3.3 Stored Procedures and Exception Handling

This section will guide you to:

- Set up Eclipse to work with JDBC
- Create a database and a table in MySQL
- Create a stored procedure in MySQL
- Create an HTML page to call a servlet
- Create a servlet that calls the stored procedure using JDBC

Development Environment

- Eclipse IDE for Enterprise Java Developers v2019-03 (4.11.0)
- Apache Tomcat Server v9.0
- JRE: OpenJDK Runtime Environment 11.0.2
- MySQL Connector for Java 8.0.16

This guide has fourteen subsections, namely:

3.3.13 Running the project

3.3.14 Pushing the code to your GitHub repositories

Step 3.3.1: Creating a database in MySQL and creating a table in it

- MySQL is already installed in your practice lab. (Refer QA to QE: Lab Guide - Phase 1)
- Log in to the MySQL command line console
- Type **CREATE DATABASE ecommerce** and press **Enter**
- Type **USE ecommerce** and press **Enter**
- Type **CREATE TABLE eproduct (ID bigint primary key auto_increment, name varchar(100), price decimal(10,2), date_added timestamp default now())** and press **Enter**
- We will now add some rows into the table
- Type **INSERT INTO eproduct(name, 'HP Laptop ABC', 12000)** and press **Enter**
- Type **INSERT INTO eproduct(name, 'Acer Laptop ABC', 14000)** and press **Enter**
- Type **INSERT INTO eproduct(name, 'Lenovo Laptop ABC', 12000)** and press **Enter**
- Type **SELECT * from eproduct** and press **Enter** to confirm that the rows have been added
- Type **EXIT** to exit the MySQL command console

Step 3.3.2: Creating a stored procedure add_product in MySQL

- Log in to the MySQL command line console
- Type the following script:

```
DELIMITER $$
CREATE PROCEDURE add_product(IN pname varchar(100), IN pprice decimal(10,2))
INSERT INTO eproduct (name, price) VALUES (pname, pprice)
$$
DELIMITER ;
```

Step 3.3.3: Creating a dynamic web project

- Open Eclipse
- Go to the **File** menu. Choose **New->Dynamic Web Project**
- Enter the project name as **JDBCSetup**. Click on **Next**
- Enter nothing in the next screen and click on **Next**

- Check the checkbox **Generate web.xml deployment descriptor** and click on **Finish**
- This will create the project files in the Project Explorer

Step 3.3.4: Adding the jar files for MySQL connection for Java

- mysql-connector-java.jar is already present in your lab. (Refer FSD: Lab Guide - Phase 1)
- Take mysql-connector-java.jar file from the folder mentioned in the lab guide for phase 1 and add it to the project's WebContent/WEB-INF/lib folder

Step 3.3.5: Creating an HTML page index.html

- In the Project Explorer, expand the project **JDBCSetup**
- Expand **WebContent**. Right click on **WebContent**. Choose **New->HTML File**
- Enter the filename as index.html and click on **Finish**
- Enter the following code:

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>JDBC Stored Procedures</title>
</head>
<body>
<a href="list">Product Info</a><br>

</body>
</html>
```

- Click on the **Save** icon

Step 3.3.6: Creating a DBConnection class to initiate a JDBC connection in code

- In the Project Explorer, expand **JDBCSetup->Java Resources**
- Right click on **src** and choose **New->Class**
- In **Package,** enter **com.ecommerce** and in **Name** enter **DBConnection** and click on **Finish**
- Enter the following code:

```
package com.ecommerce;


```

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

public class DBConnection {

	private Connection connection;

	public DBConnection(String dbURL, String user, String pwd) throws ClassNotFoundException,
SQLException{

		Class.forName("com.mysql.jdbc.Driver");
		this.connection = DriverManager.getConnection(dbURL, user, pwd);
	}

	public Connection getConnection(){
		return this.connection;
	}

	public void closeConnection() throws SQLException {
		if (this.connection != null)
			this.connection.close();
	}
}
```

Step 3.3.7: Creating a config.properties file to store JDBC credentials

- In the Project Explorer, expand the project **JDBCSetup**
- Expand **WebContent**. Right click on **WebContent**. Choose **New->File**
- Enter the filename as config.properties and click on **Finish**
- Enter the following data:

```
url=jdbc:mysql://localhost:3306/ecommerce
userid=root
password=master
```

Step 3.3.8: Creating a ProductDetails servlet

- In the Project Explorer, expand **JDBCSetup->Java Resources**
- Right click on **src** and choose **New->Servlet**
- In **Class Name,** enter **ProductDetails** and click on **Finish**
- Enter the following code:

```
import java.io.IOException;
import java.io.InputStream;
import java.io.PrintWriter;
import java.math.BigDecimal;
```

```java
import java.sql.CallableStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.Properties;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import com.ecommerce.DBConnection;

/**
* Servlet implementation class ProductDetails
*/
@WebServlet("/ProductDetails")
public class ProductDetails extends HttpServlet {
	private static final long serialVersionUID = 1L;

    /**
* @see HttpServlet#HttpServlet()
*/
    public ProductDetails() {
        super();
        // TODO Auto-generated constructor stub
    }

	/**
	 * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)
	 */
	protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
		// TODO Auto-generated method stub

		try {
			 PrintWriter out = response.getWriter();
			 out.println("<html><body>");

			InputStream in = getServletContext().getResourceAsStream("/WEB-INF/config.properties");
			Properties props = new Properties();
			props.load(in);


			DBConnection conn = new DBConnection(props.getProperty("url"), props.getProperty("userid"),
props.getProperty("password"));
			CallableStatement stmt = conn.getConnection().prepareCall("{call add_product(?, ?)}");
			stmt.setString(1, "new product");
			stmt.setBigDecimal(2, new BigDecimal(1900.50));
			stmt.executeUpdate();

			out.println("Stored procedure has been executed.<Br>");
			stmt.close();
```

```
                out.println("</body></html>");
                conn.closeConnection();

            } catch (ClassNotFoundException e) {
                e.printStackTrace();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }

        /**
         * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)
         */
        protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
            // TODO Auto-generated method stub
            doGet(request, response);
        }

}
```

Step 3.3.9: Configuring web.xml

- In the Project Explorer, expand **JDBCSetup->WebContent->WEB-INF**
- Double click on **web.xml** to open it in the editor
- Enter the following script:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://xmlns.jcp.org/xml/ns/javaee" xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
http://xmlns.jcp.org/xml/ns/javaee/web-app_4_0.xsd" id="WebApp_ID" version="4.0">
  <display-name>JDBC Stored Procedures</display-name>
  <welcome-file-list>
    <welcome-file>index.html</welcome-file>
    <welcome-file>index.htm</welcome-file>
    <welcome-file>index.jsp</welcome-file>
    <welcome-file>default.html</welcome-file>
    <welcome-file>default.htm</welcome-file>
    <welcome-file>default.jsp</welcome-file>
  </welcome-file-list>
  <servlet>
    <servlet-name>ProductDetails</servlet-name>
    <servlet-class>ProductDetails</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>ProductDetails</servlet-name>
    <url-pattern>/list</url-pattern>
  </servlet-mapping>

</web-app>
```

Step 3.3.10: Checking for servlet-api.jar

- Before building the project, we need to confirm that **servlet-api.jar** has been added to the project
- In the Project Explorer, right click on **JDBCSetup** and choose **Properties**
- Select **Java Build Path** from the options on the left
- Click on **Libraries** tab on the right
- Under **ClassPath,** expand the node that says **Apache Tomcat**
- If there is an existing entry for **servlet-api.jar,** then click on **Cancel** and exit the window
- If it is not there, then click on **Classpath** entry and click on **Add External JARs** button on the right
- From the **file** list, select **servlet-api.jar** file and click on **Ok**
- Click on **Apply and Close**

Step 3.3.11: Building the project

- From the **Project** menu at the top, click on **Build**
- If any compile errors are shown, fix them as required

Step 3.3.12: Publishing and starting the project

- If you do not see the **Servers** tab near the bottom of the IDE, go to **Window** menu and click on **Show View->Servers**
- Right click the **Server** entry and choose **Add and Remove**
- Click the **Add** button to move **JDBCSetup** from the **Available** list to the **Configured** list
- Click on **Finish**
- Right click the **Server** entry and click on **Publish**
- Right click the **Server** entry and click on **Start**
- This will start the server

Step 3.3.13: Running the project

- To run the project, open a web browser and type: **http://localhost:8080/JDBCSetup**

Step 3.3.14: Pushing the code to your GitHub repositories

Open your command prompt and navigate to the folder where you have created your files.
cd jdbc_Demo
Initialize your repository using the following command:
git init
Add all the files to your git repository using the following command:
git add .
Commit the changes using the following command:
git commit . -m “Changes have been committed.”
Push the files to the folder you initially created using the following command:
git push -u origin master