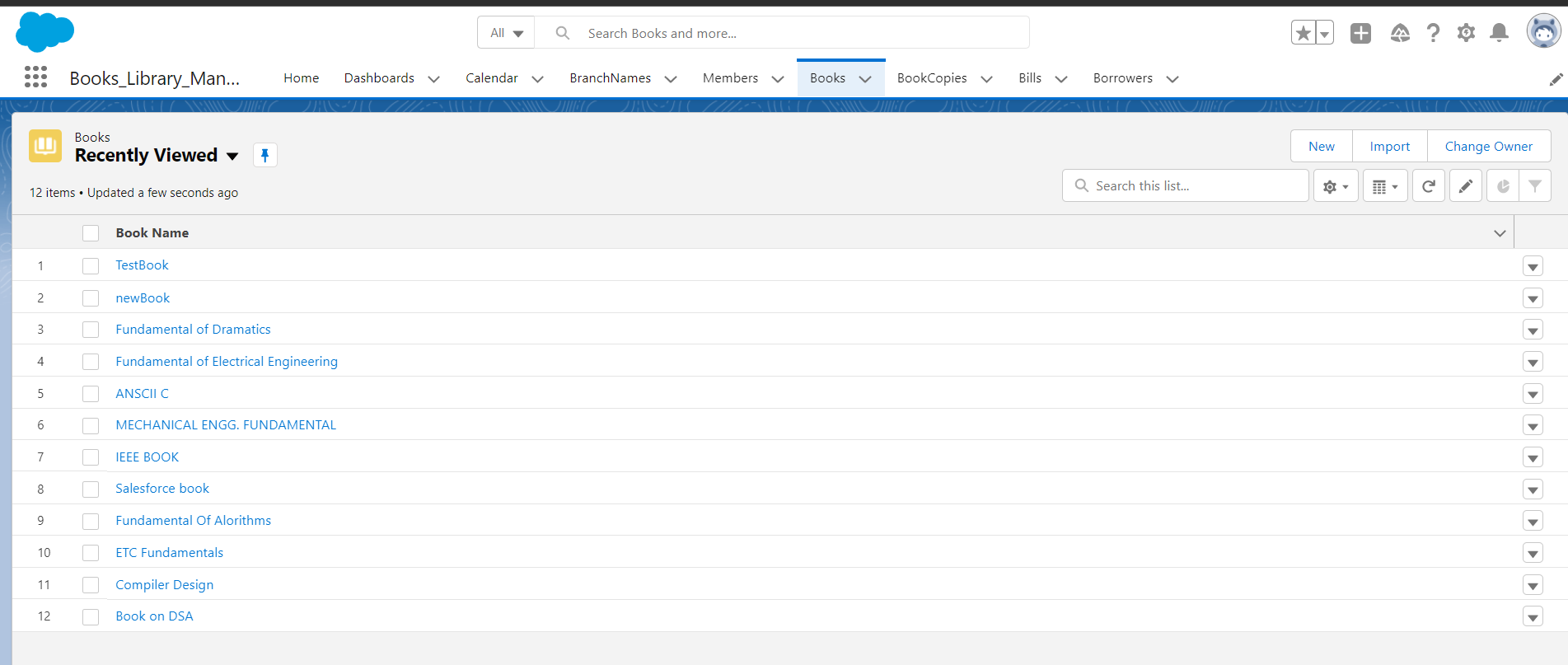
}

insert posts;

}

}

}



**Conclusion :**

Hence I have successfully executed the DML commands on saesforce objects.

**Name : Prasanna Rajendra Anjankar Registration No : 19010886**

**Roll No : 65 Class : 6CTA**

**Course Name : CRM Course Code : CT2370**

**PRACTICAL NO : 09B A65\_PRASANNA ANJANKAR**

**Aim:** Write an Apex Code to display the list of Books with its issuing Members with starting letters of Books.

**Theory:**Visualforce pages are webpages that belong to Salesforce. These webpages are created using a unique tag-based Mark-up language. It is similar to HTML but it's primary use is to access, display and update the organization’s data. The page is accessed by using a URL similar to that of a traditional webserver page.

Each tag in visual force language corresponds to some user interface component like section of a page, a list view or a field of an object. Interestingly, it can be easily mixed up with HTML markup, CSS style and Java libraries, etc.

## Creating a Visualforce Page

Go to the link **developer console → File → New → Visualforce page**. The new window opens asking for a page name. Click **Save**. Then, click on **Preview**. This opens a new webpage showing the result as shown in the following screenshot.

**Standard Components or tags:**

Salesforce.com by default provides 60 standard User Interface components. These are the components provided by Salesforce and these are started with <Apex:  Tagname> </Apex: Tagname>. It creates perfect look and feel of the Salesforce without using additional CSS and HTML.

**Custom Components or Custom tags:**

There are the components created by users and starts with <c : Tagname> </c : Tagname>.

**Standard Controllers**

Standard Controllers are used to customize both Standard objects and Custom objects. These controllers have access to Salesforce database and automatically adjusts to the type of the pages based on where it is called.

We can use Standard controllers for custom objects and Standard objects.

They provide standard functionality like save, delete and create records.

Some standard object don’t have standard controllers.

**Custom Controller**

Custom controller is a Apex class that implements all the login of visualforce page without leveraging the standard functionality. These are used to create wizards, rich UI with complex data sets and so on.

|  |
| --- |
| <apex: page controller="CustomApexClass"> |

Custom Apex Class is an Apex class that controls the Visualforce Actions and Variables.

**Extension Controllers**

Extension controller in visualforce is an Apex class that is used to add and extend the functionality to Standard controller and Custom controller. It is also used to overwrite the methods.

This controller can override existing standard links and a button.

|  |
| --- |
| <apex: page standardController="Media\_\_c"         extensions="CustomApexClass, SecondCustomApexClass"> |

Visualforce Components.

* [apex:page:](https://www.tutorialkart.com/visualforce/standard-visualforce-components-apexpage-component/) **apex:page** is a Standard Visualforce component. **apex:page** component is the very important in Visualforce because every Visualforce page will start with this component. This says that a single Visualforce page and all the components must be wrapped inside apex:page component.
* [apex:pagemessage:](https://www.tutorialkart.com/visualforce/apexpagemessage-component/) **apex:pagemessage component** will generate and displays all error messages of all components on the current page, presented using the SF styling. This component will have some attributes that to be used when apex:pagemessage component is used in Visualforce page.
* [apex:Form](https://www.tutorialkart.com/visualforce/apex-form-component/): **Apex Form Component** is used to create Forms in Visualforce page. Apex input and output components are added inside <apex:form> Component. It is required to add only one <apex:form> tag instead of adding many form tags. The body of the form determines the data that is displayed and the way it is processed.
* [apex:Commandbutton](https://www.tutorialkart.com/visualforce/apexcommandbutton-component/): apex:CommandButton component is used to create a button on Visualforce page. <apex:CommandButton>must always be a child of<apex:form> component, which means to create a button apex:CommandButton must written in apex:form tag. Button in Visualforce page is rendered as an HTML output element.
* [apex:pageblockbutton:](https://www.tutorialkart.com/visualforce/apex-pageblockbuttons-component/) Apex pageblockbuttons Component is used to create  set of buttons on pageBlock component. Apex pageblockbuttons component can be styled like Standard salesforce buttons and the the component must be a child component of an **[apex:pageblock](https://www.tutorialkart.com/visualforce/apexpageblock-component/)**component. This apex pageblockbuttons tag creates buttons and can be placed on the top, bottom and on both using “Location” attribute.
* [apex:pageblockSection](https://www.tutorialkart.com/visualforce/apex-pageblocksection-component/): Apex PageBlockSection component is used to create sections within a page block to categorize different fields into different sections. It’s similar to adding a section by using a edit page layout assignment. Apex PageBlockSection component consists of two cells, one for “**Label**” and another for “**value**“. Each component found in the body of an <apex: pageblock section> is placed into the next cell in a row until the no. of columns is reached. Apex pageBlockSection must be a child component to **<apex:pageBlock>** component. **<apex:inputField>** and **<apex:outputField>** are the two components used to add fields from a Salesforce object.
* [apex:pageblockSectionItem:](https://www.tutorialkart.com/visualforce/apex-pageblocksectionitem-component/) **Apex pageblocksectionItem** component can include up to two child components where an **<apex: pageblocksection>** creates one column in one row. If there is no content included in apex pageblocksectionItem component, then the content spans both cells of the column. If two child components are specified , the content of the first is rendered in left, **“label”** cells of the column, while the content of the second is rendered in the right , **“data’’** cell of the column.

**Code :**

<apex:page standardController="Borrower\_\_c"

recordSetVar="booksCollections" >

<apex:pageBlock title="Members who issued Books: ">

Hello, {!$User.FirstName} {!$User.LastName}<br />

These is the List of Borrowers: <br /><br />

<apex:pageBlockTable value="{!booksCollections}" var="b">

<apex:column headerValue="Borrower Name" value="{!b.Name}"/>

<apex:column headerValue="Book Name" value="{!b.Book\_\_c}"/>

<apex:column headerValue="Issue Date" value="{!b.Issue\_Date\_\_c}"/>

<apex:column headerValue="Expected Return Date" value="{!b.Return\_Date\_\_c}"/>

</apex:pageBlockTable>

<apex:form >

<apex:selectList value="{!filterid}" size="1">

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

</apex:selectList>

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

<br/>

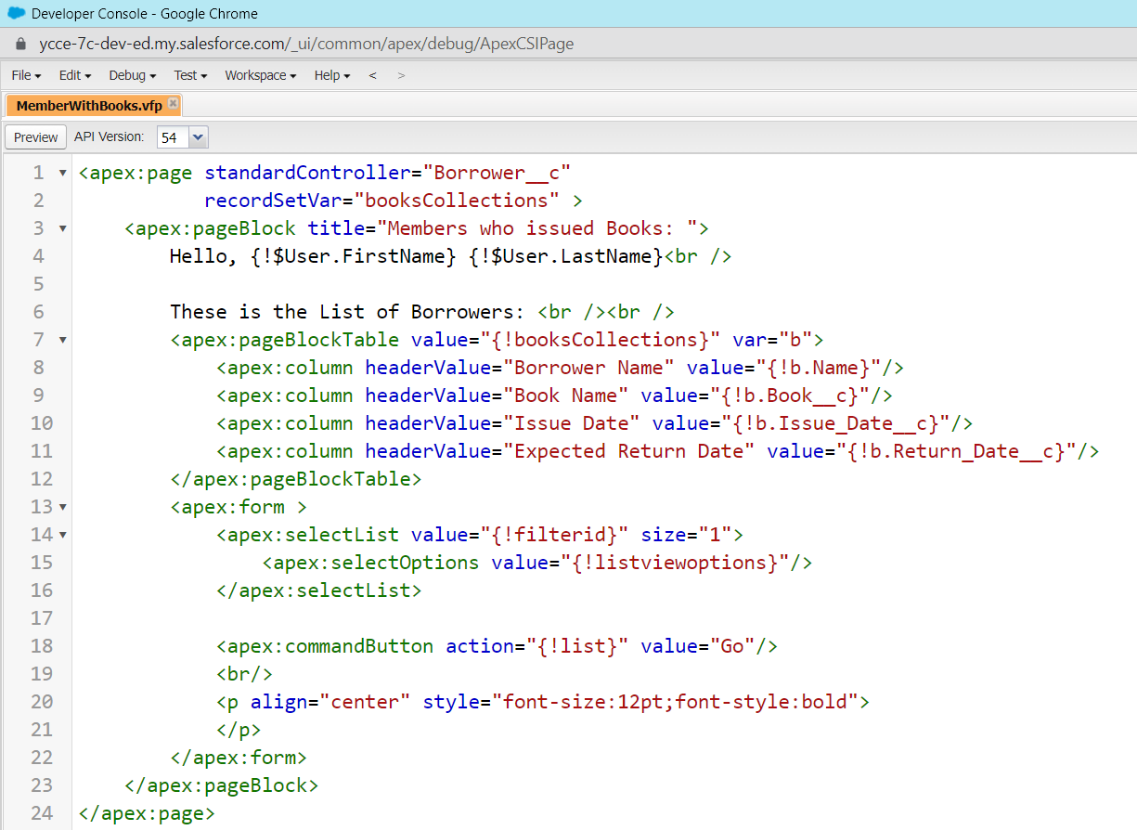
<p align="center" style="font-size:12pt;font-style:bold">

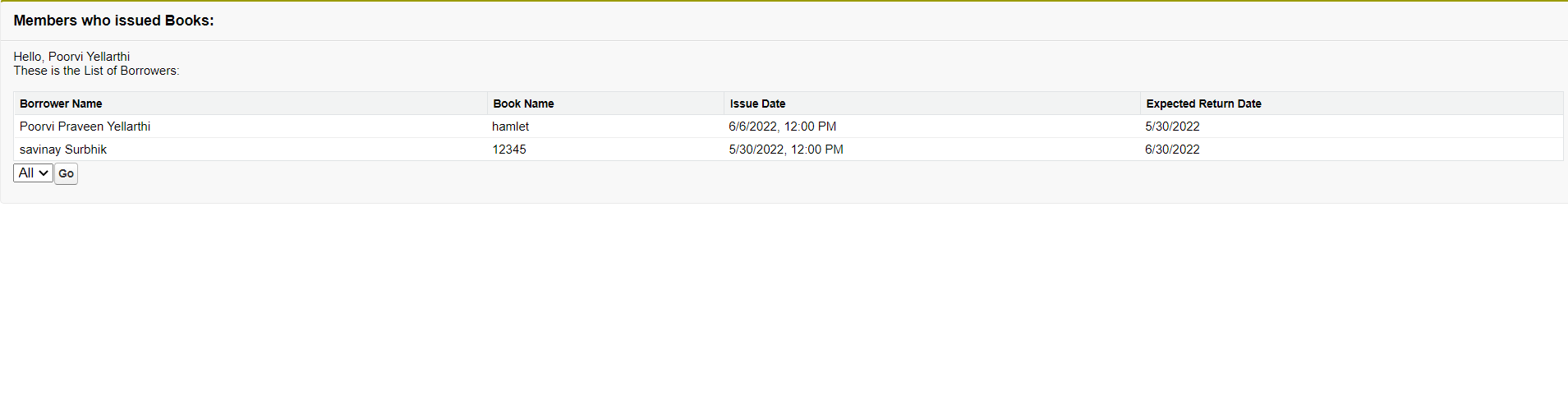
</p>

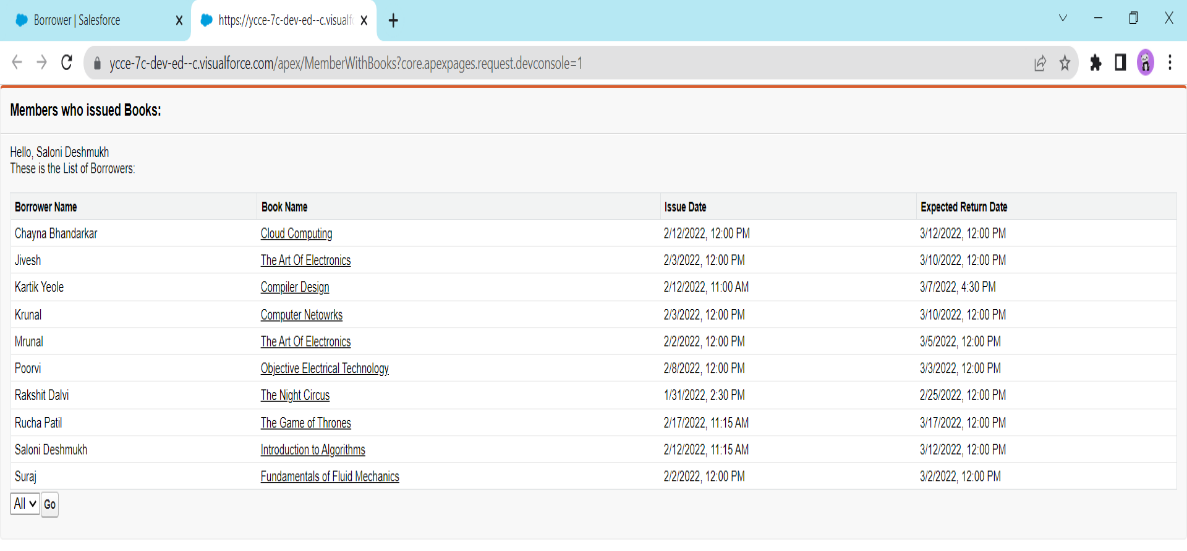
</apex:form>

</apex:pageBlock>

</apex:page>







**CONCLUSION:** Hence, I’ve written an APEX Code for a Visualforce page which displays the Borrowers’ Name, Books’ name, Issue Date and Expected Return Date according to the data in Book Library Management App.

**Name : Prasanna Rajendra Anjankar Registration No : 19010886**

**Roll No : 65 Class : 6CTA**

**Course Name : CRM Course Code : CT2370**