SQL Transformer (2.1 legacy document)

Table of contents

Table of contents

Warning:
This document was copied as is from the Cocoon 2.1 documentation, but has not yet been fully reviewed or moved to its new home.

1. Introduction

The purpose of the SQLTransformer is to query a database and translate the result to XML. To retrieve the information from the database, you are not restricted to use simple SQL statements (e.g. select, insert, update), it is also possible to use stored procedures. In combination with other transformers (e.g. FilterTransformer), this one can be very powerful.

- Name: sql
- Class: org.apache.cocoon.transformation.SQLTransformer
- Cacheable: no

2. Basic functionality

To be able to query a database, we need XML that describes exactly what we want to do. The general structure of this input XML is as follows:

```
<page> <execute-query xmlns="http://apache.org/cocoon/SQL/2.0"> <query> <!-- here comes the SQL statement or stored procedure --> </query> </execute-query> </page>
```

Nothing prevents you from putting other XML around the execute-query element. Any element not in the SQL namespace will stay untouched. The format of the SQL statement or the stored procedure is exactly the same as if you would call it directly from java with a prepared statement or a callable statement.

The query element has the following optional attributes:

1. **name**: Naming a query implicates naming the corresponding rowset (see below). When you have a sequence of queries you want to execute, it can be handy give them a name. To process the retrieved data of a certain query, you can use another transformer to check the name of the rowset and to execute the necessary business logic on it.
   usage: <query name="myName">
2. **isstoredprocedure**: When you want to use stored procedures, you have to explicitly add this attribute to the query element. By default, the transformer assumes that you want to execute a SQL statement.
   usage: <query isstoredprocedure="true">

Here is an example of how the input XML might look like:

```
<page> <title>Hello</title> <content> <para>This is my first Cocoon page filled with sql data!</para> <execute-query xmlns="http://apache.org/cocoon/SQL/2.0"> <query name="department"> select id,name from department_table </query> </execute-query> </content> </page>
```

You can use the file generator to retrieve the XML from the filesystem. To invoke the SQLTransformer you have to add following to the sitemap:

```
<map:transform type="sql"> <map:parameter name="use-connection" value="personnel"/> <map:parameter name="show-nr-of-rows" value="true"/> <map:parameter name="clob-encoding" value="UTF-8"/> </map:transform>
```

The use-connection parameter defines which connection, defined under the datasources element in cocoon.xconf, the SQLTransformer has to use to retrieve the data.

The show-nr-of-rows instructs the transformer to count the number of rows in the resultset explicitly and to set the result as attribute to the rowset element. This attribute is only useful in combination with a sql statement, not with stored procedures. If a stored procedure returns a resultset and you want to know how many rows it contains, you have to count the number of rows in another transformer or your stored procedure has to return it also (last solution is the best one).

The clob-encoding parameter defines what encoding should be used in getting content from CLOB columns.

The output XML will look as follows:

<page> <title>Hello</title> <content> <para>This is my first Cocoon page filled with sql data!</para> <rowset nrofrows="2" name="department" xmlns="http://apache.org/cocoon/SQL/2.0"> <row> <id>1</id> <name>Programmers</name> </row> <row> <id>2</id> <name>Loungers</name> </row> </rowset> </content> </page>

If you use this in combination with the simple-sql2html.xsl stylesheet,

<map:transform src="stylesheets/simple-sql2html.xsl"/>

you will get a more visually attractive page.

See below for a more in depth example with stored procedures.

By now you should be able to use the SQLTransformer, but there are some more options you might find useful...

3. Advanced functionality

3.1. Substitution

Sometimes you need more information before you can execute a query, e.g. the name of the user that is currently logged on your site. This information is only available at runtime and hence can only be substituted in the query when available.

To pass this information to the SQL statement, the input XML has to look like this:

<page xmlns:sql="http://apache.org/cocoon/SQL/2.0"> <execute-query xmlns="http://apache.org/cocoon/SQL/2.0"> <query> select id,name from employee_table where name = '<sql:substitute-value sql:name="username"/>' </query> </execute-query> </page>

The substitution is done by the SQLTransformer before it executes the query (before it calls the method prepareStatement!). For this, the transformer has to be given the necessary values via the sitemap (as parameter):

<map:transform type="sql"> <map:parameter name="use-connection" value="personnel"/> <map:parameter name="show-nr-of-rows" value="true"/> <map:parameter name="username" value="Stefano Mazzocchi"/> </map:transform>

Whenever the transformer encounters a substitute-value element for which the attribute name contains the value username, it will replace this element with the value Stefano Mazzocchi.

The output XML will be as follow:

<page xmlns:sql="http://apache.org/cocoon/SQL/2.0"> <rowset nrofrows="1" xmlns="http://apache.org/cocoon/SQL/2.0"> <row> <id>2</id> <name>Stefano

Mazzocchi</name> </row> </rowset> </page>

It is also possible to use substitution in combination with stored procedures.

3.2. Ancestors

This functionality is best described by a simple example.

Take following input XML:

<page xmlns:sql="http://apache.org/cocoon/SQL/2.0"> <execute-query xmlns="http://apache.org/cocoon/SQL/2.0"> <query name="department"> select id,name from department_table </query> <execute-query> <query name="employee"> select id,name from employee_table where department_id = <ancestor-value sql:name="id" sql:level="1"/> </query> </execute-query> </execute-query> </page>

The first query will retrieve all id's and name's from the department_table table. For each id that comes from the department_table, the second query, in which the ancestor-value element will be replaced by the id, will be executed. The above example will be transformed to the following XML:

<page xmlns:sql="http://apache.org/cocoon/SQL/2.0"> <rowset nrofrows="2" name="department" xmlns="http://apache.org/cocoon/SQL/2.0"> <row> <id>1</id> <name>Programmers</name> <rowset nrofrows="2" name="employee"> <row> <id>1</id> <name>Donald Ball</name> </row> <row> <id>2</id> <name>Stefano Mazzocchi</name> </row> </rowset> </row> <row> <id>2</id> <name>Loungers</name> <rowset nrofrows="1" name="employee"> <row> <id>3</id> <name>Pierpaolo Fumagalli</name> </row> </rowset> </row> </rowset> </page>

3.3. in- and out-parameters

Stored procedures can return data as a parameter. To make use of this functionality in java, you have to register these parameters as *out parameters*. Since this information is application specific, the SQLTransformer uses reflection to retrieve the data in the right format. For this, an extra element is needed in the input XML:

<out-parameter sql:nr="1" sql:name="code" sql:type="java.sql.Types.INTEGER"/>

where:

1. **nr**: The targeted parameter number that will return data of a certain type.
2. **type**: The type of data that will be returned (defined in java.sql.Types or in database specific drivers, e.g. oracle.jdbc.driver.OracleTypes). Once the stored procedure returns data in the parameters, the stored procedure tries to process them. If the returned parameter is an instance of ResultSet, it will be translated to XML as we saw before. In all the other situations the SQLTransformer will convert the parameter to a string.

This is an example of how to call an oracle stored procedure and process it with the SQLTransformer:

<page xmlns:sql="http://apache.org/cocoon/SQL/2.0"> <execute-query xmlns="http://apache.org/cocoon/SQL/2.0"> <query isstoredprocedure="true" name="namesearch"> begin QUICK_SEARCH.FIND_NAME('<sql:substitute-value sql:name="username"/>',?,?,?); end; </query> <out-parameter sql:nr="1" sql:name="code" sql:type="java.sql.Types.INTEGER"/> <out-parameter sql:nr="2" sql:name="nrofrows" sql:type="java.sql.Types.INTEGER"/> <out-parameter sql:nr="3" sql:name="resultset" sql:type="oracle.jdbc.driver.OracleTypes.CURSOR"/> </execute-query> </page>

The SQLTransformer will create 3 elements, respectively code, nrofrows and resultset under the element namesearch. Since the type oracle.jdbc.driver.OracleTypes.CURSOR corresponds to a

ResultSet, a rowset element will be created, containing all the data of the resultset. It is also possible to use an *in-parameter* element, e.g. <in-parameter sql:nr="1" sql:value="1"/>. This functionality is only provided to be complete, because it is available in Java itself. You can also use the *in-parameter* in combination with a SQL statement. Used in combination with an *out-parameter*, a *?-parameter* can be an *in-parameter* and an *out-parameter* at the same time.

4. Combined with other transformers

4.1. Filtertransformer

When you query a database and it returns too many rows to process at once, you might want to take a block of elements, process this block and ignore the rest for now. You can best compare it to a search on Google: they only return 10 results in one time, for more results you have to click on another block (page). It wouldn't be wise to process more than 10 elements in the pipeline if you only need to display 10 elements.

Assume that a query returns 56 row elements (by using the SQLTransformer) and that you only want to display the first 10 elements:

Output XML from the SQLTransformer:

<rowset nrofrows="56" name="test" xmlns="http://apache.org/cocoon/SQL/2.0"> <row> <!-- db record --> </row> <row> <!-- db record --> </row> ... <row> <!-- db record --> </row> </rowset>

By adding following lines to the sitemap, just under the SQLTransformer, you restrict the results to 10 elements in the first block:

<map:transform type="filter"> <map:parameter name="element-name" value="row"/> <map:parameter name="count" value="10"/> <map:parameter name="blocknr" value="1"/> </map:transform>

output XML:

<rowset nrofrows="56" name="test" xmlns="http://apache.org/cocoon/SQL/2.0"> <block id="1"> <row> <!-- db record --> </row> <!-- total of 10 rows --> <row> <!-- db record --> </row> </block> <block id="2"/> <block id="3"/> <block id="4"/> <block id="5"/> <block id="6"/> </rowset>

To make it more dynamically, put something like {reqCount} and {reqBlock} in the values for *count* and *blocknr* respectively. These can be parameters from the request and they can be passed to the sitemap with an action.

The FilterTransformer is a standalone component; you don't need to use it in combination with the SQLTransformer.

4.2. WriteDOMSessionTransformer

If you only use the FilterTransformer in combination with the SQLTransformer, you have to query the database each time the user wants to see another part of the result. You can better store the result in the session after the first request and retrieve the result from the session for the subsequent requests. This can be done by using a selector, which checks if the data is available in the session or not.

WriteDOMSessionTransformer can build a DOM starting from a given element (which will be the root of the DOM tree) and store it in the session. If you want to store the result of a query, you have to add following to the sitemap:

<map:transform type="writeDOMsession"> <map:parameter name="dom-name" value="DBresult"/> <map:parameter name="dom-root-element" value="rowset"/> </map:transform>

The transformer will build a DOM tree with rowset as root element and will store it in the session with the name DBresult.

Most of the times, it is not smart to keep the output XML of the SQLTransformer in the session. Check if it is better to do the necessary transformations first, so that you get a smaller DOM, and then put the result in the session. You probably will be able to use the FilterTransformer on the transformed XML also.

The WriteDOMSessionTransformer is a standalone component, you don't need to use it in combination with the SQLTransformer.

4.3. ReadDOMSessionTransformer

Simply transforms a DOM to SAX events, which can be used further on in the pipeline. Once you stored the result of a query in the session with the WriteDOMSessionTransformer, you can read it again with the ReadDOMSessionTransformer:

<map:transform type="readDOMsession"> <map:parameter name="dom-name" value="DBresult"/> <map:parameter name="trigger-element" value="users"/> <map:parameter name="position" value="after"/> </map:transform>

In this example the SAX events, that come from the DOM tree stored in the session with name DBresult, will be added after the users element. This means as soon that the transformer encounters the end element users, it will start to generate SAX events from the DOM tree. There are three possible positions, before, in and after:

1. **before** means that when the transformer encounters the users element, it will FIRST translate the DOM tree to SAX events and THEN it will continue to forward the other SAX events (starting with users).
2. **in** means that the transformer will forward the start element event for users and that it IMMEDIATELY starts to generate SAX events from the DOM tree. After that, it will continue to forward the child elements of users and then all the other elements.
3. **after** means that the transformer starts to generate SAX events from the DOM tree just after it has forwarded the end element users.

The ReadDOMSessionTransformer is a standalone component, you don't need to use it in combination with the WriteDOMSessionTransformer.

That's it,

Sven Beauprez

1. Comments

add your comments