XML with Virtuoso and SQLX

Introduction

Contemporary relational database vendors must provide query results in a form that can be consumed by popular XML technologies, such as parsers and XSL-T processors. The most popular mechanism for generating XML data from relational queries is known as or SQLX, an ISO /ANSI standard.

Gentle Introduction to Virtuoso

OpenLink Software's Virtuoso Universal

Server is a virtual database integration engine joined to a highly integrated web application platform. Virtuoso unifies data storage systems and provides a powerful suite of application delivery services. These features allow Virtuoso to act as a catalyst for the creation of composite applications based on Web Services standardsSQLX in Virtuoso

Virtuoso's SQLX features apply to native relational tables[1], and to tables linked via the VDB. SQLX comes to the rescue when the need arises to create XML document instances from relational data. Virtuoso's power is exemplified by its ability to invoke SQLX against attached databases that do not support native XML data types.

Becoming Familiar with SQLX

SQLX is a SQL superset introduced via function calls, rather than changes to the SQL language syntax. At first glance, SQLX may appear a bit awkward when juxtaposed with traditional SQL. Compared to other programmatic methods for generating XML, this syntax of function calls is clean, logical, and fits the workaday model of forming relational queries.

SQLX has many available functions, four of which are commonly utilized:

XMLElement, XMLForest, XMLAttributes, and XMLAgg.

XMLElement

XMLElement is the most common function called in SQLX queries. The XMLElement function creates an XML element, including attributes and/or child elements.

XMLElement is an open-ended function, allowing the arbitrary addition of elements as child nodes, by simply appending as arguments.

Example:

```
SELECT XMLELEMENT('Emp',
   XMLELEMENT('ID',
"EmployeeID"),
   XMLELEMENT('FirstName',
"FirstName"),
   XMLELEMENT('LastName',
"LastName")
)
FROM "Demo"."demo"."Employees"
WHERE
"EmployeeID" = 1
```

This example queries the Employees table for a row matching EmployeeID = 1, and creates a document with a root element of 'Emp' containing data for that row. The resulting document will result:

```
<Emp>
<ID>1</ID>
<FirstName>Nancy</FirstName>
<LastName>Davolio</LastName>
</Emp>
```

This verbose query construct is a result of XMLElement function calls used to build the children of the 'Emp' element. The XMLForest function is a more compact method to accomplish the same task.

XMLForest

XMLForest is a shortcut for generating a set of elements with only columnar content. It takes as its arguments a set of column names or aliased column names.

Example:

```
SELECT XMLELEMENT('Emp',
   XMLFOREST("EmployeeID"
AS ID, "FirstName", "LastName")
)
FROM
"Demo"."demo"."Employees"
WHERE
"EmployeeID" = 1
```

This example queries the Employees table for the row matching EmployeeID = 1, creating a document with a root element of 'Emp' containing data for that row. The resulting document will look like this:

```
<Emp>
<ID>1</ID>
<FirstName>Nancy</FirstName>
<LastName>Davolio</LastName>
</Emp>
```

The results of this query is identical to the XMLElement example, but the syntax is more compact.

Identifying information is often exposed as an attribute within a containing element. In order to promote the 'ID' element to the status of an attribute in the 'Emp' element, use the XMLAttributes function.

XMLAttributes

The XMLAttributes function operates in a similar fashion to the XMLForest method in that it generates a set of attributes using columnar content. It takes as its arguments a set of column names or aliased column names. The function must be included as the first child arguments to the XMLElement function.

Example:

```
SELECT XMLELEMENT('Emp',
XMLATTRIBUTES("EmployeeID" AS ID),
   XMLFOREST("FirstName",
"LastName")
)
FROM
"Demo"."demo"."Employees"
WHERE
"EmployeeID" = 1
```

The result of this query yields the desired effect of promoting the EmployeeID value to the status of an attribute rather than an element.

```
<Emp ID="1">
<FirstName>Nancy</FirstName>
<LastName>Davolio</LastName>
</Emp>
```

The XMLAgg (aggregate) function is used to perform a query returning multiple rows.

XMLAgg

XMLAgg aggregates a set of rows in the result set, emitting the XML specified as the XMLAgg function's argument for each row that is processed.

Example:

```
SELECT XMLELEMENT('Emps',
  XMLAGG(
  XMLELEMENT('Emp',
  XMLATTRIBUTES("EmployeeID" AS ID),
    XMLFOREST("FirstName", "LastName")
   )
  )
)
FROM "Demo"."demo"."Employees"
```

This example queries the Employees table for all rows, and then creates an XML document with a root element of 'Emps' containing several child elements. Returned rows are aggregated, creating a 'Emp' element for each. The resulting document look like this:

```
<Emps>
<Emp ID="1">
<FirstName>Nancy</FirstName>
<LastName>Davolio</LastName>
</Emp>
<Emp ID="2">
<FirstName>Andrew</FirstName>
<LastName>Fuller</LastName>
</Emp>
<Emp ID="3">
<FirstName>Janet</FirstName>
<LastName>Leverling</LastName>
</Emp>
<Emp ID="4">
<FirstName>Margaret</FirstName>
<LastName>Peacock</LastName>
</Emp>
<Emp ID="5">
<FirstName>Steven</FirstName>
<LastName>Buchanan</LastName>
</Emp>
<Emp ID="6">
<FirstName>Michael</FirstName>
<LastName>Suyama</LastName>
</Emp>
<Emp ID="7">
<FirstName>Robert</FirstName>
<LastName>King</LastName>
</Emp>
<Emp ID="8">
<FirstName>Laura</FirstName>
<LastName>Callahan</LastName>
</Emp>
<Emp ID="9">
<FirstName>Anne</FirstName>
<LastName>Dodsworth</LastName>
</Emp>
</Emps>
```

Nested Selects

The previous basic examples cover the popular query types using SQLX. The generation of XML documents with complex hierarchies requires nesting joined SELECT statements within child XMLElement and XMLAgg definitions.

Example:

```
SELECT XMLELEMENT('Emps', XMLAGG( XMLELEMENT('Emp',
  XMLATTRIBUTES("EmployeeID" AS ID),
  XMLFOREST("FirstName", "LastName"),
           (SELECT XMLELEMENT('Orders',
                              XMLAGG( XMLElement('Order',
                                                 XMLATTRIBUTES("OrderID"
                                                 XMLFOREST("OrderDate", "
                                       )
                             )
           ) FROM "Demo"."demo"."Orders" O
             WHERE O.EmployeeID = E.EmployeeID ) ) ) )
FROM "Demo"."demo"."Employees" E
```

This is similar to the previous XMLAgg example, except for creating a subelement called 'Orders', and aggregating a set of joined data representing individual Orders tied to a particular Employee. This query generates substantial output, therefore, only the beginning of the resulting output document is included:

```
<Emps>
  <Emp ID="1">
  <FirstName>Nancy</FirstName>
  <LastName>Davolio</LastName>
  <Orders>
    <Order ID="10258">
    <OrderDate>1994-08-17 00:00:00.000000</OrderDate>
    <CustomerID>ERNSH</CustomerID>
  </Order>
  <Order ID="10270">
    <OrderDate>1994-09-01 00:00:00.000000</OrderDate>
    <CustomerID>WARTH</CustomerID>
  </Order>
  <Order ID="10275">
    <OrderDate>1994-09-07 00:00:00.000000</OrderDate>
    <CustomerID>MAGAA</CustomerID>
  </Order>
...
```

Unlike standard SQL, SQLX is capable of generating a rich hierarchical output. The query that produced the example output document is comparable to a standard nested SQL SELECT statement.

Step By Step Guide

The Virtuoso Demo Database

For this demonstration, use the Virtuoso demo database included in the standard installation.

Type the following URL into your web browser:

```
http://localhost:8890/admin/
```

This will prompt for the DBA user name and password. After correctly entering username and password, you will be presented with the Virtuoso Administrator interface.

Issuing SQLX Queries in Virtuoso Administrator

As SQLX is an extension of SQL, SQLX queries may be issued using the standard SQL interfaces and APIs.

Once logged into Virtuoso's Administrator interface, click the link labeled 'Query Tools' in the tree to the left of the screen. A new set of items will appear directly below the 'Query Tools' item, click Relational Data using SQL'. You will be presented with a SQL querying interface.

Cut and paste the example below into the query editing area, and click the 'Execute' button. The result should look like this:

```
<Emp>
  <ID>1</ID>
  <FirstName>Nancy</FirstName>
  <LastName>Davolio</LastName>
</Emp>
```

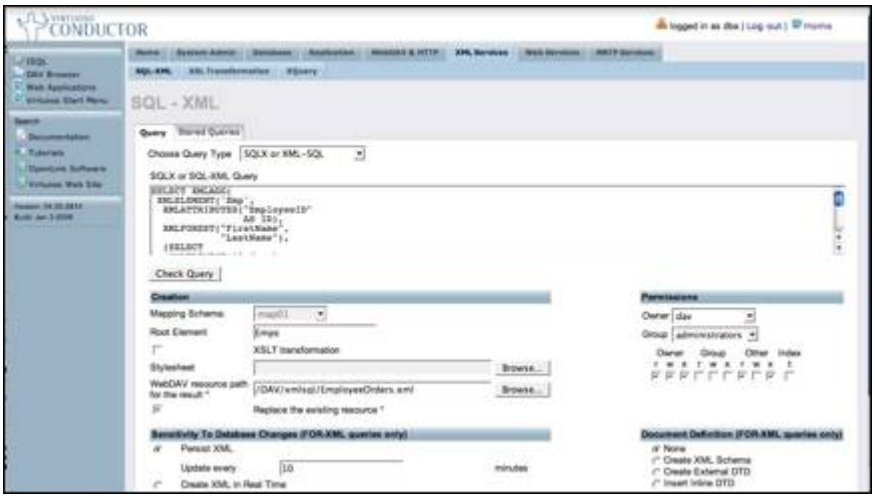
Notice that this result is different than the results demonstrated in previous sections. Canonically, the two results are identical. XML ignores white space unless explicitly directed not to.

Continue cutting, pasting, and executing the other examples. Other than formatting, the results are consistent with the examples. Try making modifications to the queries, such as renaming the elements that are produced, or promoting all of the elements to attributes. Virtuoso will check for syntax violations and report these errors.

Saving Queries as HTTP Resources

Virtuoso allows saving SQLX queries in the WebDAV repository and virtual directories. In order to access an XML query result, point a browser or XML processing tool to the previously defined URI. Virtuoso may generate this document upon request, or can periodically re-generate the document.

To create a SQLX query available by web browser, use the SQL-XML Query facilities available under XML Services->SQLX-XML tab in the Virtuoso Server Administrator, Conductor.The screen will look like this:



Set the following fields to their respective values, and click the 'Execute' button.

Field	Value
SQL Statement	`SELECT XMLAGG(XMLELEMENT('Emp', XMLATTRIBUTES("EmployeeID" AS ID), XMLFOREST("FirstName", "LastName"), (SELECT XMLELEMENT('Orders', XMLAGG(XMLElement('Order', XMLATTRIBUTES("OrderID" AS ID), XMLFOREST("OrderDate", "CustomerID")))) FROM "Demo"."demo"."Orders" O WHERE O.EmployeeID = E.EmployeeID))) FROM "Demo"."demo"."Employees" E`
Root Element	Emps
Store into	/DAV/xmlsql/EmployeeOrders.xml
Persist XML in Real Time	Selected
Owner	Dav
Group	Administrators
Permissions	Owner: rwx - Group: r - Other: r

You can now browse to the following URL to view the query results:

http://localhost:8890/DAV/xmlsql/EmployeeOrders.xml

Virtuoso WebDAV requires user authentication to browse stored queries. This is a

prudent security practice, but there may be situations that call for a query to be made publicly available.

Creating an open virtual directory as part of Virtuoso's default HTTP server is a simple task.

Navigate to the 'HTTP Hosts & Directories' sub-tab under the WebDAV & HTTP. In the 'HTTP Hosts & Directories' screen, expand the directories left of the default port and Default Web Site. by clicking on the folder icon. Once the directories are expanded click on the 'Add New Directory' Link at the top of the directory.

The screen will then look like this:

Set the following fields to their respective values, and click the 'Save Changes' button.

Field	Value
Path Arial	/xmlsql-test
Physical path is a WebDAV repository	Checked
Physical Path	/DAV/xmlsql/

You can now browse to the following URL to view the query results:

http://localhost:8890/xmlsql-test/EmployeeOrders.xml

This URL is open whereas the WebDAV URL required authentication. Programmatic Example Using JDBC

Executing queries interactively from the administrative interface is useful for diagnostic purposes, however, most database interaction with Virtuoso is accomplished through the use of application logic.

The following examples demonstrate two simple programs using the Java JDBC API. The JDBC API is a driver-managed system that connects to databases using a simple URI-based connection.

The JAR files for Virtuoso may be found in the 'jdk1.4' directory under the Virtuoso

installation folder. On a Windows system, this will generally be:

```
C:\Program Files\OpenLink\Virtuoso 3.5\jdk1.4\virtjdbc3.jar
```

Ensure that this jar is in the Java CLASSPATH environment variable. It is beyond the scope of this document to explain how to manage the CLASSPATH. In Windows, set the system's global CLASSPATH using the 'Environment Variables' dialog which is available under 'Advanced' tab in System Properties.

Simply add the full path to the Virtuoso jar to the end of any existing CLASSPATH variable, using a semi-colon (;) as a separator, or create a new environment variable if the CLASSPATH variable does not already exist.

This is a very simple example that opens a connection to the Virtuoso demo database, issues an aggregating SQLX query, and displays the textual results to standard output.

```
import java.sql.*;


public class SQLXDemo1 {


public static void main(String[] args) {

try {

// Create an instance of the Virtuoso JDBC Driver

Class.forName("virtuoso.jdbc3.Driver");


// Create a URL to the Demo Database that will be used

String url = "jdbc:virtuoso://localhost:1112/UID=dba/PWD=dba";


// Connect to the Demo Database

Connection c = DriverManager.getConnection(url);


// Build up a query to issue

StringBuffer sb = new StringBuffer();

sb.append("SELECT XMLELEMENT('Emps',\n");

sb.append("XMLAGG(\n");

sb.append("XMLELEMENT('Emp',\n");

sb.append("XMLATTRIBUTES(\"EmployeeID\" AS ID),\n");

sb.append("XMLFOREST(\"FirstName\", \"LastName\")\n");

sb.append(")\n");

sb.append(")\n");

sb.append(")\n");

sb.append("FROM \"Demo\".\"demo\".\"Employees\"\n");

String query = sb.toString();


// Create a Statement execution context

Statement s = c.createStatement();
```

Like our previous example, this program will yield the following results:

```
<Emp>
  <ID>1</ID>
  <FirstName>Nancy</FirstName>
  <LastName>Davolio</LastName>
</Emp>
```

The modest results of this program belie the power of the underlying foundation for building complex and functional JDBC applications. A simple extension to this program would to add the ability to store results into a text file, rather than displaying them on the screen.

Conclusion

As SQL is the dominant language of database applications, so shall SQLX become the extension of choice for the extraction of XML from relational resources. As a relatively simple, intuitive, and easy to learn SQL language add-on, SQLX is a great tool for the for aggregating data across the enterprise.

Additional Information Sources

More information on various technologies mentioned in this article is listed in the following table.

Resource	Location
OpenLink Virtuoso	http://virtuoso.openlinksw.com/
Extensible Markup Language	http://www.w3.org/XML/
SQL/XML	http://www.sqlx.org/
The Java Programming Language	http://java.sun.com/
Java JDBC	http://java.sun.com/products/jdbc/
Document Object Model	http://www.w3.org/DOM/

Learn More

- **Virtuoso Documentation**
- **Virtuoso Tutorials**

[1] Tables stored in Virtuoso's core database