# 1. Set Up a JDBC Environment

**Step 1.1:** 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 1.2:** Adding the jar files for MySQL connection for Java

- **mysql-connector-java.jar** is already present in your lab. (Refer the QA to QE : 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 1.3:** 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 Setup</title>
</head>
<body>
<a href="init">Initialize JDBC</a><br>
```

```
</body>
</html>
```

- Click on the **Save** icon

**Step 1.4:** 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 1.5:** 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 1.6:** Creating a DemoJDBC servlet

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

```
import java.io.IOException;
import java.io.InputStream;
import java.io.PrintWriter;
import java.math.BigDecimal;
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 DemoJDBC
*/
@WebServlet("/DemoJDBC")
public class DemoJDBC extends HttpServlet {
	private static final long serialVersionUID = 1L;
```

```
   /**
* @see HttpServlet#HttpServlet()
*/
   public DemoJDBC() {
      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"));
                  out.println("DB Connection initialized.<br>");

                  conn.closeConnection();
                  out.println("DB Connection closed.<br>");


                  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 1.7:** 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>JDBCSetup</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>DemoJDBC</servlet-name>
    <servlet-class>DemoJDBC</servlet-class>
  </servlet>
  <servlet-mapping>
```

```
  <servlet-name>DemoJDBC</servlet-name>
  <url-pattern>/init</url-pattern>
 </servlet-mapping>

</web-app>
```

**Step 1.8:** 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 1.9:** Building the project

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

**Step 1.10:** 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

# 2. JDBC Connections, Statements, and ResultSets

**Step 2.1:** 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 to 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 2.2:** 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 2.3:** Adding the jar files for MySQL Connection for Java

- **mysql-connector-java.jar** is already present in your lab. To learn about its directory path details you can refer the **lab guide for 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 2.4:** 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 Statements and Resultsets</title>
</head>
<body>
<a href="list">Product Info</a><br>

</body>
</html>
```

- Click on the **Save** icon

**Step 2.5:** 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 2.6:** Creating a config.properties file to storeJDBC 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 2.7:** 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;
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"));
			Statement stmt = conn.getConnection().createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE, ResultSet.CONCUR_READ_ONLY);
			stmt.executeUpdate("insert into eproduct (name, price, date_added) values ('New Product', 17800.00, now())");
			ResultSet rst = stmt.executeQuery("select * from eproduct");

			while (rst.next()) {
				out.println(rst.getInt("ID") + ", " + rst.getString("name") + "<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 2.8:** 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 Statements and Resultsets</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 2.9:** 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 3.2.10:** Building the project

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

**Step 2.11:** Publishing and starting the project

- If you do not see the **Servers** tab near the bottom of the IDE, go to the **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

# 3. Stored Procedures and Exception Handling

**Step 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 3.2:** 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:** 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 3.4:** 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 3.5:** 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 3.6:** 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 3.7:** 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 3.8:** 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;
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 3.9:** 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 3.10:** 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 3.11:** Building the project

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

**Step 3.12:** 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

# 4. Create, Select, and Drop a Database

Step 4.1: 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 4.2: Adding the jar files for MySQL connection for Java

- mysql-connector-java.jar is already present in your lab. To learn about its directory path details you can refer the lab guide for 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 4.3: 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 Datbase Operations</title>
</head>
<body>
<a href="dboperations">Database Operations</a><br>

</body>
</html>
```

- Click on the Save icon

Step 4.4: 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 4.5: 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 4.6: Creating a DBOperations servlet

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

```
import java.io.IOException;
import java.io.InputStream;
import java.io.PrintWriter;
import java.math.BigDecimal;
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 DBOperations
*/
@WebServlet("/DBOperations")
public class DBOperations extends HttpServlet {
```

```
    private static final long serialVersionUID = 1L;


   /**
* @see HttpServlet#HttpServlet()
*/
   public DBOperations() {
      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"));
                Statement stmt = conn.getConnection().createStatement();
                stmt.executeUpdate("create database mydatabase");
                out.println("Created database: mydatabase<br>");
                stmt.executeUpdate("use mydatabase");
                out.println("Selected database: mydatabase<br>");
                stmt.executeUpdate("drop database mydatabase");
                stmt.close();
                out.println("Dropped database: mydatabase<br>");
               conn.closeConnection();
               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 4.7: 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 DB Operations</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>DBOperations</servlet-name>
    <servlet-class>DBOperations</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>DBOperations</servlet-name>
```

```
  <url-pattern>/dboperations</url-pattern>
 </servlet-mapping>

</web-app>
```

Step 4.8: 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 4.9: Building the project

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

Step 4.10: Publishing and starting the project

- If you do not see the Servers tab near the bottom of the IDE, go to the 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 on the Server entry and click on Publish
- Right click on the Server entry and click on Start
- This will start the server

# 5.Insertion, Updation, and Deletion of Database Records

---

Step 5.1: Creating a database in MySQL and 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 5.2: 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 5.3: Adding the jar files for MySQL connection for Java

- mysql-connector-java.jar is already present in your lab. To learn about its directory path details you can refer the lab guide for 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 5.4: 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 Insert, Update, Delete</title>
</head>
<body>
<a href="list">Product Info</a><br>

</body>
</html>
```

- Click on the Save icon

Step 5.5: 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 5.6: 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 5.7: 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;
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"));
            Statement stmt =
conn.getConnection().createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
ResultSet.CONCUR_READ_ONLY);
            stmt.executeUpdate("insert into eproduct (name, price, date_added) values
('New Product', 17800.00, now())");
            out.println("Executed an insert operation<br>");

            stmt.executeUpdate("update eproduct set price=2000 where name = 'New
Product'");
            out.println("Executed an update operation<br>");

            stmt.executeUpdate("delete from eproduct where name = 'New Product'");
            out.println("Executed a delete operation<br>");

            stmt.close();

            conn.closeConnection();



            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 5.8: 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 Insert, Update, Delete</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 5.9: 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 5.10: Building the project

- From the Project menu at the top, click on Build

- If any compile errors are shown, fix them as required

Step 5.11: Publishing and starting the project

- If you do not see the Servers tab near the bottom of the IDE, go to the 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

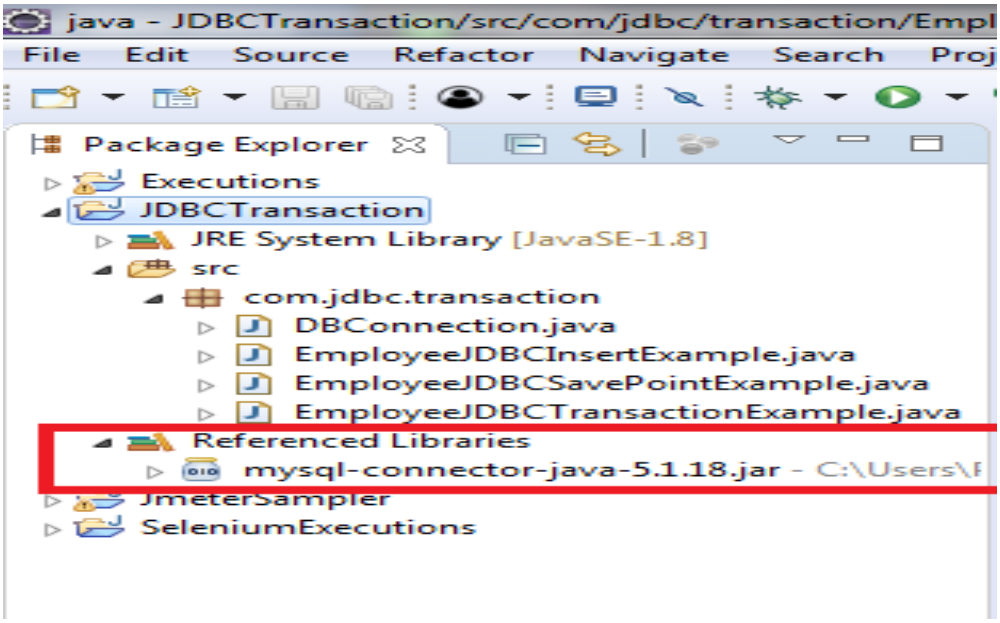
# 6 Transaction Management

**Step 6.1:** Writing a program to perform JDBC transaction management using Auto-Commit Mode.

- By default, when we create a database connection, it runs in **auto-commit** mode. It means that whenever we execute a query, the commit is fired automatically. So, every SQL query we fire is a transaction and if we are running DML or DDL queries, the changes are getting saved in the database after every SQL statement is executed .
- Sometimes we want a group of SQL queries to be part of a transaction, so that we can commit them when all the queries run successfully. If we get any exception, we have a choice to rollback all the queries executed as part of the transaction.
- Let's understand with a simple example where we want to utilize JDBC transaction management support for data integrity. Let's say we have "transaction_management" database and employee information saved in two tables. Example: I am using MySQL database.
- Create two tables 'employee' and 'address' in 'transaction_management' database using the credentials below:

```
CREATE TABLE transaction_management.employee (
  empId int(11) unsigned NOT NULL,
  name varchar(20) DEFAULT NULL,
  PRIMARY KEY (empId)) ENGINE=InnoDB DEFAULT CHARSET=utf8;

CREATE TABLE transaction_management.address (
  empId int(11) unsigned NOT NULL,
  address varchar(20) DEFAULT NULL,
  city varchar(5) DEFAULT NULL,
  country varchar(20) DEFAULT NULL,
  PRIMARY KEY (empId)) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

- Open Eclipse
- Create Java Project. Ex: JDBCTransaction
- Download "mysql-connector-java-5.1.18.jar"
- Add External jar "mysql-connector-java-5.1.18.jar" into the project




- Create a class called “DBConnection.java” and give the database credentials as below:
  - **DB_URL:** jdbc:mysql://localhost:3307/transaction_management
  - **DB_DRIVER_CLASS:** com.mysql.jdbc.Driver
  - **DB_USERNAME:** The username of database (here: **root)**
  - **DB_PASSWORD:** Password for the username (here: **root)**

```
package com.jdbc.transaction;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.SQLException;


public class DBConnection {


	public final static String DB_DRIVER_CLASS =
"com.mysql.jdbc.Driver";
	public final static String DB_URL =
"jdbc:mysql://localhost:3307/transaction_management";
	public final static String DB_USERNAME = "root";
	public final static String DB_PASSWORD = "root";
```

```
        public static Connection getConnection() throws
    ClassNotFoundException, SQLException {

                Connection con = null;

                // load the Driver Class
                Class.forName(DB_DRIVER_CLASS);

                // create the connection now
                con = DriverManager.getConnection(DB_URL,
     DB_USERNAME, DB_PASSWORD);

                System.out.println("DB Connection created
    successfully");
                return con;
        }
    }
```

- DBConnection is the class used by other classes for MYSQL database connection.
- Create another class called “EmployeeJDBCInsertExample.java”

```
    package com.jdbc.transaction;
    import java.sql.Connection;
    import java.sql.PreparedStatement;
    import java.sql.SQLException;

    public class EmployeeJDBCInsertExample {

        public static final String INSERT_EMPLOYEE_QUERY =
       "insert into Employee (empId, name) values (?,?)";
```

```
	public static final String INSERT_ADDRESS_QUERY = "insert into
Address (empId, address, city, country) values (?,?,?,?)";

	public static void main(String[] args) {

		Connection con = null;
		try {
			con = DBConnection.getConnection();

			insertEmployeeData(con, 1, "Pankaj");

			insertAddressData
(con, 1, "Albany Dr", "San Jose", "USA");
		} catch (SQLException | ClassNotFoundException e) {
			e.printStackTrace();
		} finally {

			try {
				if (con != null)
					con.close();
			} catch (SQLException e) {
				e.printStackTrace();
			}
		}
	}

	public static void insertAddressData(Connection con, int id,
String address, String city, String country) throws SQLException {
```

```
                PreparedStatement stmt =
con.prepareStatement(INSERT_ADDRESS_QUERY);
                stmt.setInt(1, id);
                stmt.setString(2, address);
                stmt.setString(3, city);
                stmt.setString(4, country);

                stmt.executeUpdate();

                System.out.println("Address Data inserted successfully
for ID=" + id);
                stmt.close();
        }

        public static void insertEmployeeData(Connection con, int id,
String name) throws SQLException {
                PreparedStatement stmt =
con.prepareStatement(INSERT_EMPLOYEE_QUERY);
                stmt.setInt(1, id);
                stmt.setString(2, name);

                stmt.executeUpdate();

                System.out.println("Employee Data inserted
successfully for ID=" + id);
                stmt.close();
        }
}
```

- By running the "EmployeeJDBCInsertExample.java" program, we will get the following output:

DB Connection created successfully

Employee Data inserted successfully for ID=1

com.mysql.jdbc.MysqlDataTruncation: Data truncation: Data too

long for column 'city' at row 1

at com.mysql.jdbc.MysqlIO.checkErrorPacket(MysqlIO.java:2939)

at com.mysql.jdbc.MysqlIO.sendCommand(MysqlIO.java:1623)

at com.mysql.jdbc.MysqlIO.sqlQueryDirect(MysqlIO.java:1715)

at com.mysql.jdbc.Connection.execSQL(Connection.java:3249)

at com.mysql.jdbc.PreparedStatement.executeInternal

(PreparedStatement.java:1268)

at com.mysql.jdbc.PreparedStatement.executeUpdate

(PreparedStatement.java:1541)

at com.mysql.jdbc.PreparedStatement.executeUpdate

(PreparedStatement.java:1455)

at com.mysql.jdbc.PreparedStatement.executeUpdate

(PreparedStatement.java:1440)

at com.journaldev.jdbc.transaction.EmployeeJDBCInsertExample.

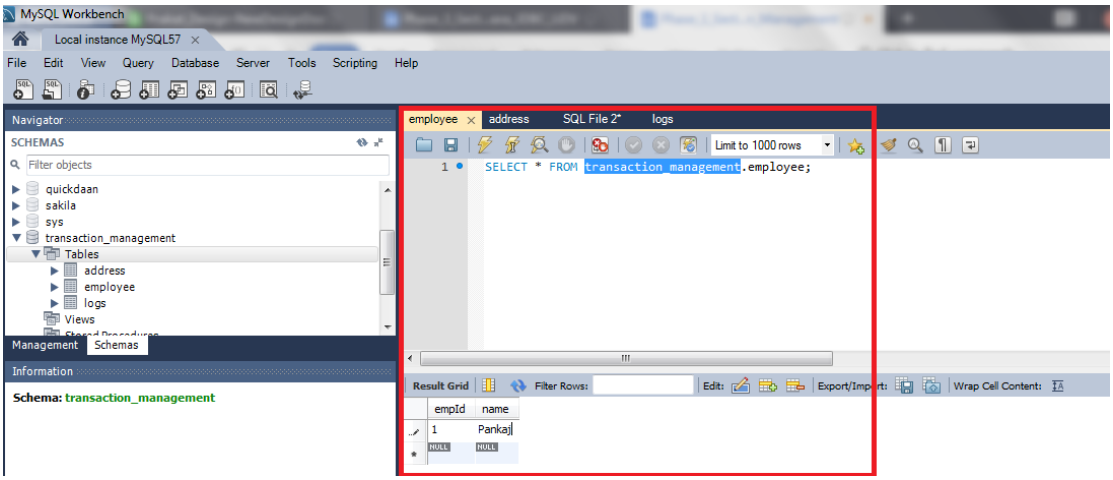
insertAddressData(EmployeeJDBCInsertExample.java:45)

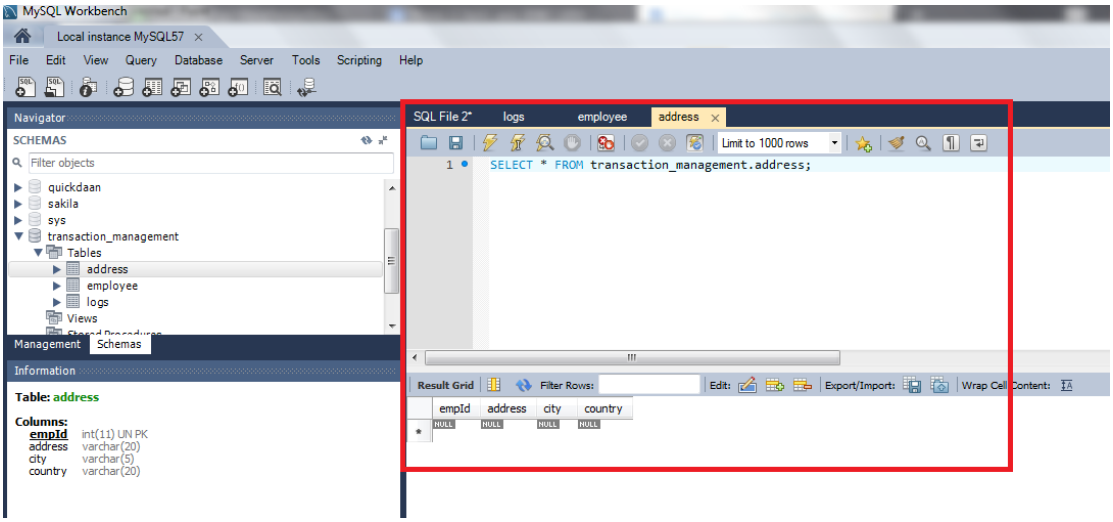
at com.journaldev.jdbc.transaction.EmployeeJDBCInsertExample.

main(EmployeeJDBCInsertExample.java:23)

- As you can see, SQLException is only raised when we are trying to insert data into the address table, because the value is bigger than the size of the column.

- If you look at the content in the employee and address tables, you will notice that data is present only in the employee table.




- By running the program again, it will try to insert employee information into the employee table again and will throw the below exception.

com.mysql.jdbc.exceptions.MySQLIntegrityConstraintViolationExceptio

n: Duplicate entry '1' for key 'PRIMARY'

at com.mysql.jdbc.SQLError.createSQLException

(SQLError.java:931)

at com.mysql.jdbc.MysqlIO.checkErrorPacket(MysqlIO.java:2941)

at com.mysql.jdbc.MysqlIO.sendCommand(MysqlIO.java:1623)

at com.mysql.jdbc.MysqlIO.sqlQueryDirect(MysqlIO.java:1715)

at com.mysql.jdbc.Connection.execSQL(Connection.java:3249)

at com.mysql.jdbc.PreparedStatement.executeInternal

(PreparedStatement.java:1268)

at com.mysql.jdbc.PreparedStatement.executeUpdate

(PreparedStatement.java:1541)

at com.mysql.jdbc.PreparedStatement.executeUpdate

(PreparedStatement.java:1455)

at com.mysql.jdbc.PreparedStatement.executeUpdate

(PreparedStatement.java:1440)

at com.journaldev.jdbc.transaction.EmployeeJDBCInsertExample.

insertEmployeeData(EmployeeJDBCInsertExample.java:57)

at com.journaldev.jdbc.transaction.EmployeeJDBCInsertExample.

main(EmployeeJDBCInsertExample.java:21

- Now, there is no way we can save the data in the address table for the Employee. Since this program leads to data integrity issues, we need transaction management to insert data into both the tables successfully or rollback everything if any exception arises.

**Step 6.2:** Writing a program to perform JDBC transaction management by disabling setAutoCommit().

- JDBC API provides the method setAutoCommit() through which we can disable the auto commit feature of the connection (should disable when it's required because the transaction will not be committed unless we call the commit() method on connection).
- Let's write another program where we will use JDBC transaction management feature to make sure data integrity is not violated.

```
package com.jdbc.transaction;


import java.sql.Connection;

import java.sql.SQLException;

public class EmployeeJDBCTransactionExample {


        public static void main(String[] args) {
```

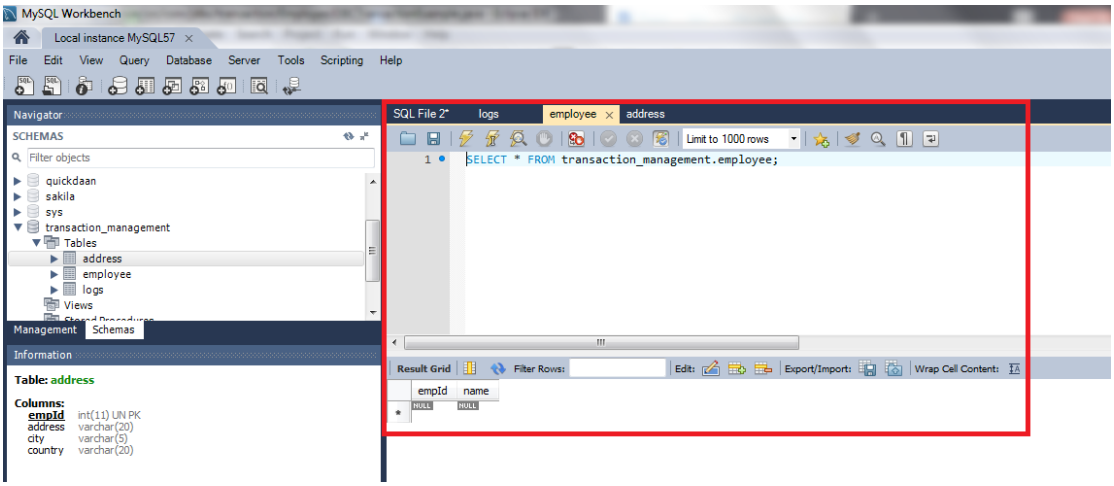
```
		Connection con = null;
		try {
			con = DBConnection.getConnection();

			//set auto commit to false
			con.setAutoCommit(false);

		EmployeeJDBCInsertExample.insertEmployee
Data(con, 1, "Pankaj");
EmployeeJDBCInsertExample.insertAddress
Data(con, 1, "Albany Dr", "San Jose", "USA");

			 //now commit transaction
			 con.commit();

		} catch (SQLException e) {
			e.printStackTrace();
			try {
				con.rollback();
				System.out.println("JDBC
Transaction rolled back successfully");
			} catch (SQLException e1) {
				System.out.println("SQLException in
rollback"+e.getMessage());
			}
		} catch (ClassNotFoundException e) {
			e.printStackTrace();
		} finally {
			try {
				if (con != null)
```

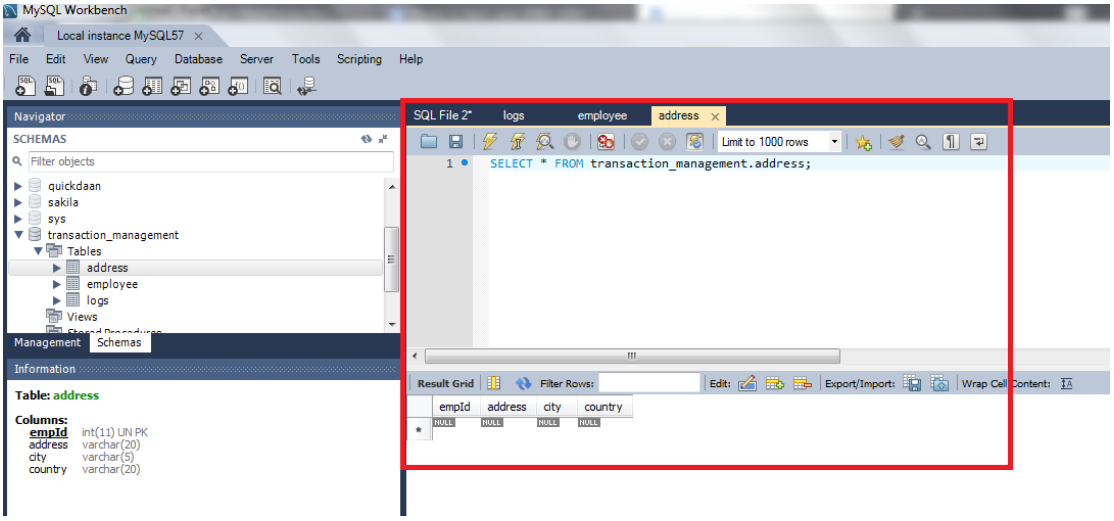
```
                    con.close();
                } catch (SQLException e) {
                    e.printStackTrace();
                }
            }
        }
    }
```

- Please make sure you remove the earlier inserted data from both the tables before running this program. By running this program, you will get the following output:

```
DB Connection created successfully
Employee Data inserted successfully for ID=1
com.mysql.jdbc.MysqlDataTruncation: Data truncation: Data too long
for column 'city' at row 1
    at com.mysql.jdbc.MysqlIO.checkErrorPacket(MysqlIO.java:2939)
    at com.mysql.jdbc.MysqlIO.sendCommand(MysqlIO.java:1623)
    at com.mysql.jdbc.MysqlIO.sqlQueryDirect(MysqlIO.java:1715)
    at com.mysql.jdbc.Connection.execSQL(Connection.java:3249)
    at com.mysql.jdbc.PreparedStatement.executeInternal
(PreparedStatement.java:1268)
    at com.mysql.jdbc.PreparedStatement.executeUpdate
(PreparedStatement.java:1541)
    at com.mysql.jdbc.PreparedStatement.executeUpdate
(PreparedStatement.java:1455)
    at com.mysql.jdbc.PreparedStatement.executeUpdate
(PreparedStatement.java:1440)
    at com.journaldev.jdbc.transaction.EmployeeJDBCInsertExample.
insertAddressData(EmployeeJDBCInsertExample.java:45)
    at com.journaldev.jdbc.transaction.EmployeeJDBCTransaction
Example.main(EmployeeJDBCTransactionExample.java:19)
JDBC Transaction rolled back successfully
```

- If you look into the database tables, you will notice that data is not inserted into both employee and address table.



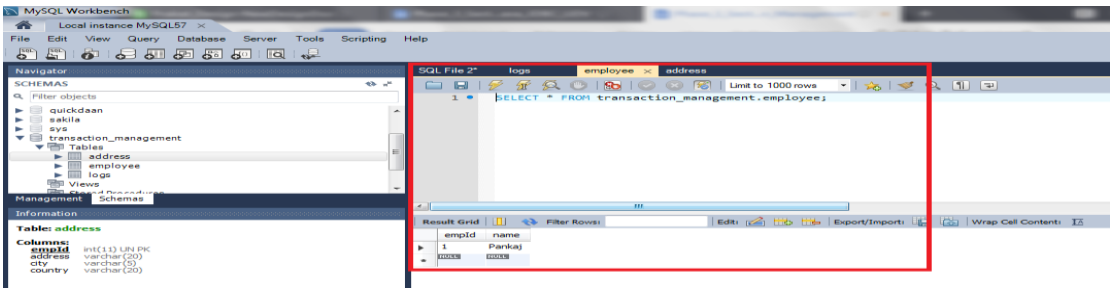



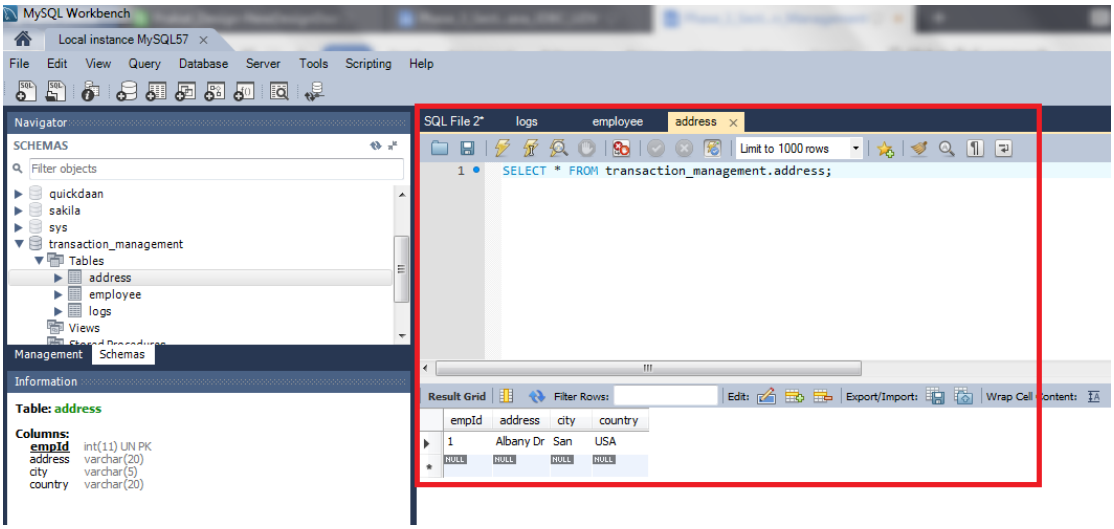

- Now we can change the city value, (here changed “San Jose” to “san” since city column size is 5) so that it can fit in the column and rerun the program to insert data into both the tables.

DB Connection created successfully

Employee Data inserted successfully for ID=1

Address Data inserted successfully for ID=1






- Notice that connection is committed only when both the inserts are executed successfully. If any of them throws an exception, we are rolling back the complete transaction.