Retrieving the Product Details Using the Product ID.

DESCRIPTION

Project objective:

Create a servlet-based application that shows a form to enter a product ID. The product ID is then validated, and product details are retrieved from the database and displayed to the user.

You need to create a product table in MySQL and prepopulate it with data. Use JDBC to do

- all database processing.

Background of the problem statement:

As a part of developing an e-commerce web application, the admin backend requires a module that can retrieve product information based on the product ID.

You must use the following:

- Eclipse as the IDE
- Apache Tomcat as the web server
- MySQL Connector for JDBC functionality

Following requirements should be met :

- Create an HTML page to take in a product ID.
- Set up JDBC to work with the application
- Create a servlet that will take the product ID and use JDBC to query the database for the product
- If the product is found, the servlet will display the product details, otherwise it will show an error message
- Document the step-by-step process involved in completing this task

Objectives:

1. Make a servlet application.

2. Create a database

3. Use productid as primary key

4. Add products to database

5. Using JDBC establish a connection between servlet and myql

6. Use the productkey to find the records of the product in the product database

JDBC:

Before inserting contents in a table we need to connect our java application to our database.

Java has its own API which JDBC API which uses JDBC drivers for database connections.

Before JDBC, ODBC API was used but it was written in C which means it was platform_dependent. JDBC API provides the applications-to-JDBC connection and JDBC driver

provides a manager-to-driver connection.

Steps for connectivity between Java program and database

1. Loading the Driver: To begin with, you first need to load the driver or register it before using it in the program. Registration is to be done once in your program. You can register a driver in one of the two ways mentioned below:

- **Class.forName():** Here we load the driver's class file into memory at the runtime. No need of using new or creation of an object. The following example uses Class.forName() to load the Oracle driver –

```
Class.forName("oracle.jdbc.driver.OracleDriver");
```

- **DriverManager.registerDriver():** DriverManager is a Java inbuilt class with a static member register. Here we call the constructor of the driver class at compile time. The following example uses DriverManager.registerDriver()to register the Oracle driver –

```
DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver())
```

2. Create the connections: After loading the driver, establish connections using:

```
Connection con = DriverManager.getConnection(url,user,password)
```

User: username from which your SQL command prompt can be accessed.

Password: password from which your SQL command prompt can be accessed.

con: is a reference to Connection interface.

url : Uniform Resource Locator. It can be created as follows:

String url = " jdbc:oracle:thin:@localhost:1521:xe"

Where oracle is the database used, thin is the driver used, @localhost is the IP Address where the database is stored, 1521 is the port number and xe is the service provider. All 3 parameters above are of String type and are to be declared by the programmer before calling the function. Use of this can be referred from the final code.

3. Create a statement: Once a connection is established you can interact with the database. The **JDBCStatement**, **CallableStatement**,

and **PreparedStatement** interfaces define the methods that enable you to send SQL commands and receive data from your database.

Use of JDBC Statement is as follows:

Statement st = con.createStatement();

Here, con is a reference to Connection interface used in the previous step.

4. Execute the query: Now comes the most important part of executing the query. The query here is an SQL Query. Now we know we can have multiple types of queries. Some of them are as follows:

- Query for updating/inserting table in a database.
- Query for retrieving data.

The executeQuery() method of Statement interface is used to execute queries of retrieving values from the database. This method returns the object of ResultSet that can be used to get all the records of a table.

The executeUpdate(SQL query) method of statement interface is used to execute

queries of updating/inserting.

Example:

```
int m = st.executeUpdate(sql);
if (m==1)
  System.out.println("inserted successfully : "+sql);
else
  System.out.println("insertion failed");
```

Here SQL is SQL query of the type String

5. Close the connections: So finally we have sent the data to the specified location and now we are on the verge of completion of our task. By closing the connection, objects of Statement and ResultSet will be closed automatically. The close() method of the Connection interface is used to close the connection.

Example:

```
con.close();
```

Conclusion:

In this way, we understand the connection between jdbc and servlet and display the products using product key.