ROHINI COLLEGE OF ENGINEERING & TECHNOLOGY
Near Anjugramam Junction, Kanyakumari Main Road, Palkulam, Variyoor P.O - 629401
Kanyakumari Dist, Tamilnadu., E-mail : admin@rcet.org.in, Website : www.rcet.org.in

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

NAME OF THE SUBJECT : Internet Programming

Subject code : CS8651

Regulation : 2017

UNIT V INTRODUCTION TO AJAX and WEB SERVICES

Unit -5

Introduction to AJAX

- AJAX stands for Asynchronous Javascript And XML.
- AJAX is not a programming language.
- AJAX is a way of using existing standards (JavaScript and XML) to make more interactive web applications.
- AJAX was popularized in 2005 by Google (with Google suggest)

An AJAX Application

- Recall the standard HTTP transaction
    1. Client opens connection to server
    2. Client sends request to server
    3. Server sends reply to client
    4. Client and server close connection

• After Step 4, the client renders the document and this may include running some JavaScript.

• In an AJAX application, the JavaScript code then communicates with the server behind the scenes.

- Communication with the server takes place asynchronously, and transparently to the user
- Data is exchanged with the server without the need for a page reload • This is accomplished through a special kind of HTTP request.

An AJAX – Examples

- Google Maps
- Google Suggest
- Gmail
- Yahoo Maps (new)

AJAX - Browser Support

All the available browsers cannot support AJAX. Here is a list of major browsers, that support AJAX.

- Mozilla Firefox 1.0 and above.
- Netscape version 7.1 and above.
- Apple Safari 1.2 and above.
- Microsoft Internet Explorer 5 and above.
- Konqueror.
- Opera 7.6 and above.

Typical AJAX Event

• A typical AJAX transaction looks like this:

1. User triggers some event (presses a key, moves mouse, ...)

2.Event handler code sends HTTP request to server

3.Server replies triggering code on client

4. Reply handler code updates web page using server's reply

• Between steps 2 and 3 the web page is still usable (event is asynchronous)

• At no point during the transaction does the browser open a new web page

Pros and Cons of AJAX

• **Pros**:

– Allows web applications to interact with data on the server

– Avoid clunky GET/POST send/receive interfaces

– web apps look more and more like real applications

– Some applications can only be realized this way

• Eg: Google Suggest offers interactive access to one of the largest data collections in the world – For office style applications, user's data is stored on a reliable server, accessable from any web browser

• **Cons:**

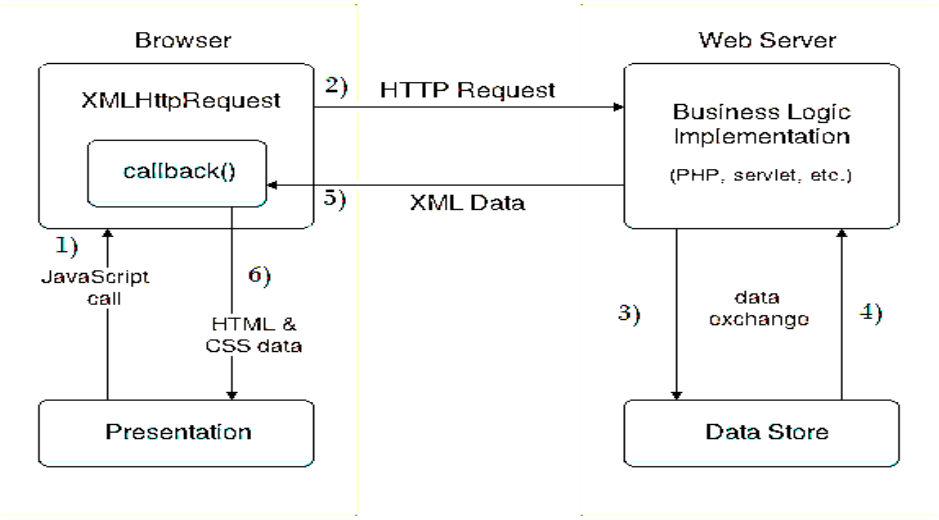
– Tough to make compatible across all browsers

– Should have a low-latency connection to the server

– Can be server intensive

• Eg: Google Suggest generates a search for every keystroke entered.

An AJAX Architecture



XMLHttpRequest object plays a important role.

1. User sends a request from the UI and a javascript call goes to XMLHttpRequest object.
2. HTTP Request is sent to the server by XMLHttpRequest object.

3. Server interacts with the database using JSP, PHP, Servlet, ASP.net etc.
4. Data is retrieved.
5. Server sends XML data or JSON data to the XMLHttpRequest callback function.
6. HTML and CSS data is displayed on the browser.

Setting up an AJAX Transaction

• Create an **XMLHTTPRequest** object

• Set up the request's **onreadystatechange** function

• Open the request

• Send the request

Creating an XMLHTTPRequest Object

```
function sendRequest()
var xmlHttp = GetXmlHttpObject();
if (!xmlHttp)
{ return false; }
xmlHttp.onreadystatechange = function()
{
if (xmlHttp.readyState == 4)
{ alert("Request complete");
}
}
var requestURI = "http://myserver.org/somepage.txt";
xmlHttp.open("GET" , requestURI, true);
xmlHttp.send(null);
}
```

The XMLHTTPRequest Object

• An XMLHTTPRequest object is in one of 5 states, as indicated by the readyState property

0. The request is not initialized

1. The request has been set up
2. The request has been sent
3. The request is in process
4. The request is complete

• Every time the readyState property changes the onreadystatechange property (a function) is called

Setting onreadystatechange

```
function sendRequest()
var xmlHttp = GetXmlHttpObject();
if (!xmlHttp) { return false; }
 xmlHttp.onreadystatechange = function()
 { if (xmlHttp.readyState == 4)
 { alert("Request complete"); }
}
var requestURI = "http://myserver.org/somepage.txt";
xmlHttp.open("GET" , requestURI, true);
 xmlHttp.send(null);
 }
```

The open and send functions

• The **open** function of an XML HTTP request takes three arguments

– xmlHttp.open(method, uri, async)

– method is either "GET" or "POST"

– uri is the (relative) URI to retrieve

– async determines whether to send the request asynchronously (true) or synchronously (false)

– The domain of the uri argument must be the same as the domain of the current page

• The **send** function takes one argument

– xmlHttp.send(content);

– content is the content to send (useful when method= "POST")

Sending the Request

```
function sendRequest()
var xmlHttp = GetXmlHttpObject();
if (!xmlHttp) { return false; }
xmlHttp.onreadystatechange = function()
{ if (xmlHttp.readyState == 4) { alert("Request complete"); } }
var requestURI = "http://myserver.org/somepage.txt";
xmlHttp.open("GET" , requestURI, true);
xmlHttp.send(null);
}
```

The responseText Property

• When an XMLHTTPRequest is complete (readyState == 4) the responseText property contains the server's response, as a String.

Example Code (Client Side)

```
function sendRequest(textNode)
var xmlHttp = GetXmlHttpObject();
if (!xmlHttp) { return false; }
xmlHttp.onreadystatechange = function()
{ if (xmlHttp.readyState == 4)
{ textNode.nodeValue = xmlHttp.responseText; } }
var requestURI = "http://greatbeyond.org/cgi-bin/request.cgi";
xmlHttp.open("GET" , requestURI, true);
xmlHttp.send(null);
}
```

Example Code (Server Side)

And we might have the following request.cgi in the cgi-bin directory of greatbeyond.org

```
#!/usr/bin/perl
print("Content-type: text/plain\n\n");
print("57 channels and nuthin' on");
```

Respond with XML

- We can look at the XML text within a response using the responseText property
- Even better, we can use the responseXML property to access the XML
- Best, responseXML.documentElement contains the document tree for the XML
- This is a document tree in the DOM model that we've seen before (just like document)

Example

```
function sendRequest() {

var xmlHttp = GetXmlHttpObject();

if (!xmlHttp) { return false; }

 xmlHttp.onreadystatechange = function() {

 if (xmlHttp.readyState == 4) {

 var xmlDoc = xmlHttp.responseXML.documentElement;

} }

var requestURI = xmlURI;

xmlHttp.open("GET" , requestURI, true);

xmlHttp.send(null);

}
```

Web Services

Web services are open standard (XML, SOAP, HTTP, etc.) based web applications that interact with other web applications for the purpose of exchanging data.

Web services can convert your existing applications into web applications.

- Is available over the Internet or private (intranet) networks
- Uses a standardized XML messaging system
- Is not tied to any one operating system or programming language
- Is self-describing via a common XML grammar
- Is discoverable via a simple find mechanism

Components of Web Services

The basic web services platform is XML + HTTP. All the standard web services work using the following components −

- SOAP (Simple Object Access Protocol)
- UDDI (Universal Description, Discovery and Integration)
- WSDL (Web Services Description Language)

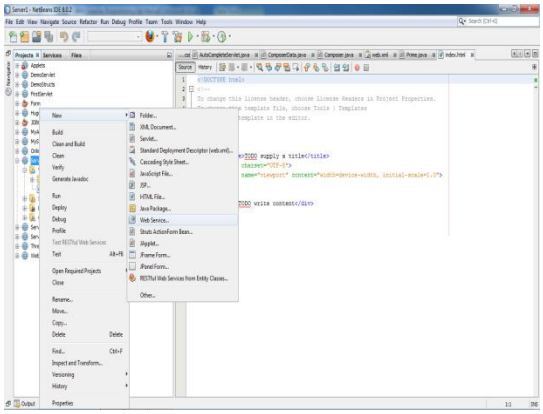
A web service enables communication among various applications by using open standards such as HTML, XML, WSDL, and SOAP. A web service takes the help of −

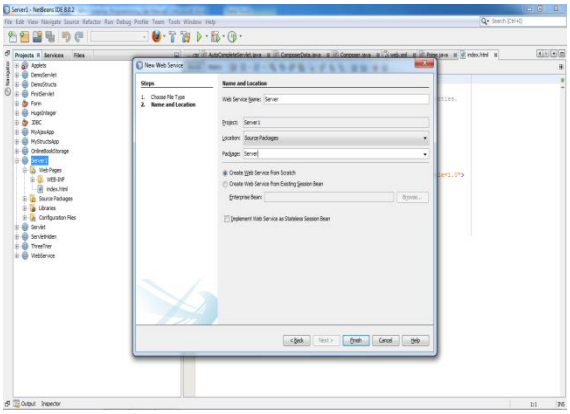
- XML to tag the data
- SOAP to transfer a message
- WSDL to describe the availability of service.

Creating, Publishing, Testing and Describing a Web Service (WSDL)

Server Part:

1. First Go to Files – > New Project or press Ctrl + shift+ N.
2. Now select **Java Web** from categories and **Web Application** from projects.
3. Now click **Next.** We are now first going to create server. So give name as **Server(**Preferably). And now click next and Click **Finish.**
4. Now Right click **Project (i.e. Server)** and **New -> Web service.** If you cannot find in the pop up click other and find there.
5. Give Web service name and Package name. Remember this name as we will be using it in further areas.(eg.Web service name is Serrver and package name is com.java.Server)

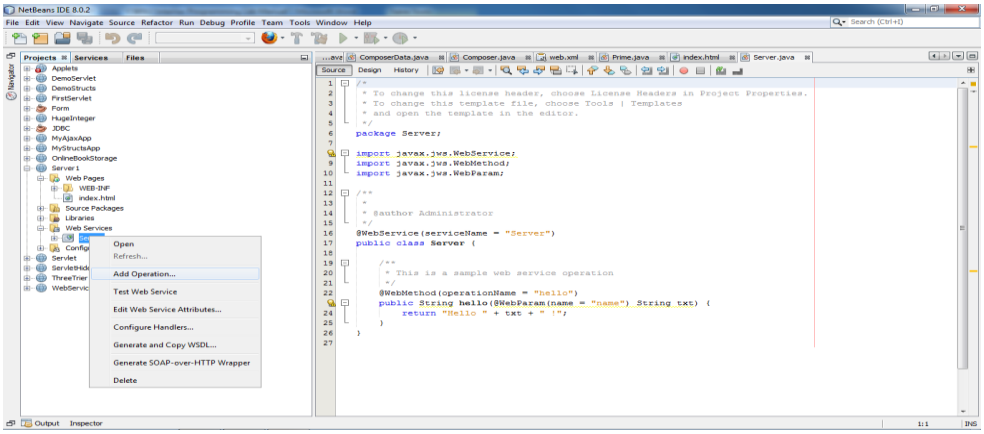


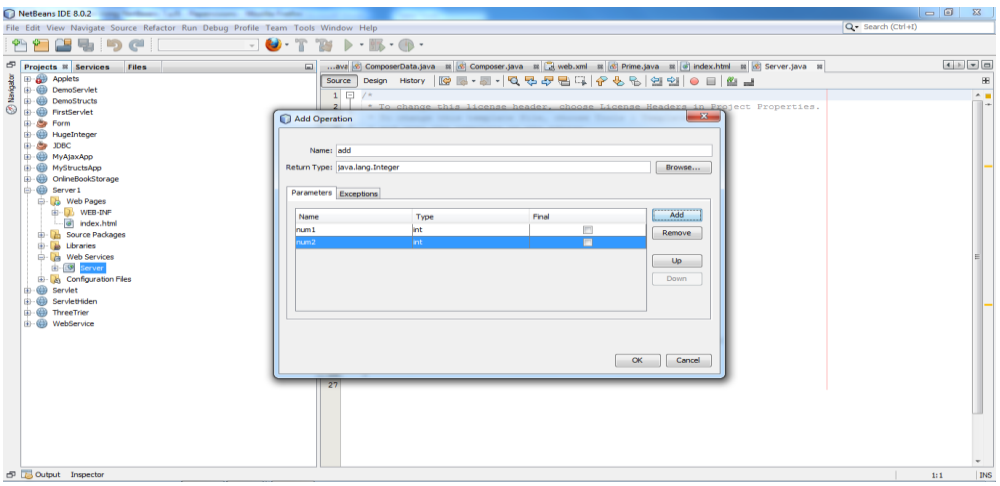


6. Now we need to add operations for the web service we have created. So **Right Click the web service and Select Add operations.(Ensure you click** the created **web service)**

7. Now add operation. For now I will teach you to do addition alone.

Give name of operation. As we are dealing with Integer return type is Integer and Click **Add** and give name as num1 and Type and also create another number as num2 for number 2. (Since addition requires two numbers). Check at left bottom for the operation signature.





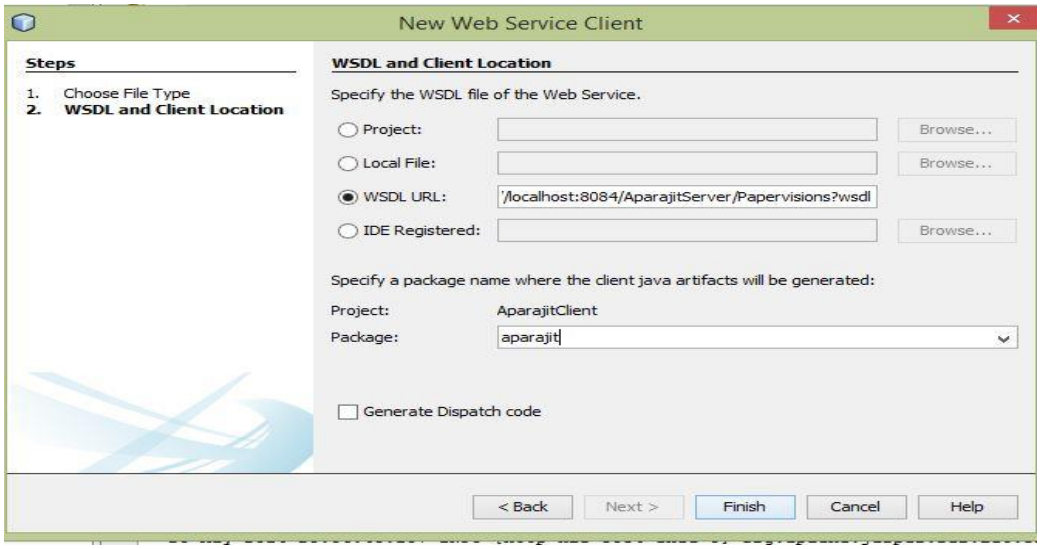
8. Click ok. And now **right click** project and click **Deploy.**

9. Now After deploying (It will take few minutes to deploy, be patient) right click the web service and click **Test Web Service.**

10. Our browser will be opened and now copy the WSDL and you will be in need of it later.

Client Service Creation

1. First Go to Files – > New Project or press Ctrl + shift+ N.
2. Now select **Java Web** from categories and **Web Application** from projects.
3. Now Click **Next**. We are now first going to create server. So give name as **Client**(Preferably). And click **Next** and **Finish**.
4. Now right click the **Project** and **select web service client**. If you cannot find in the pop up click other and find there.
5. Now select WSDL and paste the URL that you have copied.(Looks like this :http://localhost:8084/Server/Server?wsdl) And then enter the package name of the server. Refer Step 5 In server



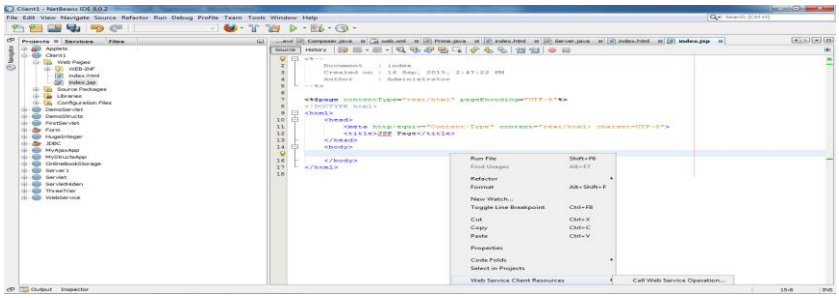
6. Right Click the Project And Create New HTML file with GUI(Form). Give a name (**Cal.html**)for HTML
7. Now we are going to design the front end. Just create two text boxes and label with Number1 and Number2. Ensure that code has action=index.jsp. and now create a new JSP fil(Right click **Client** project and select JSP). Give name as index.

```
<html>
  <head>
    <title>Calculator</title>

  </head>
  <body>
  <center>
    Calculator<sup>Beta version</sup>
    <hr><br><br><br><br><br>
    <form name ="webservice"  action="index.jsp" method="post">
    Number 1 :<input type="text" name="i1" value="" /> <br>
    Number 2 :<input type="text" name="i2" value="" /><br>
```

```
<input type="submit" value="add" name="b1" /> </center>
<br><br><br><br><br><br><br><hr>
</body>
</html>
```

8. Go to index.jsp page and delete Hello world line.



9. Call Web Service Operation and select your operation .i.e. add. You have code automatically generated .

Now num1 and num2 have 0 as value. But we need to get from user and perform this add operation.

10. So delete int num1 and int num2 value and add as follows :

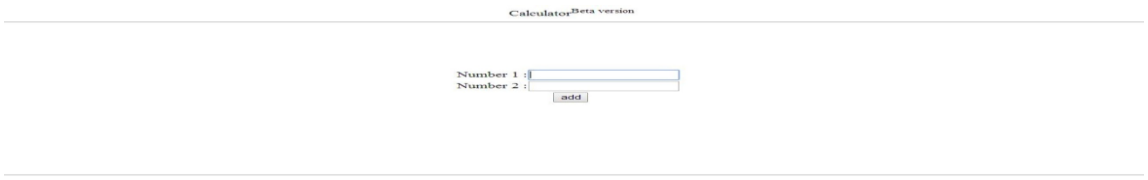
int num1 = Integer.parseInt(request.getParameter(“i1″));
int num2 = Integer.parseInt(request.getParameter(“i2″));;

i1 and i2 is the reference to the HTML document where you gave name for text box.

11.Now you need to edit the java file in server that performs the actual web service operation.In the operation add, change the return statement as **return num1+num2.**

12. Now run the calc.html file.

You will get as follows



OUTPUT:

```
Result = 7
```

Database Driven Web Service From An Application Soap

develop a simple user registration Webservices. The user registration/account registration form will be presented to the user. Once user fills in the form and clicks on the "OK" button, the serverside JSP will call the webservice to register the user.

This webservices will expose the insert user operation which will be used by the JSP client to register the user. We will use the NetBeans 6.1 IDE to develop and test the application.

The MySQL database is used to save the user registration data. You can modify the code to use any database of your choice. The existing webservices can also be modified to use the Hibernate or any other ORM technologies. You can also use the Entity beans to persist the data into database.

Software required to develop and run this example:

- JDK 1.6
- NetBeans 6.1
- MySQL Database 5 or above

Let's get started with the development of the applicaton

MySql Database Configuration In NetBeans
Let's configure MySQL database in teh NetBeans IDE and then create the required table into database.

Step 1:

- Click on the service tab in NetBeans as shown below in Fig 1.

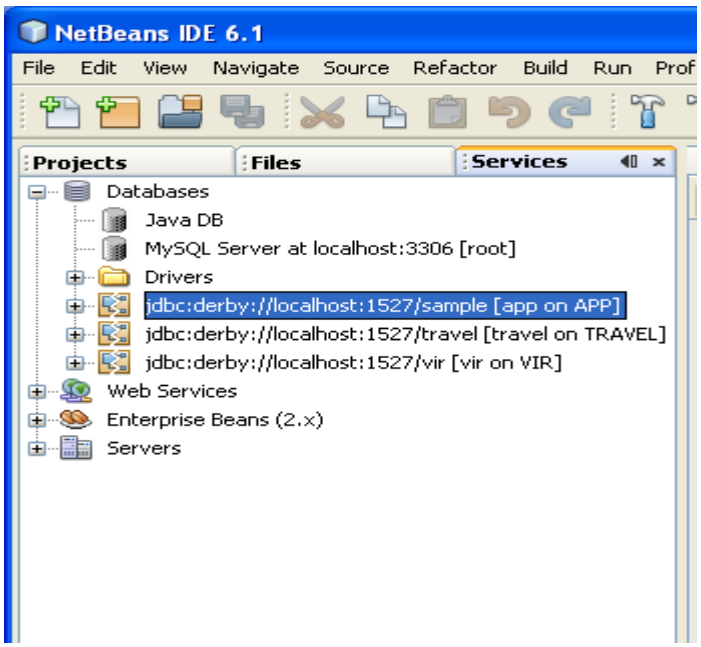


Fig. 1

Step 2:

- Right Click on the Databases
- Select New Connection as shown below in Fig 2.

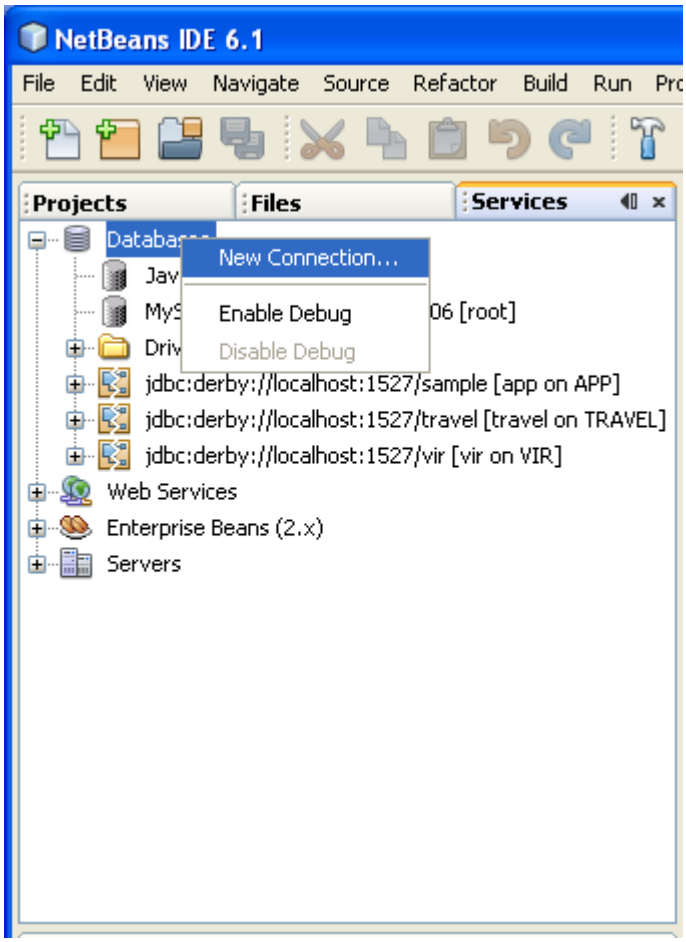


Fig. 2.

Step 3:

- It opens a dialog box for the mysql configuration.
- Type the driver name, url , user name and password as shown below in Fig. 3.

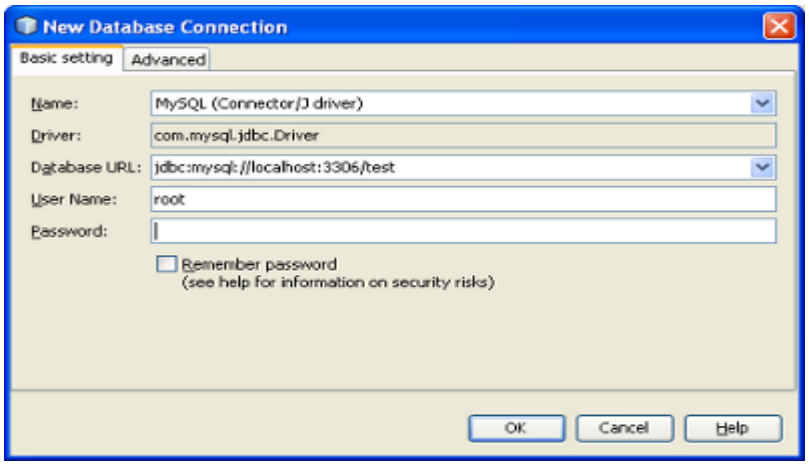


Fig. 3

Step 4:

- Click on the Ok button .
- Now expand the Newly created database connection.
- It shows the all the tables of the database test as shown below in Fig 4.

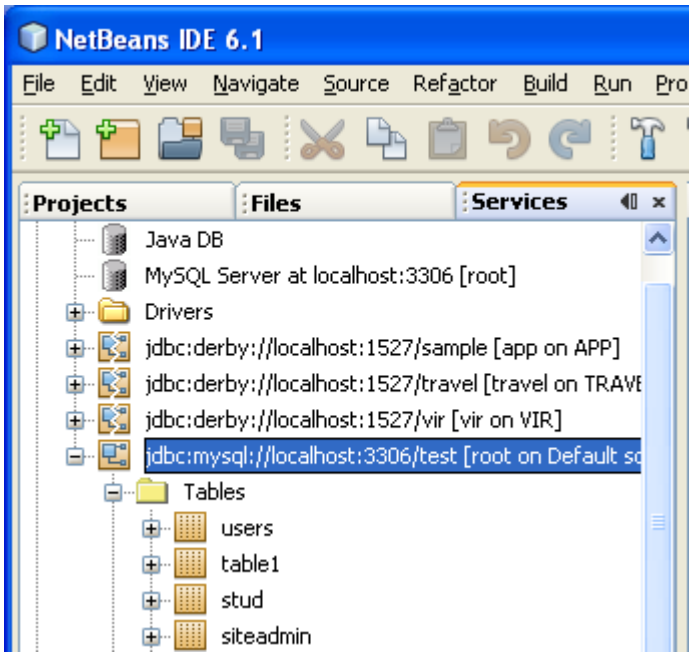


Fig. 4

Step 5:

- Create a table named login.
- Right Click on the Tables and select Create table as shown below in Fig 5

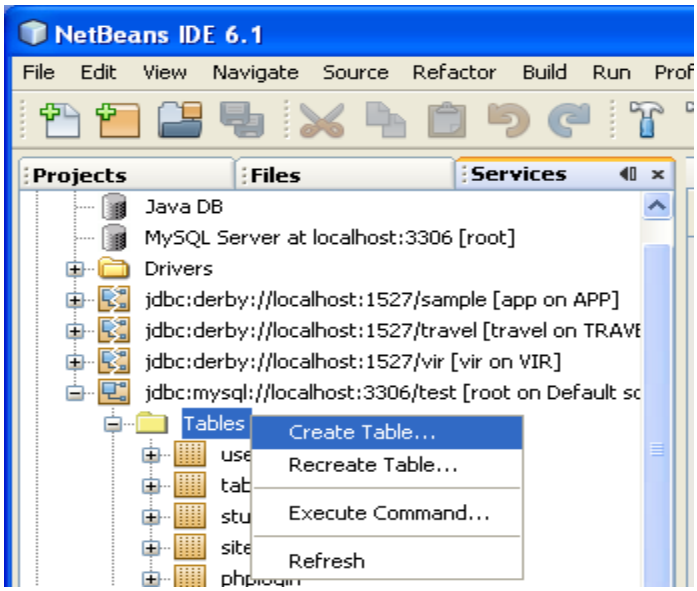


Fig. 5

Step 6:

- It opens a dialog box for giving the fields name of the table
- Now give the field name and data type as shown below in Fig 6.

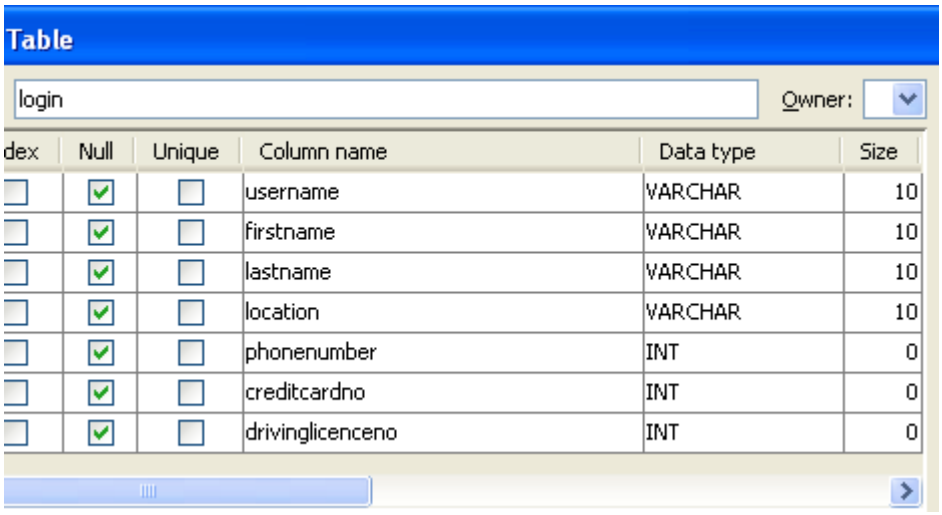


Fig. 6

Step 7:

- Click on the Ok
- It creates the table login in the test database.

Creating the WebService program for Account creation

Step 8:

- Open the netbeans 6.1
- Creat a new web project as shown below in Fig 7.

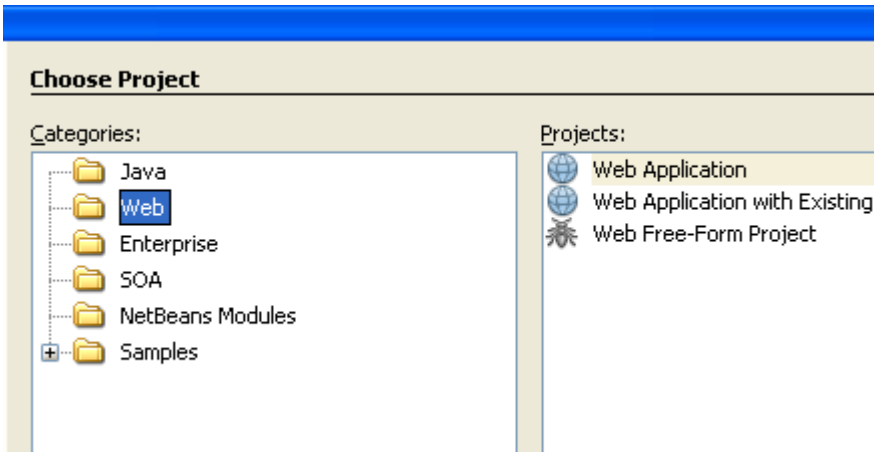


Fig. 7

Step 9:

- Type the project Name as MyAccount
- Click on the next button as shown below in Fig 8.

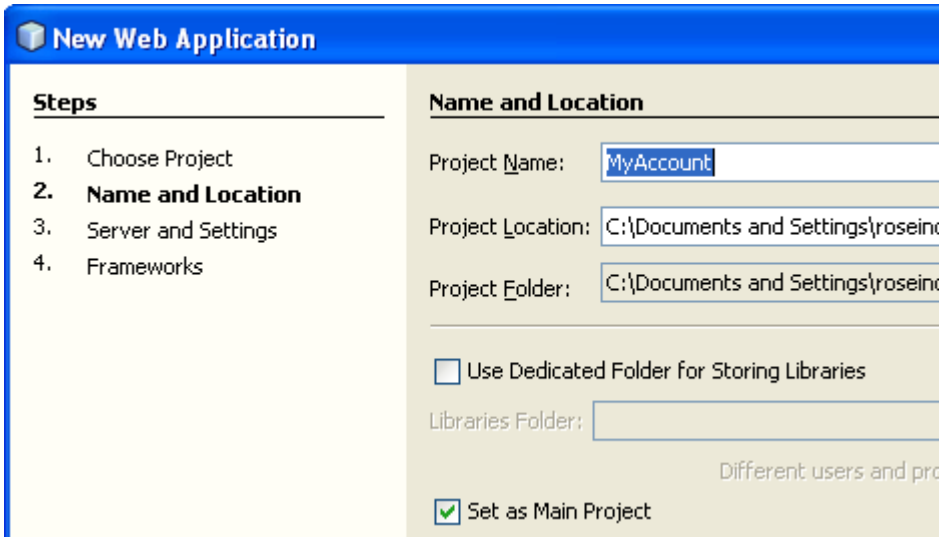


Fig. 8

Step 10:

- Select the server as Glassfish
- Click on the Next and then finish button as shown below in Fig 9.

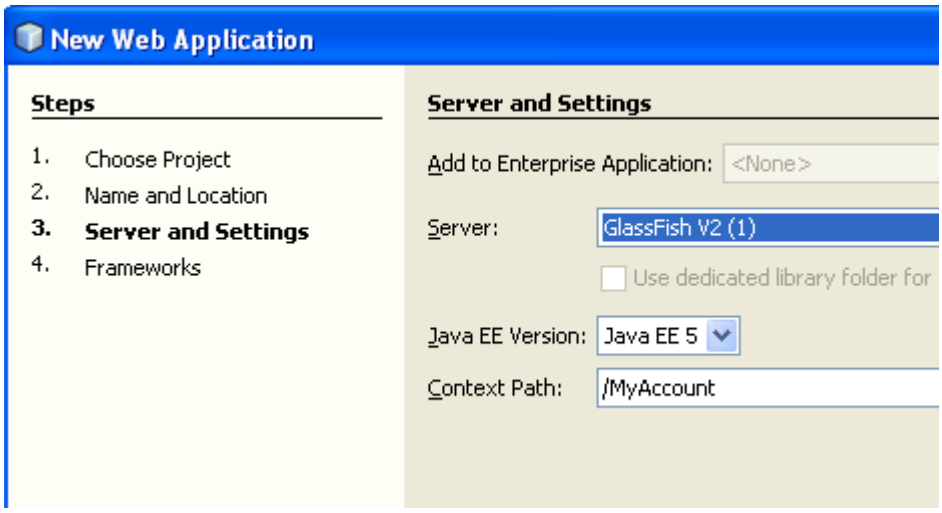


Fig. 9

Step 11:

- It creates a Web Project named MyAccount.

Creating the WebService

Step 12:

- Right Click on the project MyAccount

- Select New-->WebService as shown below in Fig 10.

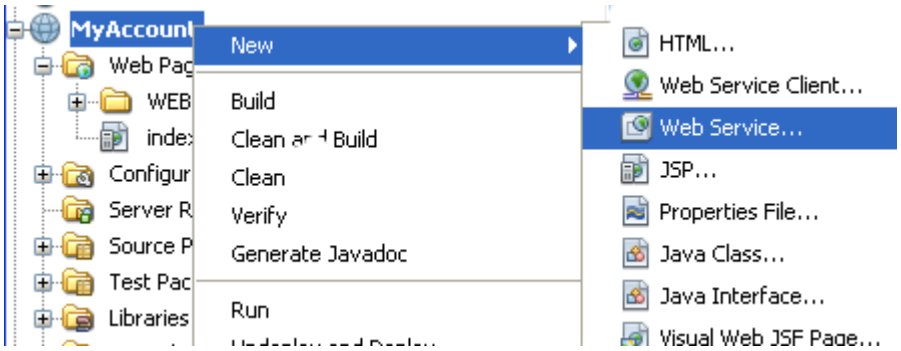


Fig. 10

Step 13:

- Type the name of the WebService as myaccount with the package as mypack.
- Click on the Finish button as shown below in Fig 11.

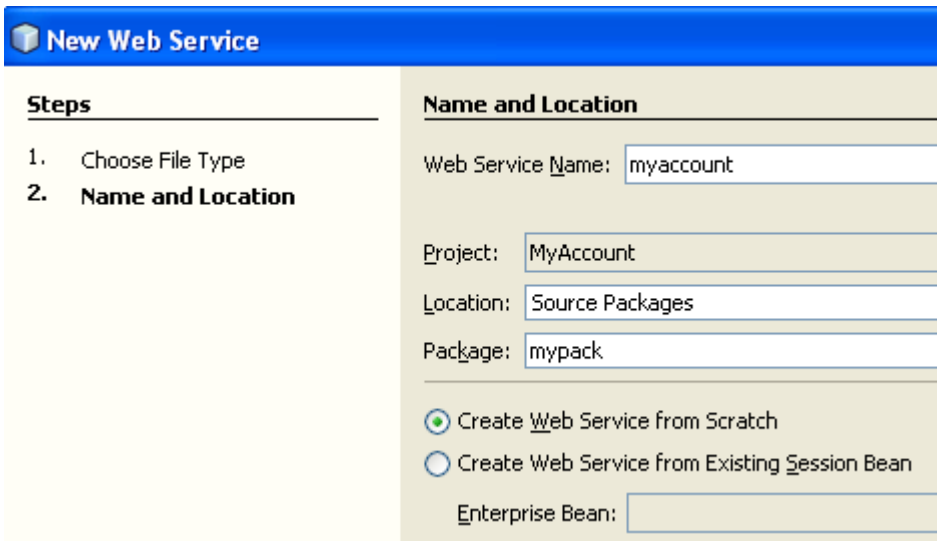


Fig. 11

Step 14:

- It creates a WebService application in design view
- Click on the Add operation as shown below in Fig 12.

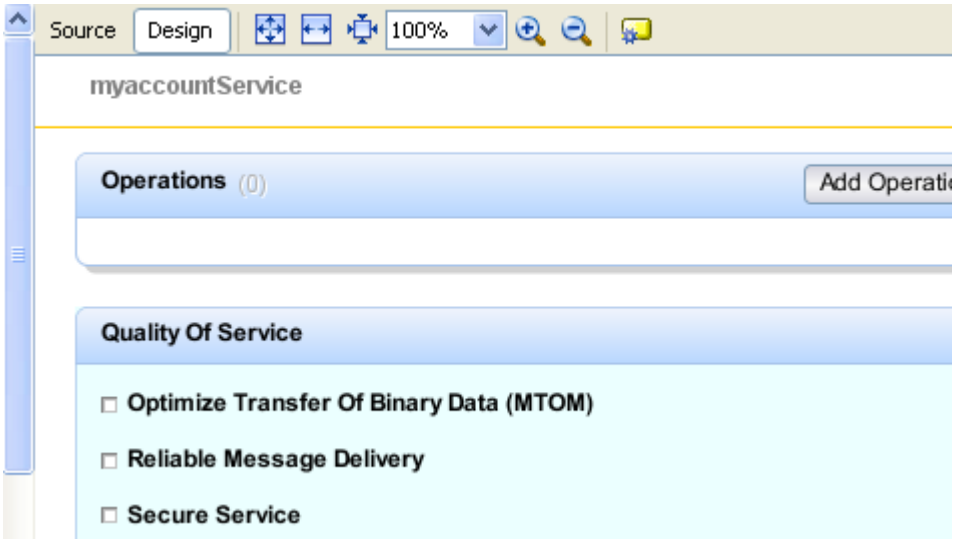


Fig. 12

Step 15:

- In the dialog box type all parameter names.
- Also select the appropriate data type.
- Click on Ok as shown below in Fig 13.

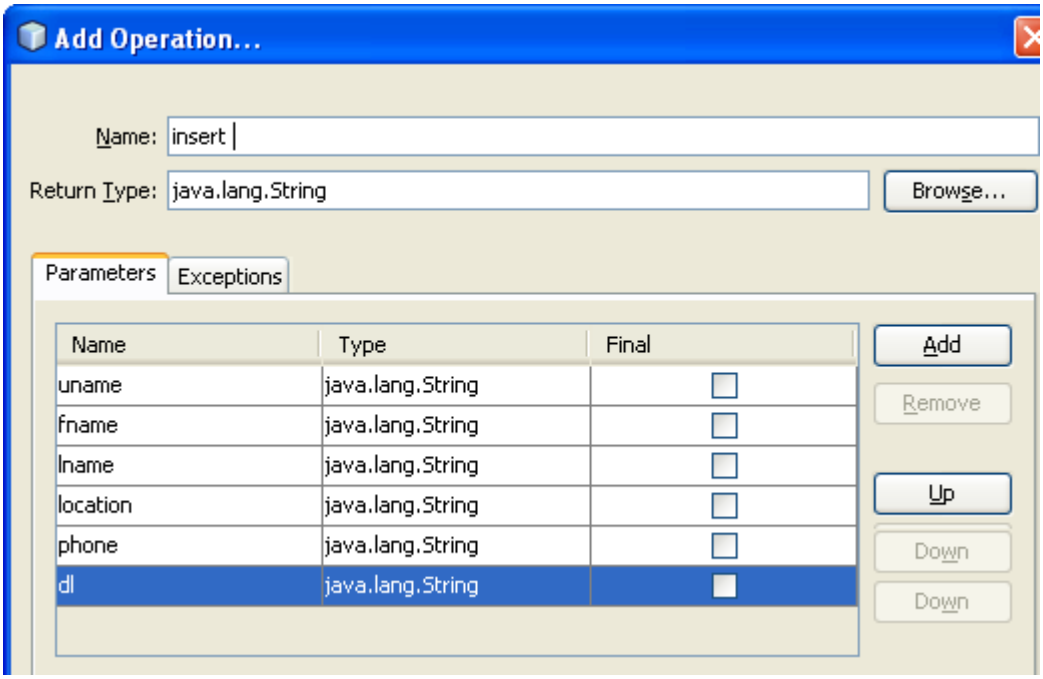


Fig. 13

Step 16:

- It creates a WebService application
- Click on the source tab as shown below in the Fig 14.

```
1   package mypack;
2
4   import javax.annotation.Resource;
4   import javax.jws.WebMethod;
5   import javax.jws.WebParam;
6   import javax.jws.WebService;
7
8   @WebService()
9   public class myaccount {
10
11      @WebMethod(operationName = "insert")
12      public String insert(@WebParam(name = "uname")
13      String uname, @WebParam(name = "fname")
14      String fname, @WebParam(name = "lname")
15      String lname, @WebParam(name = "location")
16      String location, @WebParam(name = "phone")
17      String phone, @WebParam(name = "credit")
18      String  credit, @WebParam(name = "dl")
19      String dl) {
20          //TODO write your implementation code here:
21          return null;
22      }
23  }
```

Fig. 14

Step 17:

- Now create the database source
- Right Click in the source code of myaccount.java
- Select the Enterprise Resources-->Use Database as shown below in Fig 15.

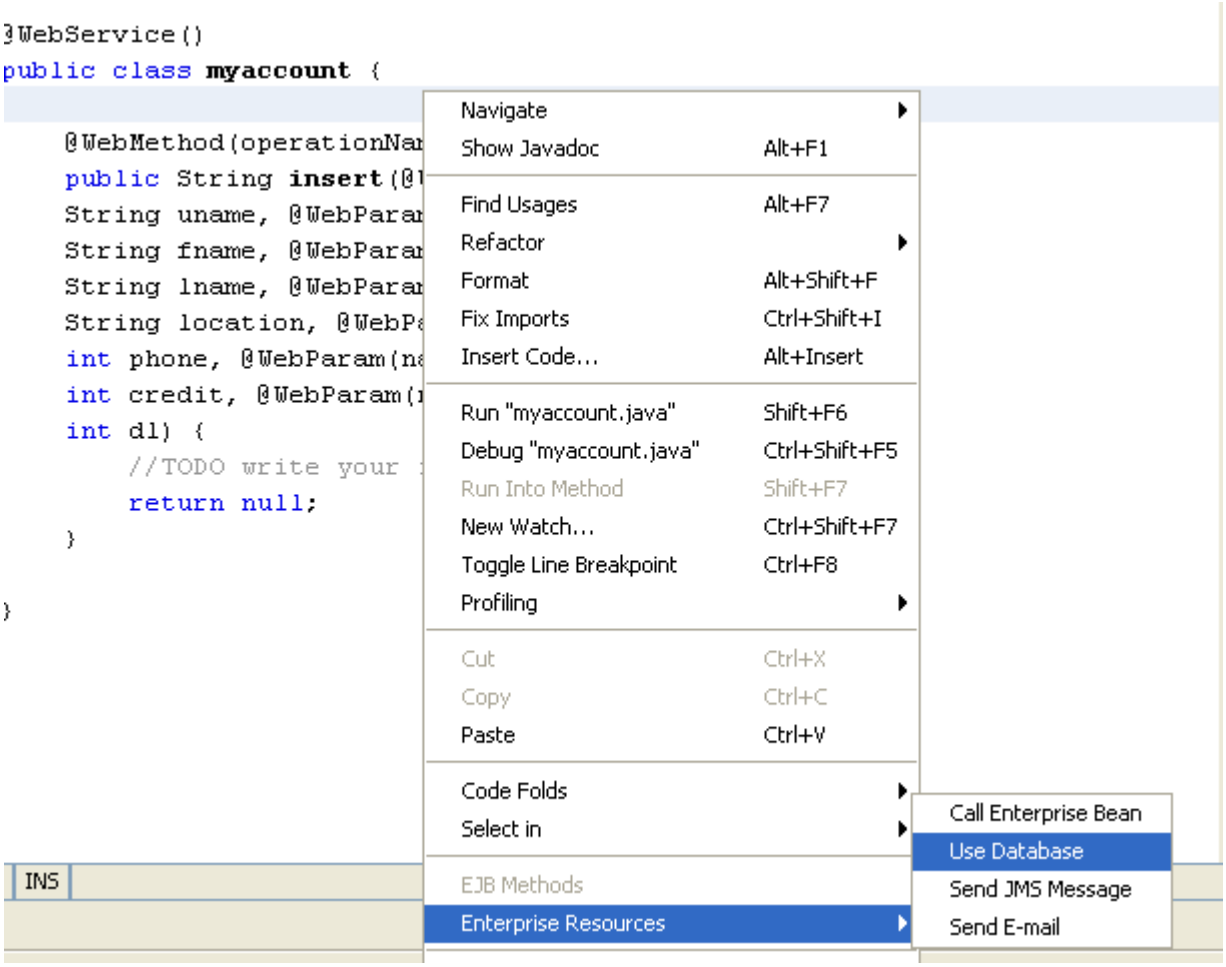


Fig. 15

Step 18:

- In the choose database select the Add button as shown below in Fig 16.

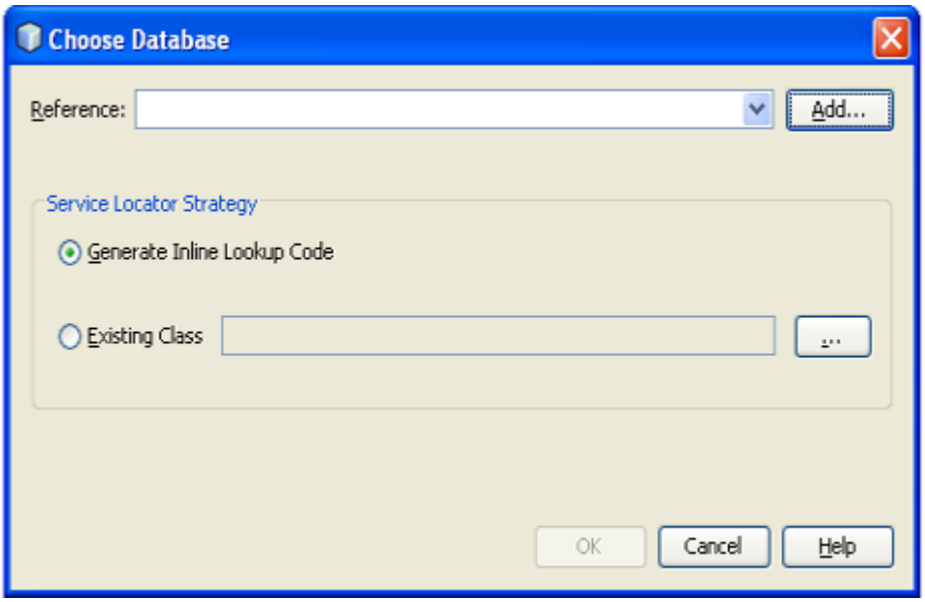


Fig. 16

Step 19:

- It opens a Add Data Source Reference.
- Type the Reference Name as data1
- For Project Data Sources Click on the Add button as shown below in Fig 17.

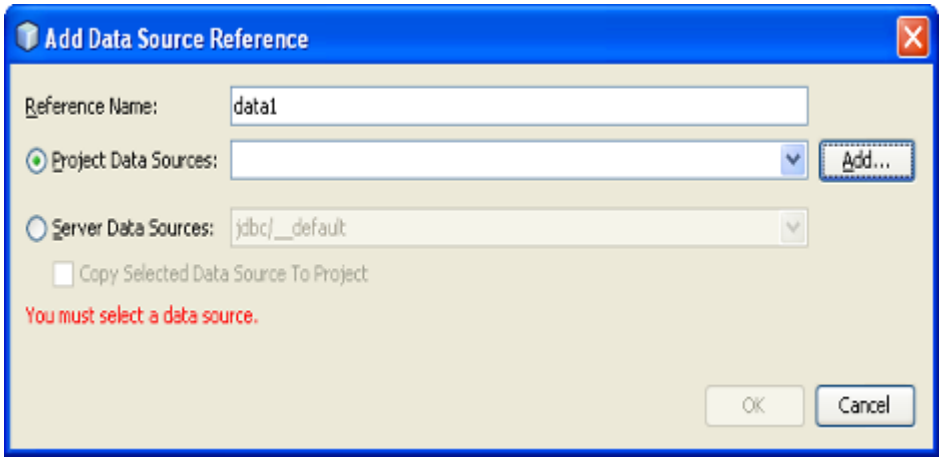


Fig. 17

Step 20:

- In the Crea Data Source type thye jndi name as jndi1
- In the database connection select the newly created database connction for the mysql. as shown below in Fig 18.

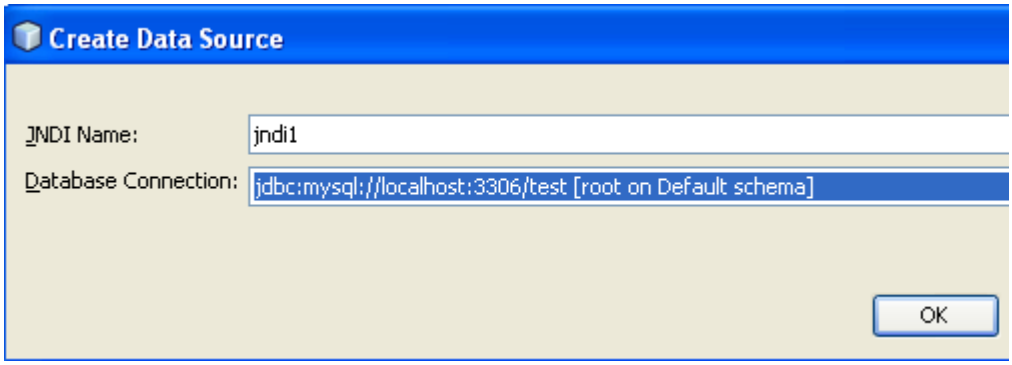


Fig. 18

Step 21:

- Click on the Ok button
- It creates the database connection gives the dialog box as shown below.
- Click on the Ok button as shown below in Fig 19.

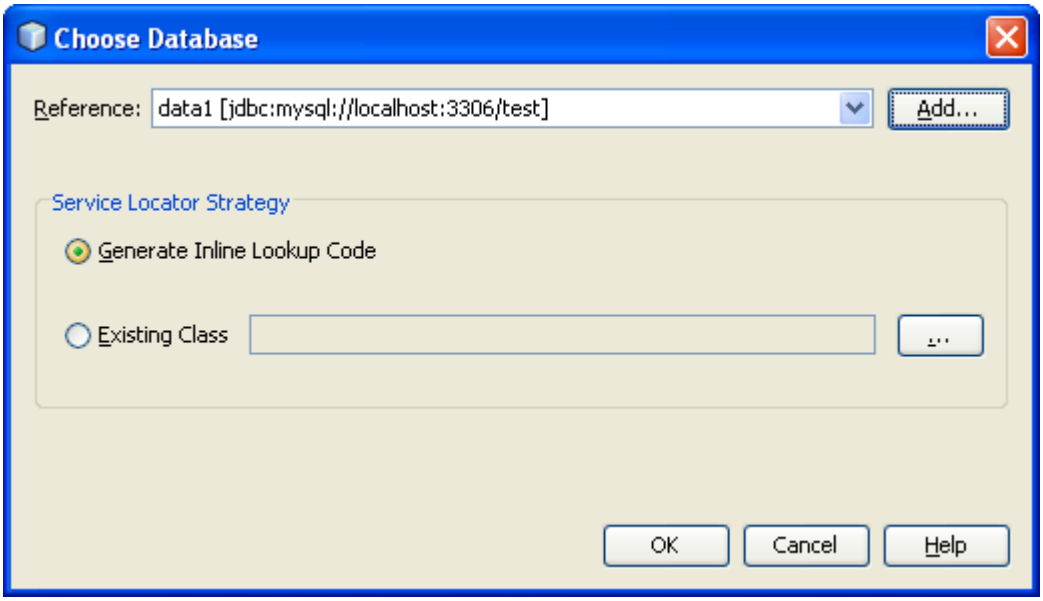


Fig. 19

Step 22:

- It creates the datasource data1 with the resource name as data1 in the code
- Edit the code and give the database connection, statement for the mysql connectivity as shown below.

```
package mypack;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import javax.annotation.Resource;
import javax.jws.WebMethod;
import javax.jws.WebParam;
import javax.jws.WebService;
import javax.sql.DataSource;

@WebService()
public class myaccount {

  @Resource(name = "data1")
  private DataSource data1;

  @WebMethod(operationName = "insert")
  public String insert(@WebParam(name = "uname") String uname,
 @WebParam(name = "fname") String fname,
@WebParam(name = "lname") String lname,
@WebParam(name = "location") String location,

@WebParam(name = "phone") String phone,
@WebParam(name = "credit") String credit,
  @WebParam(name = "dl") String dl) {
  try {
  Class.forName("com.mysql.jdbc.Driver");
  Connection con =
DriverManager.getConnection("jdbc:mysql://localhost:3306/test",
"root", "");
  PreparedStatement st =
con.prepareStatement("insert into login values(?,?,?,?,?,?,?)");
  st.setString(1, uname);
```

```
  st.setString(2, fname);
  st.setString(3, lname);
  st.setString(4, location);
  int ph = Integer.parseInt(phone);
  st.setInt(5, ph);
  int cr = Integer.parseInt(credit);
  st.setInt(6, cr);
  int d1 = Integer.parseInt(dl);
  st.setInt(7, d1);
  st.executeUpdate();
  } catch (Exception e) {
  System.out.println(e.getMessage());
  }
  return "record inserted";
  }
}
```

Step 23:

- Now deploy the project
- Right click on the project
- Select Undeploy and Deploy as shown below in Fig 20.
- It builds and deploys the MyAccount Web project on the glassfish server.

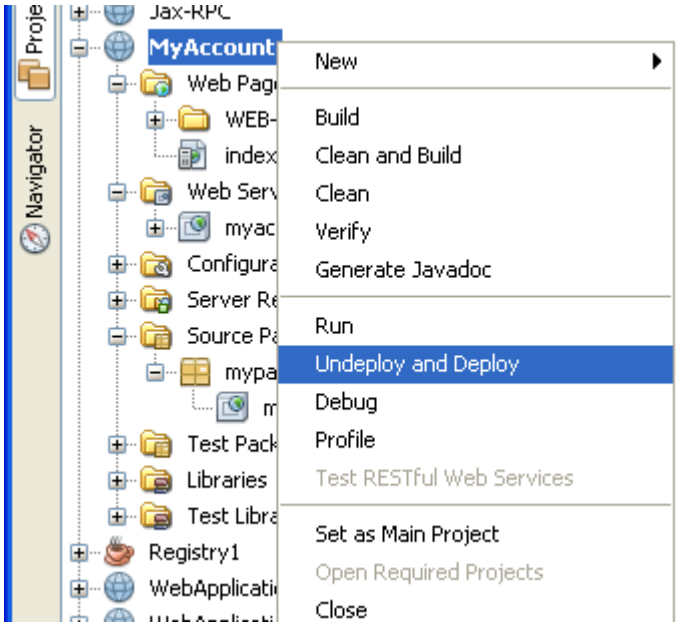


Fig. 20

Step 24:

- After deploying now we can test the Web Service created
- Right click on the myaccount webservice
- Select Test Web Service as shown below in Fig 21.

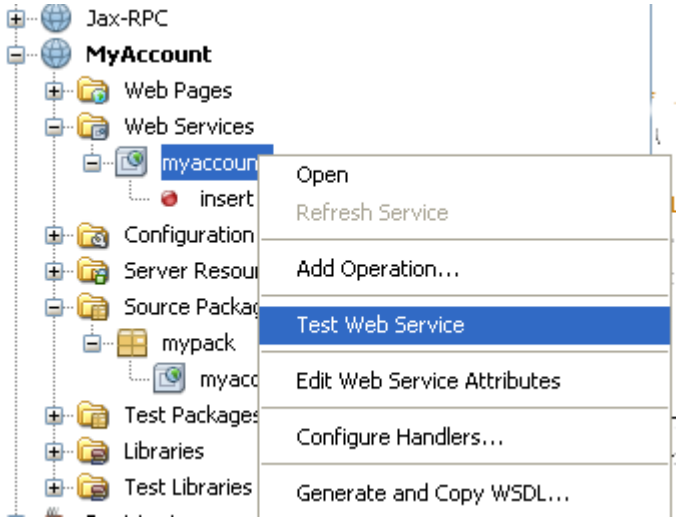


Fig. 21

Step 25:

- It runs the deployed webservice in the firefox browser.
- Type the value and click on the insert button as shown below in Fig 22.

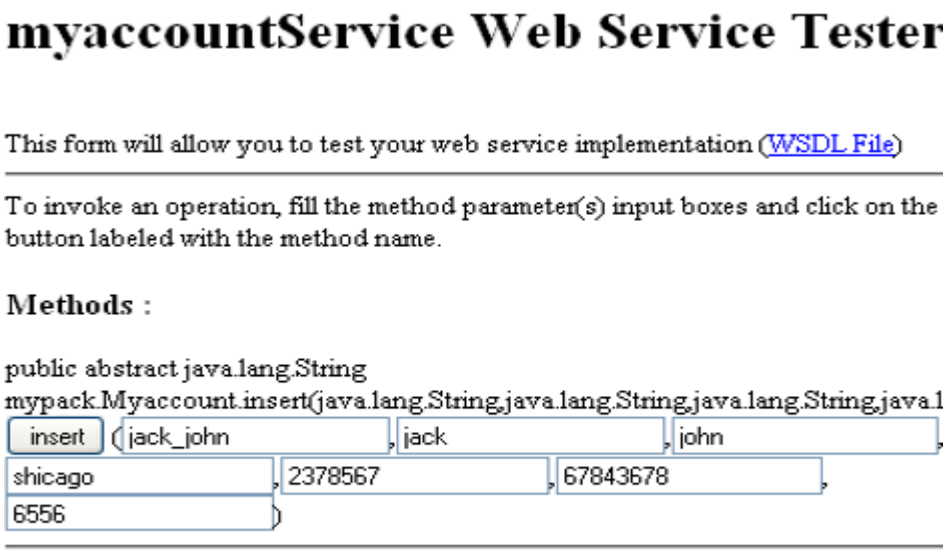


Fig. 22

Step 26:

- After clicking on the insert button it insert the values into the login table
- It open new window and displays the Metrhod parameters, SOAP request and response as shown below in Fig 23.

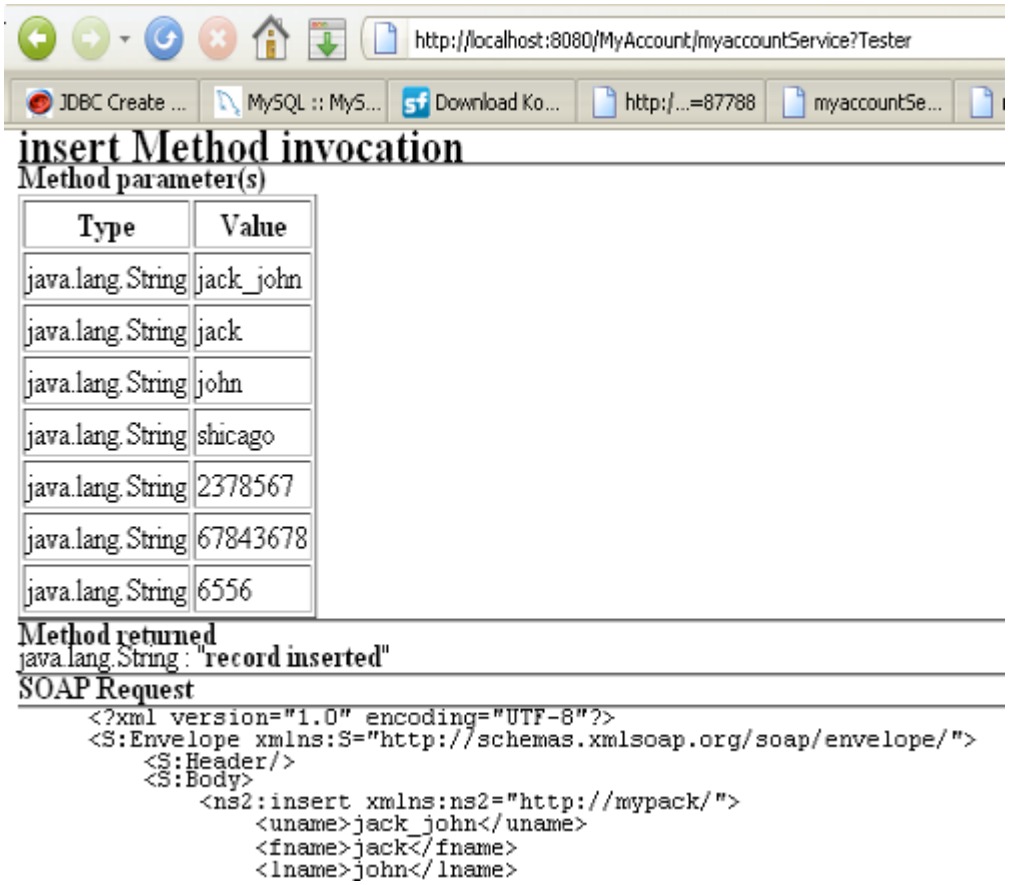


Fig. 23

Step 27:

Check the data in the database table

- In the service tab select the mysql connection
- In its table node right click on the table login
- Select the view data as shown below in Fig 24.

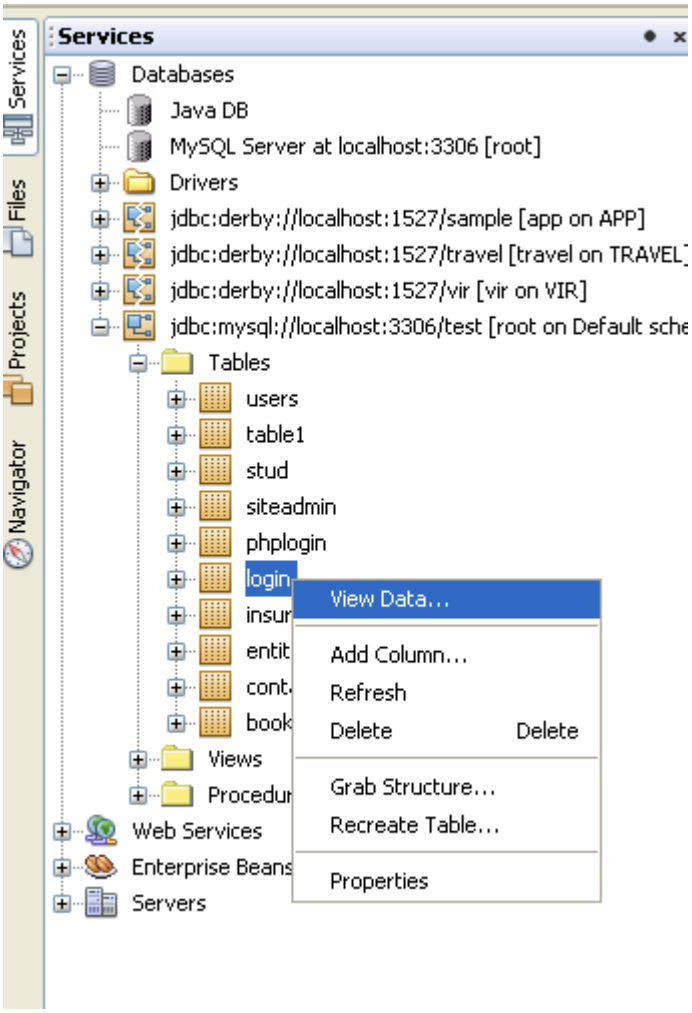


Fig. 24

Step 27:

- It opens a new window
- Then displas the data as shown below in Fig. 25.

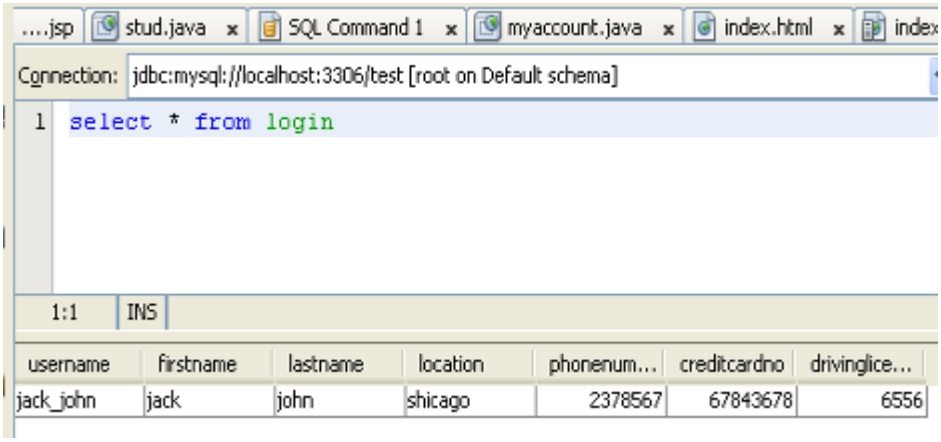


Fig. 25

Making the client of the WebService

Step 28:

- Create a new Web Project
- Type the name as MyAccountClient as shown below in Fig.26.

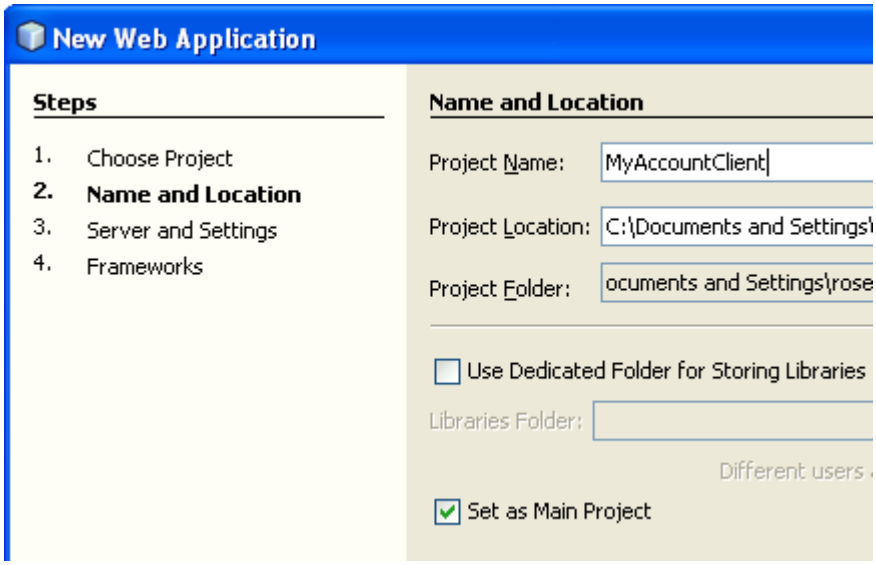


Fig. 26

Step 29:

- Click on the next button
- Select the server as glassfish as shown below in Fig. 27.

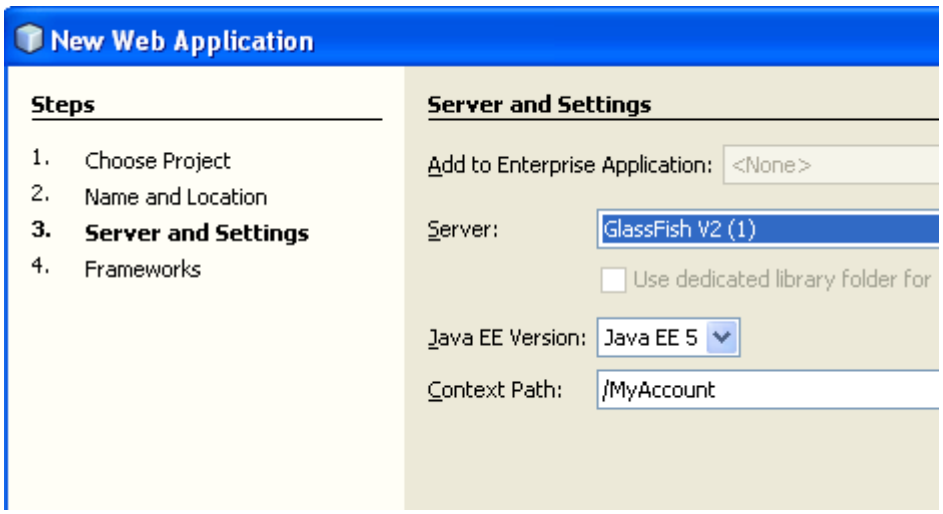


Fig. 27

Step 30:

- Click on the finish button
- It creates a web rpoject with a index.jsp file
- In the index.jsp design it with form and all its fileds
- Give the form action as Client.jsp as shown below in Fig 28.

Create WebService Client

Step 1:

- Right Click on the MyAccountClient
- Select WebService Client as shown below in Fig.28.

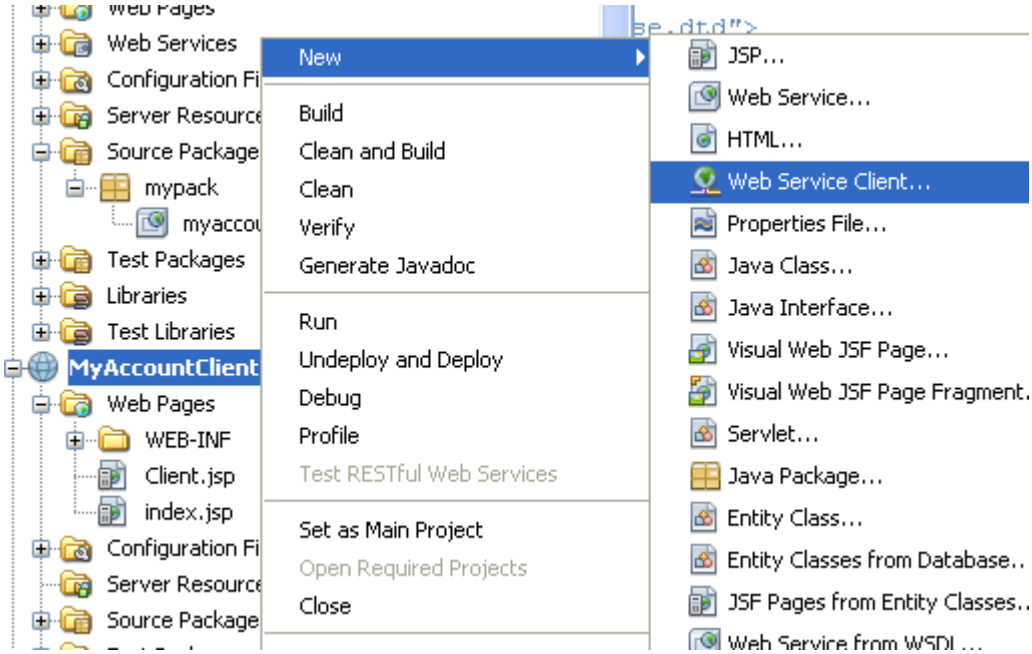


Fig. 28

Step 2:

- In the dialog box click on the browse button as shown below in Fig 29.

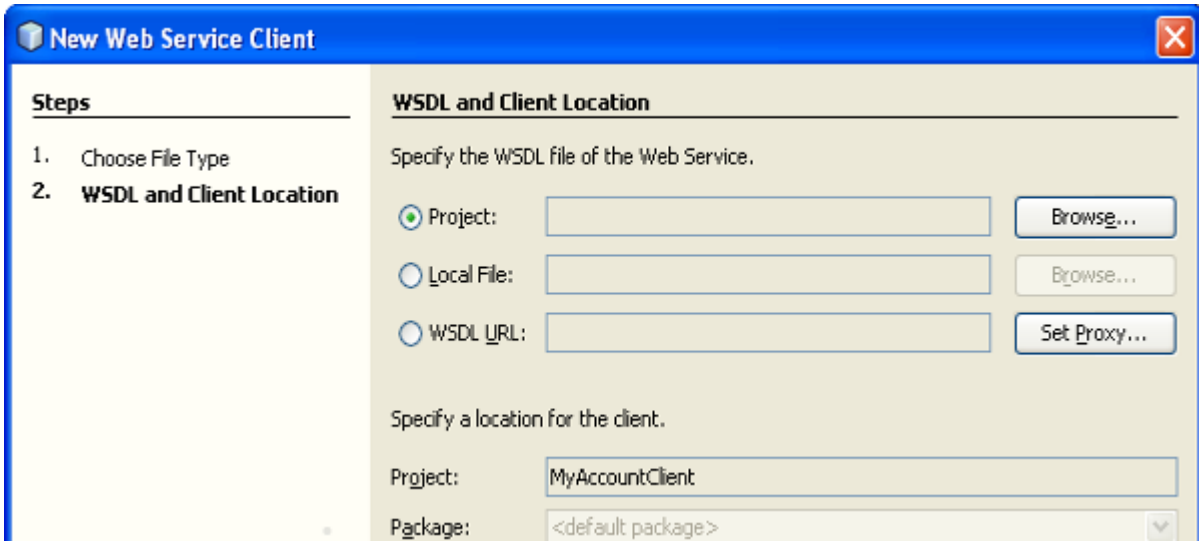


Fig. 29

Step 3:

- Select the myaccount webservice.

- Click on the OK button as shown below in Fig 30.

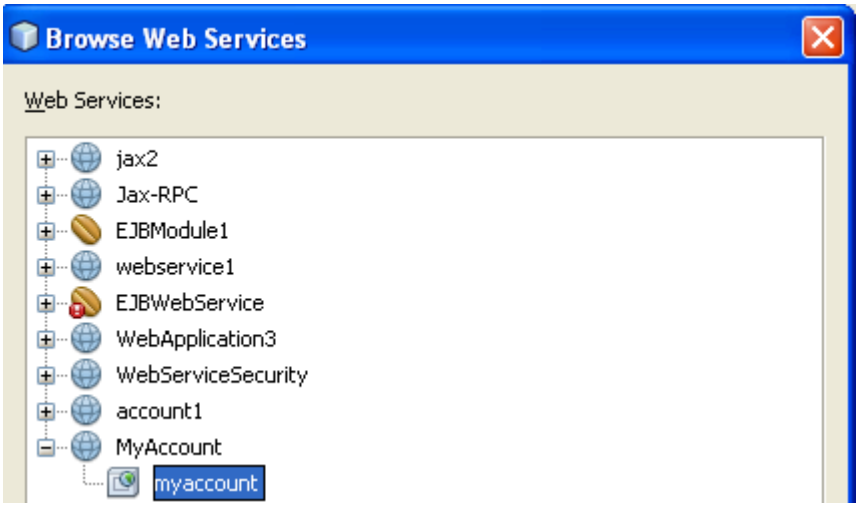


Fig. 30

Step 4:

- Now project is containing the WSDL file url.
- Click on the Ok button as shown below in Fig 31.

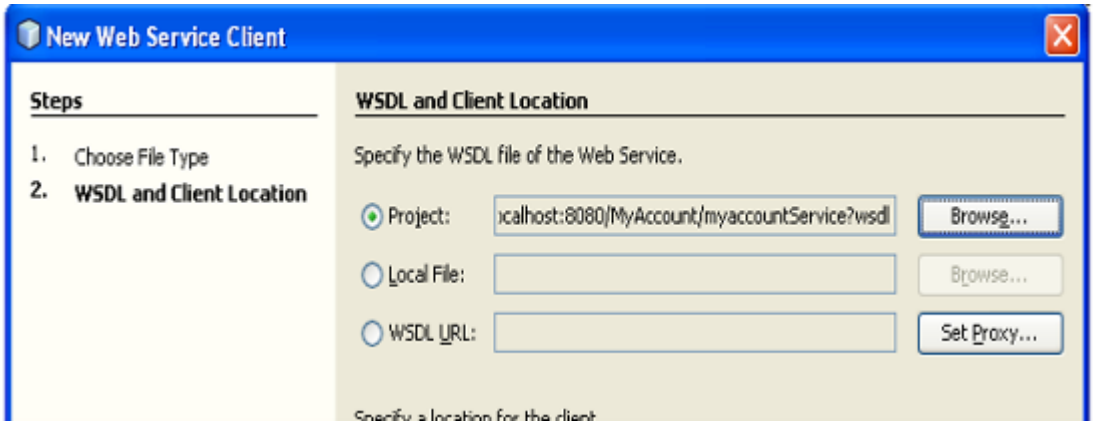


Fig. 31

Step 5:

- Right Click in the Client.jsp
- Select Web Service Client Resources-->Call WebService Operation as shown below in Fig 32.

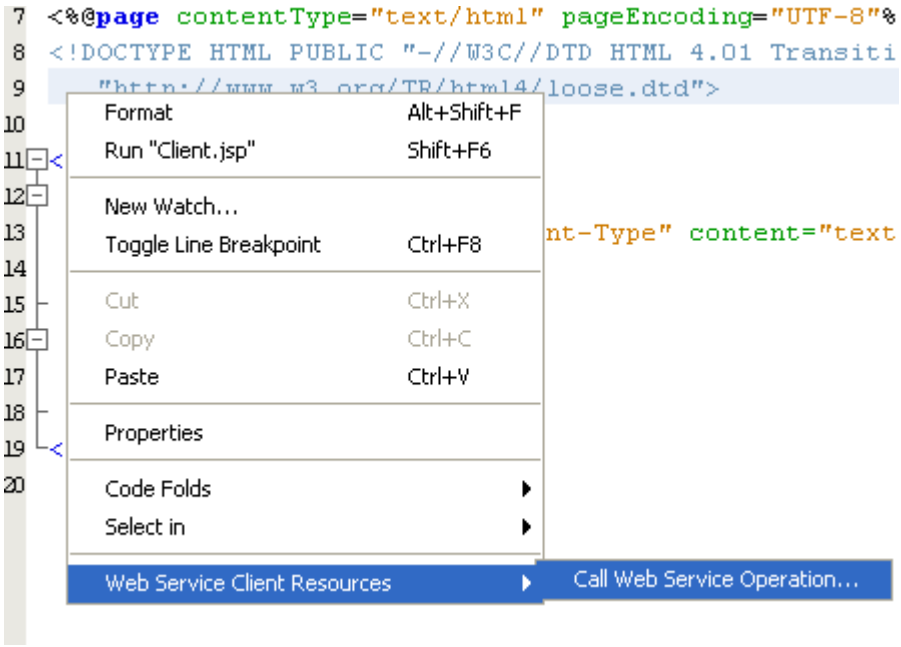


Fig. 32

Step 5:

- Select the insert operation as shown below in Fig .33.

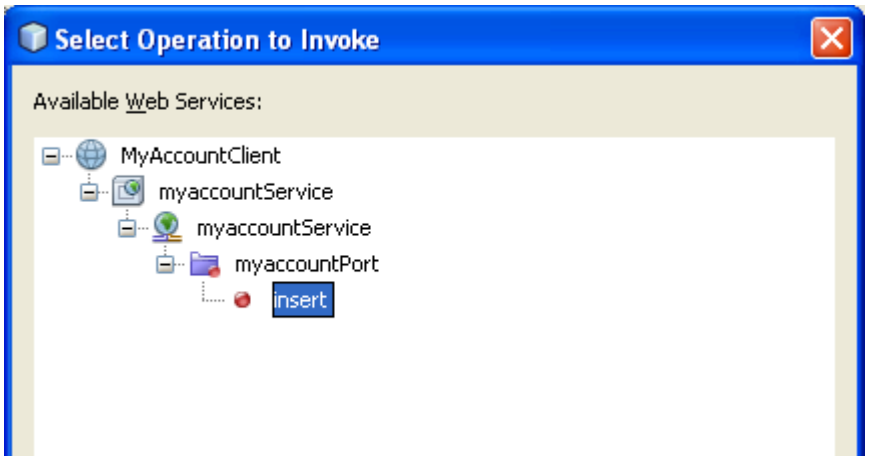


Fig. 33

Step 6:

- After insert opoeration the code becomes like as shown below in Fig 34.

```
<%
try {
    mypack.MyaccountService service = new mypack.MyaccountService();
    mypack.Myaccount port = service.getMyaccountPort();
     // TODO initialize WS operation arguments here
    java.lang.String uname = "";
    java.lang.String fname = "";
    java.lang.String lname = "";
    java.lang.String location = "";
    java.lang.String phone = "";
    java.lang.String credit = "";
    java.lang.String dl = "";
    // TODO process result here
    java.lang.String result = port.insert(uname, fname, lname, location, phone, credit
    out.println("Result = "+result);
} catch (Exception ex) {
    // TODO handle custom exceptions here
}
%>
```

Fig. 34

Step 7:

- Deploy the MyAccountClient
- Right Click and select Undeplo and Deploy as shown below in Fig.35.

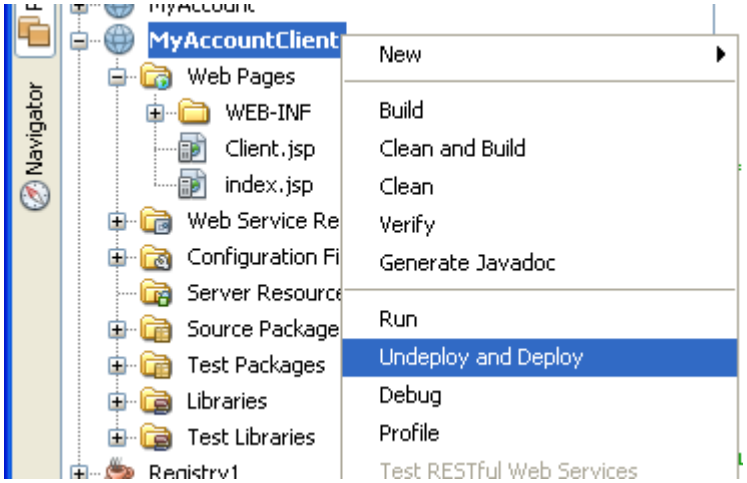


Fig. 35

Step 8:

- After deployment run the project.
- Right Click and select the run as shown below in Fig 36.

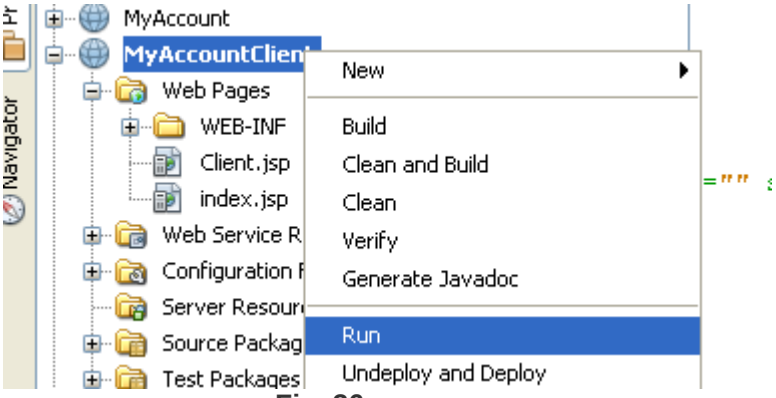


Fig. 36

Step 9:

- It runs the index.jsp in the browser as shown below in Fig 37.
- Put the data and click on the ok button

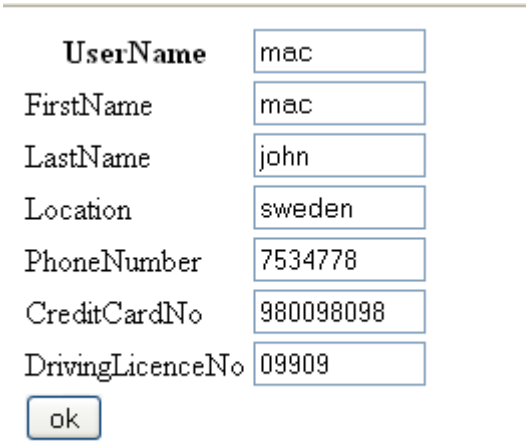


Fig. 37

Step 10:

- After this click values get inserted in the database table
- The message comes as record inserted as shown below in Fig.38

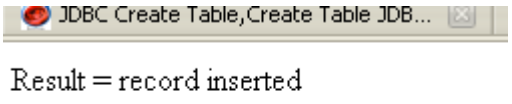


Fig. 38

Step 11:

- See the data inserted in the table.
- In the service tab select the connection created for mysql.
- In the table node select the login
- Right Click and select the view data as shown below in Fig. 39.

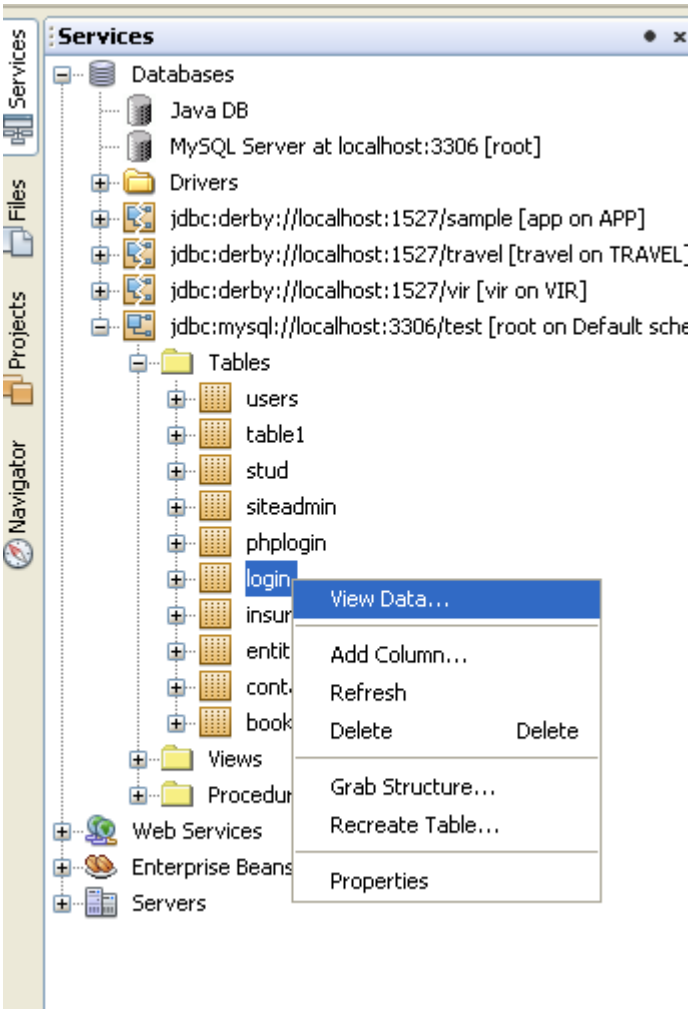


Fig. 39

Step 12:

- It displays the record inserted as shown below in Fig 40.

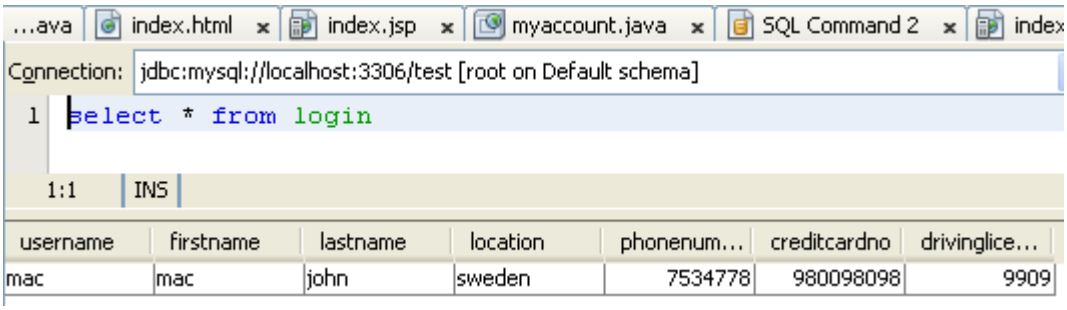


Fig. 40