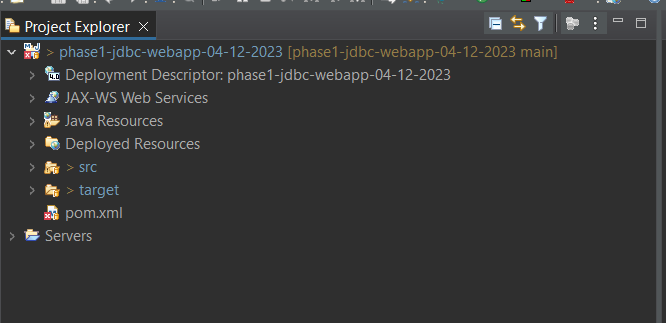
1. Set Up a JDBC Environment

**Development Environment**

* Eclipse IDE for Enterprise Java Developers
* Apache Tomcat Server v9.0
* JRE: OpenJDK Runtime Environment
* MySQL Workbench

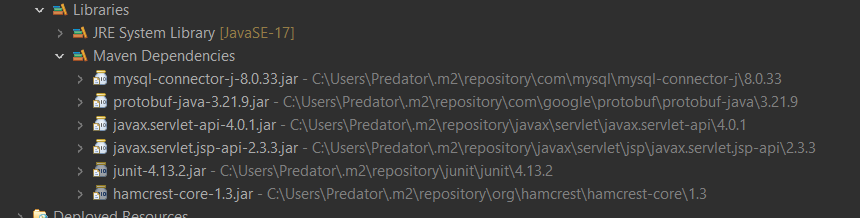
**Step 1.1:** Creating a dynamic web project

* Open Eclipse.
* Go to the **File** menu. Choose **New->Dynamic Web Project.**
* Enter the project name as **phase1-jdbc-webapp-04-12-2023**. Click on **Next.**
* Enter nothing in the next screen and click on **Next.**
* Uncheck the checkbox **Generate web.xml deployment descriptor** and click **Finish.**
* This will create the project files in the Project Explorer.



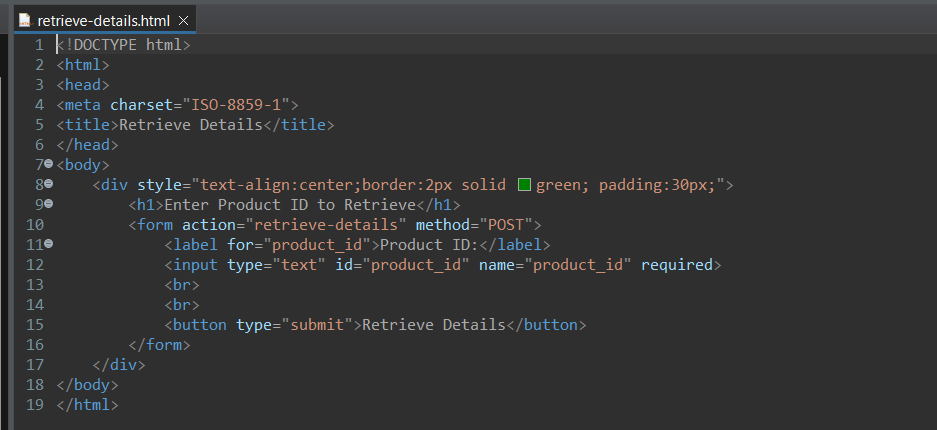
**Step 1.2:** Adding the jar files for MySQL connection for Java

* Click on **pom.xml**
* Make a tag of dependencies
* Under dependencies add dependency tag of
  + **Mysql**
  + **Javax Servlet**
  + **Javax Servlet Jsp**
  + **Junit**
* Click on Save and the files will be added automatically.



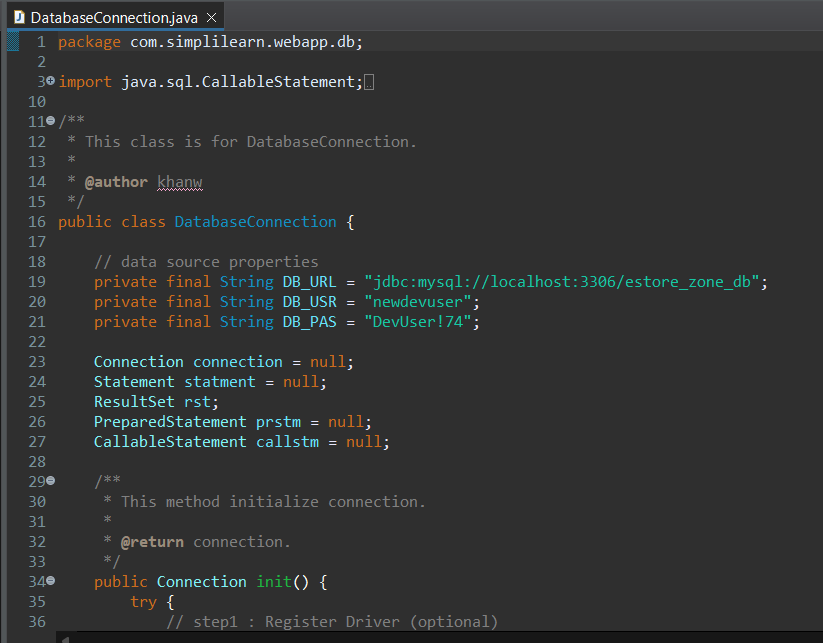
**Step 1.3:** Creating an HTML page

* In the Project Explorer, expand the project
* Expand **WebContent**. Right click on **WebContent**. Choose **New->HTML File**
* Enter the filename as **retrieve-details.html** and click on **Finish**



**Step 1.4:** Creating a JDBConnection 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/simplilearn/webapp/db** and in **Name** enter **DatabaseConnection** and click on **Finish**

****

**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**

**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**

**Step 1.7:** Configuring web.xml

* In the Project Explorer, expand **JDBCSetup->WebContent->WEB-INF**
* Double click on **pom.xml** to open it in the editor

**Step 1.8:** Checking for servlet-api.jar

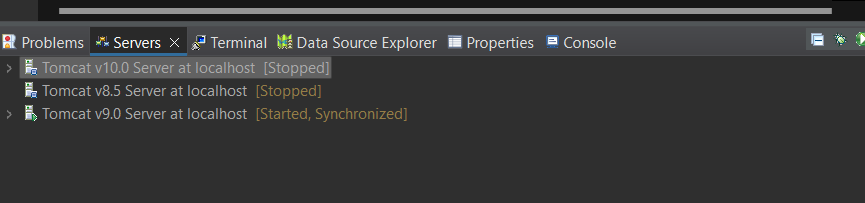
* Before building the project, we need to confirm that **servlet-api.jar** has been added to the project.
* In the Project Explorer, right click on **JDBCSetup** and choose **Properties.**
* Select **Java Build Path** from the options on the left.
* Click on **Libraries** tab on the right.
* Under **ClassPath,** 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



**Step 1.11:** Running the project

* As soon as the server is started it opens up the default browser with link: [**http://localhost:8080/**](http://localhost:8080/ServletConcept) **phase1-jdbc-webapp-04-12-2023/**

**Step 1.12:** Pushing the code to your GitHub repositories

* Open your command prompt and navigate to the folder where you have created your files

**cd <folder path>**

* 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:

1. **JDBC Connections, Statements, and ResultSets**

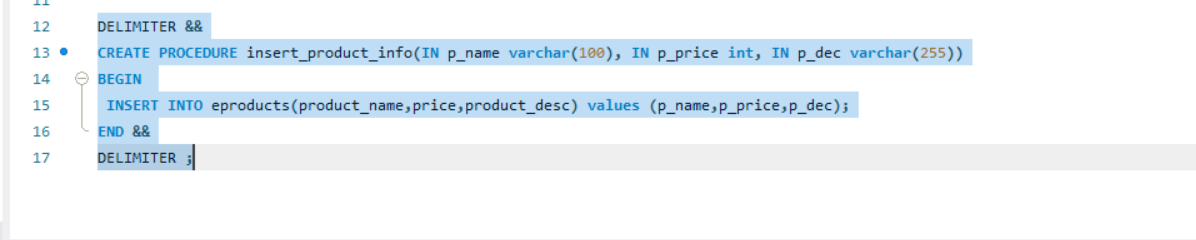
Creating a database in MySQL and creating a table in it

* 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 eproducts(product\_id int primary key auto\_increment, product\_name varchar(100), price decimal(10,2), product\_desc varchar(255), create\_at timestamp default now());**
* We will now add some rows to the table
* Type **INSERT INTO eproducts(name, ‘HP Laptop ABC’, 12000)** and press **Enter**
* Type **INSERT INTO eproducts(name, ‘Acer Laptop ABC’, 14000)** and press **Enter**
* Type **INSERT INTO eproducts(name, ‘Lenovo Laptop ABC’, 12000)** and press **Enter**
* Type **SELECT \* from eproducts** and press **Enter** to confirm that the rows have been added
* Type **EXIT** to exit the MySQL command console

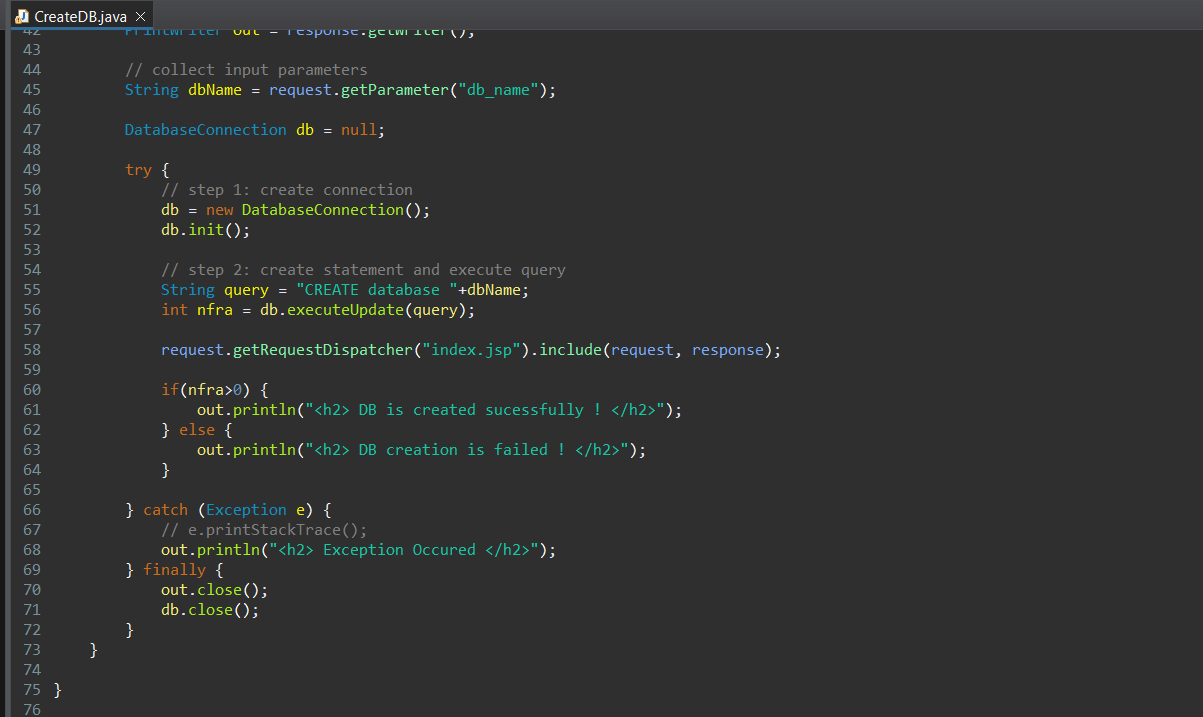
1. **Stored Procedures and Exception Handling**

**Creating a stored procedure in MySQL**

* Go to the MySQL WorkBench

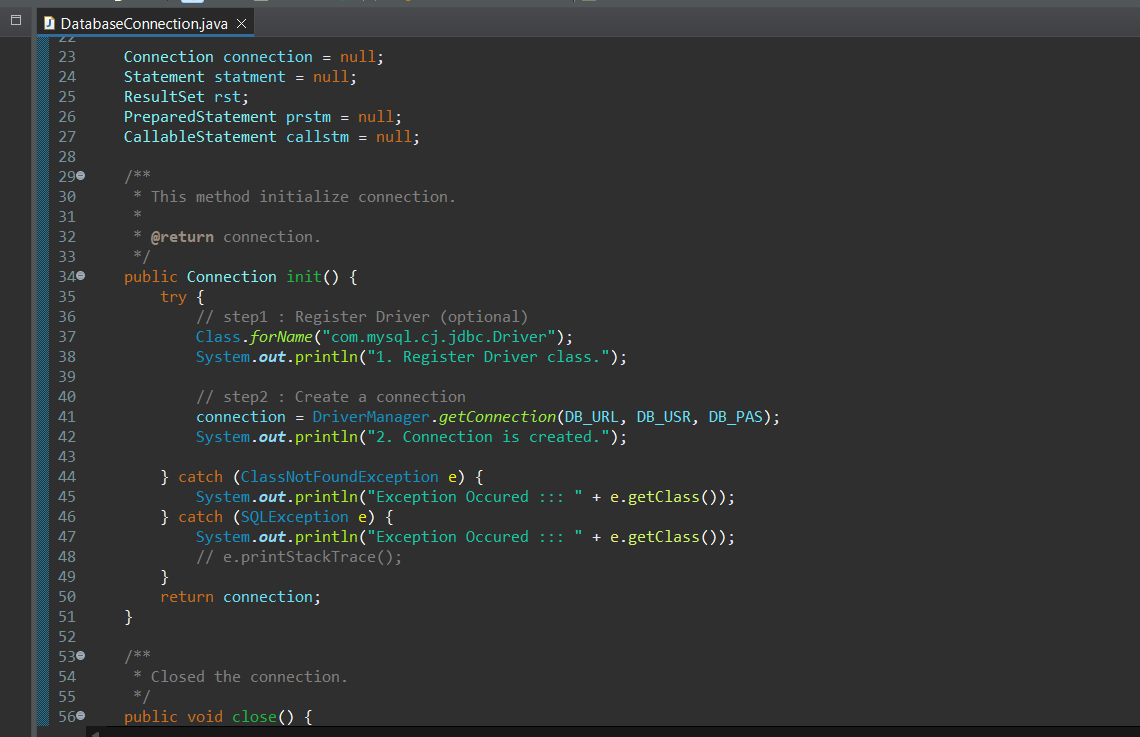
****Type the following script

**Exception Handling**

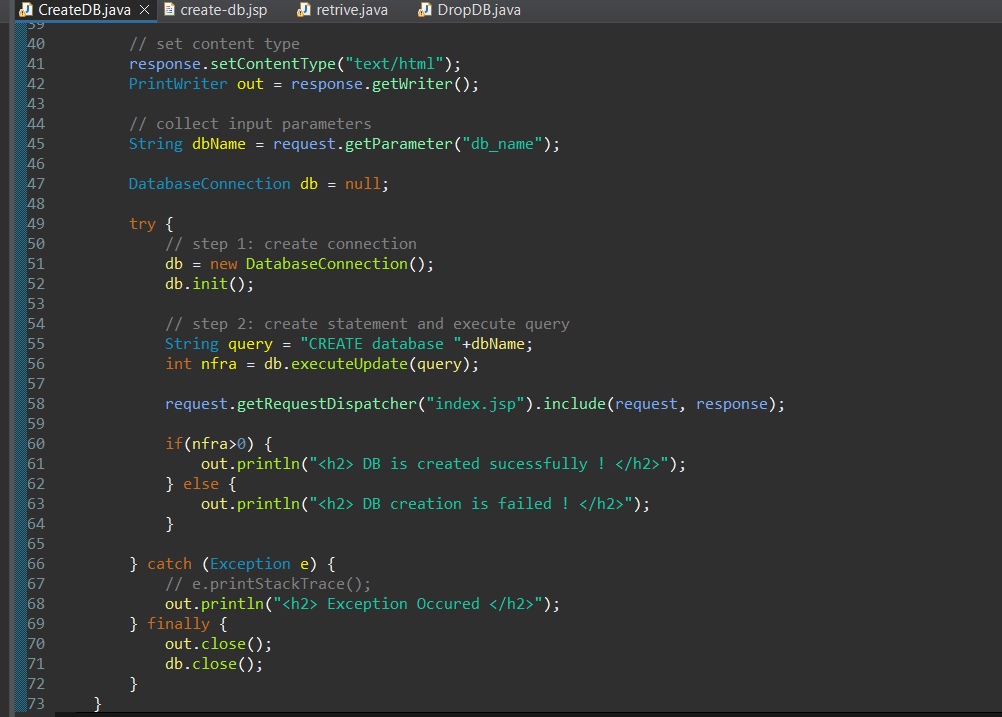


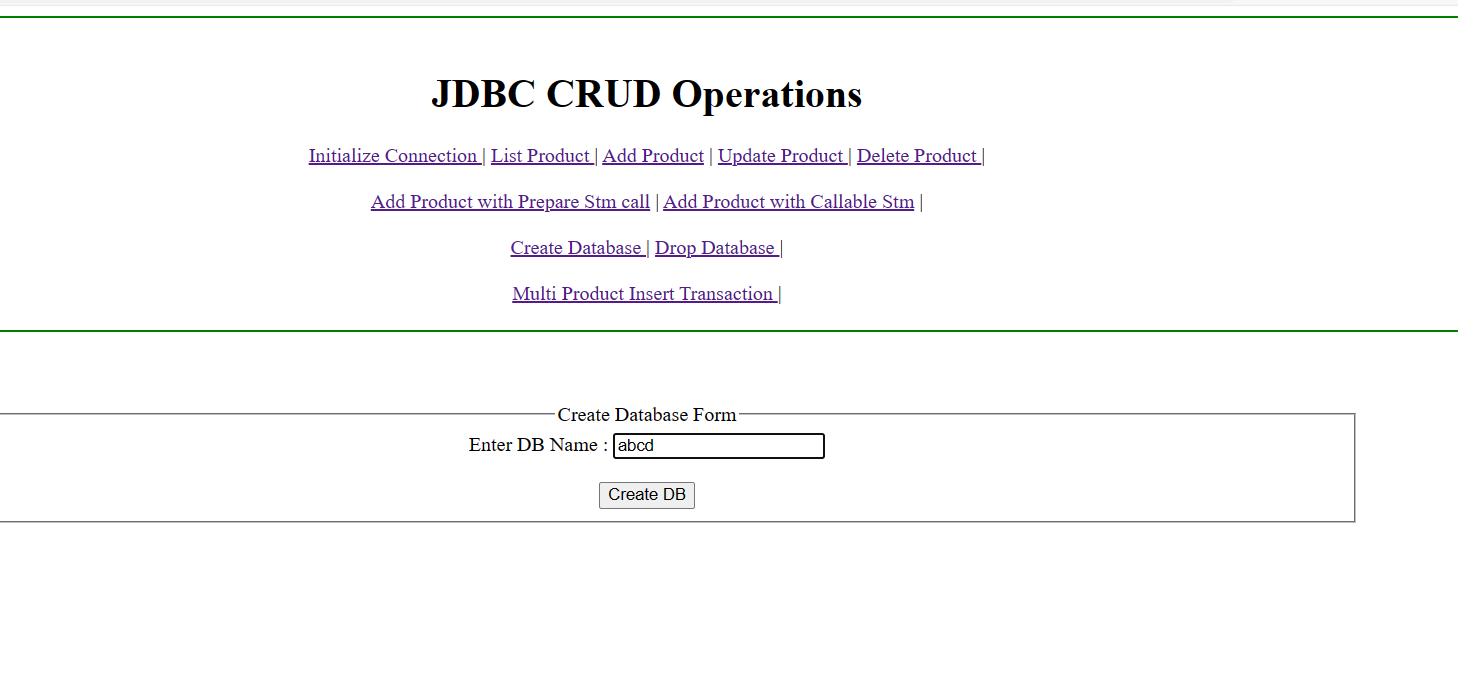
1. **Create, Select, and Drop a Database**

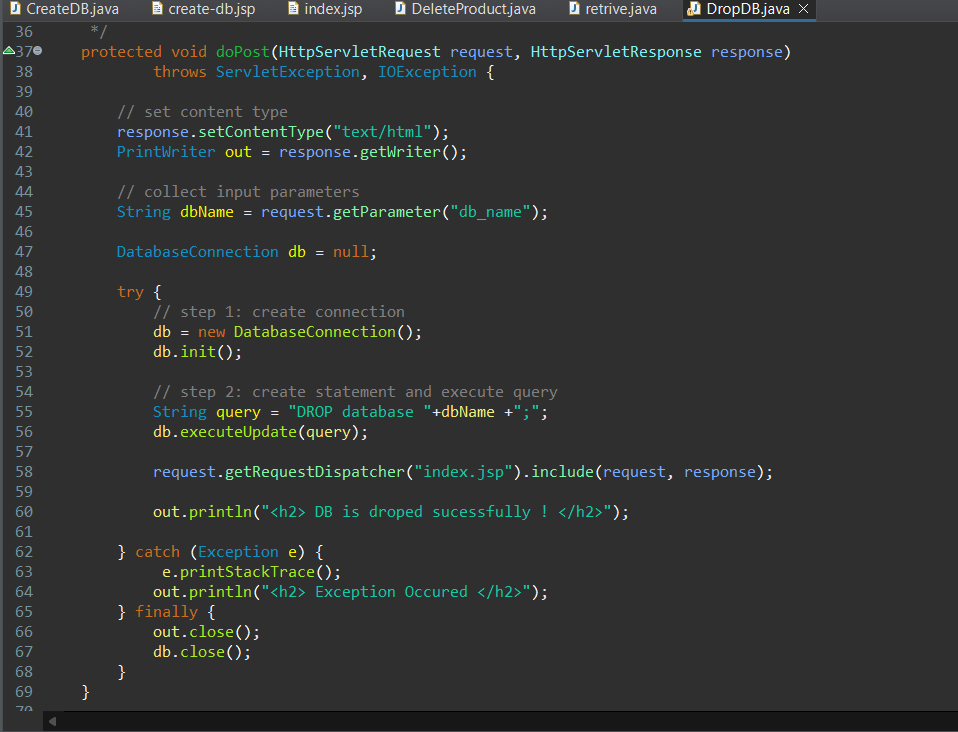
**Step1:** Create a Database Connection class to initialize connection

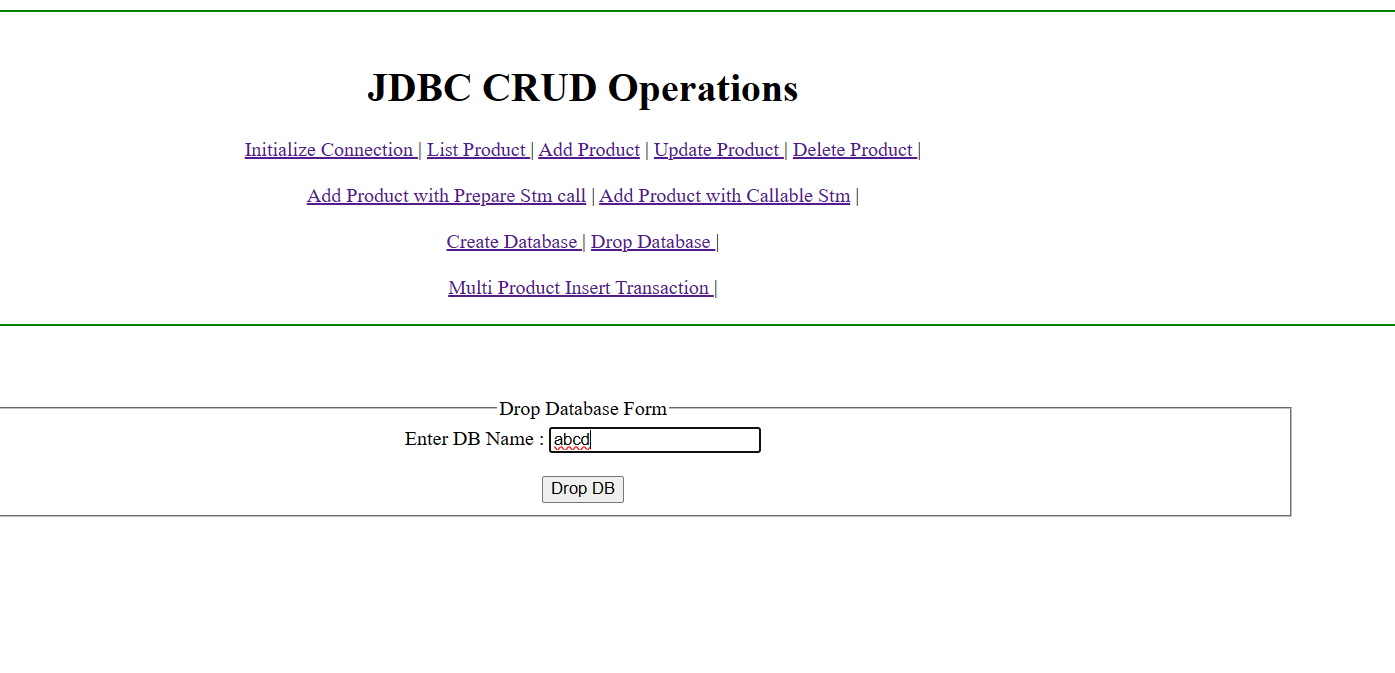
****

**Step2:** Creating ,Selecting and Deleting Database in JDBC



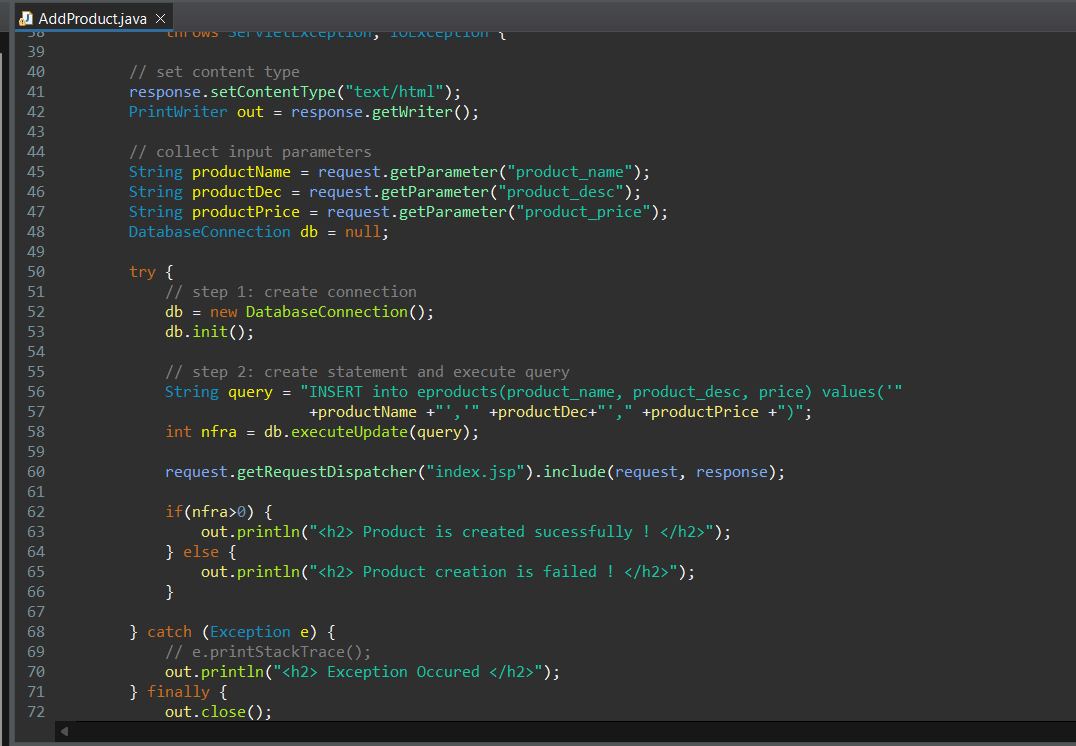




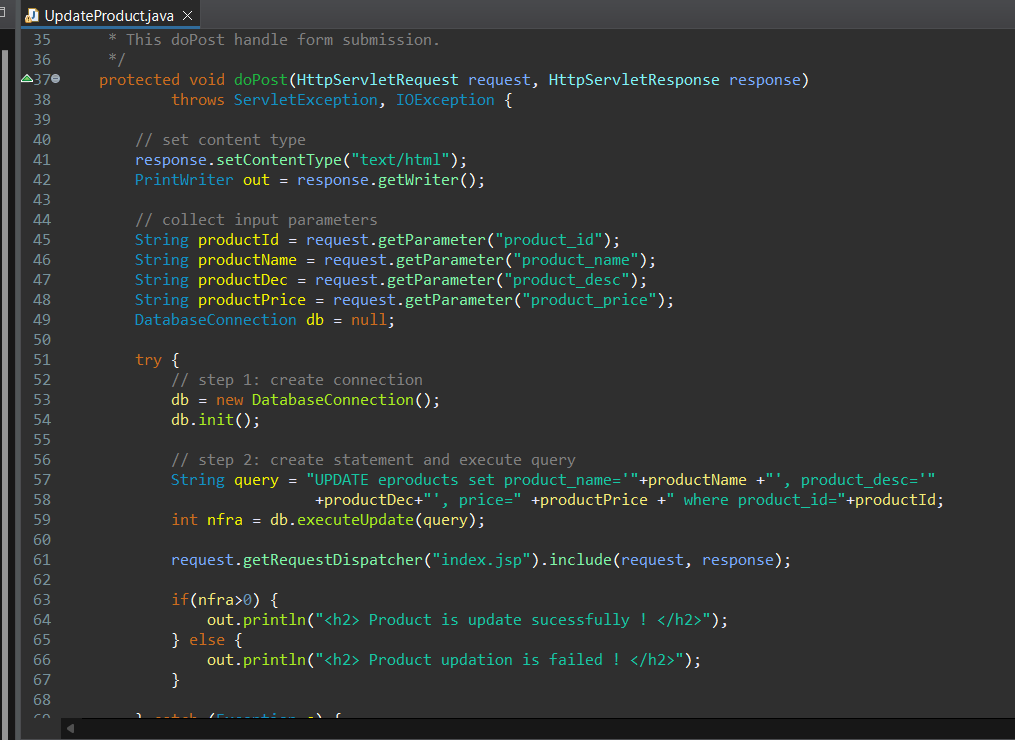


1. **Database record handling using JDBC**

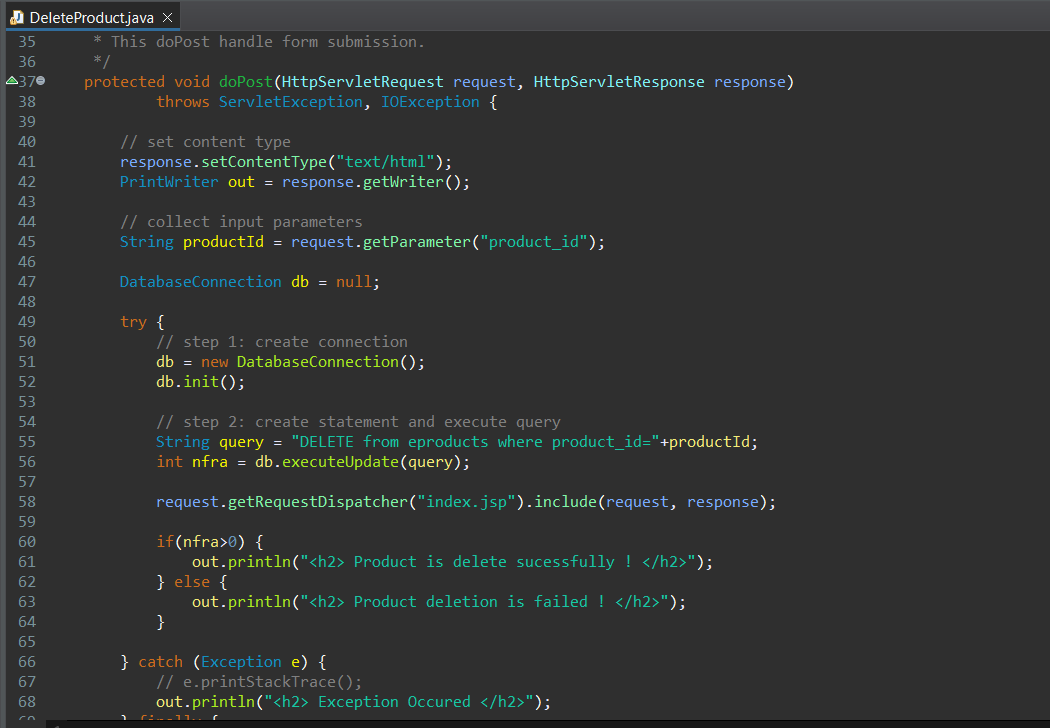
**INSERT**

****

**UPDATE**

****

**DELETE**

****

1. **Transaction Management in JDBC**

Create two tables ‘employee’ and ‘address’ in ‘transaction\_management’ database

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;

**Create a class called “DBConnection.java” and give the database credentials as below**:

* + - * **DB\_URL:** jdbc:mysql://localhost:3306/transaction\_management
      * **DB\_DRIVER\_CLASS:** com.mysql.jdbc.Driver
      * **DB\_USERNAME:** The username of database
      * **DB\_PASSWORD:** Password for the username

**DBConnection is the class used by other classes for MYSQL database connection.**

**Create another class called “EmployeeJDBCInsertExample.java”**

**Under the class put the following code**

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)

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

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.**

**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**