University of Central Lancashire
UCLan

CO3409 Distributed Enterprise Systems

Lab : Database Application: JDBC & Transactions (using SWING Controls)

Summary

Create an application using SWING controls 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. You will need to use the project found in the `tratransdb.zip` on Blackboard.

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. (e.g. `APP` and `APP`) Don't forget these. 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.

7. Right-click on the JavaDB leaf and stop the Server.

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:

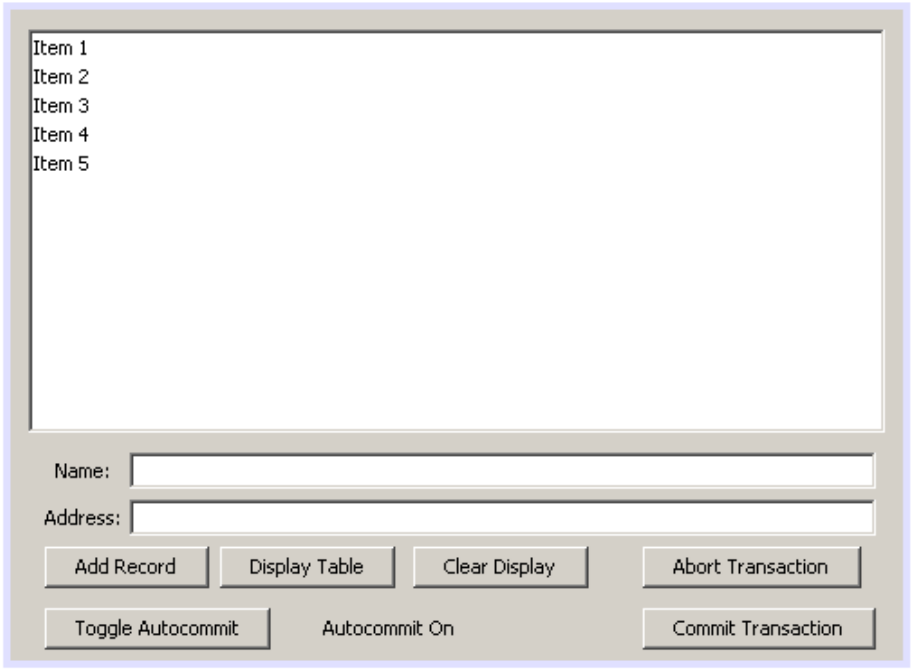
   - ☑ indicates that a check box should be ticked
   - Note: `VARCHAR` is not the same as `VARCHAR FOR BIT DATA`

- Right-click on the connection and disconnect. This is important if you are going to use an embedded Database Manager.

3. Creating an Application to access the database

1. **GUI Design**
    1. Create a new Java Application, called `transdb`. Uncheck create Main class. **Finish**.

    2. Add a `JFrame` form (right-click on the project in the Project pane, select New, Add a `JFrameForm` from Swing GUI Forms. Call the JFrame `GUIClass` in the transdb package and make it look like the following image below by dragging components from the palette and resizing them so they snap into position. See below for the components used. Don't forget to change their names.



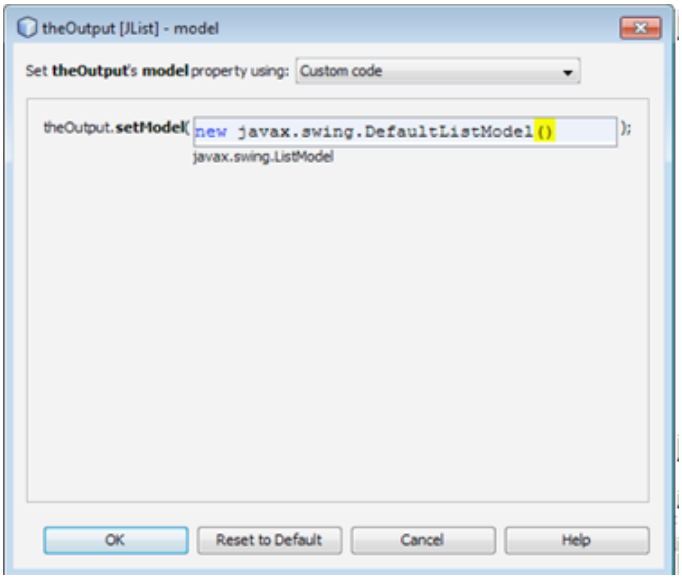
    The labels, text fields, and buttons are all on a **panel**. Change the text by right-clicking and choosing Edit Text.

    3. Rename the Text Field variables to `theName` and `theAddress` (Hint: right-click on the textfields and select **Change Variable Name**. Rename the list variable to `theOutput`. Rename the `autocommit` label to `autoCommit`. **Note**: the variable for each GUI component is the variable in the Java program that "points" to the actual object corresponding to a GUI component. The object itself has a name, which can be changed through the property inspector, but this name isn't very useful in the Java program.

    4. Select the list (theOutput) in the GUI design. Click on the dots next to the theOutput's **model** property in the properties pane and choose **set theOutput's model using: Custom code**.

    Set the model as:

```
new javax.swing.DefaultListModel()
```



5. Right-click on the project name and select **Properties**, **Libraries**. Add `derby.jar` and `derbyclient.jar` from the glassfish folder (e.g. `C:\ProgramFiles\glassfish-4.1.1\javadb\lib\derby.jar`). You may have to search for this. Use the **absolute** path. Close the Properties form.

6. In the following, it may be necessary to 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`. Try not to get them wrong if there is a choice.)

2. **Creating a connection property**

    1. Switch to the Source.
    2. Declare a connection and a statement property in the class, `GUIClass` :

    ```
    Connection conn = null;
    Statement stmt;
    ```

3. **Ensuring the database connection is closed before the program finishes**

    In the GUIClass design view, select the GUIClass JFrame. You can do this by clicking on a part of the form that isn't occupied by another component or by selecting it in the Navigator pane. Show the Events tab in the Properties Pane. Add the following code to the `windowClosing` Event.

    ```
    try {
        if (conn != null)
            conn.close();
    } catch (Exception ex) {
        ex.printStackTrace(System.err);
    }
    ```

4. **Creating a statement helper object**

    Add the following to the `GUIClass` constructor.

```
try {
    Class.forName("org.apache.derby.jdbc.ClientDriver");
    conn = DriverManager.getConnection
      ("jdbc:derby://localhost:1527/customer","UN","PW");
    stmt = conn.createStatement();
} catch (Exception ex) {
    ex.printStackTrace(System.err);
}
```

> **Note:**
>
> The driver name and the connection URL are obtained from the properties of the database (on the Services tab). **Replace UN by the user name and PW by the password for the database**.
>
> Double quotes which aren't understood by NetBeans. Replace them in the editor pane if necessary.

5. **Add Records to the Database**

   Add the following code to the Action event handler for the Add Record Button (See week 1's practical if you can't work out how to add an event handler):

```
String query = "INSERT INTO customer (name,address) VALUES ('" +
        theName.getText() + "','" +
        theAddress.getText() + "')";
try {
    int updates = stmt.executeUpdate(query);
} catch (SQLException ex) {
    ex.printStackTrace(System.err);
}
```

6. **Display the Table**

   Add the following event handler to the Display Table button:

```
try {
    ResultSet rs = stmt.executeQuery(query);
    DefaultListModel dm = (DefaultListModel) (theOutput.getModel());
    dm.clear();

    while (rs.next()) {
        String record = rs.getString(1) + " "
                      + rs.getString("Address");
        dm.addElement(record);
    }
} catch (SQLException ex) {
    ex.printStackTrace(System.err);
}
```

7. **Transactions**

   Add the following to the Toggle Autocommit button:

```
try {
    if (conn.getAutoCommit()) {
        conn.setAutoCommit(false);
        autoCommit.setText("AutoCommit Off");
    } else {
        conn.setAutoCommit(true);
        autoCommit.setText("AutoCommit On");
    }
} catch (SQLException ex) {
    ex.printStackTrace(System.err);
}
```

Add event handlers to the following.

`Abort Transaction` button:

```
try {
    conn.rollback();
} catch (SQLException ex) {
    ex.printStackTrace(System.err);
}
```

`Commit Transaction` button:

```
try {
    conn.commit();
} catch (SQLException ex) {
    ex.printStackTrace(System.err);
}
```

`Clear Display` button:

```
DefaultListModel dm = (DefaultListModel)(theOutput.getModel());
dm.clear;
```

4. 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 a transaction?
2. In your own words explain how the program works.
3. What happens if program is closed whilst a transaction is in progress?
4. What is the default behaviour (i.e. when autocommit is on)?