DB

This topic explains how to configure and apply the DB tool in SOAtest and Virtualize. This tool allows you to query databases for validating SQL statements with SOAtest.

Sections include:

Understanding the DB Tool

This tool sends an inquiry to the database you specify, and the results from the query will be returned. SOAtest and/or Virtualize formats the results into XML; this allows XML based tools (XML Data Bank, XSLT Tool, etc.) to be chained to the output.

The DB tool can be used to call stored procedures (e.g., using PL./SQL) as described in Support for PL/SQL and Stored Procedures.

A Traffic Viewer is automatically attached to each DB tool. It lets you see the end result of the query that was sent to the server (taking into consideration parameterizations, variables, and so forth). In the response, JDBC-related objects are shown.

Configuring the DB Tool

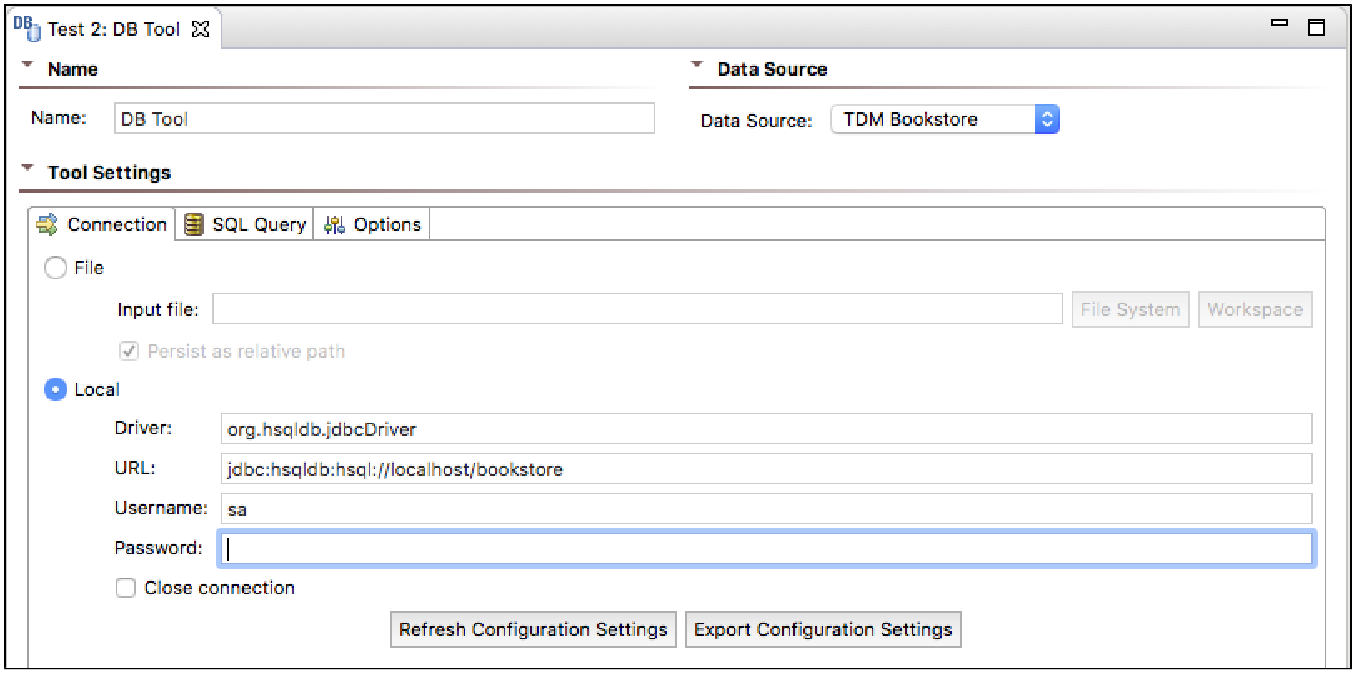
The following options can be configured in the DB tool:

General

- **Data Source:** Specifies the Data Source to be used for providing values. This menu is only available if a Data Source was added to the test or Action suite.

Connection Tab

There are several options for connecting the tool to the database. By default, you can configure a local connection using a file or by specifying the database driver settings.



Name: DB Tool

Data Source: TDM Bookstore

File

Input file:

Persist as relative path

Local

Driver: org.hsqldb.jdbcDriver

URL: jdbc:hsqldb:hsql://localhost/bookstore

Username: sa

Password:

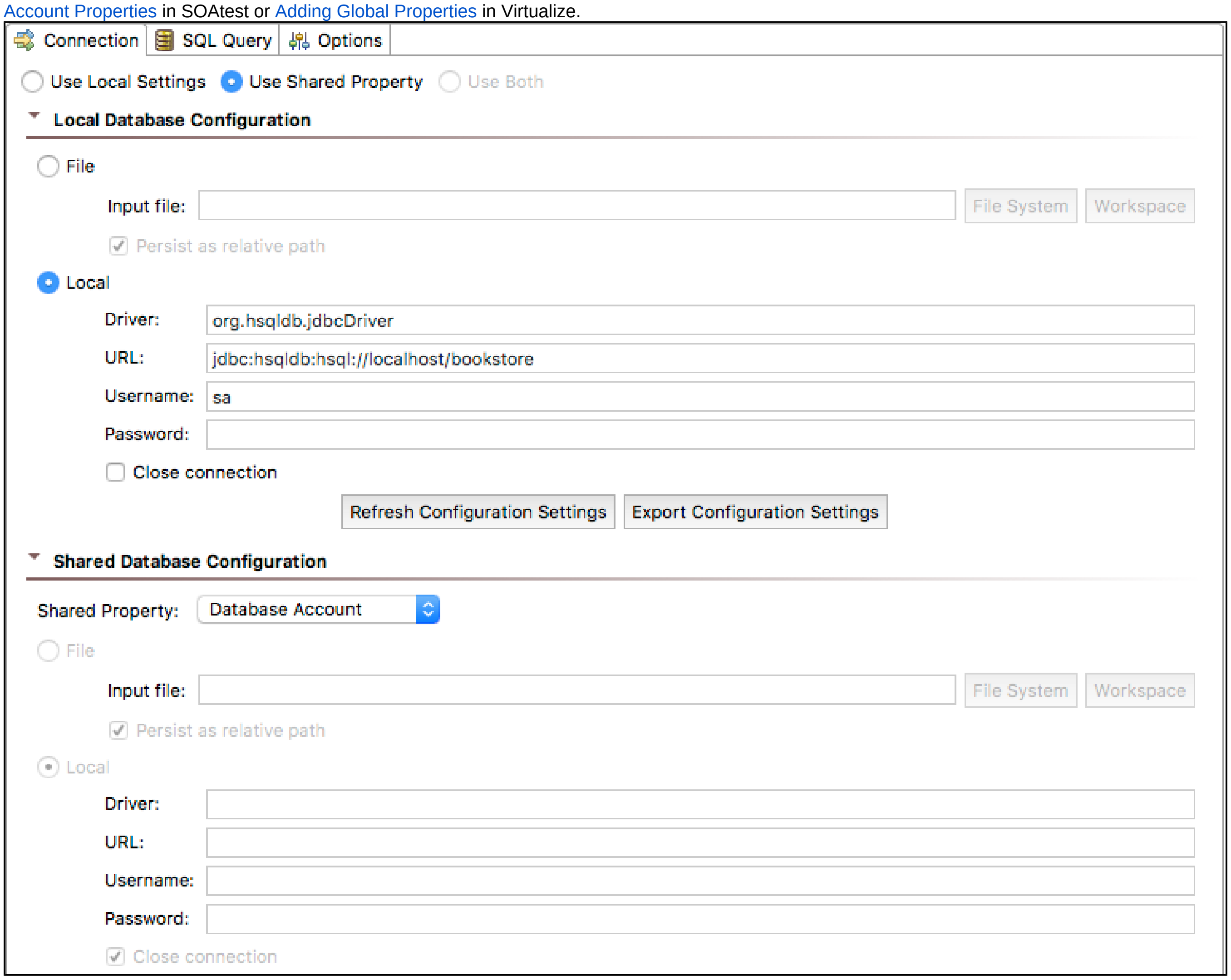
Close connection

Enable the **File** option and browse for the connection configuration file in your file system or workspace.

Enable the **Local** option and specify the **Driver**, **URL**, **Username**, and **Password** of the database to be queried. For more information on configuring database connection settings, see Database Configuration Parameters in SOAtest or Database Configuration Parameters in Virtualize.

You can also enable the **Close connection** option to close the connection if you are querying the database once and do not need to wait for another command. Do not enable this option when you plan to connect multiple DB tools to the same DB so that all tools can share a connection. Sharing the connection improves resource efficiency, as opposed to using resources on a new connection for each DB tool.

If your test suite has a Database Account shared property, you can connect **Use Local Settings,Use Shared Property**, or **Use Both.** These options enable you to use only settings within the corresponding DB tool, settings from a shared global database account (global database settings that you added to the test suite as a Global Property), or both. For more information on how to define shared properties and export them to a file, see Global Database Account Properties in SOAtest or Adding Global Properties in Virtualize.



Use Local Settings [✓ Use Shared Property] Use Both

Local Database Configuration

File

Input file:

[✓] Persist as relative path

[✓ Local]

Driver: org.hsqldb.jdbcDriver

URL: jdbc:hsqldb:hsql://localhost/bookstore

Username: sa

Password:

[ ] Close connection

Shared Database Configuration

Shared Property: Database Account

File

Input file:

[✓] Persist as relative path

[✓ Local]

Driver:

URL:

Username:

Password:

[✓] Close connection

Remember to check the Traffic tool connection ID

The Traffic tool shows a connection ID. Checking this ID will help you ensure that your DB tools are using the same connection.

SQL Query Tab

- **SQL Query:** Allows you to specify either a **Fixed** or **Parameterized** SQL statement.
  - If **Fixed** is selected, you can do finergrain parameterization by entering the special string `${column}` into the text box and selecting a data source from the **Data Source** drop-down menu. During runtime, the tool will look up the column in the specified data source. Any string value can be parameterized.
    For example, if a data source is created and has three rows that contain CA, NY, and WA, you can enter the following into the **Fixed** text box:

    ```
    SELECT *
    FROM People
    WHERE state = '${States}'
    ```

    When ran, the DB tool will create the following queries:

```
SELECT *
FROM People
WHERE state = "CA"

SELECT *
FROM People
WHERE state = "NY"

SELECT *
FROM People
WHERE state = "WA"
```

To access values from a Writable Data Source, use

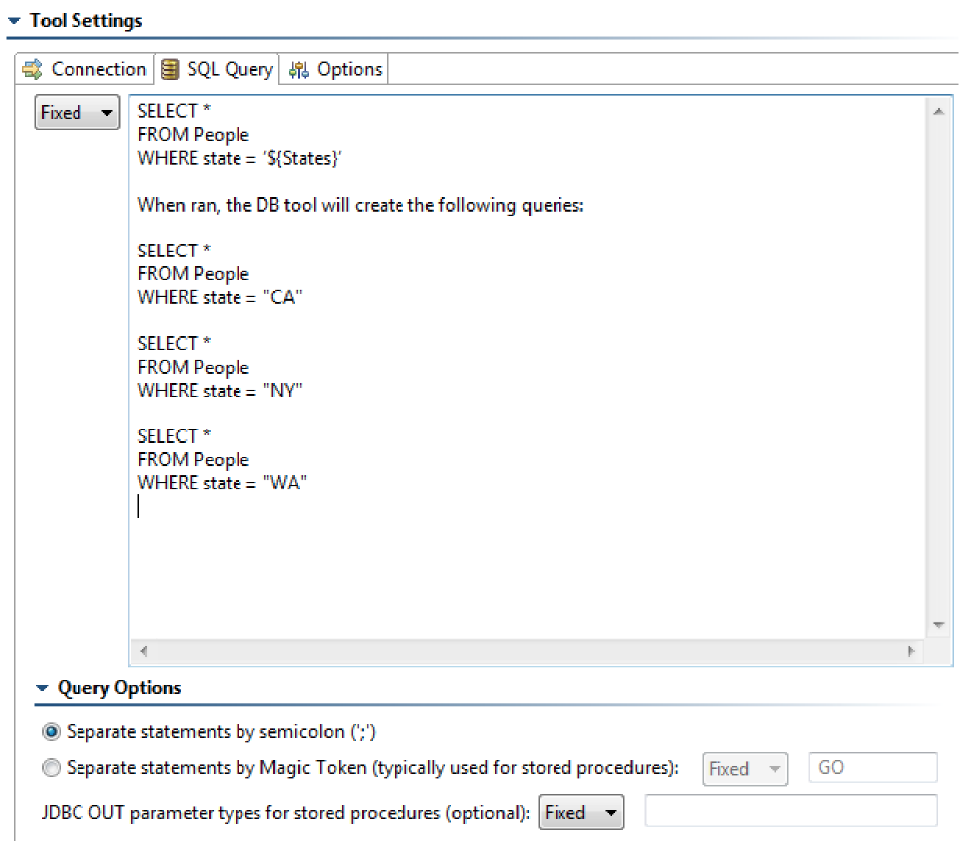
```
${<Data Source Name>: <Column Name>}
```

Note that there needs to be a space between the colon and <Column Name>

To access values from an XML Data Bank, use

```
${<Data Source Column Name mapped to Selected Element>}
```

- **Separate statements by semicolon (;):** If this option is enabled, the tool will divide the contents of the SQL query editor by semicolon. For more details, see Statement Separation Behavior.
- **Separate statements by Magic Token (typically used for stored procedures):** If this option is enabled, the tool will divide the contents of the SQL query editor by the provided magic token string. A magic token is a string that starts at the beginning of a line and contains only the magic token (with optional white space after the token). For more details, see Statement Separation Behavior.
- **JDBC OUT parameter types for stored procedures:** Allows you to invoke stored procedures and functions that take OUT parameters or return a value.
    - You can leave this field empty if you are executing a regular SQL query or a stored procedures that does not have any OUT parameters. See JDBC OUT Parameter Types for Stored Procedures below for details.
    - Note that for each DB tool, the same JDBC OUT parameters are sent to each callable statement. To use different out parameters, create a separate DB tool.



```
SELECT *
FROM People
WHERE state = '${States}'

When ran, the DB tool will create the following queries:

SELECT *
FROM People
WHERE state = "CA"

SELECT *
FROM People
WHERE state = "NY"

SELECT *
FROM People
WHERE state = "WA"
```

Query Options

(●) Separate statements by semicolon (';')
( ) Separate statements by Magic Token (typically used for stored procedures): Fixed GO
JDBC OUT parameter types for stored procedures (optional): Fixed

Statement Separation Behavior

With **Separate statements by semicolon (;)**, the tool will divide the contents of the SQL query editor by semicolon. With **Separate statements by Magic Token (typically used for stored procedures)**, the tool will divide the contents of the SQL query editor by the provided magic token string. A magic token is a string that starts at the beginning of a line and contains only the magic token (with optional white space after the token).

The separators you have selected (either semicolons or magic tokens) are *not* sent to the server. The DB tool will send each separated sub-string to the server as a separate JDBC call with the same connection. This allows a single DB tool to use multiple queries. Depending on the SQL server JDBC driver, it may be possible for multiple queries to be sent in the same JDBC call:

- **SQL Server:** Allows multiple statements in the same JDBC call (optionally separated by semicolons).
- **MySQL:** Requires "allowMultiQueries=true" to be added to the JDBC URL to allow multiple queries in the same JDBC call (separated by semicolon). For example "jdbc:mysql://boa:3306/test?allowMultiQueries=true". This feature also requires version 4.0 of the MySQL driver or later.
- **DB2:** Accepts only one statement per JDBC call.
- **Oracle:** Accepts only one statement per JDBC call.

If multiple statements need to be used by a single DB tool, they must be separated by a magic token (this is due to the restrictions in DB3 and Oracle). For example, if you choose to separate by a magic token of "GO" (the default) and you enter the following query string, two JDBC calls will be sent—each containing two SQL statements:

```
create table pr88467(dkey int, svalue varchar(30));
insert into pr88467 (dkey,svalue) values (1,"A"),(2,"B"),(3,"C");
GO
select * from pr88467;
drop table pr88467;
```

When creating or calling stored procedures, separation by magic token must be used and the stored procedure must be the only statement in its group. When creating procedures, this is required because procedures will usually contain required nested semicolons that would be removed if using the "separate by semicolon" option. When calling procedures, they must be separated because the DB tool needs to be able to detect them and send them as a CallableStatement to the JDBC driver (with the optional out parameters). For example:

```
create or replace function pr83185_func2(s NUMBER) return SYS_REFCURSOR as mycursor SYS_REFCURSOR;
begin
open mycursor for select * from EMPLOYEES where SALARY > s;
return mycursor;
end;
GO
? = call pr83185_func2(15000)
```

The above will issue two JDBC calls:

- One to create the procedure.
- A second on to call the procedure and get the results.

Options Tab

- **Fail on SQL Exception:** Specifies whether or not the tool should fail when a SQL Exception is encountered. Note that if a validation tool is chained as an output to this DB tool, the outcome of that validation will determine this tool’s success or failure, and this setting is not applicable.
- **Auto-commit database changes after running query:** Specifies whether you want to use auto-commit mode. In auto-commit mode (the default), the SQL query automatically commits changes to the database. Otherwise, changes will be visible only for the duration of the connection.
- **Rollback database changes after running query:** Specifies if rollback is enabled. When this is enabled, the changes will be immediately reverted.

  More on Auto-Commit and Rollback

  Auto-commit mode supersedes rollback mode: If auto-commit mode is on, rollback mode is automatically disabled. If you want to make database changes that are visible only for the duration of the tool, then disable auto-commit mode. If you want to revert changes mid-tool, then use the rollback option.

- **XML encoding:** Specifies whether XML characters are encoded. By default, XML values are encoded as unicode characters, but you can choose one of the following options to change how characters are encoded:
  - Choose **Unicode** to preserve all unicode characters supported by XML specifications. Only restricted XML characters, such as "<" and "&", will be encoded.
  - Choose **ASCII** to preserve all ASCII characters. Non-ASCII unicode characters, as well as restricted XML characters, will be encoded.
  - Choose **None** to disable XML encoding. Characters from the field value, including restricted XML characters, will be represented as-is in the final document.

- **Separate column names from values in the XML output:** Determines column names should be separated from values. This may result in output that is more difficult for a human to read, so it is disabled by default.
- **Use single result format if only one result:** Determines how the tool formats the output XML. If this option is enabled, the results will not be nested in a "<results>" xml tag if possible. If there are multiple results, then the multi-output format will be used even if this option is enabled.

Single output format:

```
<updatedRows>0</updatedRows>
```

Multi-output format:

```
<results>
        <updatedRows>0</updatedRows>
</results>
```

With multiple results:

```
<results>
        <updatedRows>0</updatedRows>
        <updatedRows>0</updatedRows>
        <updatedRows>0</updatedRows>
</results>
```

JDBC OUT Parameter Types for Stored Procedures

The **JDBC OUT parameter types for stored procedures** field allows you to invoke stored procedures and functions that take OUT parameters or return a value. You can leave this field empty if you are executing a regular SQL query or a stored procedures that does not have any OUT parameters.

The types you list in that field can be separated with spaces or commas, and the type names correspond to the field names in the class java.sql.Types (http://docs.oracle.com/javase/8/docs/api/java/sql/Types.html) or oracle.jdbc.OracleTypes class fields (http://download.oracle.com/docs/cd/A97329_03/web.902/q20224/oracle/jdbc/OracleTypes.html). will look for the type you provide in java.sql.Types first. If it is not found there, it will look for it in oracle.jdbc.OracleTypes. The type name you provide is passed to the JDBC CallableStatement.registerOutParameter().

Examples

- To invoke a stored procedure that takes 4 IN parameters, use `CALL MYPROC(XMLTYPE('<hello>Some XML</hello>'), 55, 'character large object', 'some text')`
- To invoke a stored procedure that takes 4 parameters (Oracle XMLType, INTEGER, CLOB and VARCHAR respectively)—assuming that the 2nd (INTEGER) and 4th (VARCHAR) parameters are OUT parameters— then the SQL query content you provide can look like
  `CALL MYPROC(XMLTYPE('<hello>some XML</hello>'), ?, 'character large object', ?)`
  and the JDBC OUT parameter types field would have the content
  `INTEGER, VARCHAR`
- To invoke a function that returns a VARCHAR and takes 5 OUT parameters with the types OUT INTEGER, IN INTEGER, Oracle XMLTYPE, Oracle CLOB, VARCHAR and DATE, the SQL query would look like
  `? = CALL MYFUNCTION(XMLTYPE('<hello>Some XML</hello>'), 'character large object', ?, 33, ?, ?, ?, ?)`
  and the JDBC OUT parameter types field would have the content
  `VARCHAR, INTEGER, SYS.XMLTYPE, CLOB, VARCHAR, DATE`
- To invoke the same function as above, but using data source values for some of IN parameters, use
  `? = CALL MYFUNCTION(XMLTYPE('${My CML Content Columna name}'), 'character large object', ?, ${Column Name 2}, ?, ?, ?, ?)`
  DB tool will serialize the function return value and the OUT parameters in XML, similar to how it does with regular SQL queries. This allows you to see the execution results and chain XML validation/extraction tools such as XML Assertor and XML Data Bank.
  If the return value or any of the OUT parameters is a CURSOR type, the return value will be handled as a JDBC ResultSet, and serialize it into XML nested within the overall result XML—the same way that it operates on regular query results.
- To invoke a function that takes an IN integer parameter and returns a CURSOR, you would use
  `? = CALL MYFUNC(15000)`
  and the parameter types field would contain
  `CURSOR`
  In the case of CURSOR return or OUT parameters, DB tool returns them in a ResultSet XML format embedded within the overall XML output of DB tool, which includes column names associated with the CURSOR. Note that if you receive a message like SQLException "Bigger type length than maximum," you are probably encountering a bug in the Oracle driver. If this occurs, please try a newer Oracle JDBC driver. Parasoft has verified that ojdbc14.jar version 10.2.0.4 does not suffer from this issue.

Oracle Type Extensions

Oracle databases support various proprietary data types. If your database utilizes these data types, you must add the appropriate jar(s) to classpath in the Preferences panel (under Parasoft> JDBC Drivers).

For example, Oracle supports oracle.xdb.XMLType. This is available with xdb.jar and xmlparserv2.jar, which are shipped with specific Oracle applications. It is also available from the XML Developer's Kit (XDK) Java, which is available at http://www.oracle.com (click Downloads, click the link for the XML Developer Kit, then select the kit for any platform). Note that this kit contains other files, but you need only the 2 listed above.

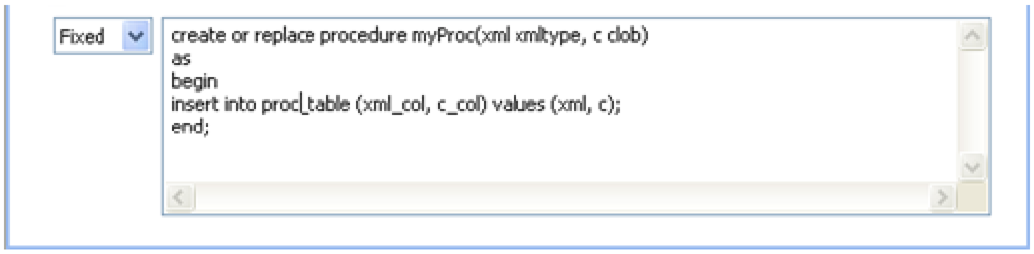
Both of these jars should be added to the JDBC classpath in the Preferences panel (under Parasoft> JDBC Drivers).

Support for PL/SQL and Stored Procedures

Since stored procedures can drive the business logic of today's systems, they need validation—just like any other component in the system. Procedures (also known as functions, procedures, packages, and triggers) to an Oracle database can also be stored. It can also call those procedures over and over. This is especially useful for complex queries that need to be executed frequently.

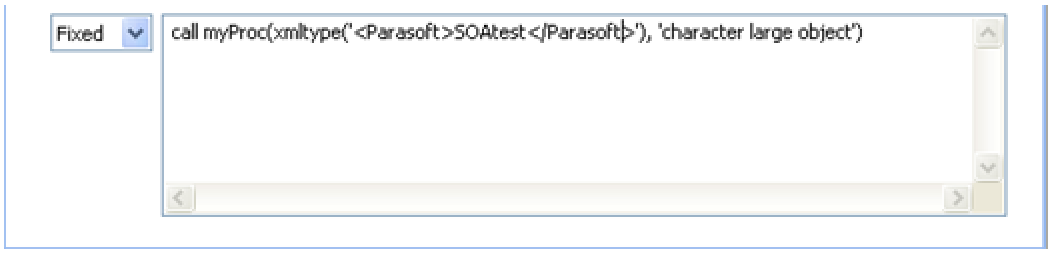
PL/SQL (Procedural Language/Structured Query Language), Oracle's proprietary procedural extension to the SQL database language, is used in the Oracle database. Other SQL database management systems offer similar extensions to the SQL language.

The following image shows an example of how to configure a DB tool query (in the SQL Query tab) to create or replace a stored procedure:



```
create or replace procedure myProc(xml xmltype, c clob)
as
begin
insert into proc_table (xml_col, c_col) values (xml, c);
end;
```

The following image shows an example of how to configure a DB tool query to call a stored procedure:



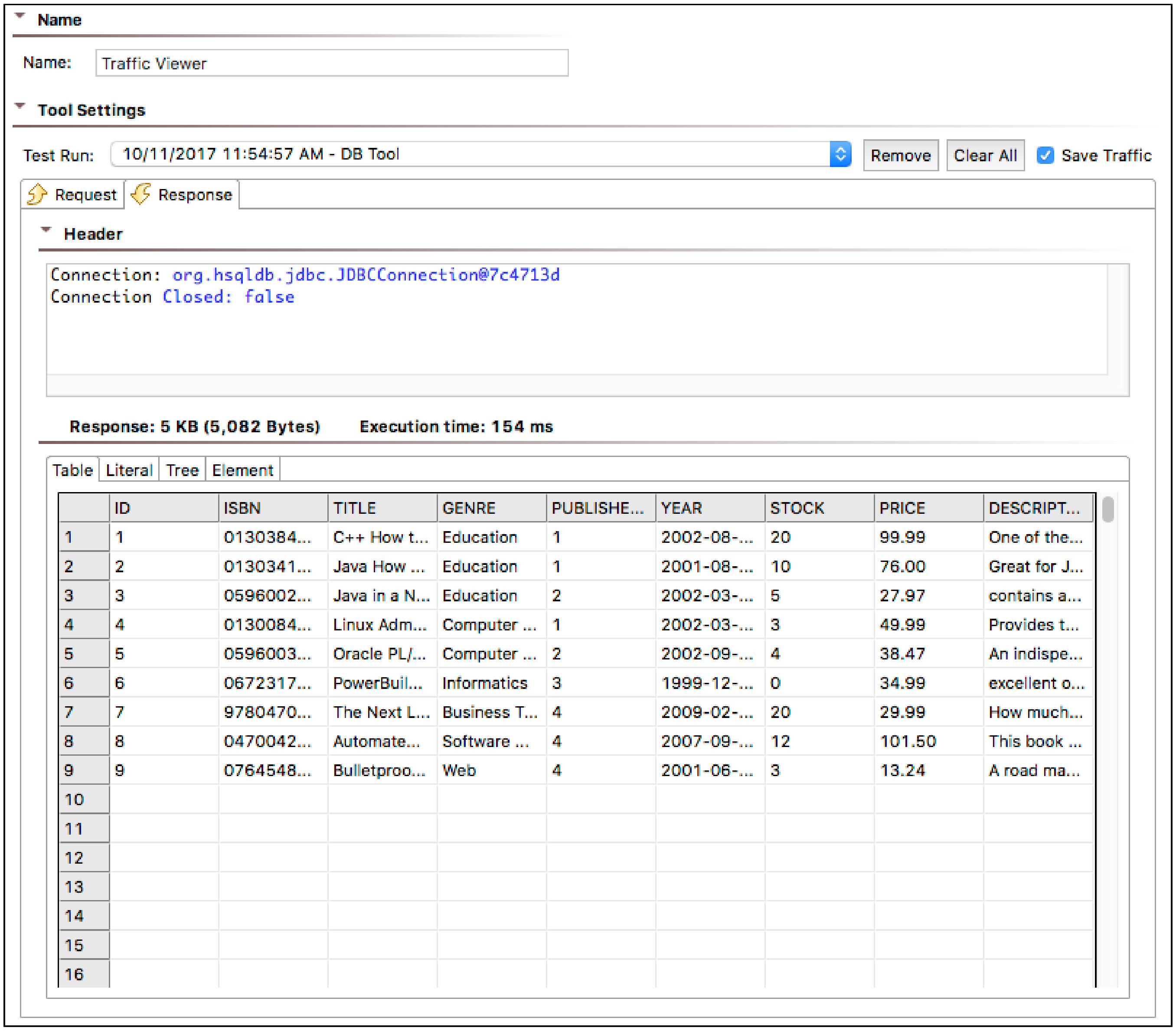
```
call myProc(xmltype('<Parasoft>SOAtest</Parasoft>'), 'character large object')
```

Viewing Query Results in the Traffic Viewer

When a Traffic Viewer tool is attached to a DB tool and a "SELECT" statement is executed by the DB tool, the Traffic Viewer tool's Response area will show a table that contains the results of the DB query.

When a DB tool executes multiple SQL statements and the XML response includes multiple result sets, you can use the combo box (above the table when there are multiple result sets) to determine which result set's traffic is shown.

Here, the traffic viewer is showing the results for result set 2:



Name

Name: Traffic Viewer

Tool Settings

Test Run: 10/11/2017 11:54:57 AM - DB Tool   Remove   Clear All   ✓ Save Traffic

Request   Response

Header

```
Connection: org.hsqldb.jdbc.JDBCConnection@7c4713d
Connection Closed: false
```

Response: 5 KB (5,082 Bytes)   **Execution time: 154 ms**

Table   Literal   Tree   Element

	ID	ISBN	TITLE	GENRE	PUBLISHE...	YEAR	STOCK	PRICE	DESCRIPT...
1	1	0130384...	C++ How t...	Education	1	2002-08-...	20	99.99	One of the...
2	2	0130341...	Java How ...	Education	1	2001-08-...	10	76.00	Great for J...
3	3	0596002...	Java in a N...	Education	2	2002-03-...	5	27.97	contains a...
4	4	0130084...	Linux Adm...	Computer ...	1	2002-03-...	3	49.99	Provides t...
5	5	0596003...	Oracle PL/...	Computer ...	2	2002-09-...	4	38.47	An indispe...
6	6	0672317...	PowerBuil...	Informatics	3	1999-12-...	0	34.99	excellent o...
7	7	9780470...	The Next L...	Business T...	4	2009-02-...	20	29.99	How much...
8	8	0470042...	Automate...	Software ...	4	2007-09-...	12	101.50	This book ...
9	9	0764548...	Bulletproo...	Web	4	2001-06-...	3	13.24	A road ma...
10									
11									
12									
13									
14									
15									
16									