

Using JSF and JDBC to Access a MySQL Database

Overview

@author R.L. Martinez, Ph.D.

In 1994, a number of database vendors began to release software drivers adhering to the Open Database Connectivity (ODBC) specification which enabled connection to their database servers from a variety of sources. The ODBC specification provided the open standards which supported various client-to-database connectivity solutions. Developers could use ODBC drivers to connect to their database of choice. However, the connection ubiquity and convenience of ODBC came at the price of excessive driver size and degraded performance. To improve connections, database vendors worked to produce drivers that were environment specific. JDBC was developed in response to the ODBC limitations and has evolved into four types of drivers.

Java Database Connectivity (JDBC)

The four types of JDBC drivers are listed in the table.

Four Types of JDBC Drivers	
Type	Description
1	Drivers that implement the JDBC API as a mapping to another data access API, such as ODBC (Open Database Connectivity). Drivers of this type are generally dependent on a native library, which limits their portability. The JDBC-ODBC Bridge is an example of a Type 1 driver.
2	Drivers that are written partly in the Java programming language and partly in native code. These drivers use a native client library specific to the data source to which they connect. Again, because of the native code, their portability is limited. Oracle's OCI (Oracle Call Interface) client-side driver is an example of a Type 2 driver.
3	Drivers that use a pure Java client and communicate with a middleware server using a database-independent protocol. The middleware server then communicates the client's requests to the data source.
4	Drivers that are pure Java and implement the network protocol for a specific data source. The client connects directly to the data source.
<p>Note: The JDBC-ODBC Bridge should be considered a transitional solution. It is not supported by Oracle. Consider using this only if your DBMS does not offer a Java-only JDBC driver.</p> <p>Note: the descriptions above are from Oracle: http://docs.oracle.com/javase/tutorial/jdbc/basics/gettingstarted.html</p>	

Using JSF and JDBC to Access a MySQL Database

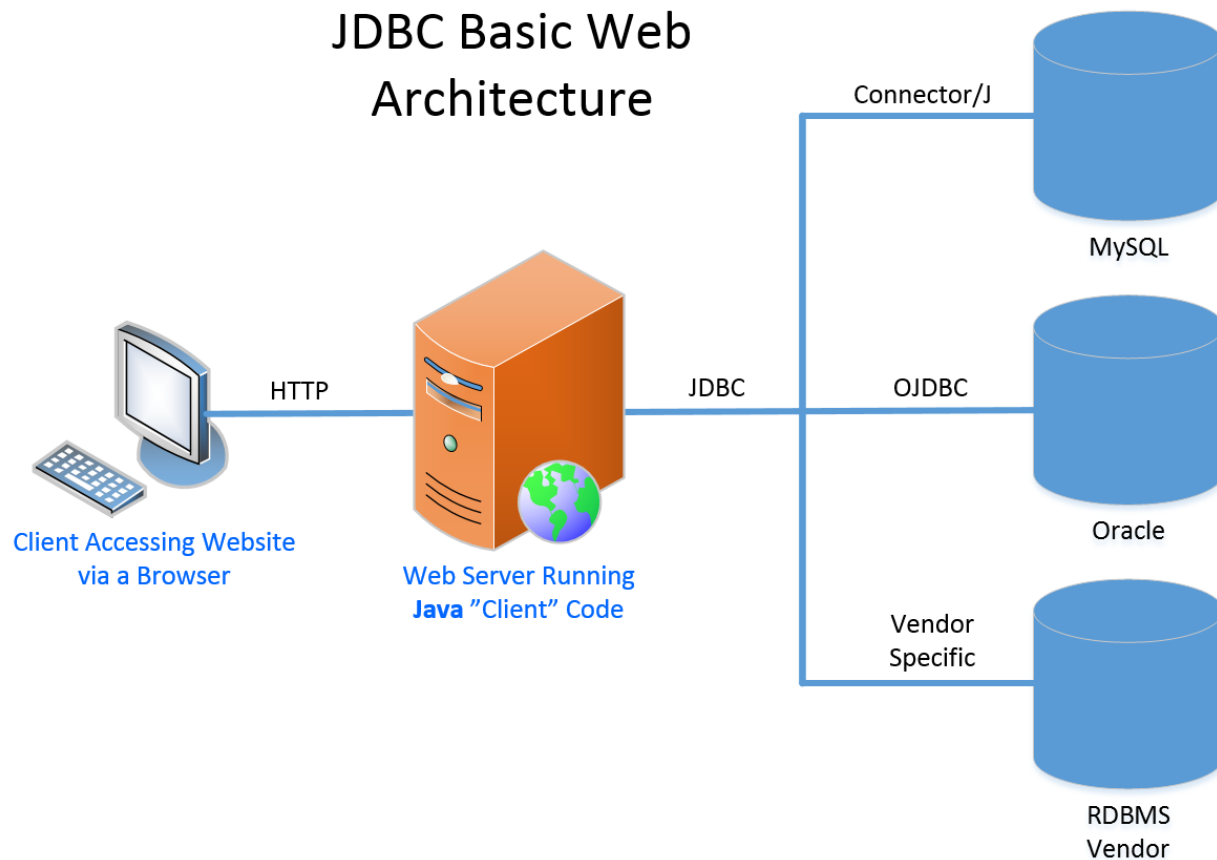
The Connector/J driver which is the specific API for Java to MySQL communication is a Type 4 driver. When developing with Java and accessing a database, a Type 4 driver should be selected if available. It should be noted in the table above that the word client, for web development, refers to software on the web server which acts as a client connecting to the database host. Clients are often considered the device on which the browser is installed but in the table above the client software is in relation to the database server and not the web server.

JDBC Web Architecture

The basic web architecture of JDBC is shown in the diagram below. Java programs running on the server use a driver developed specifically for the Relational Database Management System (RDBMS) vendor (usually by each vendor). As shown, Connector/J is the driver of choice when using Java to connect to MySQL. Note that there is a client running a browser. There is also “client” code (with respect to the database) running on the web server as discussed above.

If the code type running on the server in the diagram were changed from Java to another server-side web programming platform, then the drivers listed would also need to change. For instance, if the server code were PHP or ASP.NET, then the driver to connect to MySQL would be mysqlnd or Connector/NET respectively. All of the drivers listed, Connector/J, OJDBC, mysqlnd, and Connector/NET are JDBC drivers. Each driver supplies a solution for specific client and database connection point combinations.

Using JSF and JDBC to Access a MySQL Database



Create Read Update Delete (CRUD)

The four primary activities associated with most databases are create, read, update, and delete. Records are created in the database with an insert statement (or batch data loads). Data is read from the database using select statements. Update and delete statements complete the foursome. The combination of the four have become known as CRUD (an albeit non-flattering but easy to remember acronym). The web application in this tutorial uses the JSF framework and JDBC to implement and demonstrate each of these four operations.

Since this is the third portion of an advanced Java course (third in a series of three courses), only the code specific to JSF and JDBC is addressed with a possible few exceptions. Furthermore, topics covered in previous modules in this course will not be reviewed. For instance, the `h:commandButton` control appears on the `index.xhtml` page. Since it was discussed (at length) in the JSFTags tutorial, it will

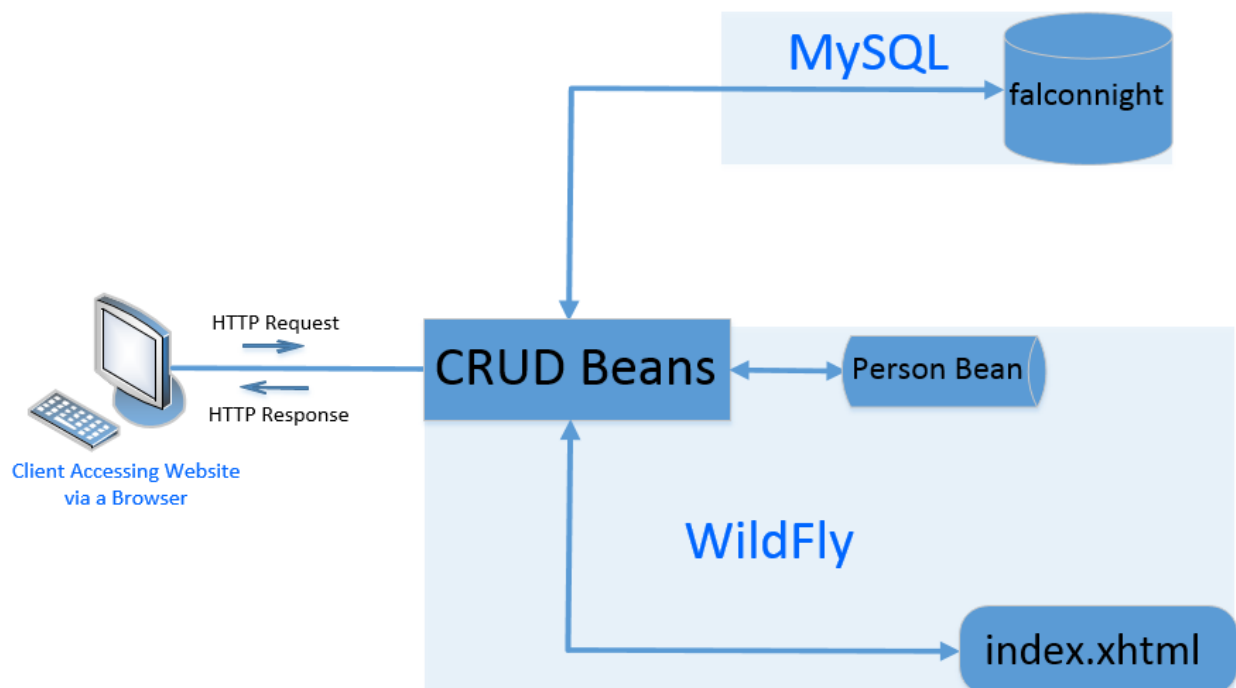
Using JSF and JDBC to Access a MySQL Database

not be explicated again. It might be beneficial for the student to review previous modules in the course in addition to the JSP modules in course ITSE 2317 Intermediate Java.

JSF-JDBC Architecture

The architecture of the JSF-JDBC application to be developed in this tutorial is shown in the diagram below. MVC stands for Model-View-Controller which is a software design pattern that will be reviewed in the next tutorial. For now, observe the various components of the application. Each of the CRUD operations will have a managed bean which will act as a controller, sending requests to the database (model), updating the Person bean local in-memory storage (model), and making the data available to index.xhtml (view).

MVC Architecture of JSF-JDBC Web Application



Developing the JSF-JDBC Web Application

In the DatabaseSetup tutorial we accomplished the following:

Using JSF and JDBC to Access a MySQL Database

1. Created the falconnight database in MySQL
2. Created a connection (and connection string) to falconnight in NetBeans

The next step is to develop the JSF-JDBC web application (henceforth known as JSF-JDBC) which will utilize the work we did in the DatabaseSetup tutorial.

Recall that the falconnight database contains four tables. JSF-JDBC only uses the person table from the database. The hobby table is required for an assignment in the course. The person-hobby and person-history tables were only supplied for educational purposes. The People web page shown is the output of the JSF-JDBC web application which we build below.

Using JSF and JDBC to Access a MySQL Database

People

Create

Person to Create

Read

Person ID	Name	Nickname	Date
1	Joe	HandyMan	15-May-2014
2	Bob	Poodle	15-May-2014
3	Bill	Scout	15-May-2014

Update

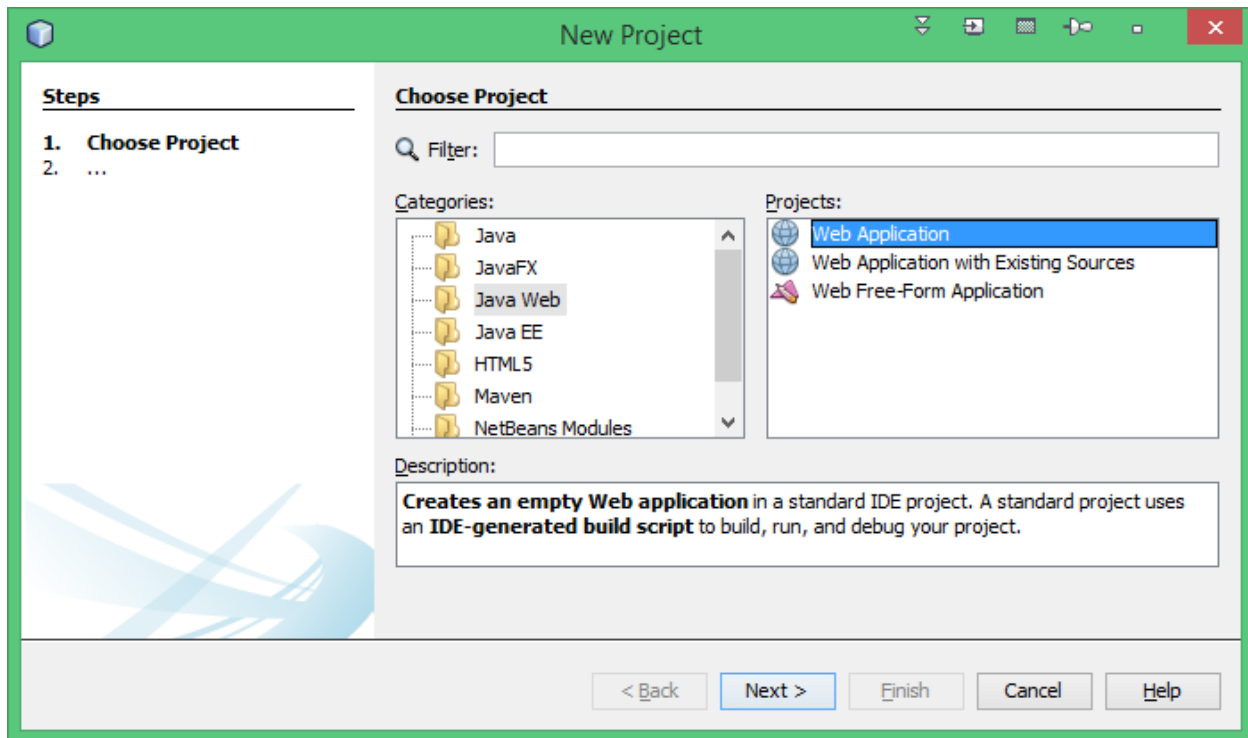
Person Info to Update

Delete

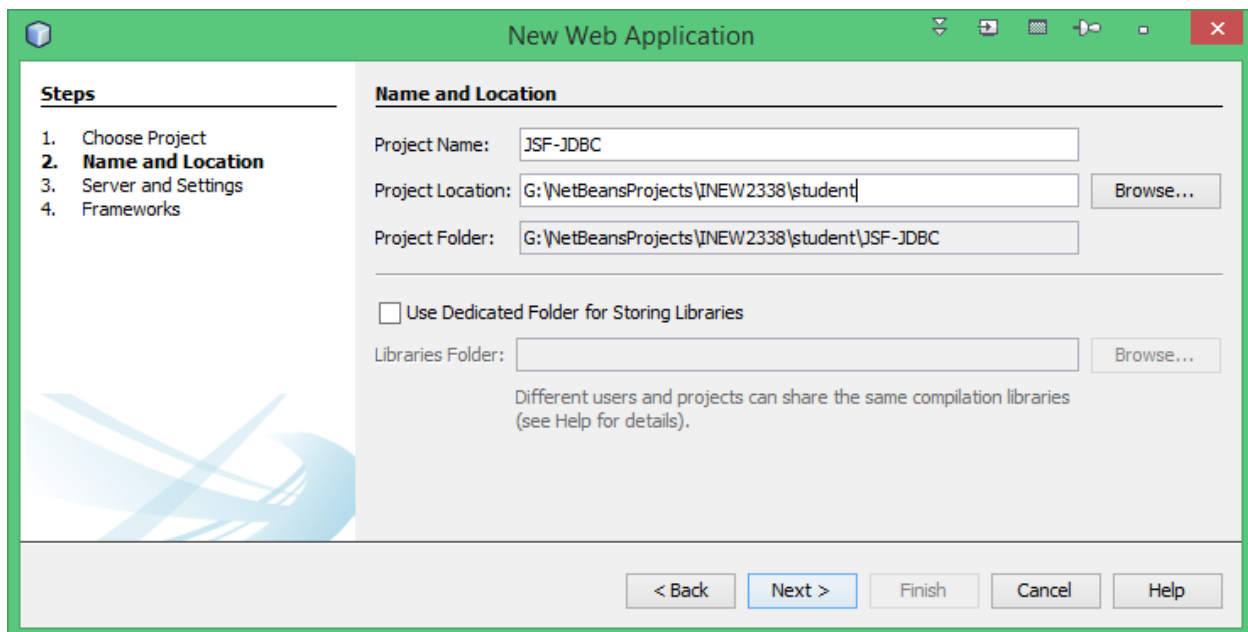
Person ID to Delete

1. Create a new project in NetBeans by File | New Project | Java Web | Web Application | Next.

Using JSF and JDBC to Access a MySQL Database

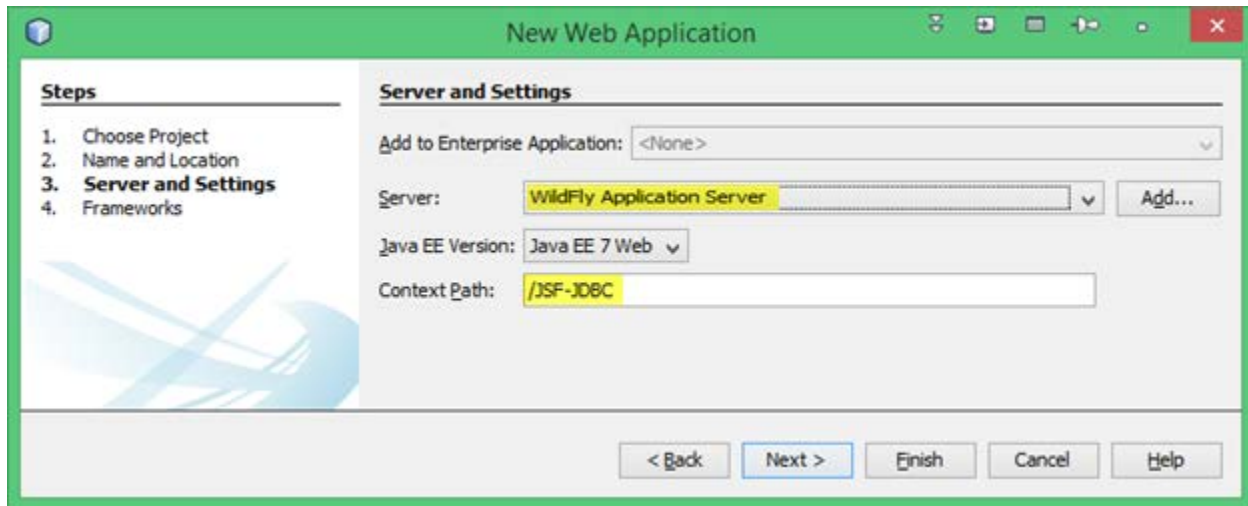


2. Make the appropriate selections. Be sure to organize your work. Superior organization of files/folders/projects and other work is essential for the professional web developer.



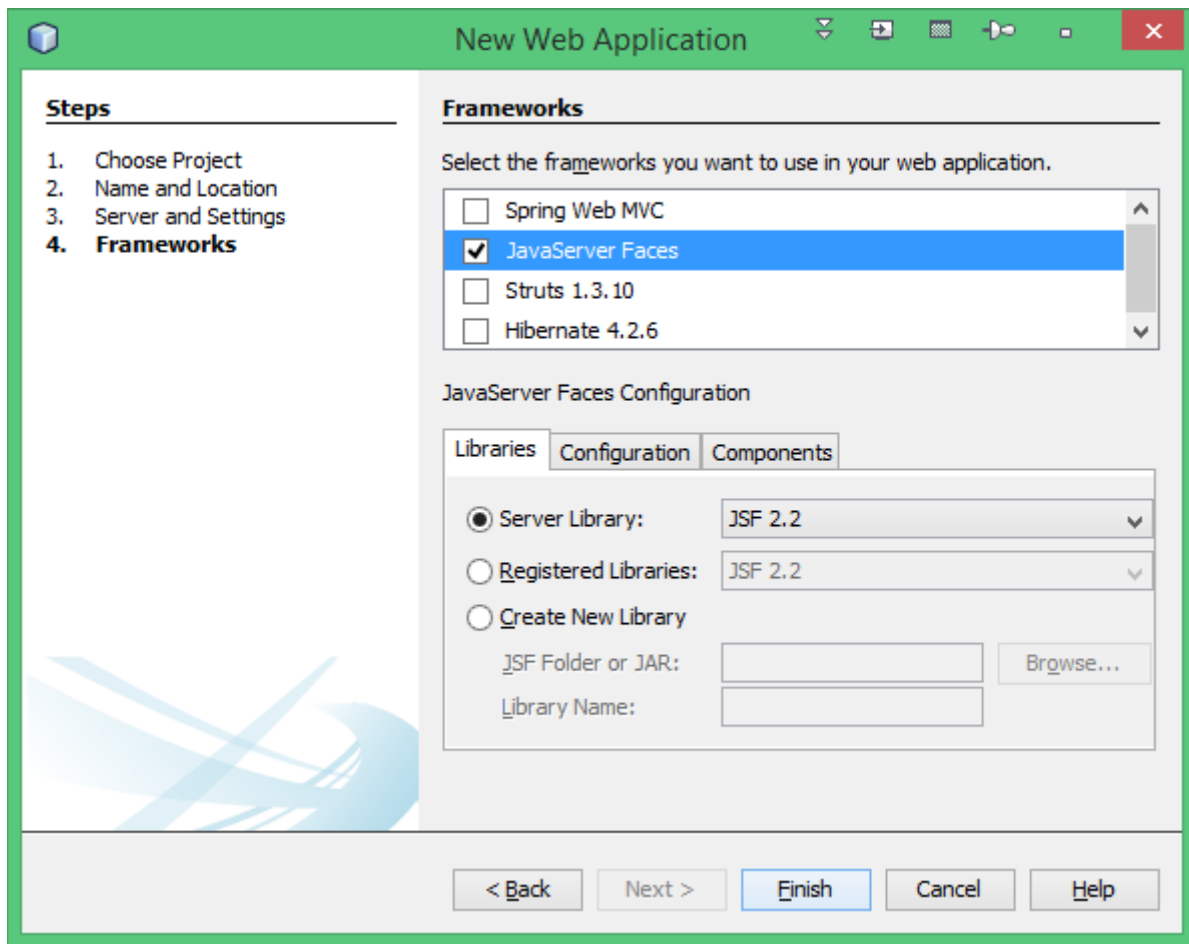
Using JSF and JDBC to Access a MySQL Database

3. Make the appropriate selections in the Server and Settings dialog. This tutorial uses the WildFly web server. While other alternatives exist such as TomEE and GlassFish, WildFly is fully Java EE 7 compliant, certified, and is very fast.



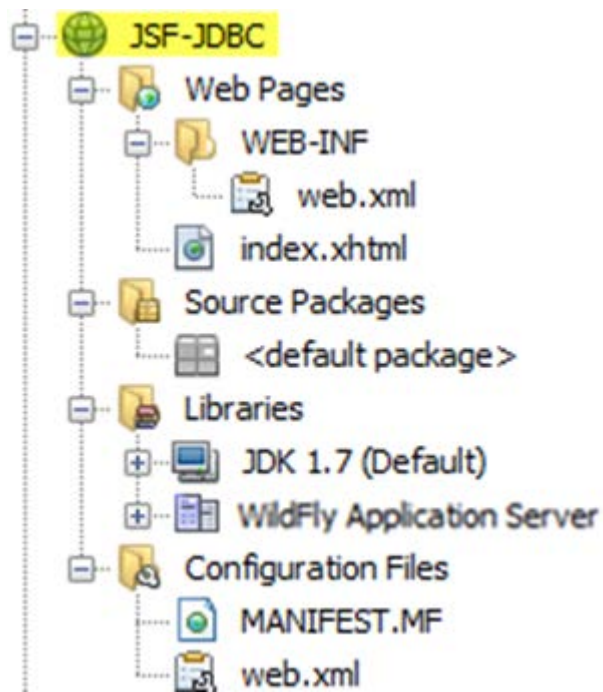
Using JSF and JDBC to Access a MySQL Database

4. Select the JSF framework | Finish.



5. Take this opportunity to confirm the project structure that NetBeans has built. The index.xhtml is familiar from previous tutorials and will be replaced shortly. The expanded project window is shown. We will be adding the com.mysite package and five files therein.

Using JSF and JDBC to Access a MySQL Database



Using JSF and JDBC to Access a MySQL Database

6. Double-click to open web.xml. The content should appear like that shown.
Also, I recommend that the session-timeout be increased to avoid unnecessary timeout issues during development and testing.

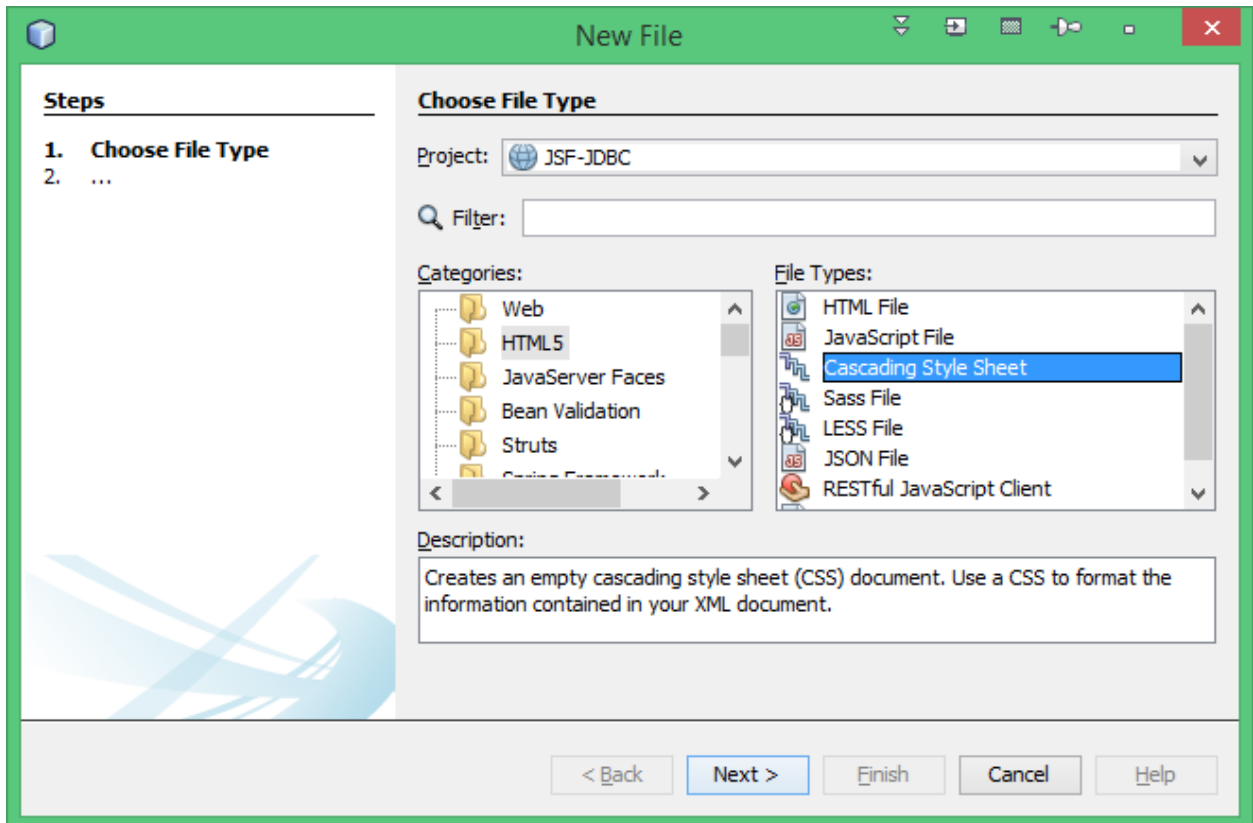
```

1  <?xml version="1.0" encoding="UTF-8"?>
2  <!-- web.xml -->
3  <web-app version="3.1" xmlns="http://xmlns.jcp.org/xml/ns/javaee"
4         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
5         xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
6         http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd">
7      <context-param>
8          <param-name>javax.faces.PROJECT_STAGE</param-name>
9          <param-value>Development</param-value>
10     </context-param>
11     <servlet>
12         <servlet-name>Faces Servlet</servlet-name>
13         <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
14         <load-on-startup>1</load-on-startup>
15     </servlet>
16     <servlet-mapping>
17         <servlet-name>Faces Servlet</servlet-name>
18         <url-pattern>/faces/*</url-pattern>
19     </servlet-mapping>
20     <session-config>
21         <session-timeout>
22             330
23         </session-timeout>
24     </session-config>
25     <welcome-file-list>
26         <welcome-file>faces/index.xhtml</welcome-file>
27     </welcome-file-list>
28 </web-app>

```

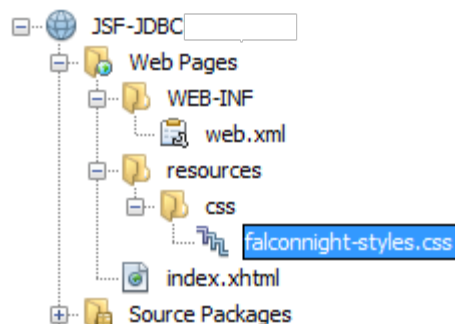
Using JSF and JDBC to Access a MySQL Database

7. R-click the Web Pages folder in the Projects window | New | Folder | name the folder resources. R-click the resources folder | New | Folder | css. R-click the css folder | New | Other | HTML5 | Cascading Style Sheet | Next | name the file falconnight-styles.css | Finish.



8. By creating the directory structure as shown, the file is accessible from line 9 of index.xhtml via the library attribute.

9 | `<h:outputStylesheet library="css" name="falconnight-styles.css"/>`



Using JSF and JDBC to Access a MySQL Database

```

1  /* falconnight-styles.css @author R.L. Martinez, Ph.D.*/
2  body{
3      background-color: khaki;
4  }
5  h1{
6      text-align:center;
7      color: blue;
8  }
9  fieldset{
10     margin-bottom: 15px;
11     border-color: blue;
12 }
13 legend{
14     font-weight: bold;
15     font-family: arial;
16     color: blue;
17 }
18 #container{
19     display: table;
20     margin: 20px auto;
21     text-align:center;
22 }
23 .people-table{
24     border-collapse:collapse;
25     margin: 20px auto;
26 }
27 .people-table-header{
28     text-align:center;
29     color: #DDD;
30     background-color: #333F48;
31     border-bottom:1px solid #BBB;
32     padding:18px;
33 }
34 .people-table-odd-row{
35     text-align:center;
36     background-color: #FFF;
37     border-top:1px solid #BBB;
38     padding-right: 5px;
39 }
40 .people-table-even-row{
41     text-align:center;
42     background-color: #CFF;
43     border-top:1px solid #BBB;
44     padding-right: 5px;
45 }
46 .datestyle{
47     padding-right: 15px;
48 }
49 .inputID{
50     text-align: center;
51     font-weight: bold;
52     width: 40px;
53     margin-right: 5px;
54     background-color: moccasin;
55 }
56 .commandButton{
57     background-color: #F2A900;
58     font-weight: bold;
59 }

```

9. Modify the falconnight-styles.css file as shown.

Using JSF and JDBC to Access a MySQL Database

```

1  <?xml version="1.0" encoding="UTF-8"?>
2  <!-- index.xhtml @author R.L. Martinez, Ph.D. -->
3  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
4  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
5  <html xmlns="http://www.w3.org/1999/xhtml"
6  xmlns:h="http://java.sun.com/jsf/html"
7  xmlns:f="http://java.sun.com/jsf/core">
8  <h:head>
9  <h:outputStylesheet library="css" name="falconnight-styles.css"/>
10 </h:head>
11
12 <h:body>
13 <div id='container'>
14 <h1>People</h1>
15 <fieldset>
16 <legend>Create</legend>
17 <h:form id="jdbcCreateForm">
18 <h3>Person to Create</h3>
19 <h:inputText value="#{createBean.person.name}"/>
20 <h:inputText value="#{createBean.person.nickname}"/>
21 <h:commandButton value="Create" styleClass="commandButton"
22 action="#{createBean.performCreate()}" />
23 </h:form>
24 </fieldset>
25 <fieldset>
26 <legend>Read</legend>
27 <h:dataTable value="#{readBean.performRead()}" var="p"
28 headerClass="people-table-header"
29 styleClass="people-table"
30 rowClasses="people-table-odd-row, people-table-even-row">
31 <h:column>
32 <f:facet name="header">
33 Person ID
34 </f:facet>
35 #{p.personID}
36 </h:column>
37 <h:column>
38 <f:facet name="header">
39 Name
40 </f:facet>
41 #{p.name}
42 </h:column>
43 <h:column>
44 <f:facet name="header">
45 Nickname
46 </f:facet>
47 #{p.nickname}
48 </h:column>
49 <h:column>
50 <f:facet name="header">
51 Date
52 </f:facet>
53 <h:outputText value="#{p.created_date}" styleClass="datestyle">
54 <f:convertDateTime pattern="dd-MMM-yyyy" />
55 </h:outputText>
56 </h:column>
57 </h:dataTable>
58 </fieldset>
59 </fieldset>

```

10. Modify the index.xhtml file as shown. We will discuss the code in index.xhtml and the .java files after the code is depicted (written).

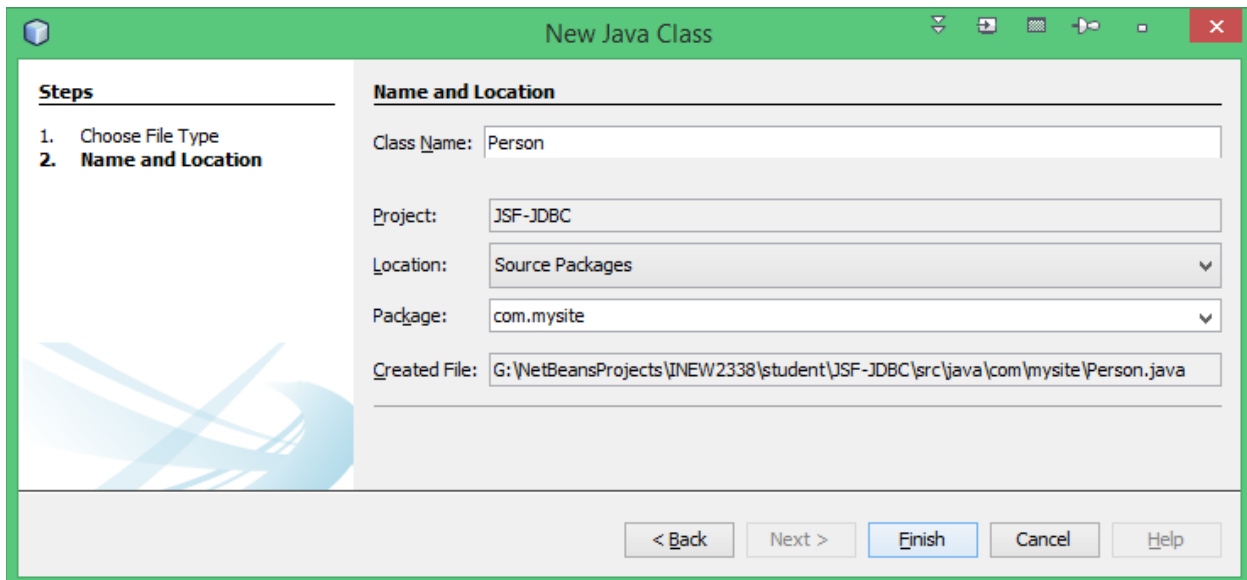
Using JSF and JDBC to Access a MySQL Database

```

60      <legend>Update</legend>
61      <h:form id="jdbcUpdateForm">
62          <h3>Person Info to Update</h3>
63          <h:inputText value="#{updateBean.person.personID}" styleClass="inputID"/>
64          <h:inputText value="#{updateBean.person.name}"/>
65          <h:inputText value="#{updateBean.person.nickname}"/>
66          <h:commandButton value="Update" styleClass="commandButton"
67                          action="#{updateBean.performUpdate()}" />
68      </h:form>
69  </fieldset>
70  <fieldset>
71      <legend>Delete</legend>
72      <h:form id="jdbcDeleteForm">
73          <h3>Person ID to Delete</h3>
74          <h:inputText value="#{deleteBean.person.personID}" styleClass="inputID"/>
75          <h:commandButton value="Delete" styleClass="commandButton"
76                          action="#{deleteBean.performDelete()}" />
77      </h:form>
78  </fieldset>
79 </div>
80 </h:body>
81
82 </html>

```

11. R-click on JSF-JDBC in the Project window | New | Java Class. Make settings as shown.



Steps	
1.	Choose File Type
2.	Name and Location

Name and Location	
Class Name:	Person
Project:	JSF-JDBC
Location:	Source Packages
Package:	com.mysite
Created File:	G:\NetBeansProjects\JNEW2338\student\JSF-JDBC\src\java\com\mysite\Person.java

Using JSF and JDBC to Access a MySQL Database

12. Modify Person.java as shown.

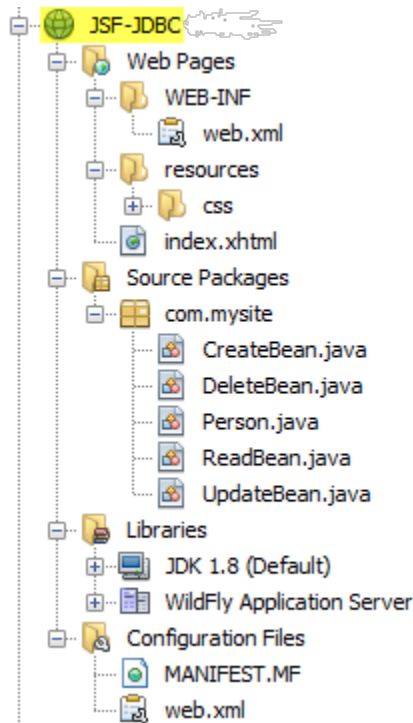
```

1  /* Person.java @author R.L. Martinez, Ph.D. */
2  package com.mysite;
3
4  import java.io.Serializable;
5  import java.util.Date;
6
7  public class Person implements Serializable {
8
9      private long personID;
10     private String name = "Enter Name";
11     private String nickname = "Enter Nickname";
12     private Date created_date;
13
14     public long getPersonID() {
15         return personID;
16     }
17
18     public void setPersonID(long personID) {
19         this.personID = personID;
20     }
21
22     public String getName() {
23         return name;
24     }
25
26     public void setName(String name) {
27         this.name = name;
28     }
29
30     public String getNickname() {
31         return nickname;
32     }
33
34     public void setNickname(String address) {
35         this.nickname = address;
36     }
37
38     public Date getCreated_date() {
39         return created_date;
40     }
41
42     public void setCreated_date(Date created_date) {
43         this.created_date = created_date;
44     }
45 }

```


Using JSF and JDBC to Access a MySQL Database

13.Repeat step 11 for CreateBean.java, ReadBean.java, UpdateBean.java, and DeleteBean.java. Your project directory tree structure should look like the following.



14.Modify the four managed bean files per the code provided below. Note that you have created a backing bean (or managed bean) for each of the four CRUD operations. The Person.java bean is used for intermediary storage of record data.

NOTE: In the four class files that follow, replace the line:

@Resource (lookup = “java:jboss/datasources/falconnight”)

with:

@Resource (lookup = “java:jboss/datasources/MySQLDS”)

Using JSF and JDBC to Access a MySQL Database

```

1  /* CreateBean.java @author R.L. Martinez, Ph.D.*/
2  package com.mysite;
3
4  import java.io.Serializable;
5  import java.sql.*;
6  import javax.annotation.Resource;
7  import javax.inject.Named;
8  import javax.enterprise.context.RequestScoped;
9  import javax.sql.DataSource;
10
11  @Named(value = "createBean")
12  @RequestScoped
13
14  public class CreateBean implements Serializable {
15
16      @Resource(lookup = "java:jboss/datasources/falconnight")
17      private DataSource dp;
18
19      Person person = new Person();
20
21      public Person getPerson() {
22          return person;
23      }
24
25      public void setPerson(Person person) {
26          this.person = person;
27      }
28
29      public void performCreate() throws SQLException {
30          if (dp == null) {
31              throw new SQLException("Cannot access data pool");
32          }
33          try (Connection con = dp.getConnection()) {
34              if (con == null) {
35                  throw new SQLException("Cannot establish connection to database");
36              }
37              java.util.Date date_now = new java.util.Date();
38              java.sql.Date sqlDate = new java.sql.Date(date_now.getTime());
39              String sql = "insert into person values (NULL, ?, ?, ?)";
40              PreparedStatement stmt = con.prepareStatement(sql);
41              stmt.setString(1, person.getName());
42              stmt.setString(2, person.getNickname());
43              stmt.setDate(3, sqlDate);
44
45              int result = stmt.executeUpdate();
46              if (result > 0) {
47                  System.out.println(" Record Inserted");
48              } else {
49                  System.out.println(" Record not Inserted");
50              }
51          }
52      }
53  }

```

Using JSF and JDBC to Access a MySQL Database

```

1  /* ReadBean.java @author R.L. Martinez, Ph.D.*/
2  package com.mysite;
3
4  import java.io.Serializable;
5  import java.sql.*;
6  import java.util.ArrayList;
7  import java.util.List;
8  import javax.annotation.Resource;
9  import javax.inject.Named;
10 import javax.enterprise.context.RequestScoped;
11 import javax.sql.DataSource;
12
13 @Named(value = "readBean")
14 @RequestScoped
15
16 public class ReadBean implements Serializable {
17
18     @Resource(lookup = "java:jboss/datasources/falconnight")
19     private DataSource dp;
20
21     public List<Person> performRead() throws SQLException {
22         if (dp == null) {
23             throw new SQLException("Cannot access data pool");
24         }
25         List<Person> list;
26         try (Connection con = dp.getConnection()) {
27             if (con == null) {
28                 throw new SQLException("Cannot establish connection to database");
29             }
30             PreparedStatement ps = con.prepareStatement(
31                 "select person_id, name, nickname, created_date from person");
32             ResultSet result = ps.executeQuery();
33             list = new ArrayList<>();
34             while (result.next()) {
35                 Person per = new Person();
36                 per.setPersonID(result.getInt("person_id"));
37                 per.setName(result.getString("name"));
38                 per.setNickname(result.getString("nickname"));
39                 per.setCreated_date(result.getDate("created_date"));
40                 list.add(per);
41             }
42         }
43         return list;
44     }
45 }

```

Using JSF and JDBC to Access a MySQL Database

```

1  /* UpdateBean.java @author R.L. Martinez, Ph.D.*/
2  package com.mysite;
3
4  import java.io.Serializable;
5  import java.sql.*;
6  import javax.annotation.Resource;
7  import javax.inject.Named;
8  import javax.enterprise.context.RequestScoped;
9  import javax.sql.DataSource;
10
11  @Named(value = "updateBean")
12  @RequestScoped
13
14  public class UpdateBean implements Serializable {
15
16      @Resource(lookup = "java:jboss/datasources/falconnight")
17      private DataSource dp;
18
19      Person person = new Person();
20
21      public Person getPerson() {
22          return person;
23      }
24
25      public void setPerson(Person person) {
26          this.person = person;
27      }
28
29      public void performUpdate() throws SQLException {
30          if (dp == null) {
31              throw new SQLException("Cannot access data pool");
32          }
33          try (Connection con = dp.getConnection()) {
34              if (con == null) {
35                  throw new SQLException("Cannot establish connection to database");
36              }
37              java.util.Date date_now = new java.util.Date();
38              java.sql.Date sqlDate = new java.sql.Date(date_now.getTime());
39              String sql = "update person "
40                  + "set name = ?, "
41                  + "nickname = ?, "
42                  + "created_date = ? "
43                  + "where person_id = ?";
44              PreparedStatement stmt = con.prepareStatement(sql);
45              stmt.setString(1, person.getName());
46              stmt.setString(2, person.getNickname());
47              stmt.setDate(3, sqlDate);
48              stmt.setInt(4, person.getPersonID());
49
50              int result = stmt.executeUpdate();
51              if (result > 0) {
52                  System.out.println("Record Updated");
53              } else {
54                  System.out.println("Record not Updated");
55              }
56          }
57      }
58  }

```

Using JSF and JDBC to Access a MySQL Database

```

1  /* DeleteBean.java @author R.L. Martinez, Ph.D.*/
2  package com.mysite;
3
4  import java.io.Serializable;
5  import java.sql.*;
6  import javax.annotation.Resource;
7  import javax.inject.Named;
8  import javax.enterprise.context.RequestScoped;
9  import javax.sql.DataSource;
10
11  @Named(value = "deleteBean")
12  @RequestScoped
13
14  public class DeleteBean implements Serializable {
15
16      @Resource(lookup = "java:jboss/datasources/falconnight")
17      private DataSource dp;
18
19      Person person = new Person();
20
21      public Person getPerson() {
22          return person;
23      }
24
25      public void setPerson(Person person) {
26          this.person = person;
27      }
28
29      public void performDelete() throws SQLException {
30          if (dp == null) {
31              throw new SQLException("Cannot access data pool");
32          }
33          try (Connection con = dp.getConnection()) {
34              if (con == null) {
35                  throw new SQLException("Cannot establish connection to database");
36              }
37              String sql = "delete from person where person_id = ?";
38              PreparedStatement stmt = con.prepareStatement(sql);
39              stmt.setInt(1, person.getPersonID());
40
41              int result = stmt.executeUpdate();
42              if (result > 0) {
43                  System.out.println("Record Deleted");
44              } else {
45                  System.out.println("Record not Deleted");
46              }
47          }
48      }
49  }

```

Using JSF and JDBC to Access a MySQL Database

15. After database and connection setup and coding the eight project files, we are ready to test the application. R-click on index.xhtml | Run File. You should see the output below.

People

Create

Person to Create

Read

Person ID	Name	Nickname	Date
1	Joe	HandyMan	15-May-2014
2	Bob	Poodle	15-May-2014
3	Bill	Scout	15-May-2014

Update

Person Info to Update

Delete

Person ID to Delete

Using JSF and JDBC to Access a MySQL Database

Code Review

Now let's discuss the code. If you have two monitors, you should open the code view in one and the discussion in another. Or, print the discussion and review the document while reviewing the code on the monitor (or vice versa). The first observation is to review the application output shown on the People page above. Notice that each CRUD operation is segmented on the page using `<fieldset>` elements and named with `<legend>` elements. Correlate each of the `<fieldset>` elements on the output page with the code that produces it in `index.xhtml`. For instance, the create output is produced by lines 16-24, and the read output by lines 24-58, etc. What is the source of the initial values Enter Name and Enter Nickname for the text controls? See if you can find the logical answer to that question.

There are three forms on the page; one for create, update, and delete. The read operation does not require a form because there is no submission to the database by the select statement. The application could use only one form. However, it is preferable to use three in this case. Do you recall why three forms are more appropriate than one in this case? Go back and review the JSFTags tutorial if that is not clear.

By the way, if you have questions about a particular topic you should be quick to conduct research. For instance, I am not going into further detail about the HTML `<fieldset>` element. If the element is new to you, its purpose should be somewhat evident by simple observation. However, if you have questions, there are vast resources within a few keystrokes. My search for `html fieldset` returned 14 million hits in 0.22 seconds. Any of the first ten results are helpful. We are so fortunate in the 21st century to have much of the world's body of knowledge at our finger tips. If your "Google-Fu" is not yet strong, you should work to change that.

Correlate the three controls on the `jdbcCreateForm` with those shown on the People page. There are two `h:inputText` tags and one `h:commandButton`. The text tags are bound to data elements in the `CreateBean` managed bean via the EL. Can you locate the reason the beans are referenced with a lowercase letter even though their names begin with capitals? The line of code that answers that question exists in each of the beans.

Using JSF and JDBC to Access a MySQL Database

This is the first time we have seen the double dot operator in the course. Recall from your studies in prerequisite courses that objects can contain three things: functionality (methods), data, and other objects. Just as the single dot operator provides access to object methods and data, the first dot operator provides access to an object contained within another object and the second dot operator provides access to methods and data within the contained object.

Take a look at the CreateBean.java code. Notice on line 19 an object of type Person is created named person. That is the object and its data that is referenced on lines 19-20 of index.xhtml. Review Person.java to see the familiar structure of a JSF managed bean. The Person class is made serializable which means that the container can convert the object into a string and move it to persistent storage if necessary. We are not working directly with serialization in this course but the servlet container (WildFly) may perform serialization as necessary as it manages resources. That is one of the reasons the objects are referred to as managed beans.

The Person class contains four data elements; all of which are made private to remain consistent with the information hiding principles of object orientation and the need-to-know principle of security management. Each of the data elements have the required getters and setters. Do you recall when the getters and setters are called? Recollect from the JSFTags tutorial that JSF automatically calls the getter method when the bound control is processed on index.xhtml and the setter is called upon form submission.

The jdbcUpdateForm and jdbcDeleteForm both use the same approach as that reviewed in jdbcCreateForm. That is, instantiate a person object in the bean for intermediate storage, bind JSF controls on index.xhtml to object data elements, populate the JSF elements with object data, and update the object data when the form is submitted by the h:commandButton which calls the “perform” method of each object.

Back to CreateBean. Lines 11 and 12 utilize a powerful concept known as Context and Dependency Injection (CDI). CDI was introduced with Java EE 6 in Dec2009. The @Named annotation is used to specify the name that other components of the application will use to access the managed bean. The value property is optional but is supplied here for semantic and declarative purposes. Note that createBean has a

Using JSF and JDBC to Access a MySQL Database

lowercase c. If a value is not supplied, the @Named property will be implicitly assigned the name of the bean with the first letter as lowercase which is what was supplied in value. So, one can infer that value is useful to divert from the default. The @RequestScoped CDI annotation has been previously discussed.

On line 16 an elegant capability is demonstrated known as resource (or dependency) injection. Resource injection is also part of the CDI specification. In the injection on line 16, the resource being injected by JSF is the JNDI name “java:jboss/datasources/MySQLDS”. Recall that this is the name of the JDBC resource we created in the MySQL local tutorial. That JDBC resource also implements a connection pool. JNDI uses a software design pattern known as service locator. When the service locator pattern is used, the service details are maintained by a central repository or registry. For our purposes, that is WildFly.

Since the resource was established and registered in WildFly, it can simply be “injected” into the managed bean with the one line of code (line 16). Without resource injection, a context object would be required and something like the following would be required to obtain a DataSource object dynamically:

```
try {
    Context context = new InitialContext();
    dp = (DataSource) context.lookup
        ("jdbc:mysql://localhost:3306/peoplehobbies?zeroDateTimeBehavior=convertToNull");
} catch (NamingException e) {
    e.printStackTrace();
}
```

Let’s move ahead. The performCreate() method on line 29 is called from line 22 index.xhtml when the h:commandButton is selected. On line 33 a Connection object is instantiated from the DataSource object which is mapped to a pool resource in WildFly.

Just as resource injection and the use of a context object provide different approaches to obtaining a DataSource, we also have an alternative to the DataSource object itself. The older, less preferred approach would be to establish a database connection using the DriverManager object which would look like this:

Using JSF and JDBC to Access a MySQL Database

```
Connection con = null;
String jdbcUrl = "jdbc:mysql://localhost:3306/peoplehobbies?zeroDateTimeBehavior=convertToNull";
con = DriverManager.getConnection(jdbcUrl, username, password);
```

Note the undesirable exposure of the username and password in the argument list.

By working with a DataSource object, the application does not require knowledge of the properties required to connect to the database such as username, password, or URL. Those properties were registered with the container and are therefore available with a simple resource injection reference or use of a context object.

Lines 37-38 of CreateBean convert a Java date to a SQL date. The conversion is required due to incompatibilities between the two formats. Line 39 is the SQL statement used to insert a record into the person table. Based on your knowledge of the person table, can you determine why the value of NULL is supplied?

The code on line 40 creates a prepared statement from the SQL supplied on line 39. Prepared statements should always be used prior to sending SQL to the database. They perform the important role of preventing the very dangerous hacking attack known as SQL injection. Prepared statements consist of two steps.

The first step is performed on line 40 which sends the SQL to the database server for syntax checking and compilation. The next step is performed on line 45 which executes the query. The three question marks in the argument list (known as anonymous, positional placeholders) in line 39 correspond to the values 1, 2, and 3 in lines 41-43. Notice that the values are extracted from the person object using the appropriate methods. Recall that the person object of the createBean contains data from the controls in the jdbcCreateForm on index.xhtml when performCreate() is called. Test the create operation by entering a new name and nickname and clicking Create. Notice the new in the Read table on the page.

Let's now consider the Read section on index.xhtml which is between lines 25-58. Read uses an h:dataTable control which provides a quickly way to display data in table format. The contents of the h:dataTable are populated by the value attribute which is performRead() method of the readBean managed bean. The var attribute is used to set an alias for the h:dataTable. The class attributes are CSS settings which are stored in the CSS file linked on line 9.

Using JSF and JDBC to Access a MySQL Database

There are four columns specified for the table and each has an f:facet which are used to associate a name with container components. Values for the data elements are supplied by results from the call to the `performRead()` method which executes a database select query. The `performRead()` method returns a list of person objects. The list object is declared on line 25 of `ReadBean.java`, instantiated with an `ArrayList` on line 33, and populated with a person object with each iterative call to `add()` on line 40.

A new person object is created for each row of the `ResultSet` returned by the `executeQuery()` method. A `ResultSet` is a collection of database records (rows) returned by the query operation. Note that `ReadBean.java` uses `executeQuery()` but the other three CRUD operations use `executeUpdate()`. `executeUpdate()` actually performs a change to the database whereas `executeQuery()` just reads from the database.

There is one more point to make about the Create code in `index.xhtml`. On line 54 a convertor JSF tag is used to reformat the SQL date into that shown. The standard SQL date is displayed as numerically as “YYYY-MM-DD” followed by the time. The convertor tag outputs the date in the format specified.

The `UpdateBean.java` and `DeleteBean.java` managed beans are very similar to the code already reviewed for `CreateBean.java`. The notable exception is that appropriate update and delete SQL statements are executed in the respective beans. By the way, the `System.out.println()` statements can be viewed in the console output to confirm expected operations.

That’s it for our current discussion of the JSF-JDBC web application. We will revisit the project again in the MVC tutorial where we take a closer look at some of the architectural benefits provided by the JSF framework.

Assignment Output

This is the output that should result from the JSF-JDBC assignment. The assignment details are in Blackboard.

Using JSF and JDBC to Access a MySQL Database

Hobbies

Create

Hobby to Create

Read

Hobby ID	Name	Description	Date
1	XBox One	Multi-Player, Big Screen	16-May-2014
2	Software Dev	Many Hours of Fun!	16-May-2014
3	Cruising	Life at Sea	16-May-2014

Update

Hobby Info to Update

Delete

Hobby ID to Delete