**Creating a Simple Web Application Using a MySQL Database**

**https://netbeans.org/kb/docs/web/mysql-webapp.html**

Written by Troy Giunipero

This document describes how to create a simple web application that connects to a MySQL database server. It also covers some basic ideas and technologies in web development, such as [JavaServer Pages](http://java.sun.com/products/jsp) (JSP), [JavaServer Pages Standard Tag Library](http://java.sun.com/products/jsp/jstl/) (JSTL), the [Java Database Connectivity](http://java.sun.com/products/jdbc/overview.html) (JDBC) API, and two-tier, client-server architecture. This tutorial is designed for beginners who have a basic understanding of web development and are looking to apply their knowledge using a MySQL database.

[MySQL](http://www.mysql.com) is a popular open source database management system commonly used in web applications due to its speed, flexibility and reliability. MySQL employs SQL, or *Structured Query Language*, for accessing and processing data contained in databases.

This tutorial is a continuation from the [Connecting to a MySQL Database](https://netbeans.org/kb/docs/ide/mysql.html) tutorial and assumes that you have already created a MySQL database named MyNewDatabase, which you have registered a connection for in the NetBeans IDE. The table data used in that tutorial is contained in [ifpwafcad.sql](https://netbeans.org/projects/samples/downloads/download/Samples%252FJava%2520Web%252Fifpwafcad.sql) and is also required for this tutorial. This SQL file creates two tables, Subject and Counselor, then populates them with sample data. If needed, save this file to your computer, then open it in the NetBeans IDE and run it on the MySQL database named MyNewDatabase.

**Contents**



* [Planning the Structure](https://netbeans.org/kb/docs/web/mysql-webapp.html#planStructure)
* [Creating a New Project](https://netbeans.org/kb/docs/web/mysql-webapp.html#createProject)
* [±](https://netbeans.org/kb/docs/web/mysql-webapp.html) [Preparing the Web Interface](https://netbeans.org/kb/docs/web/mysql-webapp.html#prepareInterface)
* [±](https://netbeans.org/kb/docs/web/mysql-webapp.html) [Preparing Communication between the Application and Database](https://netbeans.org/kb/docs/web/mysql-webapp.html#prepareCommunication)
* [±](https://netbeans.org/kb/docs/web/mysql-webapp.html) [Adding Dynamic Logic](https://netbeans.org/kb/docs/web/mysql-webapp.html#addLogic)
* [Running the Completed Application](https://netbeans.org/kb/docs/web/mysql-webapp.html#run)
* [±](https://netbeans.org/kb/docs/web/mysql-webapp.html) [Troubleshooting](https://netbeans.org/kb/docs/web/mysql-webapp.html#troubleshoot)
* [See Also](https://netbeans.org/kb/docs/web/mysql-webapp.html#seeAlso)

**To follow this tutorial, you need the following software and resources.**

|  |  |
| --- | --- |
| **Software or Resource** | **Version Required** |
| [NetBeans IDE](https://netbeans.org/downloads/index.html) | 7.0, 7.1, 7.2, 7.3, Java EE bundle |
| [Java Development Kit (JDK)](http://www.oracle.com/technetwork/java/javase/downloads/index.html) | version 6 or 7 |
| [MySQL database server](http://dev.mysql.com/downloads/mysql/) | 5.x |
| MySQL Connector/J JDBC Driver | version 5.x |
| GlassFish Server Open Source Edition | 3.x |

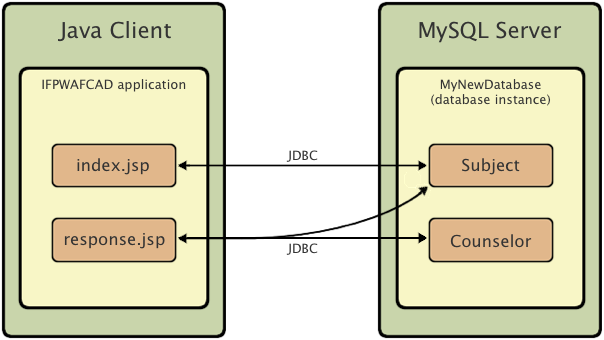
**Notes:**

* The Java download bundle of the NetBeans IDE enables you to install the GlassFish server. You require the GlassFish server to work through this tutorial.
* The [MySQL Connector/J JDBC Driver](http://dev.mysql.com/downloads/connector/j/), necessary for communication between Java platforms and the MySQL database protocol, is included in the NetBeans IDE.
* If you need to compare your project with a working solution, you can [download the sample application](https://netbeans.org/projects/samples/downloads/download/Samples%252FJava%2520Web%252FIFPWAFCAD.zip).

**Planning the Structure**

Simple web applications can be designed using a *two-tier architecture*, in which a client communicates directly with a server. In this tutorial, a Java web application communicates directly with a MySQL database using the Java Database Connectivity API. Essentially, it is the [MySQL Connector/J JDBC Driver](http://dev.mysql.com/downloads/connector/j/) that enables communication between the Java code understood by the application server (the GlassFish server), and any content in SQL, the language understood by the database server (MySQL).

The application you build in this tutorial involves the creation of two JSP pages. In each of these pages you use HTML and CSS to implement a simple interface, and apply JSTL technology to perform the logic that directly queries the database and inserts the retrieved data into the two pages. The two database tables, Subject and Counselor, are contained in the MySQL database, MyNewDatabase, which you create by completing the [Connecting to a MySQL Database](https://netbeans.org/kb/docs/ide/mysql.html) tutorial. Consider the following two-tier scenario.



The welcome page (index.jsp) presents the user with a simple HTML form. When a browser requests index.jsp, the JSTL code within the page initiates a query on MyNewDatabase. It retrieves data from the Subject database table, and inserts it into to the page before it is sent to the browser. When the user submits his or her selection in the welcome page's HTML form, the submit initiates a request for the response page (response.jsp). Again, the JSTL code within the page initiates a query on MyNewDatabase. This time, it retrieves data from both the Subject and Counselor tables and inserts it into to the page, allowing the user to view data based upon his or her selection when the page is returned to the browser.

In order to implement the scenario described above, you develop a simple application for a fictitious organization named IFPWAFCAD, The International Former Professional Wrestlers' Association for Counseling and Development.

**index.jsp**

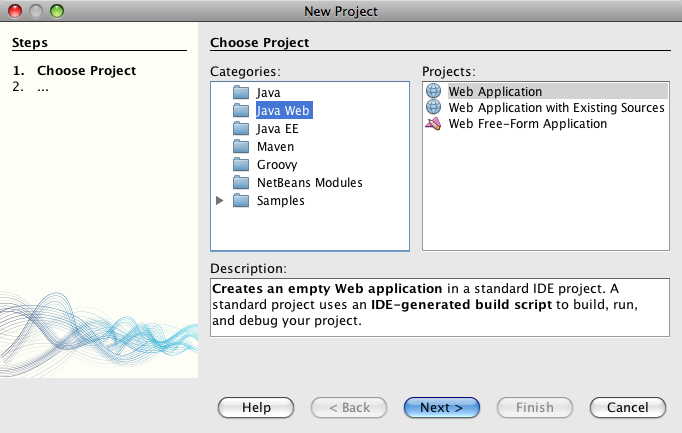


**response.jsp**



**Creating a New Project**

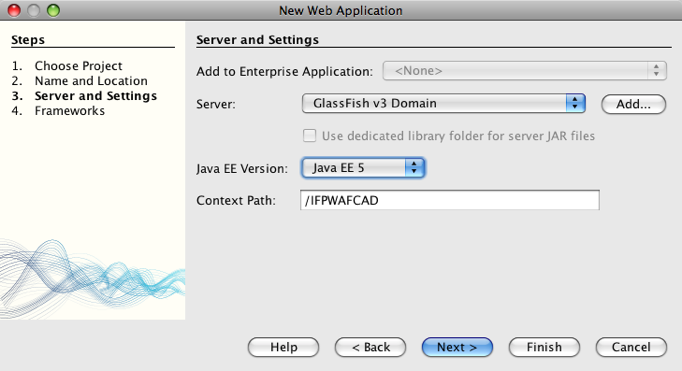
Begin by creating a new Java web project in the IDE:

1. Choose File > New Project (Ctrl-Shift-N; ⌘-Shift-N on Mac) from the main menu. Select the Java Web category, then select Web Application. Click Next.   
   

The New Project wizard allows you to create an empty web application in a standard IDE project. The standard project uses an IDE-generated Ant build script to compile, deploy, and run the application.

1. In Project Name, enter **IFPWAFCAD**. Also, specify the location for the project on your computer. (By default, the IDE places projects in a NetBeansProjects folder located in your home directory.) Click Next.
2. In the Server and Settings panel, specify the GlassFish server as server which will be used to run the application.

**Note.** The GlassFish server displays in the Server drop-down field if you installed the Java version of the [NetBeans IDE](https://netbeans.org/downloads/index.html). Because the GlassFish server is included in the download, it is automatically registered with the IDE. If you want to use a different server for this project, click the Add button located next to the Server drop-down field, and register a different server with the IDE. However, working with servers other than the GlassFish server is beyond the scope of this tutorial.

1. In the Java EE Version field, select **Java EE 5**.   
   

Java EE 6 web projects do not require the use of the web.xml deployment descriptor, and the NetBeans project template does not include the web.xml file in Java EE 6 projects. However, this tutorial demonstrates how to declare a data source in the deployment descriptor, and it does not rely on any features specific to Java EE 6, so you can set the project version to Java EE 5.

**Note.** You could equally set the project version to Java EE 6, then create a web.xml deployment descriptor. (From the New File wizard, select the Web category, then Standard Deployment Descriptor.)

1. Click Finish. The IDE creates a project template for the entire application, and opens an empty JSP page (index.jsp) in the editor. The index.jsp file serves as the welcome page for the application. The new project is structured according to Sun [Java BluePrints](http://java.sun.com/blueprints/code/projectconventions.html) guidelines.

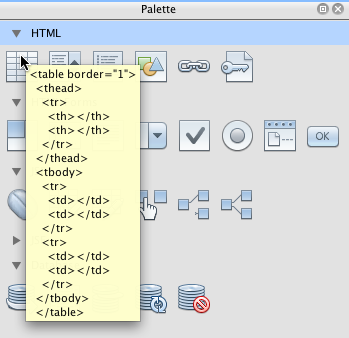
**Preparing the Web Interface**

Begin by preparing the welcome (index.jsp) and response (response.jsp) pages. The welcome page implements an HTML form that is used to capture user data. Both pages implement an HTML table to display data in a structured fashion. In this section, you also create a stylesheet that enhances the appearance of both pages.

* [Setting up the welcome page](https://netbeans.org/kb/docs/web/mysql-webapp.html#welcomePage)
* [Creating the response page](https://netbeans.org/kb/docs/web/mysql-webapp.html#responsePage)
* [Creating a stylesheet](https://netbeans.org/kb/docs/web/mysql-webapp.html#stylesheet)

**Setting up the welcome page**

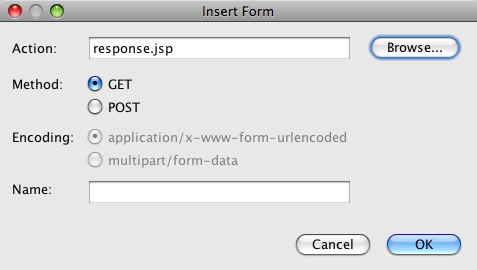
Make sure index.jsp is opened in the editor. If it is not already open, double-click index.jsp from IFPWAFCAD > Web Pages in the Projects window.

1. In the editor, change the text between the <title> tags to: IFPWAFCAD Homepage.
2. Change the text between the <h1> tags to: 'Welcome to IFPWAFCAD, the International Former Professional Wrestlers' Association for Counseling and Development!.
3. Open the IDE's Palette by choosing Window > Palette (Ctrl-Shift-8; ⌘-Shift-8 on Mac) from the main menu. Hover your pointer over the Table icon from the HTML category and note that the default code snippet for the item displays.   
     
   You can configure the Palette to your liking - right-click in the Palette and choose Show Big Icons and Hide Item Names to have it display as in the image above.
4. Place your cursor at a point just after the <h1> tags. (This is where you want to implement the new HTML table.) Then, in the Palette, double-click the Table icon.
5. In the Insert Table dialog that displays, specify the following values then click OK:
   * **Rows**: 2
   * **Columns**: 1
   * **Border Size**: 0

The HTML table code is generated and added to your page.

1. Add the following content to the table heading and the cell of the first table row (new content shown in **bold**):
2. <table border="0">
3. <thead>
4. <tr>
5. <th>**IFPWAFCAD offers expert counseling in a wide range of fields.**</th>
6. </tr>
7. </thead>
8. <tbody>
9. <tr>
10. <td>**To view the contact details of an IFPWAFCAD certified former**
11. **professional wrestler in your area, select a subject below:**</td>

</tr>

1. For the bottom row of the table, insert an HTML form. To do so, place your cursor between the second pair of <td> tags, then double-click the HTML form ( HTML form icon) icon in the Palette. In the Insert Form dialog, type in response.jsp in the Action text field, then click OK.   
   
2. Type in the following content between the <form> tags (new content shown in **bold**):
3. <tr>
4. <td>
5. <form action="response.jsp">
6. **<strong>Select a subject:</strong>**
7. </form>
8. </td>

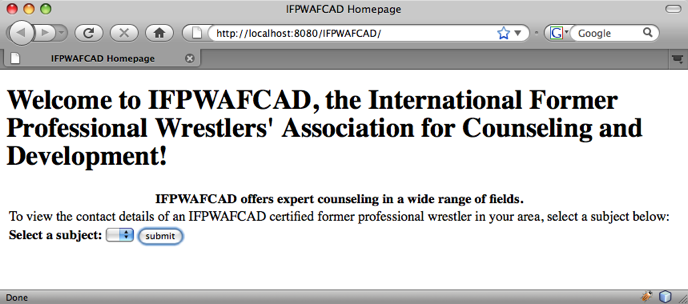
</tr>

1. Press Enter to add an empty line after the content you just added and then double-click Drop-down List in the Palette to open the Insert Drop-down dialog box.
2. Type subject\_id for the Name text field in the Insert Drop-down dialog and click OK. Note that the code snippet for the drop-down list is added to the form.

The number of options for the drop-down is currently not important. Later in the tutorial you will add JSTL tags that dynamically generate options based on the data gathered from the Subject database table.

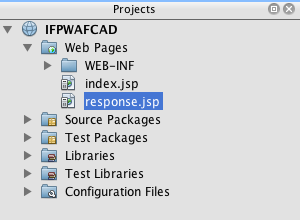
1. Add a submit button item ( Submit button icon) to a point just after the drop-down list you just added. You can either use the Palette to do this, or invoke the editor's code completion as illustrated in the previous step. In the Insert Button dialog, enter submit for both the Label and Name text fields, then click OK.
2. To format your code, right-click in the editor and choose Format (Alt-Shift-F; Ctrl-Shift-F on Mac). Your code is automatically formatted, and should now look similar to the following:
3. <body>
4. <h2>Welcome to <strong>IFPWAFCAD</strong>, the International Former
5. Professional Wrestlers' Association for Counseling and Development!
6. </h2>
7. <table border="0">
8. <thead>
9. <tr>
10. <th>IFPWAFCAD offers expert counseling in a wide range of fields.</th>
11. </tr>
12. </thead>
13. <tbody>
14. <tr>
15. <td>To view the contact details of an IFPWAFCAD certified former
16. professional wrestler in your area, select a subject below:</td>
17. </tr>
18. <tr>
19. <td>
20. <form action="response.jsp">
21. <strong>Select a subject:</strong>
22. <select name="subject\_id">
23. <option></option>
24. </select>
25. <input type="submit" value="submit" name="submit" />
26. </form>
27. </td>
28. </tr>
29. </tbody>
30. </table>

</body>

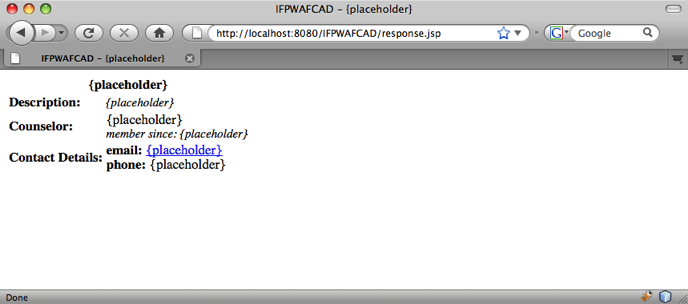
To view this page in a browser, right-click in the editor and choose Run File (Shift-F6; Fn-Shift-F6 on Mac). When you do this, the JSP page is automatically compiled and deployed to your server. The IDE opens your default browser to display the page from its deployed location.   


**Creating the response page**

In order to prepare the interface for response.jsp you must first create the file in your project. Note that most of the content that displays in this page is generated dynamically using JSP technology. Therefore, in the following steps you add *placeholders* which you will later substitute for the JSP code.

1. Right-click the IFPWAFCAD project node in the Projects window and choose New > JSP. The New JSP File dialog opens.
2. In the JSP File Name field, enter response. Note that Web Pages is currently selected for the Location field, meaning that the file will be created in the project's web directory. This is the same location as where the index.jsp welcome page resides.
3. Accept any other default settings and click Finish. A template for the new response.jsp page is generated and opens in the editor. A new JSP node also displays under Web Pages in the Projects window.   
   
4. In the editor, change the title to: IFPWAFCAD - {placeholder}.
5. Remove the <h1>Hello World!</h1> line between the <body> tags, then copy and paste the following HTML table into the body of the page:
6. <table border="0">
7. <thead>
8. <tr>
9. <th colspan="2">{placeholder}</th>
10. </tr>
11. </thead>
12. <tbody>
13. <tr>
14. <td><strong>Description: </strong></td>
15. <td><span style="font-size:smaller; font-style:italic;">{placeholder}</span></td>
16. </tr>
17. <tr>
18. <td><strong>Counselor: </strong></td>
19. <td>{placeholder}
20. <br>
21. <span style="font-size:smaller; font-style:italic;">
22. member since: {placeholder}</span>
23. </td>
24. </tr>
25. <tr>
26. <td><strong>Contact Details: </strong></td>
27. <td><strong>email: </strong>
28. <a href="mailto:{placeholder}">{placeholder}</a>
29. <br><strong>phone: </strong>{placeholder}
30. </td>
31. </tr>
32. </tbody>

</table>

To view this page in a browser, right-click in the editor and choose Run File (Shift-F6; Fn-Shift-F6 on Mac). The page compiles, is deployed to the GlassFish server, and opens in your default browser.   


**Creating a stylesheet**

Create a simple stylesheet that enhances the display of the web interface. This tutorial assumes that you understand how style rules function, and how they affect corresponding HTML elements found in index.jsp and response.jsp.

1. Open the New File wizard by pressing the New File ( New File button) button in the IDE's main toolbar. Select the Web category, then select Cascading Style Sheet and click Next.
2. Type style for CSS File Name and click Finish. The IDE creates an empty CSS file and places it in the same project location as index.jsp and response.jsp. Note that a node for style.css now displays within the project in the Projects window, and the file opens in the editor.
3. In the editor, add the following content to the style.css file:
4. body {
5. font-family: Verdana, Arial, sans-serif;
6. font-size: smaller;
7. padding: 50px;
8. color: #555;
9. }
10. h1 {
11. text-align: left;
12. letter-spacing: 6px;
13. font-size: 1.4em;
14. color: #be7429;
15. font-weight: normal;
16. width: 450px;
17. }
18. table {
19. width: 580px;
20. padding: 10px;
21. background-color: #c5e7e0;
22. }
23. th {
24. text-align: left;
25. border-bottom: 1px solid;
26. }
27. td {
28. padding: 10px;
29. }
30. a:link {
31. color: #be7429;
32. font-weight: normal;
33. text-decoration: none;
34. }
35. a:link:hover {
36. color: #be7429;
37. font-weight: normal;
38. text-decoration: underline;

}

1. Link the stylesheet to index.jsp and response.jsp. In both pages, add the following line between the <head> tags:

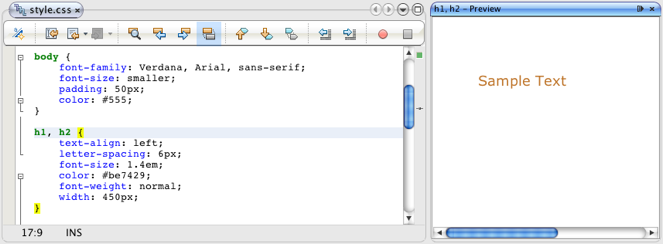
<link rel="stylesheet" type="text/css" href="style.css">

To quickly navigate between files that are open in the editor, press Ctrl-Tab, then select the file you are wanting.

**NetBeans CSS Support**

When working with CSS in the IDE, you can take advantage of the CSS Style Builder and CSS Preview. Together, these tools provide extensive support for creating style rules and viewing elements when coupled with style attributes.

For example, place your cursor within the h1 rule in style.css, then open CSS Preview (Window > Other > CSS Preview):



CSS Preview demonstrates how an element renders in a browser. Also note that the preview automatically refreshes as you make changes to a rule, providing a real-time textual representation of style elements from the IDE.

**Preparing Communication between the Application and Database**

The most efficient way to implement communication between the server and database is to set up a database *connection pool*. Creating a new connection for each client request can be very time-consuming, especially for applications that continuously receive a large number of requests. To remedy this, numerous connections are created and maintained in a connection pool. Any incoming requests that require access to the application's data layer use an already-created connection from the pool. Likewise, when a request is completed, the connection is not closed down, but returned to the pool.

After preparing the data source and connection pool for the server, you then need to instruct the application to use the data source. This is typically done by creating an entry in the application's web.xml deployment descriptor. Finally, you need to ensure that the database driver (MySQL Connector/J JDBC Driver) is accessible to the server.

**Important:** From this point forward, you need you ensure that you have a MySQL database instance named MyNewDatabase set up that contains sample data provided in [ifpwafcad.sql](https://netbeans.org/projects/samples/downloads/download/Samples%252FJava%2520Web%252Fifpwafcad.sql). This SQL file creates two tables, Subject and Counselor, then populates them with sample data. If you have not already done this, or if you need help with this task, see [Connecting to a MySQL Database](https://netbeans.org/kb/docs/ide/mysql.html) before proceeding further.

Also, your database needs to be password-protected to create a data source and work with the GlassFish server in this tutorial. If you are using the default MySQL root account with an empty password, you can set the password from a command-line prompt.   
  
This tutorial uses nbuser as an example password. To set your password to *nbuser*, navigate to your MySQL installation's bin directory in the command-line prompt and enter the following:

shell> mysql -u root

mysql> UPDATE mysql.user SET Password = PASSWORD('*nbuser*')

-> WHERE User = 'root';

mysql> FLUSH PRIVILEGES;

For more information, see the official MySQL Reference Manual: [Securing the Initial MySQL Accounts](http://dev.mysql.com/doc/refman/5.1/en/default-privileges.html).

1. [Setting up a JDBC data source and connection pool](https://netbeans.org/kb/docs/web/mysql-webapp.html#setUpJDBC)
2. [Referencing the data source from the application](https://netbeans.org/kb/docs/web/mysql-webapp.html#referenceDataSource)
3. [Adding the database driver's JAR file to the server](https://netbeans.org/kb/docs/web/mysql-webapp.html#addJar)

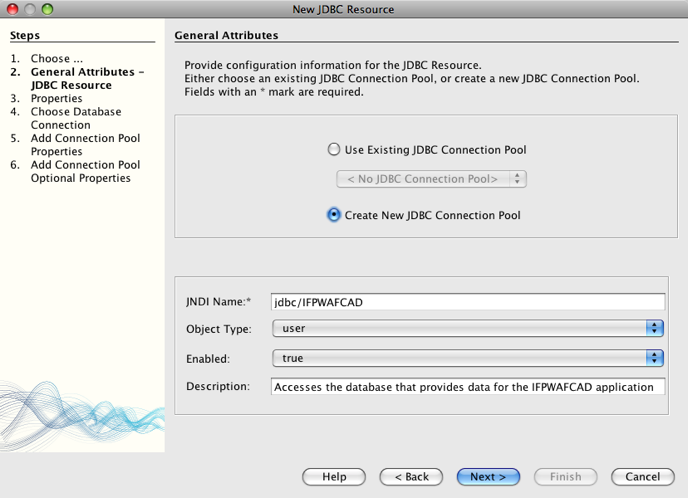
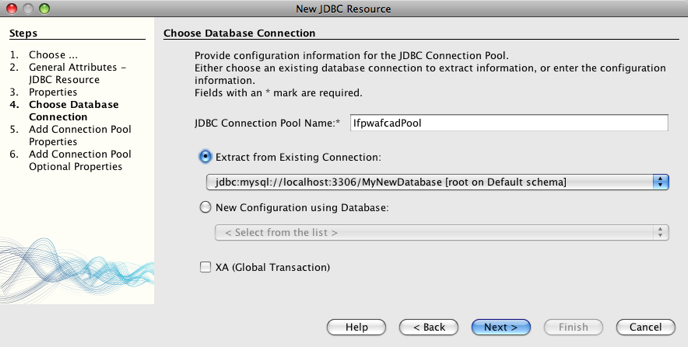
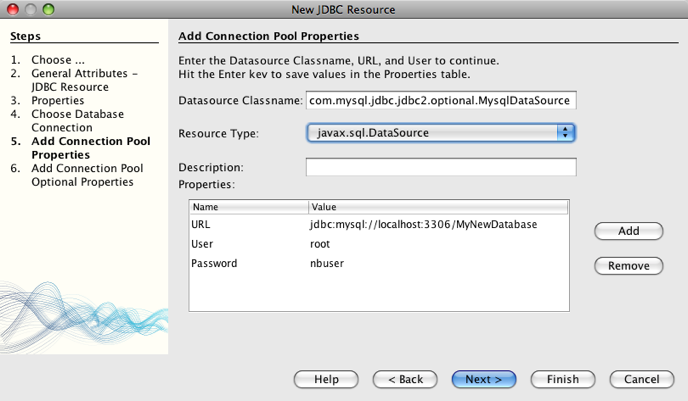
**Setting up a JDBC data source and connection pool**

The GlassFish Server Open Source Edition contains Database Connection Pooling (DBCP) libraries that provide connection pooling functionality in a way that is transparent to you as a developer. To take advantage of this, you need to configure a [JDBC](http://java.sun.com/products/jdbc/overview.html) (Java Database Connectivity) *data source* for the server which your application can use for connection pooling.

For more information on JDBC technology, see [The Java Tutorials: JDBC Basics](http://java.sun.com/docs/books/tutorial/jdbc/basics/index.html).

You could configure the data source directly within the GlassFish server Admin Console, or, as described below, you can declare the resources that your application needs in a sun-resources.xml file. When the application is deployed, the server reads in the resource declarations, and creates the necessary resources.

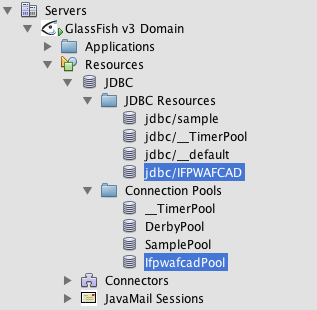
The following steps demonstrate how to declare a connection pool, and a data source that relies on the connection pool. The NetBeans JDBC Resource wizard allows you to perform both actions.

1. Open the New File wizard by pressing the New File ( New File button) button in the IDE's main toolbar. Select the GlassFish server category, then select JDBC Resource and click Next.
2. In step 2, General Attributes, choose the Create New JDBC Connection Pool option, then in the JNDI Name text field, type in **jdbc/IFPWAFCAD**.   
     
   The JDBC data source relies on [JNDI](http://java.sun.com/javase/technologies/core/jndi/index.jsp), the Java Naming and Directory Interface. The JNDI API provides a uniform way for applications to find and access data sources. For more information, see [The JNDI Tutorial](http://java.sun.com/products/jndi/tutorial/).
3. Optionally, add a description for the data source. For example, type in: Accesses the database that provides data for the IFPWAFCAD application.
4. Click Next, then click Next again to skip step 3, Additional Properties.
5. In Step 4, type in **IfpwafcadPool** for JDBC Connection Pool Name. Make sure the Extract from Existing Connection option is selected, and choose jdbc:mysql://localhost:3306/MyNewDatabase from the drop-down list.   
     
     
   **Note:** The wizard detects any database connections that have been set up in the IDE. Therefore, you need to have already created a connection to the MyNewDatabase database at this point. You can verify what connections have been created by opening the Services window (Ctrl-5; ⌘-5 on Mac) and looking for connection nodes ( database connection node icon) under the Databases category.
6. Click Next. In Step 5, note that the IDE extracts information from the database connection you specified in the previous step, and sets name-value properties for the new connection pool.   
   
7. Click Finish. The wizard generates a glassfish-resources.xml file that contains entries for the data source and connection pool you specified.

In the Projects window, you can open the newly created Server Resources > glassfish-resources.xml file and note that, within the <resources> tags, a data source and connection pool have been declared containing the values you previously specified.

**Note.** In NetBeans IDE 6.9 and earlier versions of the IDE, the name of the file generated by the IDE is sun-resources.xml.

To confirm that a new data source and connection pool are indeed registered with the GlassFish server, you can deploy the project to the server, then locate the resources in the IDE's Services window:

1. In the Projects window, right-click the IFPWAFCAD project node and choose Deploy. The server starts up if not already running, and the project is compiled and deployed to it.
2. Open the Services window (Ctrl-5; ⌘-5 on Mac) and expand the Servers > GlassFish > Resources > JDBC > JDBC Resources and Connection Pools nodes. Note that the new data source and connection pool are now displayed:   
   

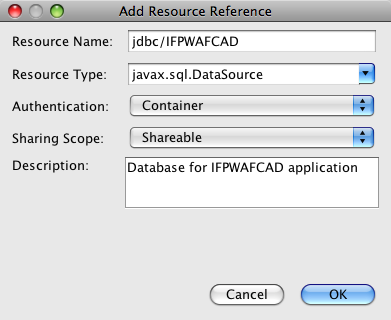
**Referencing the data source from the application**

You need to reference the JDBC resource you just configured from the web application. To do so, you can create an entry in the application's web.xml deployment descriptor.

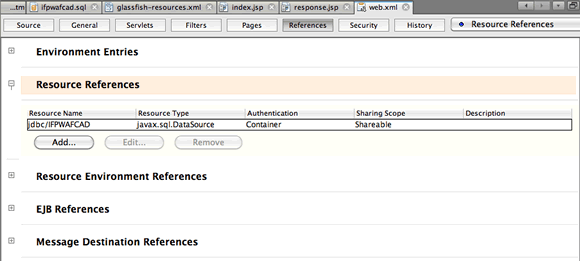
Deployment descriptors are XML-based text files that contain information describing how an application is to be deployed to a specific environment. For example, they are normally used to specify application context parameters and behavioral patterns, security settings, as well as mappings for servlets, filters and listeners.

**Note.** If you specified Java EE 6 as the Java version when you created the project, you need to create the deployment descriptor file by choosing Web > Standard Deployment Descriptor in the New File wizard.

Perform the following steps to reference the data source in the application's deployment descriptor.

1. In the Projects window, expand the Configuration Files folder and double-click web.xml to open the file in the editor.
2. Click the References tab located along the top of the editor.
3. Expand the Resource References heading and click Add to open the Add Resource Reference dialog.
4. For Resource Name, enter the resource name that you gave when configuring the data source for the server above (jdbc/IFPWAFCAD). The Description field is optional, but you can enter a human-readable description of the resource, e.g., Database for IFPWAFCAD application. Click OK.   
   

Click OK. The new resource is now listed under the Resource References heading.



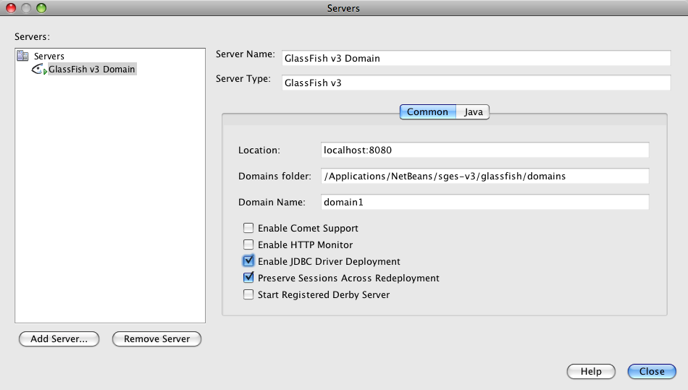
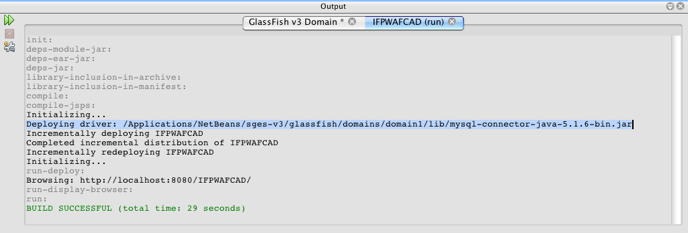
1. To verify that the resource is now added to the web.xml file, click the Source tab located along the top of the editor. Notice that the following <resource-ref> tags are now included.
2. <resource-ref>
3. <description>Database for IFPWAFCAD application</description>
4. <res-ref-name>jdbc/IFPWAFCAD</res-ref-name>
5. <res-type>javax.sql.dataSource</res-type>
6. <res-auth>Container</res-auth>
7. <res-sharing-scope>Shareable</res-sharing-scope>

</resource-ref>

**Adding the database driver's JAR file to the server**

Adding the database driver's JAR file is another step that is vital to enabling the server to communicate with your database. Ordinarily, you would need to locate your database driver's installation directory and copy the mysql-connector-java-5.1.6-bin.jar file from the driver's root directory into the library folder of the server you are using. Fortunately, the IDE's server management is able to detect at deployment whether the JAR file has been added - and if not, it does so automatically.

In order to demonstrate this, open the Servers manager (Choose Tools > Servers). The IDE provides a JDBC driver deployment option. If the option is enabled, it initiates a check to determine whether any drivers are required for the server's deployed applications. In the case of MySQL, if the driver is required and it is missing, the IDE's bundled driver is deployed to the appropriate location on the server.

1. Choose Tools > Servers to open the Servers manager. Select the GlassFish server in the left pane.
2. In the main pane, select the Enable JDBC Driver Deployment option.   
   
3. Before you close the Servers manager, make a note of the path indicated in the Domains folder text field. When you connect to the GlassFish server in the IDE, you are actually connecting to an *instance* of the application server. Each instance runs applications in a unique domain, and the Domain Name field indicates the name of the domain your server is using. As shown in the image above, the driver JAR file should be located within domain1, which is the default domain created upon installing the GlassFish server.
4. Click Close to exit the Servers manager.
5. On your computer, navigate to the GlassFish server installation directory and drill into the domains > domain1 > lib subfolder. Because you should have already deployed the IFPWAFCAD project to the server, you should see the mysql-connector-java-5.1.6-bin.jar file. If you do not see the driver JAR file, perform the following step.
6. Deploy your project to the server. In the IDE's Projects window, choose Deploy from the right-click menu of the project node. You can view progress in the IDE's Output window (Ctrl-4; ⌘-4 on Mac). The output indicates that the MySQL driver is deployed to a location in the GlassFish server.   
     
   Now, if you return to the domain1/lib subfolder on your computer, you can see that the mysql-connector-java-5.1.6-bin.jar file has been automatically added.

**Adding Dynamic Logic**

Returning to the index.jsp and response.jsp placeholders that you created earlier in the tutorial, you can now implement the JSTL code that enables pages to generate content *dynamically*, i.e., based on user input. To do so, perform the following three tasks.

1. [Add the JSTL library to the project's classpath](https://netbeans.org/kb/docs/web/mysql-webapp.html#addJSTL)
2. [Implement JSTL code](https://netbeans.org/kb/docs/web/mysql-webapp.html#implementCode)

**Adding the JSTL library to the project's classpath**

You can apply the [JavaServer Pages Standard Tag Library](http://java.sun.com/products/jsp/jstl/) (JSTL) to access and display data taken from the database. The GlassFish server includes the JSTL library by default. You can verify this by expanding the GlassFish Server 3.1.2 node under the Libraries node in the Projects window, and searching for the javax.servlet.jsp.jstl.jar library. (Older versions of the GlassFish server use the jstl-impl.jar library.) Because the GlassFish server libraries are by default added to your project's classpath, you do not have to perform any steps for this task.

JSTL provides the following four basic areas of functionality.

* core: common, structural tasks such as iterators and conditionals for handling flow control
* fmt: internationalization and localization message formatting
* sql: simple database access
* xml: handling of XML content

This tutorial focuses on usage of the core and sql tag libraries.

For more information on JSTL, see [The Java EE 5 Tutorial, Chapter 7: JavaServer Pages Standard Tag Library](http://java.sun.com/javaee/5/docs/tutorial/doc/bnakc.html).

**Implementing JSTL code**

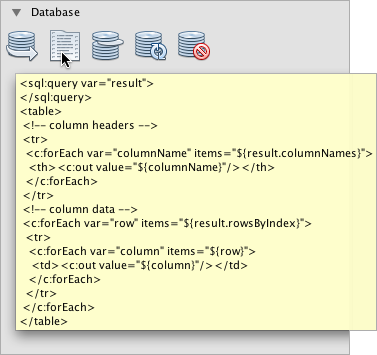
Now you can implement the code that dynamically retrieves and displays data for each page. Both pages require that you implement an SQL query that utilizes the data source created earlier in the tutorial.

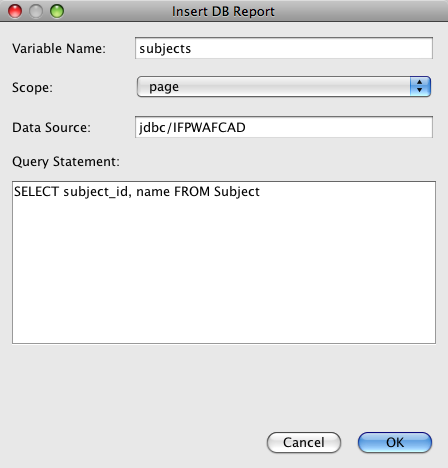
The IDE provides several database-specific JSTL snippets which you can select from the Palette (Ctrl-Shift-8; ⌘-Shift-8 on Mac).



**index.jsp**

In order to dynamically display the contents of the form in index.jsp, you need to access all names from the Subject database table.

1. Hover your mouse over the DB Report item in the Palette.   
     
   The DB Report item uses the <sql:query> tag to create an SQL query, then it uses the <c:forEach> tag to loop through the query's resultset and output the retrieved data.
2. Place your cursor above the <%@page ... %> declaration (line 7), then double-click the DB Report item in the Palette. In the dialog that displays, enter the following details:
   * **Variable Name:** subjects
   * **Scope:** page
   * **Data Source:** jdbc/IFPWAFCAD
   * **Query Statement:** SELECT subject\_id, name FROM Subject



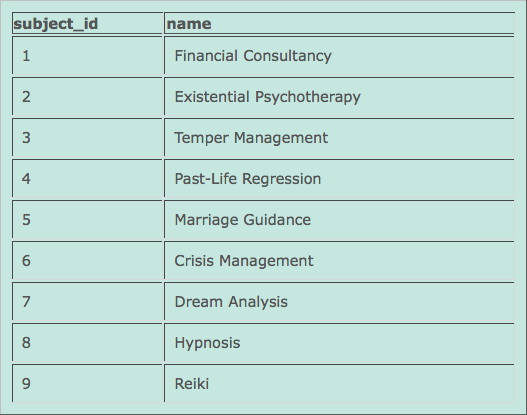
1. Click OK. The following content is generated in the index.jsp file. (New content shown in **bold**.)
2. **<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>**
3. **<%@taglib prefix="sql" uri="http://java.sun.com/jsp/jstl/sql"%>**
4. <%--
5. Document : index
6. Created on : Dec 22, 2009, 7:39:49 PM
7. Author : nbuser
8. --%>
9. **<sql:query var="subjects" dataSource="jdbc/IFPWAFCAD">**
10. **SELECT subject\_id, name FROM Subject**
11. **</sql:query>**
12. **<table border="1">**
13. **<!-- column headers -->**
14. **<tr>**
15. **<c:forEach var="columnName" items="${subjects.columnNames}">**
16. **<th><c:out value="${columnName}"/></th>**
17. **</c:forEach>**
18. **</tr>**
19. **<!-- column data -->**
20. **<c:forEach var="row" items="${subjects.rowsByIndex}">**
21. **<tr>**
22. **<c:forEach var="column" items="${row}">**
23. **<td><c:out value="${column}"/></td>**
24. **</c:forEach>**
25. **</tr>**
26. **</c:forEach>**
27. **</table>**
28. <%@page contentType="text/html" pageEncoding="UTF-8"%>
29. <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"

"http://www.w3.org/TR/html4/loose.dtd">

Note that the IDE automatically added taglib directives needed for the JSTL tags used in the generated content (<sql:query> and <c:forEach>). A taglib directive declares that the JSP page uses custom (i.e., JSTL) tags, names the tag library that defines them, and specifies their tag prefix.

1. Run the project to see how it displays in a browser. Right-click the project node in the Projects window and choose Run.

When you choose Run, the IDE deploys the project to the GlassFish server, the index page is compiled into a servlet, and the welcome page opens in your default browser. The code generated from the DB Report item creates the following table in the welcome page.



As you can see, the DB Report item enables you to quickly test your database connection, and enables you to view table data from the database in your browser. This can be particularly useful when prototyping.

The following steps demonstrate how to integrate the generated code into the HTML drop-down list you created earlier in the tutorial.

1. Examine the column data in the generated code. Two <c:forEach> tags are used; one is nested inside the other. This causes the JSP container (i.e., the GlassFish server) to perform a loop on all table rows, and for each row, it loops through all columns. In this manner, data for the entire table is displayed.
2. Integrate the <c:forEach> tags into the HTML form as follows. The value of each item becomes the subject\_id, and the output text becomes the name, as recorded in the database. (Changes are displayed in **bold**).
3. <form action="response.jsp">
4. <strong>Select a subject:</strong>
5. <select name="subject\_id">
6. **<c:forEach var="row" items="${subjects.rowsByIndex}">**
7. **<c:forEach var="column" items="${row}">**
8. <option **value="<c:out value="${column}"/>"**>**<c:out value="${column}"/>**</option>
9. **</c:forEach>**
10. **</c:forEach>**
11. </select>
12. <input type="submit" value="submit" name="submit" />

</form>

An alternative, simpler way to integrate the <c:forEach> tags into the HTML form would be as follows.

<form action="response.jsp">

<strong>Select a subject:</strong>

<select name="subject\_id">

**<c:forEach var="row" items="${subjects.rows}">**

<option **value="${row.subject\_id}"**>**${row.name}**</option>

**</c:forEach>**

</select>

<input type="submit" value="submit" name="submit" />

</form>

In either case, the <c:forEach> tags loop through all subject\_id and name values from the SQL query, and insert each pair into the HTML <option> tags. In this manner, the form's drop-down list is populated with data.

1. Delete the table that was generated from the DB Report item. (Deletion shown below as **~~strike-through text~~**.)
2. <%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
3. <%@taglib prefix="sql" uri="http://java.sun.com/jsp/jstl/sql"%>
4. <%--
5. Document : index
6. Created on : Dec 22, 2009, 7:39:49 PM
7. Author : nbuser
8. --%>
9. <sql:query var="subjects" dataSource="jdbc/IFPWAFCAD">
10. SELECT subject\_id, name FROM Subject
11. </sql:query>
12. **~~<table border="1">~~**
13. **~~<!-- column headers -->~~**
14. **~~<tr>~~**
15. **~~<c:forEach var="columnName" items="${subjects.columnNames}">~~**
16. **~~<th><c:out value="${columnName}"/></th>~~**
17. **~~</c:forEach>~~**
18. **~~</tr>~~**
19. **~~<!-- column data -->~~**
20. **~~<c:forEach var="row" items="${subjects.rowsByIndex}">~~**
21. **~~<tr>~~**
22. **~~<c:forEach var="column" items="${row}">~~**
23. **~~<td><c:out value="${column}"/></td>~~**
24. **~~</c:forEach>~~**
25. **~~</tr>~~**
26. **~~</c:forEach>~~**
27. **~~</table>~~**
28. <%@page contentType="text/html" pageEncoding="UTF-8"%>
29. <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"

"http://www.w3.org/TR/html4/loose.dtd">

1. Save your changes (Ctrl-S; ⌘-S on Mac).
2. Refresh the welcome page of the project in your browser.

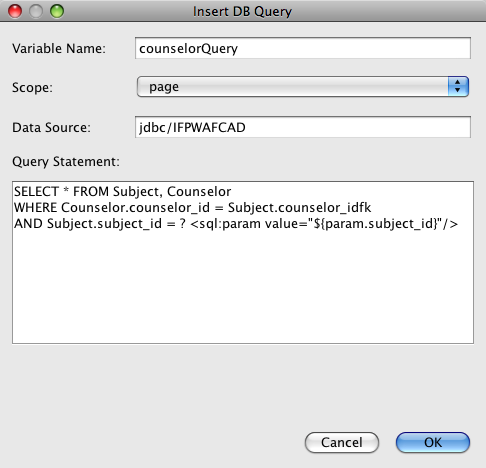
Note that the drop-down list in the browser now contains subject names that were retrieved from the database.

You do not need to redeploy your project because compile-on-save is enabled for your project by default. This means that when you modify and save a file, the file is automatically compiled and deployed and you do not need to recompile the entire project. You can enable and disable compile-on-save for your project in the Compiling category of the Properties window of the project.

**response.jsp**

The response page provides details for the counselor who corresponds to the subject chosen in the welcome page. The query you create must select the counselor record whose counselor\_id matches the counselor\_idfk from the selected subject record.

1. Place your cursor above the <%@page ... %> declaration (line 7), and double-click DB Query in the Palette to open the Insert DB Query dialog box.
2. Enter the following details in the Insert DB Query dialog box.
   * **Variable Name:** counselorQuery
   * **Scope:** page
   * **Data Source:** jdbc/IFPWAFCAD
   * **Query Statement:** SELECT \* FROM Subject, Counselor WHERE Counselor.counselor\_id = Subject.counselor\_idfk AND Subject.subject\_id = ? <sql:param value="${param.subject\_id}"/>



1. Click OK. The following content is generated in the response.jsp file. (New content shown in **bold**.)
2. **<%@taglib prefix="sql" uri="http://java.sun.com/jsp/jstl/sql"%>**
3. <%--
4. Document : response
5. Created on : Dec 22, 2009, 8:52:57 PM
6. Author : nbuser
7. --%>
8. **<sql:query var="counselorQuery" dataSource="jdbc/IFPWAFCAD">**
9. **SELECT \* FROM Subject, Counselor**
10. **WHERE Counselor.counselor\_id = Subject.counselor\_idfk**
11. **AND Subject.subject\_id = ? <sql:param value="${param.subject\_id}"/>**
12. **</sql:query>**
13. <%@page contentType="text/html" pageEncoding="UTF-8"%>
14. <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"

"http://www.w3.org/TR/html4/loose.dtd">

Note that the IDE automatically added the taglib directive needed for the <sql:query> tag. Also, note that you used an <sql:param> tag directly within the query. Because this query relies on the subject\_id value that was submitted from index.jsp, you can extract the value using an EL (Expression Language) statement in the form of ${param.subject\_id}, and then pass it to the <sql:param> tag so that it can be used in place of the SQL question mark (?) during runtime.

1. Use a <c:set> tag to set a variable that corresponds to the first record (i.e., row) of the resultset returned from the query. (New content shown in **bold**.)
2. <sql:query var="counselorQuery" dataSource="jdbc/IFPWAFCAD">
3. SELECT \* FROM Subject, Counselor
4. WHERE Counselor.counselor\_id = Subject.counselor\_idfk
5. AND Subject.subject\_id = ? <sql:param value="${param.subject\_id}"/>
6. </sql:query>

**<c:set var="counselorDetails" value="${counselorQuery.rows[0]}"/>**

Although the resultset returned from the query should only contain a single record, this is a necessary step because the page needs to access values from the record using EL (Expression Language) statements. Recall that in index.jsp, you were able to access values from the resultset simply by using a <c:forEach> tag. However, the <c:forEach> tag operates by setting a variable for the rows contained in the query, thus enabling you to extract values by including the row variable in EL statements.

1. Add the taglib directive for the JSTL core library to the top of the file, so that the <c:set> tag is understood. (New content shown in **bold**.)
2. **<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>**

<%@taglib prefix="sql" uri="http://java.sun.com/jsp/jstl/sql"%>

1. In the HTML markup, replace all placeholders with EL statements code that display the data held in the counselorDetails variable. (Changes below shown in **bold**):
2. <html>
3. <head>
4. <meta http-equiv="Content-Type" content="text/html; charset=UTF-8"/>
5. <link rel="stylesheet" type="text/css" href="style.css">
6. <title>**${counselorDetails.name}**</title>
7. </head>
8. <body>
9. <table>
10. <tr>
11. <th colspan="2">**${counselorDetails.name}**</th>
12. </tr>
13. <tr>
14. <td><strong>Description: </strong></td>
15. <td><span style="font-size:smaller; font-style:italic;">**${counselorDetails.description}**</span></td>
16. </tr>
17. <tr>
18. <td><strong>Counselor: </strong></td>
19. <td><strong>**${counselorDetails.first\_name} ${counselorDetails.nick\_name} ${counselorDetails.last\_name}**</strong>
20. <br><span style="font-size:smaller; font-style:italic;">
21. <em>member since: **${counselorDetails.member\_since}**</em></span></td>
22. </tr>
23. <tr>
24. <td><strong>Contact Details: </strong></td>
25. <td><strong>email: </strong>
26. <a href="mailto:**${counselorDetails.email}**">**${counselorDetails.email}**</a>
27. <br><strong>phone: </strong>**${counselorDetails.telephone}**</td>
28. </tr>
29. </table>
30. </body>

</html>

**Running the Completed Application**

You've now completed the application. Try running it again to see how it displays in a browser. Note that because of NetBeans' Compile on Save feature, you do not need to worry about compiling or redeploying the application. When you run a project, you can be sure the deployment contains your latest changes.

Click the Run Project ( Run Project button) button in the main toolbar. The index.jsp page opens in the IDE's default browser.

When index.jsp displays in the browser, select a subject from the drop-down list and click submit. You should now be forwarded to the response.jsp page, showing details corresponding to your selection.



This concludes the Creating a Simple Web Application Using a MySQL Database tutorial. This document demonstrated how to create a simple web application that connects to a MySQL database. It also demonstrated how to construct an application using a basic two-tier architecture, and utilized numerous technologies including JSP, JSTL, JDBC, and JNDI as a means of accessing and displaying data dynamically.

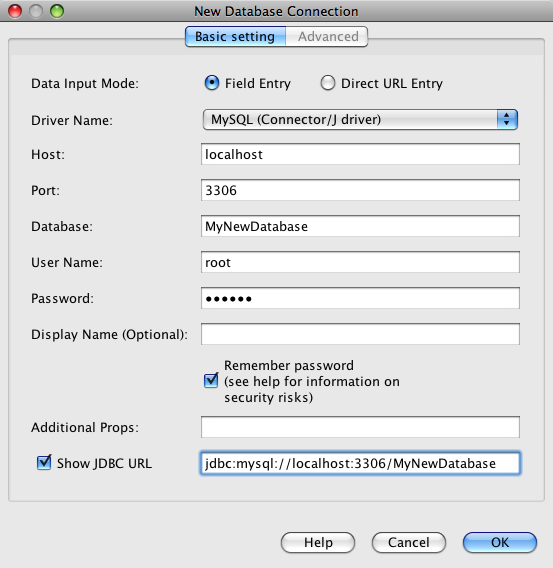
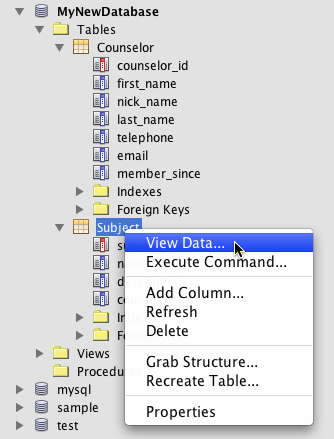
**Troubleshooting**

Most of the problems that occur with the tutorial application are due to communication difficulties between the GlassFish Server Open Source Edition and the MySQL database server. If your application does not display correctly, or if you are receiving a server error, the following examinations may be useful.

* [Do database resources exist?](https://netbeans.org/kb/docs/web/mysql-webapp.html#access?)
* [Do the connection pool and data source exist on the server?](https://netbeans.org/kb/docs/web/mysql-webapp.html#datasource?)
* [Is the MySQL Connector/J driver accessible to the GlassFish server?](https://netbeans.org/kb/docs/web/mysql-webapp.html#driver?)
* [Is the database password-protected?](https://netbeans.org/kb/docs/web/mysql-webapp.html#password?)
* [Are the connection pool properties correctly set?](https://netbeans.org/kb/docs/web/mysql-webapp.html#ping?)

**Do database resources exist?**

Use the IDE's Services window (Ctrl-5; ⌘-5 on Mac) to ensure that the MySQL server is running, and that MyNewDatabase is accessible and contains appropriate table data.

* To connect to the MySQL database server, right-click the MySQL Server node and choose Connect.
* If a connection node ( Database connection node) for MyNewDatabase does not display in the Services window, you can create a connection by right-clicking the MySQL driver node ( Database driver node) and choosing Connect Using. Enter the required details in the dialog that displays.   
    
  The fields provided in the New Database Connection dialog mirror the URL string entered in the Show JDBC URL option. Therefore, if you know the URL (e.g., jdbc:mysql://localhost:3306/MyNewDatabase) you can paste it into the Show JDBC URL field, and the remaining dialog fields become automatically populated.
* To ensure that the Subject and Counselor tables exist and that they contain sample data, expand the MyNewDatabase connection node ( Database connection node) and locate the MyNewDatabase catalog node ( Database catalog node). Expand the catalog node to view existing tables. You can view table data by right-clicking a table node and choosing View Data.   
  

**Do the connection pool and data source exist on the server?**

After deploying the application to the GlassFish server, the glassfish-resources.xml contained in the project should instruct the server to create a JDBC resource and connection pool. You can determine whether these exist from the Servers node in the Services window.

* Expand the Servers > the GlassFish Server > Resources node. Expand JDBC Resources to view the jdbc/IFPWAFCAD data source that was created from glassfish-resources.xml. Expand the Connection Pools node to view the IfpwafcadPool connection pool that was created from glassfish-resources.xml. (This is [demonstrated above](https://netbeans.org/kb/docs/web/mysql-webapp.html#view-connection-pool).)

**Is the MySQL Connector/J driver accessible to the GlassFish server?**

Make sure that the MySQL Connector/J driver has been deployed to the GlassFish server. (This is discussed in [Adding the database driver's JAR file to the server](https://netbeans.org/kb/docs/web/mysql-webapp.html#addJar).)

* Locate the GlassFish server installation folder on your computer and drill down into the GlassFish domains/domain1/lib subfolder. Here you should find the mysql-connector-java-5.1.6-bin.jar file.

**Is the database password-protected?**

The database needs to be password-protected to enable the GlassFish server data source to work properly in this tutorial. If you are using the default MySQL root account with an empty password, you can set the password from a command-line prompt.

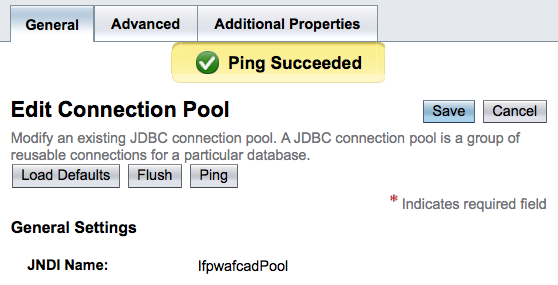
* To set your password to *nbuser*, navigate to your MySQL installation's bin directory in the command-line prompt and enter the following:
* shell> mysql -u root
* mysql> UPDATE mysql.user SET Password = PASSWORD('*nbuser*')
* -> WHERE User = 'root';

mysql> FLUSH PRIVILEGES;

For more information, see the official MySQL Reference Manual: [Securing the Initial MySQL Accounts](http://dev.mysql.com/doc/refman/5.1/en/default-privileges.html).

**Are the connection pool properties correctly set?**

Ensure that the connection pool is working correctly for the server.

1. Open the Services window (Ctrl-5; ⌘-5 on Mac) and expand the Servers node.
2. Right-click the GlassFish server node and choose View Admin Console.
3. Enter admin/adminadmin for the default username/password.
4. In the tree on the left side of the console, expand the Resources > JDBC > JDBC Connection Pools > IfpwafcadPool node. Details for the IfpwafcadPool connection pool display in the main window.
5. Click the Ping button. If the connection pool is set up correctly, you will see a 'Ping Succeeded' message.   
   
6. If the ping fails, click the Additional Properties tab and ensure that the listed property values are correctly set.