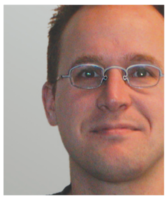
Customizing XML Storage in DB2

Tailor XML storage to the needs of your application



Matthias Nicola

(www.matthiasnicola.de) is a senior engineer for DB2 pureXML at the IBM Silicon Valley Lab. He also works closely with customers and business partners, assisting them in the design, implementation, and optimization of XML solutions.

DB2 pureXML makes storing and querying XML data easy: just define a column of type XML and then insert or load XML documents into that column. But what if you need to go beyond the basics? With IBM DB2 for z/OS and DB2 for Linux, UNIX, and Windows (LUW), that's no problem. Let's review some best practices for when and how to customize your XML storage.

To get started, we'll use the XML document in Figure 1, which represents an order with an ID, date, customer ID, and multiple items. Note that items can vary in the elements that describe them, such as size or color. Let's assume that we need to manage many such documents in DB2.

```
<order OrderID="9001" OrderDate="2009-10-18">
  <customerID>26914</customerID>
  <item id="LK-486">
    <name>Magic Potion</name>
    <size>300ml</size>
    <price>19.99</price>
  </item>
  <item id="VF-145">
    <name>Crystal Ball, Deluxe</name>
    <color>crystal clear</color>
    <price>295.00</price>
  </item>
</order>
```

Figure 1: *A sample XML document*

How to split and reassemble XML documents

The first tip in my article "15 best practices for pureXML performance in DB2" (see sidebar, "Resources") is that you should choose your document granularity wisely. Essentially, this means that the XML documents you store in DB2 should match the logical business objects of your application and the predominant granularity of access.

In our example, let's assume that orders are subject to frequent changes and that reading, adding, or deleting individual items within an order are the most critical operations that require optimal performance. In this case you could consider splitting order documents upon insert and storing each item as a separate document in a separate row. The advantage of this storage approach (compared to intact storage of the original order documents) is that it allows easier and faster manipulation of the stored data:

- You can *retrieve* an item with a single row read, and without extracting the item from an order document.
- You can *remove* an item from an order simply by deleting a row from the items table. Manipulating an entire order document is not required.
- You can *add* an item to an order by inserting another item that has the appropriate order ID, customer ID, date, and sequence number. Again, manipulating an order document is not required.

This ease of adding and removing items in an order is particularly valuable in DB2 9 for z/OS, which doesn't support inserting or deleting elements within an existing XML document.

Figure 2 shows a possible table definition and INSERT statement for splitting an order document in DB2 for z/OS and DB2 for LUW. Relational columns store the order ID, customer ID, order date, and a sequence number for the items. Item information is stored in XML format because items may have different elements and attributes (see Figure 3).

The INSERT statement contains a fullselect with an XMLTABLE function. This function extracts values from the incoming XML document for insertion into the columns of the items table, and it splits the incoming XML document and produces separate item documents.

The XMLTABLE function contains a parameter marker through which the application can pass an order document. With the XPath

```
CREATE TABLE items(ordID INTEGER, custID INTEGER,
                   odate DATE, seqNo INTEGER, item XML);

INSERT INTO items(ordID, custID, odate, seqno, item)
 SELECT T.ordID, T.custID, T.odate, T.seqno, XMLDOCUMENT( T.item)
 FROM
  XMLTABLE('$d/order/item' PASSING cast(? AS XML) "d"
   COLUMNS
    ordID     INTEGER   PATH   '../@OrderID',
    custID    INTEGER   PATH   '../customerID',
    odate     DATE      PATH   '../@OrderDate',
    seqNo     FOR ORDINALITY,
    item      XML       PATH   '.') AS T;
```

Figure 2: *Table and INSERT statement to store each item in a separate row*

```
ORDID   CUSTID  ODATE        SEQNO    ITEM
------- ------- ------------- -------- ----------------------------------
 9001   26914   10/18/2009          1  <item id="LK-486">
                                          <name>Magic Potion</name>
                                          <size>300ml</size>
                                          <price>19.99</price>
                                       </item>
 9001    26914  10/18/2009          2  <item id="VF-145">
                                          <name>Crystal Ball, Deluxe</name>
                                          <color>crystal clear</color>
                                           <price>295.00</price>
                                       </item>
2 record(s) selected.
```

Figure 3: *Contents of the items table after inserting the sample document*

expression $d/order/item, the XMLTABLE function produces one row for each item element in the input document, then extracts the order ID, customer ID, and order date. The special column definition FOR ORDINALITY numbers the generated rows. (For details on the XMLTABLE function, see "XMLTABLE by example, Part 1" in the Resources sidebar.) The XMLDOCUMENT function ensures that each item fragment can be inserted as a stand-alone XML document.

Figure 3 shows the data in the items table after our sample document has been inserted using the INSERT statement in Figure 2.

Figure 4 shows how you can reconstruct the original order document in Figure 1, if needed. The functions XMLELEMENT and XMLATTRIBUTES construct the top of the document with values from the relational columns of the items table. The function XMLAGG combines all items for a given order in the constructed document. Note that XMLAGG contains an optional ORDER BY clause on the column seqno.

```
SELECT XMLELEMENT(name "order",
              XMLATTRIBUTES(ordID AS "OrderID", odate as "OrderDate"),
              XMLELEMENT(name "customerID", custID),
              XMLAGG(item ORDER BY seqno)  )
FROM items
WHERE ordID = 9001
GROUP BY ordID, odate, custID;
```

Figure 4: *Reassembling the original order document*

This ensures that the items appear in the same sequence as in the original document.

Use generated columns to your advantage

The new IBM DB2 pureXML features in DB2 9.7 for LUW allow you to use XML columns together with the Database Partitioning Feature (DPF), range-partitioned tables, and multidimensional clustering (MDC) tables. However, the partitioning or clustering keys must consist of relational columns. You just saw how to use INSERT and XMLTABLE to extract values from XML documents into relational columns. You can certainly use those relational columns to partition or cluster your table. If you prefer that your application uses simpler INSERT statements (such as INSERT INTO orders(order) VALUES(?);) and is unaware of the extraction, consider using a generated column.

DB2 9.7 supports XML parameters in user-defined functions (UDFs), enabling you to define generated columns that are automatically populated with values from inserted XML documents. Figure 5 shows a UDF that takes an XML document, such as the sample order in Figure 1, as input. This UDF uses the functions XMLCAST and XMLQUERY to extract the OrderDate attribute from the input document.

```
CREATE FUNCTION extractDate(doc XML)
   RETURNS DATE
   LANGUAGE SQL CONTAINS SQL
   NO EXTERNAL ACTION DETERMINISTIC
   RETURN XMLCAST(XMLQUERY('$d/order/@OrderDate'
          PASSING doc AS "d") AS DATE);
```

Figure 5: *Scalar UDF with parameter of type XML*

You can use this UDF in SELECT queries and other SQL statements, but also to define a generated column. For the following example, let's assume that inserting and retrieving complete orders are the most critical operations. In this case, storing the order documents intact is a good choice. Figure 6 defines a table that stores orders in an XML column and automatically extracts the order date into a generated relational column (odate). An INSERT statement can now simply insert an XML document into the order column and does not need to be concerned with extracting values into relational columns.

```
CREATE TABLE orders(
  order XML,
  odate DATE GENERATED ALWAYS AS (extractDate(order)));
```

Figure 6: *Table definition with a generated column*

If you store many orders on an ongoing basis, you may need to archive (roll off) old orders. This is best accomplished with range partitioning. The table in Figure 7 is partitioned by values in the column odate, which is generated from the XML column. Similarly, you can use generated columns as the distribution key in a partitioned database (with DPF) or as the clustering key for MDC tables.

```
CREATE TABLE order2(
   order XML,
   odate DATE GENERATED ALWAYS AS (extractDate(order)) NOT NULL)
 PARTITION BY RANGE (odate)
 (PART   q109   STARTING('01-01-2009')   ENDING ('03-31-2009') INCLUSIVE,
  PART   q209   ENDING ('06-30-2009')   INCLUSIVE,
  PART   q309   ENDING ('09-30-2009')   INCLUSIVE,
  PART   q409   ENDING ('12-31-2009')   INCLUSIVE);
```

Figure 7: *Range-partitioned table based on a generated column*

Keep XML storage under control

Customizing your XML storage has various benefits. Splitting large XML documents into smaller documents can allow for simpler and more efficient manipulation of XML data. Using UDFs to define generated columns simplifies the extraction of XML values into relational columns. These columns then help you manage XML in partitioned databases, range-partitioned tables, or MDC tables. You can find other useful best practices for XML data in *The DB2 pureXML Cookbook* and the articles listed in the Resources sidebar. ✱

RESOURCES

15 best practices for pureXML performance in DB2: **ibm.com**/developerworks/db2/library/techarticle/dm-0610nicola

The DB2 pureXML Cookbook (IBM Press, 2009): ibm.com/developerworks/wikis/display/db2xml/DB2+pureXML+Cookbook

XMLTABLE by example, Part 1: ibm.com/developerworks/db2/library/techarticle/dm-0708nicola

Enhance business insight and scalability of XML data with new DB2 9.7 pureXML features: ibm.com/developerworks/data/library/techarticle/dm-0904db297purexml

Exploit XML indexes for XML query performance in DB2 9: **ibm.com**/developerworks/data/library/techarticle/dm-0611nicola

Updating XML in DB2 9.5: ibm.com/developerworks/data/library/techarticle/dm-0710nicola

DB2 pureXML wiki: ibm.com/developerworks/wikis/display/db2xml/Home