Creating a Query Tool

Table Of Contents

Table Of Contents

Introduction
Quick Start
Setting up The Environment
Installing the Query Tool
Getting Down to Business: Using Cloudscape with Enhydra
Setting Up the Environment
Creating the Application
Query_Tool.html
SubmitQuery.jsp
Section 1: Collecting Parameters
Section 2: Submitting the Query
Section 3: Display the Results
DemoConnect.java
Register and Start Your Application
Summary
Resource Links (A list of the links referenced in this paper)

Introduction

This paper describes how to create a JavaServer Pages (JSP) based query tool for your Cloudscape 3.5 database. It include
code and details on developing a Web Application using Java. For this paper, we will use the following tools:

- Enhydra 3.0.1 Java Application Server
- Redhat 6.1 Linux
- Blackdown JDK 1.2.2

These tools were selected because they are all available for free download. The application and Cloudscape should work v
any Java application server that supports JSP, any version 2 JDK, and any OS that supports a version 2 JDK. (See the
Cloudscape JVM Compatibility Chart).

We will examine how to use Cloudscape as the database engine for your web-based application by looking at a simple data
query tool using an HTML file, JSP and a bean.

To just get the query tool application up and running quickly, go to "Quick Start". To get step-by-step details, skip to "Gett
Down To Business".

Quick Start

The rest of this paper details the anatomy of the query tool application. If you just want to get it running here are some qu
start steps.

The examples use **/home** as the root location.

Setting Up the Environment

1. Install the Blackdown JDK 1.2.2 in /home/jdk1.2.2Blackdown

   * Any 1.2 JDK should work

2. Edit your path to include **/home/jdk1.2.2Blackdown/bin** :

   a. Bash/ksh: **export PATH=$PATH:/home/jdk1.2.2Blackdown/bin**

3. Install Cloudscape in /home/cloudscape:

   a. Download Cloudscape: http://www.cloudscape.com/Evaluations/index.html

   b. Follow the directions in the install.html file that accompanies your software or see Installing Cloudscape.

4. Create a Directory to store the database

   mkdir –p /home/cloudscape/data

5. Install Enhydra

   a. Download Enhydra3.0.1

   b. Run: rpm –i Enhydra-3.0.1-1.i386.rpm

   The default directory for Enhydra 3.0.1 is /usr/local/enhydra3.0.1.

6. Configure Enhydra:

   /usr/local/enhydra3.0.1/configure /home/jdk1.2.2Blackdown

7. Edit /usr/local/enhydra3.0.1/bin/multiserver

   Change
   ```
   CLASSPATH=${NEWCP}
   ```
   to:
   ```
   CLASSPATH=${NEWCP}:/home/cloudscape/lib/cloudscape.jar:/home
   ```

8. Edit your CLASSPATH (for bash/ksh)

   export CLASSPATH= $CLASSPATH:/home/cloudscape/lib/cloudscape.jar:/home

Note: If you're going through "Getting Down To Business", go back to "Setting Up Your Environment" at this point.

Installing The Query Tool

1. Download the QueryTool.tar:

   QueryTool.tar

2. Expand the tar file into /home

   tar –xf /home QueryTool.tar

3. If you do not want your database created in /home/cloudscape/data

   a. Edit /home/jdbc/Paper/DemoConnect.java

   b. Change:
   ```
   p.put("cloudscape.system.home", "/home/cloudscape/data");
   ```
   to:
   ```
   p.put("cloudscape.system.home", "/{yourPath}/data");
   ```

   c. Recompile:
   ```
   Javac DemoConnect.java
   ```

4. As root, Start Enhydra:

   /usr/local/enhydra3.0.1/bin/multiserver &

5. Open a web browser and open the Enhydra administration interface:

   http://{yourSystem}:8001

6. Log in using the default username and password:

   Username: admin
   Password: enhydra

7. Click **Add**.

8. Select **"War"**.

9. Enter:

   Name = QueryTool
   Doc Root = /home/public_html/Paper

10. Click **OK**.

11. Select the Connections tab
12. Click **Create**
13. Enter:

    URL Prefix = /Cloudscape
    Port Number = 1000
14. Click **OK**
15. Click **OK** again
16. Click **Save State** to save the changes.
17. Click **OK**
18. Click **OK** again
19. Select **QueryTool** from the list
20. Click **Start**.

Test The Query Tool

1. Your application is now running. To use the application, open a web browser and open the URL:

   http://{yourSystem}:1000/Cloudscape/Query_Tool.html.
2. In the text window enter the query "Select * from sys.systables"

   (Do not put a semi-colon ";" at the end of the query text)
3. Click Submit Query.

   NOTE: When you run the application for the first time, it creates and start the database which might take a f
   minutes.
4. Your query tool is done! If you would like to know how it works read-on.

Getting Down to Business: Using Cloudscape with Enhydra

Using Java Server Pages (JSP) allows for quick development and easy integration with Cloudscape. To demonstrate this, w
create a simple tool. Our first application will be a JSP, based query tool for Cloudscape. In this section, we will:

1. Install:
   JDK 1.2.2
   Cloudscape 3.5
   Enhydra 3.0.1
2. Configure Enhydra.
3. Create the application files.
4. Start the application.
5. Run a query using the newly created query tool.

Setting Up the Environment

Before we create our application, we must set up the environment necessary to run the application. We will start by creati
directories for the application files:
Note: The examples will install everything in /home

1. Go to Quick Start and complete the steps in Setting Up the Enviornment.
2. Create a directory for the .java and .class files:

   mkdir –p /home/jdbc/Paper
3. Create a directory for the *.jsp, *.gif, *.html files

   mkdir –p /home/public_html/Paper
4. Start Enhydra:

   /usr/local/enhydra3.0.1/bin/multiserver &
   The Enhydra server is now running.
5. Open a web browser and go to the Enhydra administration interface:

http://{yourSystem}:8001

6. Log in using the default username and password:

   Username: admin
   Password: enhydra

If you can log into the Enhydra Administration tool, you have successfully set up Enhydra. Now that we have the environme
ready, it is now time to create our query tool.

Creating the Application

To create the query tool we, must create 3 files:

File	**Description**
Query_Tool.html	This is the input form for the application.
SubmitQuery.jsp	This is the Java Server page which will execute the query and display the results as HTML.
DemoConnect.java	This will create a connection to the database and instantiate an instance of the Cloudscape JDBC driver if needed.
Note:	These files are available for download from files/QueryTool.tar

Query_Tool.html

We will start with the HTML file which is used to enter and submit the query to the JSP. This HTML file contains a single for
with an input text box, drop down list of databases, Submit and Reset buttons.

```
<html>
<head>
<title>Cloudscape Query Tool</title>
</head>

<body bgcolor="#6699FF" background="clouds.gif" bgproperties="fixed">

<p><i><b><font color="#333399" size="7">
        Cloudscape Query Tool:</font></b></i></p>
<hr>
<form method="POST" action="SubmitQuery.jsp">
    <textarea rows="15" name="query_text" cols="71"></textarea>
    <p><b>Database:</b> <select size="1" name="database">
        <option>demoDB</option></select></p>
    <p><input type="submit" value="Submit Query" name="SubmitQuery" >
    <input type="reset" value="Reset" name="B2"></p>
</form>

</body>
</html>
```

This form will post the fields “query_text” and “database” to SubmitQuery.jsp for processing. When you are done creating Query_Tool.html, copy it to **/home/public_html/Paper/Query_Tool.html**.

SubmitQuery.jsp

This JSP actually submits the query to the database and displays the results in an HTML table with column headers. In this there are three sections to the code: Collecting the parameters, submitting the query and displaying the results. We will loo each section in detail.

Section 1: Collecting Parameters

```
<%
    int maxRows = 50;
    //We will limit the number of output rows
    boolean maxRowsHit = false;
    String queryText = "";
    String database = "";
    Java.sql.Connection connection = null;

    //Get the parameters from query_tool.html
    //The "request" object is a JSP built in to
    //access data sent by POST
    queryText  = request.getParameter("query_text");

    //We Cannot Run a query without Text So let's
    //Catch this Situation
    if ( queryText.equalsIgnoreCase("") )
    {
        //NOTE: "out" is a built in HTML printwriter for JSP
        out.println("<br><br>Please Enter Your SQL Query");
        out.println("<br>Use the BACK button on your browser “+
                    “and try again.\n");
        return;
    }

    //We need the database name
    database  = request.getParameter("database");

%>
```

Section 2: Submitting the Query

```
<html>
<head>
    <title>Query Results</title>
</head>
<center><b><font size=+1>Your Query Results:</font></b></center>
<br>
<body bgcolor=#ffffff> <font face="Comic Sans MS" size+=4 color=red>
<CENTER>
<TABLE Border="2" Cellpadding="3" Cellspacing="3">


<%


    try {
        //We will use our DemoConnect Class to get
        //a new connection to the database
        jdbc.Paper.DemoConnect newConn = new jdbc.Paper.DemoConnect();

        connection = newConn.getConnection(database);


        // Send the query to the database
        // Normally for performance reasons we would
        // use a PreparedStatement or a
        // stored prepared statement in an application.
        // In this case the query
        // is Dynamic so we will use a statement.
        Java.sql.Statement statement =
                                connection.createStatement();
        statement.execute(queryText);
```

Section 3: Displaying The Results

```
// Process results of the select.
Java.sql.ResultSet result = statement.getResultSet();
int numRows = 1;
int numCols = 2;

//*****************************************************
//Print Header Information
//We will use the ResultSetMetaData to get the column names
//and print them as the first row of our table
Java.sql.ResultSetMetaData dataParams =
                        result.getMetaData();
out.print("<TR><TD Align=\"center\">" +
            dataParams.getColumnName(1));

//We build this in a loop because we do not
//know how many columns of output were requested.
for ( numCols = 2; numCols <= dataParams.getColumnCount();
                                numCols++ )
    out.print("</TD Align=\"center\"> “+
              “<TD Align=\"center\">" +
              dataParams.getColumnName(numCols));

//Print the closing info for the row
out.println("</TD Align=\"center\"></TR>");
//*****************************************************
//Print the query Results
String resultData = "";

while (result.next())
{
    resultData = "<TR><TD Align=\"center\">" +
                    result.getString(1);
    for ( numCols = 2;
          numCols <= dataParams.getColumnCount();
          numCols++ )
    {
        resultData = resultData +
        "</TD Align=\"center\"><TD Align=\"center\">"+
        result.getString(numCols);
    } //End for

    resultData = resultData + "</TD Align=\"center\">";
    out.println(resultData);

    //Keep track of the number of Rows processed
    if ( numRows == maxRows )
    {
        maxRowsHit = true;
        break;
    }
    numRows++;

}//End While more data rows

//Explicitly close all statements, connecitons etc
```

```
        result.close();
        statement.close();
    }
    catch(Exception ex){
        out.println("<p><br>Exception during SQL query: "+ex);
    }
    //Make sure the connection is closed even if there is a failure
    finally {
        connection.close();
    }

    out.println("</TABLE></CENTER>");
    //Print a message if we hit the maximum number of output rows
    if ( maxRowsHit == true)
        out.println("<CENTER><br>Return A Maxium of "+
                       maxRows+" Rows<br></CENTER>");

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

Section 1: Collecting Parameters

We start the JSP by collecting the parameters “database” and “query_text” posted from the HTML form. JSP support a buil “request” object which is the same as **javax.servlet.ServletRequest**.

```
queryText  = request.getParameter("query_text");
database  = request.getParameter("database");
```

For simplicity, the only error we look for is the existence of query_text. For this condition, we print the message to HTML ar the processing of the JSP. In a JSP, “out” is a built-in HTML java.io.PrintWriter. Using out.println we can write the error to browser by writing:

```
out.println("<br><br>Please Enter Your SQL Query");
```

If query text was provided, we are ready to submit the query to the database.

Section 2: Submitting the Query

To submit a query to the database we must obtain a connection, create a statement, and execute the statement.

To obtain a connection, we call DemoConnect.getConnection() which returns a connection object. We will discuss DemoConnect.class in the next section. DemoConnect.getConnection() takes a String containing the database name as an argument:

```
connection = newConn.getConnection(database);
```

Once we have a connection object, we can create a statement. First, we create a statement object:

```
Statement statement = connection.createStatement();
```

Then, we execute the statement:

```
statement.execute(queryText);
```

We are using **java.sql.Statement()** in this case because we do not know ahead of time what the query will be. In most applications, the text of the query is known ahead of time. For queries with known text, use java.sql.PeparedStatement or, yet, Cloudscape stored prepared statements.

Take a look at the Tuning Cloudscape for information on these statement types. The Stored Prepared Statements Perform Tips contains information about the performance impacts of using the different statement types.

We have collected the parameters and submitted the query, now we need to display the results of the query.

Section 3: Display the Results

We will display the query results in an HTML table. To display the results, we will use a **java.sql.resultSet()**.

Since this is an ad-hoc query tool, we do not know how many columns of data are expected. To determine the number of columns and the text for the column headers, we will use **java.sql.resultSetMetaData()**. Using the **resultSetMetaDat** methods **getColumnCount()** and **getColumnName()** we create the header row for the HTML table. (See Code Section "Display The Results")

We start by printing the start of the row (<tr>) and the first column header:

```
ResultSetMetaData dataParams = result.getMetaData();
out.print("<TR><TD Align=\"center\">" +
                dataParams.getColumnName(1));
```

Then, we print the rest of the column headers:

```
for ( numCols = 2; numCols <= dataParams.getColumnCount(); numCols++ )
    out.print("</TD Align=\"center\"><TD Align=\"center\">" +
                                    dataParams.getColumnName(numCols));
```

When we are done with column headers we print the end-of-row information for the HTML table:

```
out.println("</TD Align=\"center\"></TR>");
```

Then we process the rest of the result set in the same manner until we run out of rows or we reach maxRows. Except this instead of looking at the metaData we use the resultSet.get... methods.

When we are done with any **resultSet**, **Connection** or **Statement** it should be explicitly closed. We will first close result statement:

```
result.close();
statement.close();
```

The Connection should be closed in the **finally{...}** section of the **try{...}** block. This way if there is an error in the proce of the query, the database connection is not left open:

```
finally {
    connection.close();
}
```

The last thing we do is print a message if we reached the maximum number of rows so we verify that we are seeing the complete result:

```
if ( maxRowsHit == true)
    out.println("<CENTER><br>Return A Maxium of "+
                                    maxRows+" Rows<br></CENTER>");
```

When you are done creating SubmitQuery.jsp, copy it to /home/public_html/Paper/SubmitQuery.jsp. Now that you have cr the JSP to process and display the results, you need to create the DemoConnect.java which is referenced in SubmitQuery.js

DemoConnect.java

DemoConnect.java returns a connection to the Cloudscape database. We only need to instantiate the Cloudscape JDBC c
when one does not exist.

```
Package jdbc.Paper;

import Java.sql.*;
import Java.util.*;

public class DemoConnect
{
    public DemoConnect ()
    {

    }

    public Connection getConnection(String database)
        throws SQLException
    {
        Connection conn = null;

        //Open a connection to the database
        try {
            //We will connect to the database or
            //create a database if one does not exist
            conn =   DriverManager.getConnection("jdbc:cloudscape:"+
                                database+";create=true","","");
        }
        //If there is no Driver start one and try the connection again

        catch (SQLException ex) {
            try {
                Properties p = System.getProperties();
                //We set cloudscape.system.home to the
                //directory where the database is or should exist
                p.put("cloudscape.system.home",
                            "/home/cloudscape/data");
                //We are running in embedded mode using
                //the cloudscape JDBCDriver

          Class.forName("COM.cloudscape.core.JDBCDriver").newInstance
();
                conn =
                    DriverManager.getConnection("jdbc:cloudscape:"+

database+";create=true","","");
            }  catch (Exception e) {
                System.out.println("Exception in Start Driver: "+e );
            }

        } catch (Exception e) {
            System.out.println("Exception in DemoConnect: "+e );
        }

        return(conn);
    }

}
```

We created a method called **getConnection()** which takes a **String** containing the database name as an argument. Most the time there will be an instance of the driver so we will first try to establish a connection.

```
Connection conn = null;

conn = DriverManager.getConnection("jdbc:cloudscape:"+
     database+";create=true","","");
```

In most cases, this is all we need to do before returning the connection object. If **java.sql.Connection** throws an SQLException, we will assume that this meant there was no instance if the JDBC driver, so we will instantiate the JDBC driv try the connection again. Since we are using Cloudscape in the same JVM as our application and the Enhydra server, we ar using the embedded JDBC driver for the connection. The **cloudscape.system.home** property must be set before instanti the driver. Properties can be set in your application like we did here or in the **cloudscape.properties** file.

```
p.put("cloudscape.system.home","/home/cloudscape/data");
```

Then, we instantiate the driver:

```
Class.forName("COM.cloudscape.core.JDBCDriver").newInstance();
```

Now that the driver is started, we attempt the connection again and return the connection object.

When you are done creating the **DemoConnect.jar** file compile it to create a DemoConnect.class.

```
 Javac  DemoConnect.java
```

When you are done creating **DemoConnect.class**, copy **DemoConnect.*** to **/home/jdbc/Paper/***. We now have al files necessary to start our application. Let's register our query tool with Enhydra.

Register and Start Your Application:

Open a web browser and go to the Enhydra administration interface:

1. http://{yourSystem}:8001
2. Log in using the default username and password:

   Username: admin
   Password: enhydra
3. In the Multiserver Admin bar on the left click **Add**.

   The "Add New Application/Servlet " dialogue appears
4. Select **War**.

   Fields will be added to the dialogue.
5. Enter

   Name = QueryTool
   Doc Root = /home/public_html/Paper
6. Click **OK** on the application successfully added dialogue.
7. Select the **Connections** tab
8. Click **Create**.

   The "Add New Connection" dialogue appears"
9. Enter:

   URL Prefix = /CloudscapePort
   Number = 1000

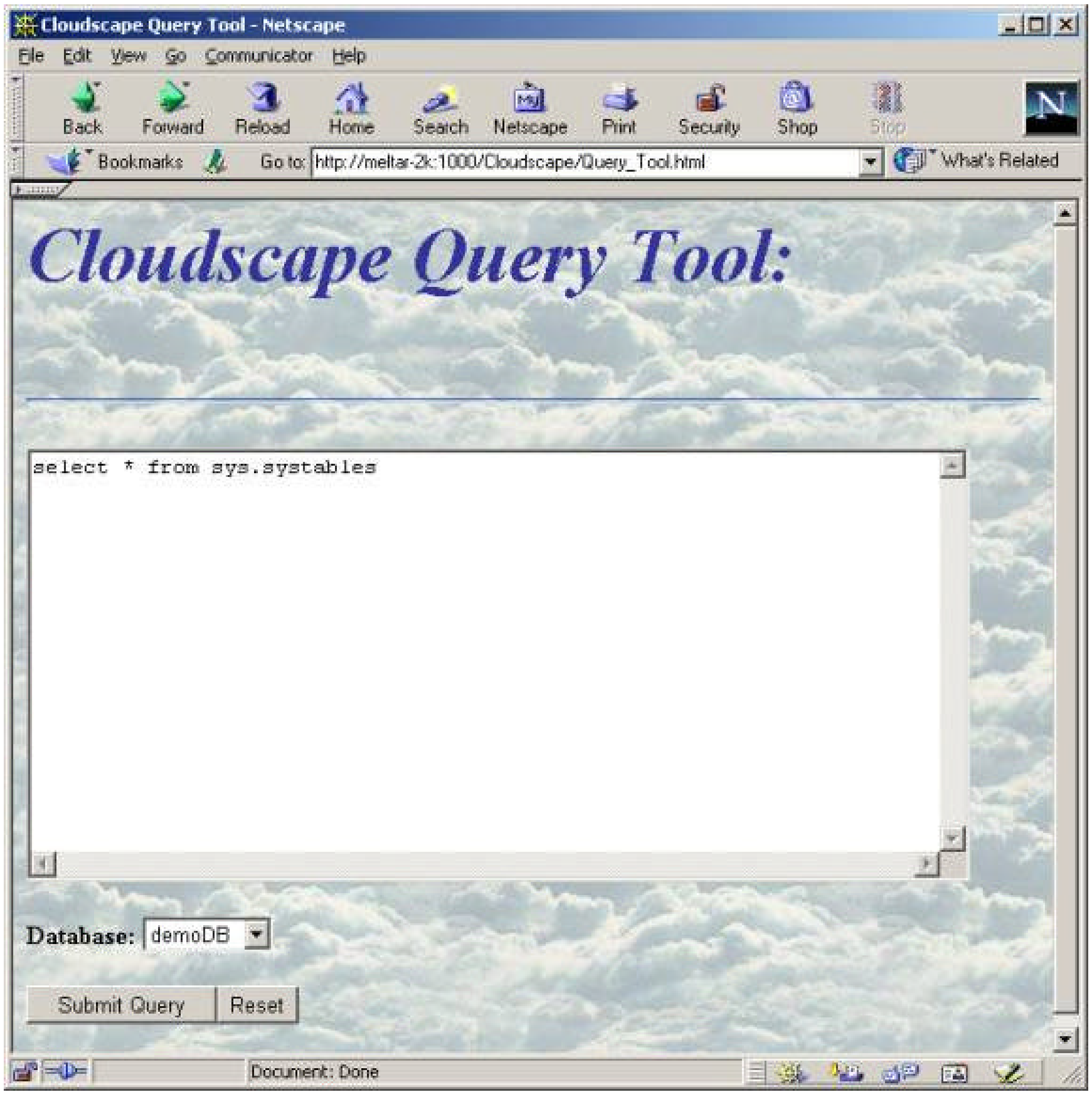
10. Click **OK**.

    The success dialog appears.
11. Click **OK** again.
12. In the Multiserver Admin bar on the left click **Save State** to save the changes
13. Click **OK**.

    The write warning dialogue appears.
14. Click **OK** again.

    The success dialog appears.
15. Select **QueryTool** from the list.
16. Click **Start**

Your application is now running. To use the application open a web browser and enter the URL: http://{yourSystem}:1000/Cloudscape/Query_Tool.html. You should see the HTML page you created.

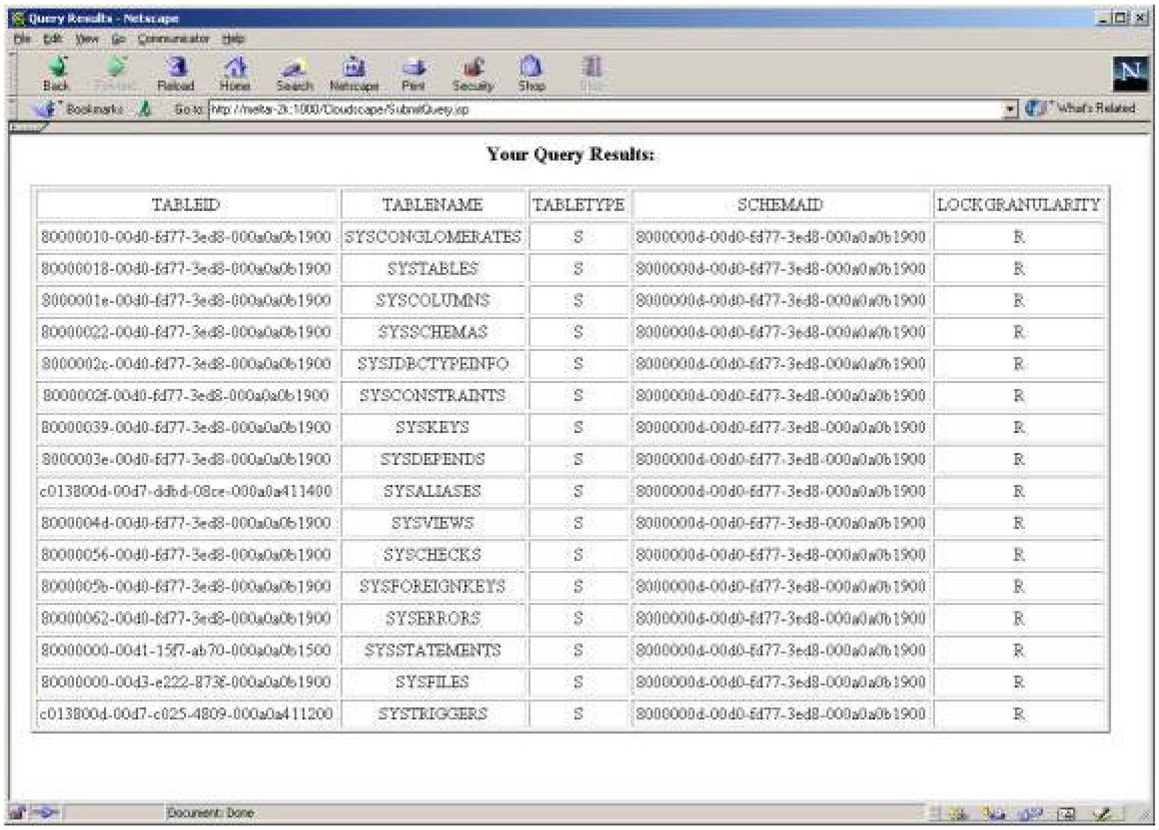

Cloudscape Query Tool - Netscape
File Edit View Go Communicator Help
Back Forward Reload Home Search Netscape Print Security Shop Stop
Bookmarks Go to: http://meltar-2k:1000/Cloudscape/Query_Tool.html What's Related
Cloudscape Query Tool:
Database: demoDB
Submit Query Reset
Document: Done

```
select * from sys.systables
```

Figure 1

Type the query “Select * from sys.systables” (Do not put a semi-colon “;” at the end of the query text) into the text windo

Click **Submit Query** to send the request to **SubmitQuery.jsp**. Your query results should look like Figure 2. Remember, th
time you run this tool, it will take a few minutes to complete the query because it must first create and start the database.


Query Results - Netscape
File Edit View Go Communicator Help
Back Forward Reload Home Search Netscape Print Security Shop Stop
Bookmarks Go to: http://nelta-2k:1000/Cloudscape/SubmitQuery.jsp
What's Related
Document: Done

Your Query Results:

TABLEID	TABLENAME	TABLETYPE	SCHEMAID	LOCKGRANULARITY
80000010-00d0-fd77-3ed8-000a0a0b1900	SYSCONGLOMERATES	S	8000000d-00d0-fd77-3ed8-000a0a0b1900	R
80000018-00d0-fd77-3ed8-000a0a0b1900	SYSTABLES	S	8000000d-00d0-fd77-3ed8-000a0a0b1900	R
8000001e-00d0-fd77-3ed8-000a0a0b1900	SYSCOLUMNS	S	8000000d-00d0-fd77-3ed8-000a0a0b1900	R
80000022-00d0-fd77-3ed8-000a0a0b1900	SYSSCHEMAS	S	8000000d-00d0-fd77-3ed8-000a0a0b1900	R
8000002c-00d0-fd77-3ed8-000a0a0b1900	SYSJDBCTYPEINFO	S	8000000d-00d0-fd77-3ed8-000a0a0b1900	R
8000002f-00d0-fd77-3ed8-000a0a0b1900	SYSCONSTRAINTS	S	8000000d-00d0-fd77-3ed8-000a0a0b1900	R
80000039-00d0-fd77-3ed8-000a0a0b1900	SYSKEYS	S	8000000d-00d0-fd77-3ed8-000a0a0b1900	R
8000003e-00d0-fd77-3ed8-000a0a0b1900	SYSDEPENDS	S	8000000d-00d0-fd77-3ed8-000a0a0b1900	R
c013800d-00d7-ddbd-08ce-000a0a411400	SYSALIASES	S	8000000d-00d0-fd77-3ed8-000a0a0b1900	R
8000004d-00d0-fd77-3ed8-000a0a0b1900	SYSVIEWS	S	8000000d-00d0-fd77-3ed8-000a0a0b1900	R
80000056-00d0-fd77-3ed8-000a0a0b1900	SYSCHECKS	S	8000000d-00d0-fd77-3ed8-000a0a0b1900	R
8000005b-00d0-fd77-3ed8-000a0a0b1900	SYSFOREIGNKEYS	S	8000000d-00d0-fd77-3ed8-000a0a0b1900	R
80000062-00d0-fd77-3ed8-000a0a0b1900	SYSERRORS	S	8000000d-00d0-fd77-3ed8-000a0a0b1900	R
80000000-00d1-15f7-ab70-000a0a0b1500	SYSSTATEMENTS	S	8000000d-00d0-fd77-3ed8-000a0a0b1900	R
80000000-00d3-e222-873f-000a0a0b1900	SYSFILES	S	8000000d-00d0-fd77-3ed8-000a0a0b1900	R
c013800d-00d7-c025-4809-000a0a411200	SYSTRIGGERS	S	8000000d-00d0-fd77-3ed8-000a0a0b1900	R

Figure 2

You have now created a useful query tool you can use with your Cloudscape Database.

Summary

In this example, we created a simple yet useful web-based query tool for your Cloudscape database. You can use the methodology used here to create any type of web application with dynamic content. For an example of an eBusiness applic
using this methodology, see "Bubbas CloudBooks: A Cloudscape ePerformance Test".

Resource Links

Downloads

Enhydra:
http://www.enhydra.org/software/downloads/enhydra/index.html
Blackdown JDK:
http://www.blackdown.org/Java-linux/mirrors.html
Red Hat Linux:
http://www.redhat.com/apps/download/

Reference

Installing Cloudscape:

http://www.cloudscape.com/support/doc_35/install/

Cloudscape JVM Compatibility Chart
http://www.cloudscape.com/Engineering/vm.html

Performance Tip: Using Stored Prepared Statements
i.CloudscapePerf-PrepStmts.html

Bubbas CloudBooks: A Cloudscape ePerformance Test
i-Cloudscape-Bubbas.html

Author:
Scott Fadden
Cloudsscape Performance
Informix Software, Inc.
scottfa@informix.com