Build a servlet-based application that executes SQL statements against a database

by Joel Murach - *(While he may not be a JavaRanch regular, yet, he did provide us with this article to be included in our Newsletter.)*

This tutorial shows how to use a Java servlet, a JavaServer Page (JSP), and a static Java class to create an application that can be used to interactively execute any standard SQL statement against a database that's running on a server. You can use an application like this one to work with a database as you're developing an application. In this article, this application will be referred to as the SQL Gateway application.

If you're working with a database that's hosted by an Internet Service Provider (ISP), the ISP will usually include an HTML-based way to work with the database that's similar to this application. If not, you can upload this application to work with the database. Either way, this application shows how to use JDBC within a servlet to work with a database that's running on a server.

Prerequisites

This tutorial is an excerpt from chapter 11 of Murach's Java Servlets and JSP by Andrea Steelman and Joel Murach. It assumes that you have a basic understanding of the Java language, servlets, and JavaServer Pages. If you don't, you might be interested in Murach's Beginning Java 2 or Murach's Java Servlets and JSP.

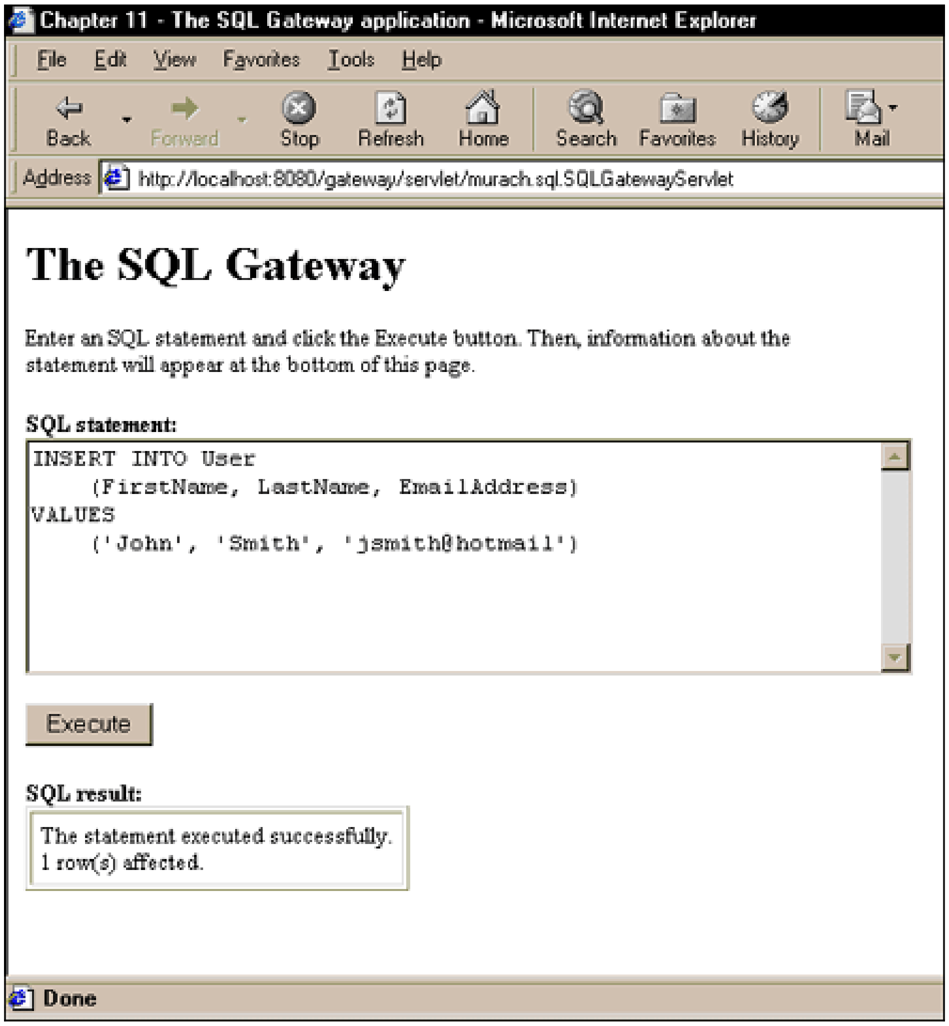
This tutorial also assumes that an appropriate servlet/JSP container and database server are available on the server that you're using. The source code for this tutorial has been tested using Tomcat 4.0 and MySQL, but it should work with other servlet/JSP containers, and it should work for most database servers if you supply a valid driver and connection string for that database.

Source code

Before you begin this tutorial, you may want to download the source code for this application. That way, you can open the source code in your favorite text editor to see how it all fits together.

The user interface

Here's the user interface for the SQL Gateway application after it has executed an INSERT statement:


Chapter 11 - The SQL Gateway application - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search Favorites History Mail
Address http://localhost:8080/gateway/servlet/murach.sql.SQLGatewayServlet
Execute
Done

The SQL Gateway

Enter an SQL statement and click the Execute button. Then, information about the statement will appear at the bottom of this page.

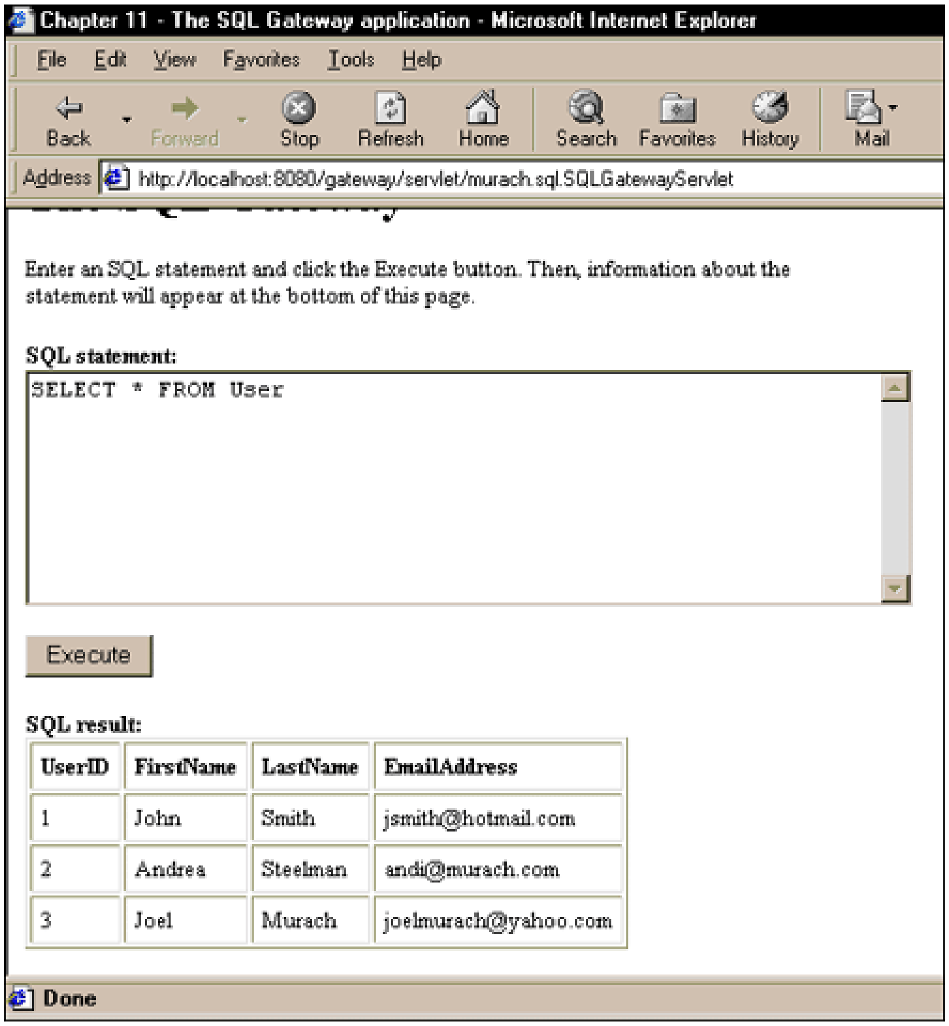
SQL statement:

```
INSERT INTO User
    (FirstName, LastName, EmailAddress)
VALUES
    ('John', 'Smith', 'jsmith@hotmail')
```

SQL result:

The statement executed successfully.
1 row(s) affected.

As you can see, the bottom of the page shows a message that indicates the number of rows that were affected by the statement. If the SQL statement is a SELECT statement that runs successfully, the result set will be displayed within an HTML table as shown here:


Chapter 11 - The SQL Gateway application - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search Favorites History Mail
Address http://localhost:8080/gateway/servlet/murach.sql.SQLGatewayServlet

Enter an SQL statement and click the Execute button. Then, information about the statement will appear at the bottom of this page.

SQL statement:

```
SELECT * FROM User
```

SQL result:

UserID	FirstName	LastName	EmailAddress
1	John	Smith	jsmith@hotmail.com
2	Andrea	Steelman	andi@murach.com
3	Joel	Murach	joelmurach@yahoo.com

Of course, if the SQL statement doesn't execute successfully, the result will be a message that displays information about the exception that was thrown.

The code for the JSP

The code for the JSP starts with a scriptlet that contains Java code that retrieves two attributes from the session object:

```
<!doctype html public "-//W3C//DTD HTML 4.0
Transitional//EN">
<%
   String sqlStatement =
      (String) session.getAttribute("sqlStatement");
   if (sqlStatement == null)
      sqlStatement = "";
   String message =
      (String) session.getAttribute("message");
   if (message == null)
      message = "";
%>
```

The first attribute is the string that contains the SQL statement, and the second attribute is the string that contains the result message. If these attributes contain null values, they haven't been set yet so this code sets the sqlStatement and message variables to empty strings.

This JSP also contains an HTML form that contains a text area and a submit button:

```
<form action="../servlet/murach.sql.SQLGatewayServlet"
method="post">
   <b>SQL statement:</b><br>
   <textarea name="sqlStatement" cols=60 rows=8>
      <%= sqlStatement %>
   <textarea><br>
   <br>
   <input type="submit" value="Execute">
<form>
```

Here, the text area allows the user to enter the SQL statement. This code creates a text area that's approximately 60 characters wide and 8 lines tall. Within this area, the sqlStatement variable is displayed, which is empty the first time this JSP is run. Then, when the user clicks the submit button, this JSP calls the SQLGatewayServlet that's described later in this article.

The table near the end of the JSP displays the message string that contains the result of the SQL statement:

```
<b>SQL result:</b><br>
<table cellpadding="5" border="1">
  <%= message %>
<table>
```

Since this message contains the rows and columns for an HTML table, it's coded within the Table tags.

The code for the servlet

The SQLGatewayServlet, which is stored in the murach.sql package, starts by importing the java.sql package so it can use the JDBC classes. In addition, it declares a Connection object so the database connection can be used by all of the methods in the servlet:

```
package murach.sql;

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.sql.*;

public class SQLGatewayServlet extends HttpServlet{

    private Connection connection;
```

When the servlet engine places this servlet into service, the init method opens the connection to the database.

```
    public void init() throws ServletException{
        try{
            Class.forName("org.gjt.mm.mysql.Driver");
            String dbURL =
"jdbc:mysql://localhost/murach";
            String username = "root";
            String password = "";
            connection = DriverManager.getConnection(
                dbURL, username, password);
        }
        catch(ClassNotFoundException e){
            System.out.println("Database driver not
found.");
        }
        catch(SQLException e){
            System.out.println(
              "Error opening the db connection: "
                + e.getMessage());
        }
    }
```

Usually, this occurs when the first user uses the application. That way, the database connection will be open and available for all subsequent users. Then, a new thread is spawned for each user that uses this servlet.

In this example, the servlet uses a driver for the MySQL database to open a connection to a database named "murach" that's running on the same server as the servlet. In addition, this servlet uses MySQL's default username of "root" and a blank password. However, you can modify this code to connect to just about any type of database running on any type of server. Either way, you'll need to make sure that an appropriate driver for the database is installed on the server. For more information about getting, installing, and configuring MySQL, you can go to www.mysql.com. In addition, there's an introduction to MySQL in chapter 10 of Murach's Java Servlets and JSP.

Before the servlet engine takes a servlet out of service, the destroy method closes the database connection and frees up the resources required by the connection.

```
    public void destroy() {
        try{
            connection.close();
        }
        catch(SQLException e){
            System.out.println(
              "Error closing the db connection: "
                + e.getMessage());
        }
    }
```

When the JSP shown earlier calls the doPost method, this method calls the doGet method.

```
    public void doPost(HttpServletRequest request,
                    HttpServletResponse response)
                    throws IOException,
ServletException{
        doGet(request, response);
    }
```

Within the doGet method, the first statement gets the SQL statement that the user entered in the JSP, and the second statement declares the message variable.

```
    public void doGet(HttpServletRequest request,
                     HttpServletResponse response)
                     throws IOException,
ServletException{

        String sqlStatement =
            request.getParameter("sqlStatement");
        String message = "";
```

Then, within the try block, the first statement uses the Connection object to create a Statement object, and the next two statements use the trim and substring methods of a String object to return the first six letters of the SQL statement that the user entered.

```
    try{
        Statement statement =
connection.createStatement();
        sqlStatement = sqlStatement.trim();
        String sqlType = sqlStatement.substring(0, 6);
```

If the first six letters of the SQL statement are "select", the executeQuery method of the Statement object returns a ResultSet object. Then, this object is passed to the getHtmlRows method of the SQLUtil class that's shown later in this article, and it returns the result set formatted with the HTML tags for rows and columns.

```
    if (sqlType.equalsIgnoreCase("select")){
        ResultSet resultSet =
            statement.executeQuery(sqlStatement);
        message =
            SQLUtil.getHtmlRows(resultSet);
    }
```

However, if the first six letters of the SQL statement aren't "select", the executeUpdate method of the Statement object is called, which returns the number of rows that were affected. If the number of rows is 0, the SQL statement was a DDL statement like a DROP TABLE or

CREATE TABLE statement. Otherwise, the SQL statement was an INSERT, UPDATE, or DELETE statement. Either way, the code sets the message variable to an appropriate message.

```
    else{
        int i = statement.executeUpdate(sqlStatement);
        if (i == 0) // this is a DDL statement
          message =
            "<tr><td>" +
              "The statement executed successfully." +
            "</td></tr>";
        else        // this is a DML statement
          message =
            "<tr><td>" +
              "The statement executed successfully.<br>"
+
              i + " row(s) affected." +
            "</td></tr>";
        }
        statement.close();
    }
```

If any of the statements within the try block throw an SQLException, the catch block sets the message variable to display information about the SQLException. If, for example, you enter an SQL statement that contains incorrect syntax, this message will help you troubleshoot your syntax problem.

```
    catch(SQLException e){
        message = "Error executing the SQL statement:
<br>"
                + e.getMessage();
    }
```

After the catch block, the next three statements get the session object and set the sqlStatement and message variables as attributes of that object.

```
        HttpSession session = request.getSession();
        session.setAttribute("message", message);
        session.setAttribute("sqlStatement",
sqlStatement);
```

Then, the last two statements return a RequestDispatcher object that forwards the request and response objects to the JSP shown earlier in this article.

```
        RequestDispatcher dispatcher =
            getServletContext().getRequestDispatcher(
                "/sql/sql_gateway.jsp");
        dispatcher.forward(request, response);
    }
}
```

The code for the utility class

The code for the utility class named SQLUtil is shown below. This class contains a static method named getHtmlRows that is called by the servlet shown eariler. Like the SQLGatewayServlet, this class is stored in the murach.sql package.

```
package murach.sql;

import java.sql.*;

public class SQLUtil{
```

The getHtmlRows method accepts a ResultSet object and returns a String object that contains the HTML code for all of the column headings and rows in the result set. To build the information for that String object, the getHtmlRows declares a StringBuffer object named htmlRows and appends data to it as the method is executed. At the end of the method, the toString method is used to convert the StringBuffer object to the String object that is returned to the servlet.

```
    public static synchronized String
    getHtmlRows(ResultSet results) throws SQLException{
        StringBuffer htmlRows = new StringBuffer();
        ResultSetMetaData metaData =
results.getMetaData();
        int columnCount = metaData.getColumnCount();

        htmlRows.append("<tr>");
        for (int i = 1; i <= columnCount; i++)
            htmlRows.append("<td><b>"
                + metaData.getColumnName(i) + "</td>");
        htmlRows.append("</tr>");

        while (results.next()){
            htmlRows.append("<tr>");
            for (int i = 1; i <= columnCount; i++)
                htmlRows.append("<td>"
                    + results.getString(i) + "</td>");
        }
        htmlRows.append("</tr>");
        return htmlRows.toString();
    }
}
```

To get the column headings that are returned, the getHtmlRows method uses the getMetaData method of the ResultSet object to create a ResultSetMetaData object. This type of object contains information about the result set including the number of columns and the names of the columns. To get that information, the getHtmlRows method uses the getHtmlRows and getColumnName methods of the ResultSetMetaData object.

To get the data from the result set, the getHtmlRows method uses a for loop within a while loop to get the data for each column in each row. Within these loops, the code uses the getString method of the result set to get the data for each field. That converts the data to a string no matter what data type the field is.

Please note that this method is declared with the synchronized keyword. This prevents two or more threads of a servlet from executing this method at the same time.

Thanks for reading. Please let me know if you found this tutorial helpful.

Joel Murach
joelmurach@yahoo.com

Related Downloads and Resources

Source Code - Free Download
http://www.murach.com/books/jsps/download_sql_gateway.htm
Feel free to download the code for this application. Then, you can install and configure it to work on your system.

Murach's Java Servlets and JSP
http://www.murach.com/books/jsps
If you want to learn more about servlet and JSP development using a MySQL database, JDBC, and connection pooling, this book uses a unique format to methodically present these topics. In addition, it presents other essential topics such as JavaBeans, custom JSP tags, JavaMail, SSL, security, and XML.

Murach's Beginning Java 2
http://www.murach.com/books/java
If you had any trouble understanding the Java syntax presented in this tutorial, this book provides great introduction to the Java language and also makes a great reference.