

XQuery in IBM DB2 10 for z/OS

Introduction

In November 2011, support was added for XQuery in IBM® DB2® 10 for z/OS® through APARs PM47617 and PM47618.

XQuery extends XPath with new language constructs, adding expressiveness to the IBM pureXML® features of DB2. Many XQuery queries can be rewritten using a combination of XPath and SQL/XML, but often XQuery is the preferred tool, as it is an easier to code and more readable alternative.

Like XPath, XQuery is a W3C standard, and its addition is a further step in ensuring compatibility within the DB2 family of products. XQuery has been available in DB2 for Linux, UNIX, and Windows since 2006.

Constructs of XQuery

XQuery introduces new types of expressions that can be used in `XMLQUERY`, `XMLTABLE`, and `XMLEXISTS` function calls. The `XMLQUERY` function is used in the `SELECT` clause of an SQL query to specify which XML data to retrieve. The `XMLTABLE` function is used in the `FROM` clause to extract XML data in a table format. The `XMLEXISTS` function is used in the `WHERE` clause to specify under which conditions to retrieve the data.

XQuery expressions are most applicable with `XMLQUERY`, because arguments to `XMLEXISTS` can be specified by XPath expressions, which are indexable, while XQuery expressions are not. `XMLQUERY` is not indexable. XQuery can be used in `XMLTABLE` as the row expression or column expressions, but is usually not needed for the relational result. XQuery could be used to construct new XML values for the XML column results.

There are three types of expressions that can be used alone or in combination:

- FLWOR expressions
 - A FLWOR expression is a loop construct for manipulating XML documents in an application-like manner. The name (pronounced *flower*) is an acronym for the keywords used in the expression.

The keywords provide the functions described in the following table.

Table 1. FLWOR functions

<i>for</i>	Allows a variable to loop through a sequence of values. These can be literals, XPath expressions, and so on. The looping variable is prefixed with \$, as with other variables in XML expressions. An example is as follows: for \$il in /Invoice/InvoiceLine
<i>let</i>	Assigns a variable a single value. This can be a literal, an XPath expression, and so on. The variable is prefixed with \$. An example is as follows: let \$oref := /Invoice/OrderReference
<i>where</i>	Defines the criteria for which values are to be returned, as with an SQL query. An example is as follows: where \$il/eOrdrLineReference = \$ol/ID
<i>order by</i>	Orders the output specified in the return clause, as with an SQL query. An example is as follows: order by \$il/OrderLineReference
<i>return</i>	Specifies what is to be returned from each iteration of the FLWOR expression. The final result is their concatenation. return \$il

- Direct XML constructors

Instead of using publishing functions for creating XML elements, documents, and other XML constructs, you can now write them as literals anywhere that you can write an expression of the same type.

- Conditional expressions

You can use IF-THEN-ELSE logic anywhere you can use an expression within an XQuery expression.

FLWOR expressions

To illustrate a simple FLWOR expression, assume you have a table named INVOICE with an XML column named INVDOC holding invoices, as shown in the following code:

```
<Invoice>
  <OrderReference>1234</OrderReference>
  <DeliveredBy>XY Shipping</DeliveredBy>
  <InvoiceLine>
    <OrderLineReference>678</OrderLineReference>
    <InvoicedQuantity>18</InvoicedQuantity>
    <Item>
      <Name>Carmen curlers</Name>
    </Item>
  </InvoiceLine>
  <InvoiceLine>
    <OrderLineReference>679</OrderLineReference>
    <InvoicedQuantity>10</InvoicedQuantity>
    <Item>
      <Name>GHD hair straightener</Name>
    </Item>
  </InvoiceLine>
</Invoice>
```

You want to extract all the item names and quantities from the invoices where the items have been delivered by XY Shipping. To accomplish this task, loop through the invoice lines using the `for` keyword. Use the `let` clause to establish a reference to the `DeliveredBy` element, which you use in the *where* clause to select only those clauses with a value of XY Shipping. Use the `order by` clause to sort the items alphabetically by name, and use the `return` clause to return quantity and name for each invoice line. Your query should look as follows:

```
SELECT XMLQUERY (
'for $il in $in/Invoice/InvoiceLine
let $del := $in/Invoice/DeliveredBy
where $del = "XY Shipping"
order by $il/Item/Name
return
  fn:concat($il/InvoicedQuantity, " ", $il/Item/Name)'
PASSING INVDOC AS "in")
FROM INVOICE
```

The result of running the query on the example single invoice document is as follows:

```
18 Carmen curlers 10 GHD hair straightener
```

You could rewrite this example using XPath; this example is included here to show the basic principles of the FLWOR expression. The real strength of the FLWOR expression is its ability to combine data from different XML documents.

Assume that apart from the invoices, you also have a table named `PURCHASE_ORDERS` with an XML column named `PODOC` that contains orders like the one specified in the following query. The result of this query may not be very readable, but we demonstrate how to make such results more readable in the Direct XML constructors section.

```
<Order>
  <ID>1234</ID>
  <OrderLine>
    <ID>678</ID>
    <Quantity>8</Quantity>
  </OrderLine>
  <OrderLine>
    <ID>679</ID>
    <Quantity>10</Quantity>
  </OrderLine>
</Order>
```

Ensure that the number of items ordered in each order line matches the number of items invoiced in the corresponding invoice line. return the order and order line number for each discrepancy. To accomplish this task, join the tables `INVOICE` and `PURCHASE_ORDER`, pairing each order with an invoice, as shown in the following query. Assume that no SQL columns of the tables provide this information, so you must perform the join the XML data on the order versus invoice level and the order line versus invoice line level.

```
SELECT XMLQUERY (
'for $ol in $po/Order/OrderLine
for $il in $in/Invoice/InvoiceLine
let $oid := $po/Order/ID
let $oref := $in/Invoice/OrderReference
where $oid = $oref
and $ol/ID = $il/OrderLineReference
and $ol/Quantity != $il/InvoicedQuantity
order by $il/OrderLineReference
return
  fn:concat("Discrepancy orderno:", $oref, "-", $il/OrderLineReference)'
PASSING PODOC AS "po", INVDOC AS "in")
```

```

FROM PURCHASE_ORDER, INVOICE
WHERE XMLEXISTS (
 '$po/Order/OrderLine[ID = $in/Invoice/InvoiceLine/OrderLineReference]'
PASSING PODOC AS "po", INVDOC AS "in")

```

It is possible to do the whole join inside XQuery, but for efficiency, we add an `XMLEXIST` predicate to the SQL query that selects only matching pairs of orders and invoices. As mentioned earlier, `XMLEXISTS` predicates are eligible for index access, while `XMLQUERY` predicates are not. By adding the predicate, you avoid having to evaluate the Cartesian product in the XQuery expression. In this example, we could omit the join on the order-invoice level from the XQuery expression, but we kept it for readability. The result of the query is as follows:

```
Discrepancy orderno:1234-678
```

Direct XML constructors

Assume that you have the same basic scenario as in the previous example, but you want the output in a slightly different format. Instead of obtaining a list of the orders and order lines, obtain XML `DiscrepancyOrderLine` elements for each of the discrepancies.

You accomplish this task by using direct XML construction in the return clause of the FLWOR expression. You can also accomplish this task by using XML publishing functions, but direct XML construction provides far more intuitive and readable XQuery expressions.

To construct an XML element, write the start and end tags as literals. The content of the XML element can be more literals (constants), or it can be a composite expression. Start and end of composite expressions are marked by escape characters '{' and '}', When you use these escape character, DB2 knows when an expression is constant and when it should be evaluated based on the values of the variables in the for and let clauses.

The structure of our example XML element is as follows:

```

SELECT XMLQUERY (
 'for $ol in $po/Order/OrderLine
 for $il in $in/Invoice/InvoiceLine
 let $oid := $po/Order/ID
 let $oref := $in/Invoice/OrderReference
 where $oid = $oref
 and $ol/ID = $il/OrderLineReference
 and $ol/Quantity != $il/InvoicedQuantity
 order by $il/OrderLineReference
 return
   <DiscrepancyOrderLine>
     {$oref}
     {$il/OrderLineReference}
     {$il/Item/Name}
     <OrderedQuantity> {$ol/Quantity/text()} </OrderedQuantity>
     {$il/InvoicedQuantity}
   </DiscrepancyOrderLine>'
PASSING PODOC AS "po", INVDOC AS "in")
FROM PURCHASE_ORDER, INVOICE
WHERE XMLEXISTS (
 '$po/Order/OrderLine[ID = $in/Invoice/InvoiceLine/OrderLineReference]'
PASSING PODOC AS "po", INVDOC AS "in")

```

The nested elements within the `DiscrepancyOrderLine` element are all included as they are, except for the quantity from the orderline, because we want to change the name of the element to `OrderedQuantity`. Write the new start and end tags, and include the contents of the element by using the `text()` function.

The result of running the query on the XML documents above is as follows:

```
<DiscrepancyOrderLine>
  <OrderReference>1234</OrderReference>
  <OrderLineReference>678</OrderLineReference>
  <Name>Carmen curlers</Name>
  <OrderedQuantity>8</OrderedQuantity>
  <InvoicedQuantity>18</InvoicedQuantity>
</DiscrepancyOrderLine>
```

Conditional expressions

Assume the same basic example, but now we want to distinguish the cases where the customer has been invoiced too much from where he has been invoiced too little.

Use the IF-THEN-ELSE expression in the return clause to construct different element names depending on whether the ordered quantity is larger or smaller than the invoiced quantity:

```
SELECT XMLQUERY (
'for $ol in $po/Order/OrderLine
for $il in $in/Invoice/InvoiceLine
let $oid := $po/Order/ID
let $oref := $in/Invoice/OrderReference
where $oid = $oref
and $ol/ID = $il/OrderLineReference
and $ol/Quantity != $il/InvoicedQuantity
order by $il/OrderLineReference
return
  if ($ol/Quantity > $il/InvoicedQuantity) then
    <DeficitOrderLine>
      {$oref}
      {$il/OrderLineReference}
      {$il/Item/Name}
      <OrderedQuantity> {$ol/Quantity/text()} </OrderedQuantity>
      {$il/InvoicedQuantity}
    </DeficitOrderLine>
  else
    <SurplusOrderLine>
      {$oref}
      {$il/OrderLineReference}
      {$il/Item/Name}
      <OrderedQuantity> {$ol/Quantity/text()} </OrderedQuantity>
      {$il/InvoicedQuantity}
    </SurplusOrderLine>
'
,
PASSING PODOC AS "po", INVDOC AS "in")
FROM PURCHASE_ORDER, INVOICE
```

The result of running this query on the two XML documents above is as follows:

```
<DeficitOrderLine>
  <OrderReference>1234</OrderReference>
  <OrderLineReference>678</OrderLineReference>
  <Name>Carmen curlers</Name>
  <OrderedQuantity>8</OrderedQuantity>
  <InvoicedQuantity>18</InvoicedQuantity>
</DeficitOrderLine>
```

XQuery performance

XQuery greatly enhances DB2 support for constructing and manipulating XML, far beyond what was expressible using SQL/XML and XPath. XQuery is allowed everywhere that XPath is allowed, including XMLQUERY, XMLEXISTS, and XMLTABLE, but there are some important considerations for performance when deciding where and how to use XQuery.

Consider the SQL WHERE predicate from the earlier examples:

```
WHERE XMLEXISTS (
 '$po/Order/OrderLine[ID = $in/Invoice/InvoiceLine/OrderLineReference]'
 PASSING PODOC AS "po", INVDOC AS "in")
```

The same predicate could have been written with an XQuery FLWOR join expression:

```
WHERE XMLEXISTS (
 'for $po in /Order/OrderLine,
     $in in /Invoice/InvoiceLine
  where $po/ID=$in/OrderLineReference
  return $po/ID'
 PASSING PODOC AS "po", INVDOC AS "in")
```

However, XQuery FLWOR expressions are not indexable by DB2. Even if qualifying indexes existed on PURCHASE_ORDER with XMLPATTERN('/Order/OrderLine/ID'), and on INVOICE with XMLPATTERN('/Invoice/InvoiceLine/OrderLineReference'), the first example using XPath can use the indexes, but the second example using XQuery FLWOR cannot. Therefore, it is a better idea to avoid FLWOR expressions, if possible, in places where DB2 uses indexes such as XMLEXISTS predicates or in XMLTABLE row expressions.

The preferred practice is to use XPath in predicate expressions to allow DB2 to use an index to qualify the rows, and then to use XQuery in the XMLQUERY function or in XMLTABLE column expressions to perform manipulation on only the qualifying rows. If you do not have such a qualifying predicate, DB2 could evaluate XQuery against many rows, which could lead to elongated query run time.

Whether you write an XML expression inside of XMLQUERY as XPath or as XQuery, where the expressions can be expressed equivalently, performance is roughly equivalent. In many cases, DB2 performs automatic rewrites of XQuery to XPath when possible to optimize performance.

Summary and reference

You have seen, through some simple examples, the basic functionality of XQuery provided in DB2 10 for z/OS.

There are many details that have not been covered here. For general functions and more information, see *Extremely pureXML in DB2 10 for z/OS*, SG24-7915 and *pureXML Guide*, SC19-2981. You can also go to the following IBM developerWorks® website found at:

<http://www.ibm.com/developerworks/wikis/display/db2xml/DB2+for+zOS+pureXML>

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service. IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing, IBM Corporation, North Castle Drive, Armonk, NY 10504-1785 U.S.A.

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you. This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk. IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you. Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products. This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurement may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

© Copyright International Business Machines Corporation 2012. All rights reserved.

Note to U.S. Government Users Restricted Rights -- Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

This document was created or updated on July 2, 2012.

Send us your comments in one of the following ways:

- Use the online **Contact us** review form found at:
ibm.com/redbooks
- Send your comments in an e-mail to:
redbook@us.ibm.com
- Mail your comments to:
IBM Corporation, International Technical Support Organization
Dept. HYTD Mail Station P099
2455 South Road
Poughkeepsie, NY 12601-5400 U.S.A.

This document is available online at <http://www.ibm.com/redbooks/abstracts/tips0896.html> .

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. These and other IBM trademarked terms are marked on their first occurrence in this information with the appropriate symbol (® or ™), indicating US registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. A current list of IBM trademarks is available on the Web at <http://www.ibm.com/legal/copytrade.shtml>

The following terms are trademarks of the International Business Machines Corporation in the United States, other countries, or both:

DB2®
developerWorks®
IBM®
pureXML®
Redbooks (logo)®
z/OS®

The following terms are trademarks of other companies:

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

Windows, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Other company, product, or service names may be trademarks or service marks of others.