IT3030 Programming Applications and Frameworks

Practical 03.2 – Database handling in web applications

This practical sheet will guide you to add database handling features to the form and grid application you developed in the practical 2. You will learn the below.

- Configure the project to contain the dependencies to connect to MySQL DB.
- Write code to connect to the MySQL DB.
- Write code based on MVC to perform CRUD operations.

Section 1: Add the dependencies

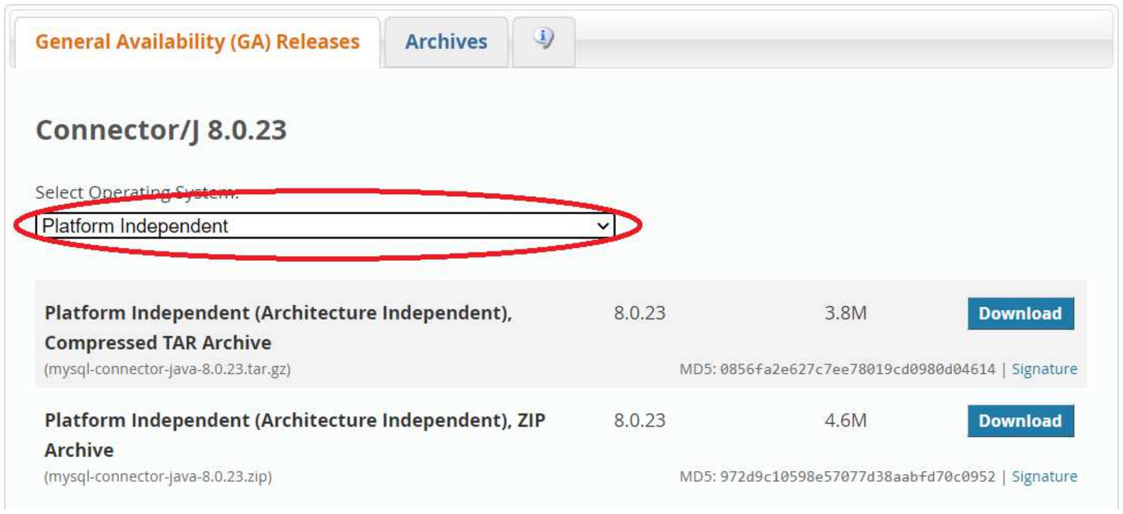
We need an external dependency name *MySQL Connector* to connect to the MySQL DB from JAVA.

NOTE: This week we will add this dependency manually. In coming weeks, we will learn how to do it using a dependency management tool.

Step 1.1: Download MySQL Connector

Go to the official web site and download the zip file.
https://dev.mysql.com/downloads/connector/j/5.1.html

Select *Platform Independent* and download the *ZIP archive*.

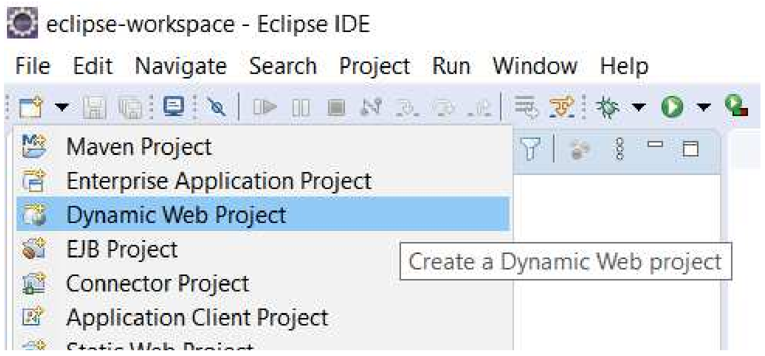


Platform Independent (Architecture Independent), Compressed TAR Archive (mysql-connector-java-8.0.23.tar.gz)	8.0.23	3.8M	Download
MD5: 0856fa2e627c7ee78019cd0980d04614 | Signature			
Platform Independent (Architecture Independent), ZIP Archive (mysql-connector-java-8.0.23.zip)	8.0.23	4.6M	Download
MD5: 972d9c10598e57077d38aabfd70c0952 | Signature			

Unzip the content and find the file with a name similar to *mysql-connector-java-x.x.xx-bin.jar.*
This file is the dependency you need to connect to MySQL using Java.

Step 1.2 Add the dependency to the project.

Create a new Dynamic Web Project using Eclipse.


eclipse-workspace - Eclipse IDE
File Edit Navigate Search Project Run Window Help
Maven Project
Enterprise Application Project
Dynamic Web Project
EJB Project
Connector Project
Application Client Project
Create a Dynamic Web project

Go to the *WebContent/WEB-INF/lib* folder inside the web project, using the file explorer.
NOTE: You can right click the folder in Eclipse and select *Show in -> System explorer*

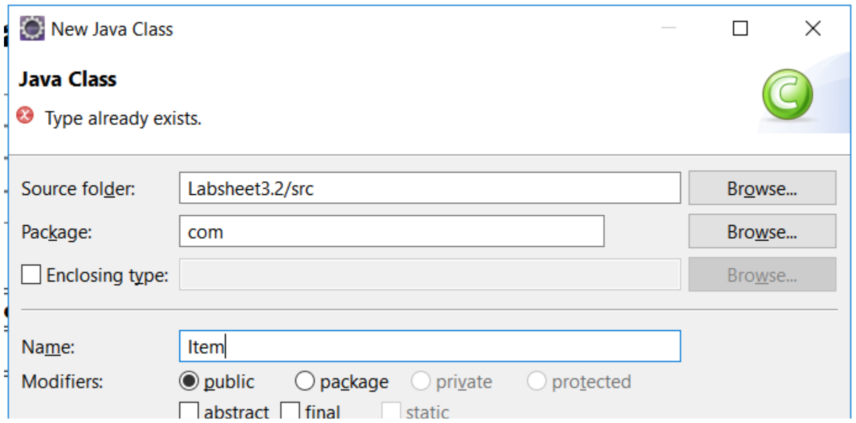
Copy the MySQL connector dependency into the lib directory.

Section 2: Connect to the MySQL DB

Step 2.1: Add a Model class

It is always good to keep the business logic (including DB handling) in separate dedicated classes, without including into the jsp pages.
Let's use a class name *Item* to write the business logic related to the items handling.

Add a new class to the project. Package: com and Name: Item.



New Java Class

Java Class

Type already exists.

Source folder: Labsheet3.2/src

Package: com

[ ] Enclosing type:

Name: Item

Modifiers: [✓ public] package private protected

[ ] abstract [ ] final [ ] static

Step 2.2: Create a Connect method

Let's code a method to contain the code to connect to the MySQL DB

Import the two classes *java.sql.Connection* and *java.sql.DriverManager*
NOTE: It's ok to import java.sql.* as we will need many classes in the sql package.

Create a method named *connect()*, which returns a *Connection*.

```
public Connection connect()
{
        Connection con = null;

         try
         {
              Class.forName("com.mysql.jdbc.Driver");
              con= DriverManager.getConnection("jdbc:mysql://127.0.0.1:3306/test",
                                                       "root", "");

              //For testing
              System.out.print("Successfully connected");
         }
         catch(Exception e)
         {
              e.printStackTrace();
         }

        return con;
}
```

Q: What are the three parameters passed to the method `getConnection()`?

Q: Where the name of the DB is specified?
Let's try testing this method and make sure that your code can connect to the database.

In the `if (request.getParameter("itemCode") != null)` section of the itemps jsp, create an object of the Item class and call the connect method.

```
if (request.getParameter("itemCode") != null)
{
	Item itemObj = new Item();
	itemObj.connect();//For testing the connect method
}
```

Run and test your application. If you get the `"Successfully connected"` message, the code can connect to the DB successfully.
Q: Will you sew the message when the page loads for the first time?
Q: Where will you see the `"Successfully connected" message?`
Q: If there are errors, what will be happened? Where will you see the error messages?

If there are errors, fix them before continue.

Section 3: Develop Insert operation

Let's try to develop the CRUD operations for the Item class.

Step 3.1 Create a method for inserting

Create a method named *insertItem*, which accepts parameters for code, name, price, and desc.

```
public String insertItem(String code, String name, String price, String desc)
```

Step 3.2: Connecting

Call the connect method and verify if connected properly.

```
Connection con = connect();

if (con == null)
{
	return "Error while connecting to the database";
}
```

Step 3.3 Prepare the SQL statement

We can run a plain SQL statement or use a special type of statements called "prepared statements".
Q: Why/when should we use prepared statements?

For this practical, let's use prepared statements.
We have to set the prepared statement and bind the values for it.

```
// create a prepared statement
String query = " insert into items
(`itemID`,`itemCode`,`itemName`,`itemPrice`,`itemDesc`)"
                                  + " values (?, ?, ?, ?, ?)";
PreparedStatement preparedStmt = con.prepareStatement(query);

// binding values
preparedStmt.setInt(1, 0);
preparedStmt.setString(2, code);
preparedStmt.setString(3, name);
preparedStmt.setDouble(4, Double.parseDouble(price));
preparedStmt.setString(5, desc);
```

Q: What are the variables used to bind the values?

Step 3.4 Execute the statement

Once the values are bound, we can execute the statement and close the connection.

```
//execute the statement
preparedStmt.execute();
con.close();
```

Q: Why do we close the connection?

Finally, we can show an output.

```
output = "Inserted successfully";
```

NOTE: It is good to write all these code inside a try block

The complete method will look like below.

```
public String insertItem(String code, String name, String price, String desc)
{
	String output = "";

	try
	{
		Connection con = connect();

		if (con == null)
		{
			return "Error while connecting to the database";
		}

		// create a prepared statement
		String query = " insert into items
	(`itemID`,`itemCode`,`itemName`,`itemPrice`,`itemDesc`)"
	+ " values (?, ?, ?, ?, ?)";

		PreparedStatement preparedStmt = con.prepareStatement(query);

		// binding values
		preparedStmt.setInt(1, 0);
		preparedStmt.setString(2, code);
		preparedStmt.setString(3, name);
		preparedStmt.setDouble(4, Double.parseDouble(price));
		preparedStmt.setString(5, desc);
```

```
            //execute the statement
            preparedStmt.execute();
            con.close();

            output = "Inserted successfully";
        }
        catch (Exception e)
        {
            output = "Error while inserting";
            System.err.println(e.getMessage());
        }

        return output;
}
```

Step 3.5 Test the feature

Add a web page named *items.jsp* to the project.

NOTE: Web pages should be added to the *WebContent* directory in the project.

Add a form to capture data to submit to the server

```
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Items Management</title>
</head>
<body>

        <h1>Items Management</h1>
        <form method="post" action="items.jsp">
            Item code: <input name="itemCode" type="text"><br>
            Item name: <input name="itemName" type="text"><br>
            Item price: <input name="itemPrice" type="text"><br>
            Item description: <input name="itemDesc" type="text"><br>
            <input name="btnSubmit" type="submit" value="Save">
        </form>

</body>
</html>
```

NOTE: the action of the form is set to the same page. We will add java code to process the post-back request to the same page.

Let’s add java code on the top of the page, before HTML.

Read the form data and call the *insertItem()* method by passing the proper parameter values.

Set the return of the *insertItem()* method into a session, so you can show the returned message on the page.

NOTE: You need a form to pass the parameters. See the final code in the last page for the HTML form.

```
if (request.getParameter("itemCode") != null)
{
	Item itemObj = new Item();
		String stsMsg = itemObj.insertItem(request.getParameter("itemCode"),
				request.getParameter("itemName"),
				request.getParameter("itemPrice"),
				request.getParameter("itemDesc"));

	session.setAttribute("statusMsg", stsMsg);
}
```

Q: Why do we need the if condition?

Show the status message on a proper place on the page (maybe below the form).

```
<% out.print(session.getAttribute("statusMsg")); %>
```

The complete code for the web page should look like below.

```
<%@page import="com.Item"%>
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
	pageEncoding="ISO-8859-1"%>

<%
	if (request.getParameter("itemCode") != null)
	{
		Item itemObj = new Item();

		String stsMsg = itemObj.insertItem(request.getParameter("itemCode"),
					request.getParameter("itemName"),
					request.getParameter("itemPrice"),
					request.getParameter("itemDesc"));

		session.setAttribute("statusMsg", stsMsg);
	}
%>

<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Items Management</title>
</head>
<body>

	<h1>Items Management</h1>
	<form method="post" action="items.jsp">
		Item code: <input name="itemCode" type="text"><br>
		Item name: <input name="itemName" type="text"><br>
		Item price: <input name="itemPrice" type="text"><br>
		Item description: <input name="itemDesc" type="text"><br>
		<input name="btnSubmit" type="submit" value="Save">
	</form>
	<%
		out.print(session.getAttribute("statusMsg"));
	%>
</body>
</html>
```

Section 4: Develop Read operation

Let's code a method to read the items in the DB and display on the page in a table

Step 4.1: Create the method

Add a method named *readItems()* into the Item class

```
public String readItems()
{
        String output = "";
        Return output;
}
```

Step 4.2: Connect to the database.

Similar to the *insertItem()* method, connect to the database.

```
try
{
        Connection con = connect();
        if (con == null)
        {
                return "Error while connecting to the database for reading.";
        }
}
catch (Exception e)
{
        output = "Error while reading the items.";
        System.err.println(e.getMessage());
}
```

Step 4.3: Prepare the html table

Since we will display the items inside an html table, we need to prepare the html for the table. Let's assign the elements of the html table and the header row into the *output* variable.

```
// Prepare the html table to be displayed
output = "<table border='1'>"
+ "<tr><th>Item Code</th><th>Item Name</th><th>Item Price</th>"
+ "<th>Item Description</th><th>Update</th><th>Remove</th></tr>";
```

Step 4.4: Read from the DB and append the rows to the table

We need an SQL statement to read the items from the DB. The values read from the DB should be assigned to a `ResultSet` variable.

We will use a loop to fetch row by row from the result set and read the cell/column values in the rows.

```
String query = "select * from items";
Statement stmt = con.createStatement();
ResultSet rs = stmt.executeQuery(query);

// iterate through the rows in the result set
while (rs.next())
{
        String itemID = Integer.toString(rs.getInt("itemID"));
        String itemCode = rs.getString("itemCode");
        String itemName = rs.getString("itemName");
        String itemPrice = Double.toString(rs.getDouble("itemPrice"));
        String itemDesc = rs.getString("itemDesc");

        // Add a row into the html table
        output += "<tr><td>" + itemCode + "</td>";
        output += "<td>" + itemName + "</td>";
        output += "<td>" + itemPrice + "</td>";
        output += "<td>" + itemDesc + "</td>";

        // buttons
        output += "<td><input name='btnUpdate' type='button' value='Update'></td>"
               + "<td><form method='post' action='items.jsp'>"
               + "<input name='btnRemove' type='submit' value='Remove'>"
               + "<input name='itemID' type='hidden' value='" + itemID + "'>"
               + "</form></td></tr>";
}
```

NOTE: The *remove* button is wrapped inside a *form* element and the type of the button is changed to a *submit* button.
NOTE: Inside the form in the remove column, a new hidden element is included, which holds the itemID. We will use this ID to remove the item.

The complete *readItems()* method is given below.

```
public String readItems()
{
        String output = "";

        try
        {
                Connection con = connect();

                if (con == null)
                {
                        return "Error while connecting to the database for reading.";
                }

                // Prepare the html table to be displayed
                output = "<table border='1'><tr><th>Item Code</th>"
                        +"<th>Item Name</th><th>Item Price</th>"
                        + "<th>Item Description</th>"
                        + "<th>Update</th><th>Remove</th></tr>";

                String query = "select * from items";
                Statement stmt = con.createStatement();
                ResultSet rs = stmt.executeQuery(query);

                // iterate through the rows in the result set
                while (rs.next())
                {
                        String itemID = Integer.toString(rs.getInt("itemID"));
                        String itemCode = rs.getString("itemCode");
                        String itemName = rs.getString("itemName");
                        String itemPrice = Double.toString(rs.getDouble("itemPrice"));
                        String itemDesc = rs.getString("itemDesc");

                        // Add a row into the html table
                        output += "<tr><td>" + itemCode + "</td>";
                        output += "<td>" + itemName + "</td>";
                        output += "<td>" + itemPrice + "</td>";
```

```
                    output += "<td>" + itemDesc + "</td>";

                    // buttons
                    output += "<td><input name='btnUpdate' "
                          + " type='button' value='Update'></td>"
                          + "<td><form method='post' action='items.jsp'>"
                          + "<input name='btnRemove' "
                          + " type='submit' value='Remove'>"
                          + "<input name='itemID' type='hidden' "
                          + " value='" + itemID + "'>" + "</form></td></tr>";
              }

              con.close();

              // Complete the html table
              output += "</table>";
       }
       catch (Exception e)
       {
              output = "Error while reading the items.";
              System.err.println(e.getMessage());
       }

       return output;
}
```

Step: 4.5 Show the output on the page

Add java code to create an *Item* object and call the *readItems()* method, the show the output on the page.

```
<%
       Item itemObj = new Item();
       out.print(itemObj.readItems());
%>
```

Find the updated version of the *items.jsp* below.

```
<%@ page import="com.Item"%>
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
       pageEncoding="ISO-8859-1"%>

<%
       //Insert item----------------------------------
       if (request.getParameter("itemCode") != null)
       {
              Item itemObj = new Item();

              String stsMsg = itemObj.insertItem(request.getParameter("itemCode"),
                          request.getParameter("itemName"),
                          request.getParameter("itemPrice"),
                          request.getParameter("itemDesc"));

              session.setAttribute("statusMsg", stsMsg);
       }
%>
```

```
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Items Management</title>
</head>
<body>

      <h1>Items Management</h1>
      <form method="post" action="items.jsp">
            Item code: <input name="itemCode" type="text"><br> Item
            name: <input name="itemName" type="text"><br> Item price:
            <input name="itemPrice" type="text"><br> Item
            description: <input name="itemDesc" type="text"><br> <input
                  name="btnSubmit" type="submit" value="Save">
      </form>
      <%
            out.print(session.getAttribute("statusMsg"));
      %>
      <br>
      <%
            Item itemObj = new Item();
            out.print(itemObj.readItems());
      %>
</body>
</html>
```

Try inserting multiple items and make sure you can see the inserted items on the page, inside the table.

Q: What happens if you enter a text value to the price field?

NOTE: Try to develop the Delete and Update features.

End of the sheet