University of Central Lancashire
UCLan

CO3409 Distributed Enterprise Systems

Lab : Database Application: JDBC & Transactions

Summary

Create a JSF application that connects to a database through a JDBC programming interface. Use it to check the effect of transactions.

Purpose

On completion of this lab you should be able to:

- Use NetBeans to create a database application
- Access a database through JDBC
- Use transactions

Things to Remember

Some things to remember...

- Run Netbeans as "Administrator".
- When running your application, ensure your database is running first (see previous lab worksheets).

Activities

Some of these activities may seem like you have done them before or something similar, You have. Some of these tasks are repetitive but necessary - I'm sorry.

1. Creating a Database using Netbeans IDE

Create a new database by following the instructions below.

1. Run NetBeans.
2. Switch to the Services Pane.
3. Expand the Databases branch.
4. Right-click on the JavaDB leaf.
5. Select Create Database [If this is not available, Right-click on the JavaDB leaf and select properties. Make sure the location of the Java DB Installation is correct].

6. Enter the database name, `**customer**`, and a user and a password. (use `APP` and `APP`) **Don't forget these**.
7. When you click OK, a separate database server application will be started and will create the database for you. When the database has been created, a connection node will appear in the Databases branch.

2. Creating a Table using Netbeans IDE

Right-click on the customer connection node and choose Connect. Expand the node. A list of schemas (starting with APP) will be displayed. Unless you used a username of APP, you will need to create a new schema. If you used APP, go straight to **2. Creating a table**.

1. **Creating a Schema**

   Right-click on the connection and select Execute Command. In the SQL command tab that appears in the Editor window, add the command:

   ```
   CREATE SCHEMA <schema name>
   ```

   Note: the `<schema name>` should be the same as your `<user name>`. For example, I used `DGC` and `DGC` for the schema name and my user name.

2. **Creating a table**

   Once the connection has been made, expand the database and right-click on the Tables node. Select Create Table. Enter the table name, **customer**, and add the following fields (columns):

Name	Type	Size	Additional
name	`VARCHAR`	50	☑ Primary Key
address	`VARCHAR`	50	

   Notes:

   - ☑ indicates that a check box should be ticked
   - Note: `VARCHAR` is not the same as `VARCHAR FOR BIT DATA`
   - Right-click on the connection and disconnect. This is important if you are going to use an embedded Database Manager.

3. Creating a Web Application to access the database

1. **User Interface Design**

   Create a new Java Web Application project called `transdbjsf`. Set the server to Payara and use Java EE 7 web. Select the Java Server Faces framework. [For advanced components, we could select Primefaces in the components tab, but not today]. Wait for the Server Library and Registered Libraries to finish searching. Select one of these with a JSF library e.g. JSF 2.2, then click **Finish**.

2. **The View**

1. Replace the contents of `index.xhtml` with the contents from ***Appendix 1***.

2. Create a new XHTML file, `edit.xhtml`. Using `index.xhtml` as an example, modify this to display a name and address, have a place to display error messages and a submit button that will call the `saveAddress` method of the `userData` object. The page should also display the `autocommit` status.

3. **The Model**

    1. Create a Java class called `Address` in package jdbc. Replace the contents with:

    ```
    package jdbc;

    public class Address {
       private String name;
       private String address;

       public Address (String name, String address) {
          this.name = name;
          this.address = address;
       }

    }
    ```

    Create Getter and Setter methods for the name and address. You can do this manually or automatically by using the insert code feature.

4. **The Backing Bean (Named Bean)**

    1. Right-click on Source Packages and create a Java class called `UserData` in package jdbc and replace the contents of the file with the contents from ***Appendix 2***.

    2. Import appropriate packages. Use the light bulb symbol that appears to the left of the code. (The appropriate classes are likely to be in `java.sql` package. You will be using context and dependency **inject**ion. Try not to get the package wrong if there is a choice.)

    3. Add a `string` variable called `message`.

    4. Right-click in the class, **Insert Code**, add **Getter and Setter** for address, name message and autoOn.

    5. Replace any `System.out.println` to use logging.

    6. Add statements to set the message variable to allow the display of message on the page.

    7. Modify `index.xhtml` to display the messages using:

    ```
    <h:outputText value = "#{userData.message}" />
    ```

5. **Style Sheet (CSS)**
    1. Create a new folder called `resources` in the `Web Pages` Folder. Create a new folder called `css` inside resources. In css, create a new Cascading Style Sheet (CSS) file called `styles.css`.

    2. Add the styles from ***Appendix 3***.

6. **Ensuring the Faces Servlet Processes the Facelets Pages.**

    Right-click on the project name and select **Properties**, **Run**. Set **the Relative URL** (the start page) to `/faces/index.xhtml`.

7. **Testing the program**
    1. On the Services tab, right-click on the Java DB tab and start the server.

    2. Run the application. Common errors are syntax errors and forgetting to start the database server.

    3. Test transactions by running two browser tabs (or two browsers) simultaneously. Turn autocommit off, records now are added as part of a transaction, which will only be completed when the Commit or Abort button is clicked. Explore whether the second copy can see the state of the database when the first copy is adding records.

    4. What happens when you commit a transaction?

    5. What happens when you abort a transaction?

    6. What happens if both copies are adding records (with/without transactions)?

Questions

Answer the following questions, remember to keep them for your own records (they will come in handy for revision).

1. What is the purpose of a connection?
2. How is the connection managed when the page is reloaded?
3. Where is the user name and password provided to the database?
4. Compare the URL used in the program with the URL displayed in the Netbeans connection in the Databases folder in the Services tab.
5. How are records added to the database? What language is used?
6. What JSF feature is used to display the table? How is the table header set?
7. Why are alternate rows of the table different colours?
8. How could you change the large green text's colour and size?
9. The backing bean is "SessionScoped". What does this mean?
10. What happens if you replace `@Named("userData")` with `@Named`?

11. What does `<h:panelGrid columns="2">` do?
12. Explain what happens when a command button is clicked?
13. Explain how `AutoCommit` is managed by the program?
14. What does `AutoCommit` do?
15. How does the application commit or abort the transaction?
16. What is the default behaviour (i.e. when autocommit is on)?
17. What is a transaction?
18. Explain how the program works.
19. What happens if program is closed whilst a transaction is in progress? (Hint: it's not very good)
20. What is ACID?
21. What happens if you use an apostrophe in the name or address. (Hint: it's not very good)
22. What security problem would this application be vulnerable to?
23. How could this be fixed? You will probably need to look this up.

Appendices

Appendix 1

```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml"
  xmlns:ui="http://xmlns.jcp.org/jsf/facelets"
  xmlns:h="http://xmlns.jcp.org/jsf/html"
  xmlns:f="http://xmlns.jcp.org/jsf/core">

  <h:head>
    <title>JSF tutorial 1</title>
    <h:outputStylesheet library="css" name="styles.css" />
  </h:head>

  <h:body>
    <h2>DataTable Example</h2>
    <h:form>
      <h:dataTable value="#{userData.addresses}"
         var="address"
         styleClass="addressTable"
         headerClass="addressTableHeader"
         rowClasses="addressTableOddRow,addressTableEvenRow">

         <h:column>
           <f:facet name="header">Name</f:facet>
           #{address.name}
         </h:column>

         <h:column>
            <f:facet name="header">Address</f:facet>
              #{address.address}
         </h:column>

         <h:column>
           <f:facet name="header">Edit</f:facet>
           <h:commandButton value="edit"
```

```
            action="#{userData.editAddress(address)}" />
          <h:commandButton value="delete"
            action="#{userData.deleteAddress(address)}"/>
        </h:column>

      </h:dataTable>
      <br/>
      <h:commandButton value="display"
         action="#{userData.doNothing()}"/>
       </h:form>
       <br/>
       <h:form styleClass="addressForm">

         <h:panelGrid columns="2">

         Name: <h:inputText id="nameInput"
                value="#{userData.name}"
                label="name min 2 max 8">
                <f:validateLength minimum="2" maximum="100" />
               </h:inputText>
         Address: <h:inputText id="addressInput"
                    value="#{userData.address}">
                  </h:inputText>
         <h:commandButton value="submit"
           action="#{userData.addAddress()}" />
         <h:message for="nameInput" style="color:red" />
         <h:message for="addressInput" style="color:red" />
         </h:panelGrid>
      </h:form>
      <h:form styleClass="addressForm">
          <h:panelGrid columns="2">
            Autocommit: <h:outputText
                        value="#{userData.autoOn}" />
            <br/>
            <h:commandButton value="Toggle autocommit"
              action="#{userData.autoCommit()}"/>

            <h:commandButton value="Commit"
              action="#{userData.commit()}"/>
            <h:commandButton value="Roll back"
              action="#{userData.rollBack()}"/>
          </h:panelGrid>
      </h:form>
  </h:body>
</html>
```

Appendix 2

```
package jdbc;

@Named
@SessionScoped
public class UserData implements Serializable {

    private static final long serialVersionUID = 1L;
    private String name;
    private String address;
```

```
    private String oldName;
    private String oldAddress;

    private Connection conn;
    private Statement stmt;
    private boolean autoOn = true;

    private static final ArrayList<Address> addresses = new ArrayList<Address>
();

    public ArrayList<Address> getAddresses() {
        try {
            addresses.clear();
            setUpStatement();
            if (stmt != null) {
                String query = "Select * from customer";

                ResultSet rs = stmt.executeQuery(query);

                while (rs.next()) {
                    Address record = new Address(rs.getString("Name"),
rs.getString("Address"));
                    addresses.add(record);
                }
            }
        } catch (ClassNotFoundException | SQLException ex) {
            Logger.getLogger(UserData.class.getName()).log(Level.SEVERE, null,
ex);
        }
        return addresses;
    }

    public void autoCommit() throws SQLException {
        try {
            setUpConnection();
            conn.setAutoCommit(!autoOn);
            autoOn = !autoOn;

        } catch (ClassNotFoundException ex) {
            Logger.getLogger(UserData.class.getName()).log(Level.SEVERE, null,
ex);
        }
    }

    public String addAddress() {
        Logger.getLogger(UserData.class.getName()).log(Level.SEVERE,
                        null, "addAddress name: " + getName() + " address: " +
getAddress());
        try {
            setUpStatement();
            if (stmt != null) {
                String query = "INSERT INTO customer (name,address) VALUES ('" +
name + "','" + address + "')";

                int updates = stmt.executeUpdate(query);
                Logger.getLogger(UserData.class.getName()).log(Level.SEVERE,
                        null, "updates: " + updates);
            }
        } catch (ClassNotFoundException | SQLException ex) {
```

```
            Logger.getLogger(UserData.class.getName()).log(Level.SEVERE, null,
ex);
        }
        return null;
    }

    private void setUpStatement() throws ClassNotFoundException, SQLException {
        setUpConnection();
        if (conn != null) {
            stmt = conn.createStatement();
        }
    }

    private void setUpConnection() throws ClassNotFoundException, SQLException {
        if (conn == null) {
            Class.forName("org.apache.derby.jdbc.ClientDriver");

            conn = DriverManager.getConnection(
                    "jdbc:derby://localhost:1527/customer;create=true", "APP",
"APP");
            if (conn != null) {
                conn.setAutoCommit(autoOn);
            }
        }
    }

    public String deleteAddress(Address delAddress) {
        try {
            setUpStatement();
            if (stmt != null) {

                String query = "delete from customer where name = '"
                        + delAddress.getName() + "'";

                int updates = stmt.executeUpdate(query);
                Logger.getLogger(UserData.class.getName()).log(Level.SEVERE,
                        null, "deletions: " + updates);
            }
        } catch (ClassNotFoundException | SQLException ex) {
            Logger.getLogger(UserData.class.getName()).log(Level.SEVERE, null,
ex);
        }
        return null;
    }

    public String commit() {
        try {
            setUpConnection();
            if (conn != null) {
                try {
                    conn.commit();
                    conn.close();
                } finally {
                    conn = null;
                }
                Logger.getLogger(UserData.class.getName()).log(Level.SEVERE,
                        null, "Commit successful");
            }
        } catch (ClassNotFoundException | SQLException ex) {
            Logger.getLogger(UserData.class.getName()).log(Level.SEVERE, null,
ex);
```

```
        }

        return null;
    }

    public String rollBack() {
        try {
            setUpConnection();
            if (conn != null) {
                try {
                    conn.rollback();
                    conn.close();
                } finally {
                    conn = null;
                }
                Logger.getLogger(UserData.class.getName()).log(Level.SEVERE,
                        null, "Rollback successful");
            }
        } catch (ClassNotFoundException | SQLException ex) {
            Logger.getLogger(UserData.class.getName()).log(Level.SEVERE, null,
ex);
        }
        return null;
    }

    public String editAddress(Address currentAddress) {
        name = currentAddress.getName();
        oldName = name;
        address = currentAddress.getAddress();
        oldAddress = address;
        return "edit";
    }

    public String doNothing() {
        return null;
    }

    public String saveAddress() {
        Logger.getLogger(UserData.class.getName()).log(Level.SEVERE,
                        null, "addAddress name: " + getName() + " address: " +
getAddress());
        try {
            setUpStatement();
            if (stmt != null) {
                String query = "UPDATE customer SET name = '" + name
                        + "', address = '" + address
                        + "' WHERE name ='" + oldName
                        + "'AND address = '" + oldAddress + "'";

                int updates = stmt.executeUpdate(query);
                Logger.getLogger(UserData.class.getName()).log(Level.SEVERE,
                        null, "updates: " + updates);
            }
        } catch (ClassNotFoundException | SQLException ex) {
            Logger.getLogger(UserData.class.getName()).log(Level.SEVERE, null,
ex);
        }

        return "index";
    }
}
```

Appendix 3

```
@charset "ISO-8859-1";

.addressForm {
   color:green;
   font-size:x-large;
}

.addressTable {
   border-collapse:collapse;
   border:1px solid #000000;
}

.addressTableHeader {
   text-align:center;
   background:none repeat scroll 0 0 #B5B5B5;
   border-bottom:1px solid #000000;
   padding:2px;
}

.addressTableOddRow {
   text-align:center;
   background:none repeat scroll 0 0 #FFFFFFF;
}

.addressTableEvenRow {
   text-align:center;
   background:none repeat scroll 0 0 #D3D3D3;
}
```