IBM Information Management software

Leveraging DB2 9 for z/OS pureXML technology

By Guogen (Gene) Zhang, PhD
DB2 z/OS XML Development

Executive summary

Keeping pace with rapidly changing requirements is a constant challenge for organizations today. One way to meet this challenge is by storing and processing XML natively. Doing so can help enterprises improve application development productivity and quality by eliminating the time-consuming mappings and schema evolution necessary for rapidly changing requirements, thus improving time to market and significantly lowering IT costs.

The breakthrough hybrid database server DB2® 9 for z/OS® integrates XML database technology into relational databases, providing unprecedented scalability and performance for both relational and XML data. The hybrid database server also helps improve performance and scalability by avoiding complex joins commonly seen in an object persistence solution with relational databases. This paper offers an overview of pureXML™ technology in DB2 9 for z/OS, its business values and technical feature details. In addition, it lists the commonalities and differences of the XML functionality between DB2 for Linux®, UNIX®, and Microsoft® Windows® and DB2 for z/OS.

DB2 9 for z/OS: An overview

DB2 9 for z/OS support for XML lets your client applications manage XML data in DB2 tables. You can store well-formed XML documents in their hierarchical form, and retrieve all or portions of those documents. Because the stored XML data is fully integrated into the DB2 database system, you can access and manage the XML data by leveraging DB2 functionality.

To efficiently manage traditional SQL data types and XML data, DB2 uses two distinct storage mechanisms. However, the underlying storage mechanism that is used for a given data type is transparent to the application. The application does not need to explicitly specify which storage mechanism to use, or to manage the physical storage for XML and non-XML objects.

XML document storage and retrieval

The XML column data type is provided for storage of XML data in DB2 tables and most SQL statements support the XML data type. This enables you to perform many common database operations with XML data, such as:

- *Creating tables with XML columns.*
- *Adding XML columns to existing tables.*
- *Creating indexes over XML columns.*
- *Creating triggers on tables with XML columns.*
- *Inserting, updating or deleting XML documents.*

Alternatively, a decomposition stored procedure lets you extract data items from an XML document and store those data items in columns of relational tables using an XML schema that has been annotated with instructions on how to store the data items.

In addition, you can use SQL to retrieve entire documents from XML columns, just as you retrieve data from any other type of column. When you need to retrieve portions of documents, you can specify XPath expressions, through SQL with XML extensions (SQL/XML).

Application development

Application development support of XML enables applications to combine XML and relational data access and storage. The following programming languages support the new XML data type:

- *Assembler*
- *C and C++ (embedded SQL or DB2 ODBC)*
- *COBOL*
- *Java™ (JDBC and SQLJ)*
- *PL/I*

Database administration

DB2 for z/OS database administration support for XML includes the following items:

XML schema repository (XSR).
The XSR is a repository for all XML schemas that are required to validate and process XML documents stored in XML columns or decomposed into relational tables.

Utility support.
You can use DB2 for z/OS utilities on XML objects. The utilities handle XML objects similar to the way they handle LOB objects. For some utilities, you need to specify certain XML keywords.

Performance.
Indexing support is available for data stored in XML columns. The use of indexes over XML data can improve the efficiency of queries that you issue against XML documents. An XML index differs from a relational index in that a relational index indexes an entire column, while an XML index indexes part of the data in a column. You can indicate which parts of an XML column are indexed by specifying an XML pattern, which is a limited XPath expression.

The business value of DB2 9

As the first hybrid data server for the industry, DB2 9 lets you store XML data in its pure, native form. Before IBM introduced pureXML technology, there were only a few options to store XML data, including:

As files in file systems.
Storing XML data as files has the advantage of preserving original documents. But it provides no database ACID properties and other database processing capabilities, such as indexing.

Decomposing, or shredding, the XML into relational or object relational form.
The decomposition approach is commonly used for regularly structured data. It has the advantage of not requiring XML functionality in databases. However, there are many disadvantages:

- *Mapping can be complex and fragile, and mapping must be predefined.*
- *It may need artificial keys to keep the parent-child relationship.*
- *It can be difficult to reconstruct, often requiring many joins, and potential poor performance.*
- *Decomposition typically applies to a single schema, and changes are usually limited. If the schema changes, the changes of the mapping could be tedious, and schema evolution may require outages.*
- *Queries are in SQL or through XPath or XQuery to SQL transformation. It is usually less productive in coding, and queries can be difficult to understand, diagnose and explain.*

Storing the XML intact in character form in a character large object (CLOB) or varchar column, and optionally extract commonly searched portions into relational tables for quick search.
The CLOB storage is the simplest to support. It has the advantage of preserving the original documents. However, it has many shortcomings:

- *Without additional indexing, the XML document must be parsed for searching either in the database server or the client, which is prohibitively expensive.*
- *When portions are extracted in relational tables with indexes for fast search, it is tedious and inefficient to keep the two in sync when there are updates.*
- *It is expensive to retrieve portions of a document.*

Other vendors also provide BLOB-based native XML storage, which stores post-parse binary representation in a BLOB. This approach suffers some disadvantages of being hard to retrieve, and inefficient to update. Furthermore, if the binary format needs to be converted to a relational representation for query processing either on the fly or persistently, it could be costly either in processing time or storage. And it becomes prohibitively expensive.

DB2 9 pureXML technology features native hierarchical storage, and native XML operators for query processing. Compared with decomposition or CLOB approach, and other vendors' relational-based technology, it has the following advantages:

Data model and storage:

- *Offers a compact value-based hierarchical storage.*
- *Can directly represent flexible hierarchical structures with explicit parent-child relationship.*
- *Avoids normalization and joins that are necessary to re-assemble the normalized tables.*
- *Provides node-level XML indexing for query performance.*
- *Delivers schema flexibility with no schema restrictions on XML columns, and therefore, schema can evolve freely.*
- *Providers a smoother transition and opportunity for best mix by managing both relational data and XML data together.*
- *Leverages mature existing infrastructure for reliability, availability and scalability.*

Query languages and processing:

- *Supports the standard declarative XML query languages SQL/XML and XPath. It provides high productivity in developing applications to process XML data that mapping approaches cannot achieve.*
- *Delivers native optimized operators and access methods that provide unprecedented performance and scalability.*
- *Eliminates impedance mismatch between applications and databases when applications are processing XML data to help drastically improve productivity.*

Overall, DB2 9 can help significantly improve productivity by eliminating tedious mapping and database schema evolution, and improve efficiency in storage and query processing, and all the properties a database server provides. The result is shorter time to market, easier maintenance for rapid changing business needs, high performance and scalability. The next few chapters offer more detailed information on the specific technical features of DB2 9.

XML type and native XML storage

DB2 9 introduced XML as a first-class SQL type and enables you to create a table with one or more XML columns. For example, the following creates a table with an XML column:

```
CREATE TABLE BASICS.PURCHASEORDERS (
   PONUMBER  VARCHAR(10) NOT NULL,
   PODATE   DATE,
   POSTATUS  CHAR(1),
    POXML    XML)
    IN DATABASE SALESDB;
```

You can also alter an existing table to add one or more XML columns:

```
ALTER TABLE BASICS.PURCHASEORDERS ADD INVOICEXML  XML;
```

For a table containing one or more XML columns, DB2 adds a hidden column named `DB2_GENERATED_DOCID_FOR_XML` in the base table, and creates a separate XML table space and an internal XML table for each XML column. The internal XML table consists of three columns (`DOCID, MIN _ NODEID, XMLDATA`). The XML table space always uses 16 KB page size, and it is a partition-by-growth table space for a simple, segmented, or partition-by-growth base table space, and a partitioned table space for a partitioned base table space. The `XMLDATA` column, with a var binary type, contains the hierarchical storage for XML data model. There are one or more rows in the internal XML table for an XML documents depending on the document size.

You can store well-formed XML documents into an XML column, and there is no XML schema constraint and no length limit associated with an XML column in DB2 9. You can insert an XML document using the INSERT statement with a string literal, a host variable, another column, or a file for an XML value. In addition, you can use the LOAD utility to load XML data.

For example, the following INSERT statement inserts a string literal XML document into an XML column. Notice that XML documents are case-sensitive.

```
INSERT INTO BASICS.PURCHASEORDERS VALUES
(‘2006040001’, CURRENT DATE, ‘A’,
 ‘<?xml version=”1.0” encoding=”UTF-8”?>
<purchaseOrder orderDate=”1999-10-20”>
  <shipTo country=”US”>
   <name>Alice Smith</name>
  </shipTo>
  <billTo country=”US”>
   <name>Robert Smith</name>
  </billTo>
   . . .
</purchaseOrder>’
,  NULL);
```

During the INSERT or LOAD process, XML values in string format are parsed and converted into the internal representation for storage. By default, insignificant whitespaces are stripped during this process. If you need to preserve whitespaces in a document, you need to invoke XMLPARSE() function explicitly and specify the `PRESERVE WHITESPACE` option as follows:

```
INSERT INTO BASICS.PURCHASEORDERS VALUES
('2006040001', CURRENT DATE, 'A',
  XMLPARSE(DOCUMENT CAST(? AS CLOB(100K)) PRESERVE WHITESPACE),
  NULL);
```

In case you need to preserve some of the whitespaces but not all, you can use an attribute called `xml:space` with a value "`preserve`" on the element in your document you want whitespaces to be preserved, which is a W3C standard mechanism, while using `STRIP WHITESPACE` option for the parsing.

You can also update an XML column with a new document. For example, the following update replaces an existing purchase order with a new one and stores an invoice at the same time.

```
UPDATE BASICS.PURCHASEORDERS SET
 POXML = :new_poxml, INVOICEXML = :invoicexml
WHERE PONUMBER = '2006040001';
```

You can also delete a row with XML just as a regular column. In the following example, the DELETE statement deletes a row with the given PONUMBER.

```
DELETE FROM BASICS.PURCHASEORDERS
WHERE PONUMBER = '2006040001';
```

For searched UPDATE and DELETE, you can specify both relational predicates and XML predicates.

DB2 9 invokes z/OS XML System Services (XMLSS) for high-performance parsing. You need to use z/OS R1.8 or later, or z/OS R1.7 with the PTF for XMLSS installed.

You can also validate a document against an XML schema before insertion.

Host language interfaces

DB2 9 provides XML host language interfaces for Assembler, C or C++ (embedded SQL or DB2 ODBC), COBOL, Java (JDBC or SQLJ), PL/I, and . NET. All the interfaces use string as the XML format. In host languages, XML host variables use a syntax that looks like a distinct type on a LOB. For example, in C or C++, you can use the following host variable declaration:

```
EXEC SQL BEGIN DECLARE SECTION;
       SQL TYPE IS XML AS CLOB(1M) xmlPo;
EXEC SQL END DECLARE SECTION;
```

You can use it in INSERT or SELECT as follows:

```
EXEC SQL INSERT INTO BASICS.PURCHASEORDERS VALUES ('200600001',
                CURRENT DATE, 'A', :xmlPo);

EXEC SQL SELECT POXML INTO :xmlPo
       FROM  BASICS.PURCHASEORDERS
       WHERE PONUMBER = '20060001';
```

For INSERT, implicit `XMLPARSE` is invoked, while for SELECT, implicit `XMLSERIALIZE` applies. You can also use explicit `XMLPARSE` and `XMLSERIALIZE` to convert between string format and internal data model format. The following is an example of `XMLSERIALIZE`:

```
EXEC SQL SELECT XMLSERIALIZE(POXML AS CLOB(100K)) INTO :clobPo
       FROM  BASICS.PURCHASEORDERS
       WHERE PONUMBER = '20060001';
```

Since XML data does not have a length limit, it is difficult to determine how much memory to allocate for a host variable to receive an XML value from DB2. In DB2 9, a new way of fetch XML and LOB data is introduced to allow for piece-by-piece fetch. The facility is the new option for `FETCH: FETCH WITH CONTINUE` and `FETCH CURRENT CONTINUE`. Check SQL reference for details.

In JDBC, the standard interface methods `setString()`, `setCharacterString()` and `getString()` etc. are expanded to support the XML type also. A new class `ibm.com.db2.jcc.DB2Xml` also provides some XML-specific methods.

XML data encoding

DB2 9 for z/OS supports XML columns in a table of any DB2-supported encoding. XML data is converted into UTF-8 at bind-in time before parsing if it is not already in UTF-8. Likewise, XML data is serialized into UTF-8 first internally at bind-out time and then converted into the encoding of the host variable or application encoding if necessary.

If an XML value is stored in a character host variable, the encoding of the host variable takes precedence over the encoding declaration inside the XML data. It is important to keep consistency between the real encoding and host variable encoding. Otherwise, the data may get corrupted or parsing may fail.

On the other hand, if an XML value is stored in a binary host var, the encoding determination process as specified by W3C for XML will apply, which includes Byte Order Mark (BOM) or internal encoding declaration.

Since XML character data are stored in UTF-8 internally in DB2 9, using UTF-8 database and application encoding, or UTF-8 encoding in binary host variables for XML data, can avoid the encoding conversion overhead and potential data loss problem during the bind-in and bind-out processes.

XML indexing

In addition to XML-related index objects (DOCID index on a base table and NODEID on an internal XML table), you can create specific XML indexes on XML columns using XPath expressions. The XML indexes supported in DB2 9 are value indexes. That is to map node values to nodes, identified by NodeIDs and RIDs of records in which the nodes reside. For example, the following example creates an XML index on the `XMLPO` column of table BASICS.`PURCHASEORDERS` using XML pattern `'/purchaseOrder/items/item/desc'`, which identifies all the descriptions of `items` within `purchaseOrder`. Notice that XPath expressions are case-sensitive.

```
CREATE INDEX ON BASICS.PURCHASEORDERS(POXML) GENERATE KEYS USING
XMLPATTERN '/purchaseOrder/items/item/desc' AS SQL VARCHAR(100);
```

Only string and numerical data types are supported for XML indexes in DB2 9 for z/OS, which uses SQL `VARCHAR(n)` or `DECFLOAT` correspondingly. An XML index is logically created on an XML column of a base table, but physically it is on the implicitly created XML table, which is reflected on catalog information and database object description.

The XML pattern is a limited subset of XPath expressions that do not have any predicate. Only element, attribute, or text nodes are allowed for indexing in DB2 9. An indexed element node can have sub-elements, but there is no composite key supported.

An XML index is different from indexes on columns of other types in that it may have zero or more index entries for each document, depending on the XML pattern specified. For example, for the index created above, there are two entries for this document illustrated below, they are `Lawnmower` and `Baby Monitor`.

```
<?xml version="1.0" encoding="UTF-8"?>
<purchaseOrder orderDate="1999-10-20">
  <shipTo country="US">
   <name>Alice Smith</name>
   . . .
  </shipTo>
  <billTo country="US">
   <name>Robert Smith</name>
   . . .
  </billTo>
  <comment>Hurry, my lawn is going wild!</comment>
  <items>
   <item partNum="872-AA">
    <desc>Lawnmower</desc>
    <quantity>1</quantity>
    <USPrice>148.95</USPrice>
    <comment>Confirm this is electric</comment>
   </item>
   <item partNum="926-AA">
    <desc>Baby Monitor</desc>
    <quantity>1</quantity>
    <USPrice>39.98</USPrice>
    <shipDate>2003-05-21</shipDate>
   </item>
  </items>
</purchaseOrder>
```

XML indexes are used in accelerating the query processing for the `XMLEXISTS()` predicate, but not `XMLQUERY()`.

Searching and retrieving XML Data

In addition to simple SELECT of columns and expressions of XML type, you can search on XML data using the `XMLEXISTS()` predicate. Except for NULL testing, `XMLEXISTS()` is the only predicate applicable to the XML type, and no direct comparison operators are available for XML at the SQL level. In `XMLEXISTS()`, you specify an XPath expression for a document to match. If the result of the XPath expression is an empty sequence, then `XMLEXISTS()` returns false. Otherwise, it returns true. For example, the following query selects POXML that has an `item` with "`Shoe`" as the description. The second example is to illustrate that you can pass a SQL expression into an XPath expression.

```
SELECT POXML
FROM  BASICS.PURCHASEORDERS
WHERE XMLEXISTS('//items/item[desc = "Shoe"]' PASSING POXML);

SELECT POXML
FROM  BASICS.PURCHASEORDERS
WHERE XMLEXISTS('//items/item[desc = $x]'
     PASSING POXML, 'Shoe' AS "x");
```

You can use XMLEXISTS predicate anywhere a predicate can be used except in the ON clause of an outer join.

You can also extract portions of an XML document using the `XMLQUERY()` function with XPath as the first argument, and other optional arguments just as in `XMLEXISTS()`. The following example extracts the `quantity` elements of “`Shoe`” item from the purchase order.

```
SELECT XMLQUERY(‘//items/item[desc=”Shoe”]/quantity’
             PASSING POXML)
FROM  BASICS.PURCHASEORDERS
WHERE XMLEXISTS(‘//items/item[desc = “Shoe”]’ PASSING POXML);
```

Functions `fn:data()` and `fn:string()` can be used to get the value of an element or attribute instead of the element or attribute itself. For example, if you want to get a list of quantities you can use the following query.

```
SELECT XMLQUERY(‘fn:data(//items/item[desc=”Shoe”]/quantity)’
             PASSING POXML)
FROM  BASICS.PURCHASEORDERS
WHERE XMLEXISTS(‘//items/item[desc = “Shoe”]’ PASSING POXML);
```

XPath support

DB2 9 for z/OS uses an XPath expression to identify portions of an XML document and is used in `XMLQUERY()`, `XMLEXISTS()` and `XMLPATTERN` of `CREATE INDEX`. Our strategy is to provide some core XML query language features that are critical to the application development in DB2 9, and expand them into the full XQuery language in the follow-on releases. We adopt the equivalent of the core XPath 1.0 language constructs and data types. However, we follow XPath 2.0 semantics and make them compatible with XQuery, including the XQuery prolog for namespace declaration that is not part of XPath but is necessary for the langauge.

The following data types are supported:

- `xs:boolean`,
- `xs:integer`,
- `xs:decimal`,
- `xs:double` *and*
- `xs:string`.

The support axes are the XQuery-required axes:

- *the forward axes:* `child`, `attribute`, `descendant`, `descendant-or-self`, `self`, `.`, `//`, `@` *and*
- `parent` *axis and its abbreviated form (..).*

The supported functions includes: `fn:abs`, `fn:boolean`, `fn:compare`, `fn:concat`, `fn:contains`, `fn:count`, `fn:data`, `fn:length`, `fn:normalize-space`, `fn:not`, `fn:round`, `fn:string`, `fn:substring`, `fn:sum`.

Please note that positional predicate is not supported.

Because DB2 9 for z/OS all XML documents are stored as untyped, you may need to use type casting (constructors for primitive types) for the correct semantics. For example,no cast is needed for the following query: *"Find all the products in the Catalog with RegPrice > 100"*, assuming we have an XML column named `XCatalog`.

```
XMLQUERY('/Catalog/Categories/Product[RegPrice > 100]' PASSING
XCatalog)
```

Similarly, there is no need for cast in the following query: *"Find all the products with more than 10% discount in the Catalog"*.

```
XMLQUERY('/Catalog/Categories/Product[RegPrice * 0.9 > SalePrice]'
PASSING XCatalog)
```

However, the following query requires a cast: *"Find all the products on sale in the Catalog"*, as comparison `RegPrice < SalePrice` will become an untypedAtomic comparison if the cast is not specified:

```
XMLQUERY('/Catalog/Categories/Product[RegPrice > xs:double(SalePrice)
]' PASSING XCatalog)
```

In the above latter two examples, there is a potential cardinality problem that may cause an error. For expression `RegPrice * 0.9` to work, there can only be one `RegPrice` element under a `Product` element. Likewise, `xs:double()` only takes one item, and it will cause an error if there are multiple `SalePrice`'s for a `Product` element. It is important in XPath programming to avoid cardinality errors. To avoid these potential errors, we can code the two XPath expressions as follows, respectively:

```
XMLQUERY('/Catalog/Categories/Product[RegPrice/(. * 0.9) > SalePrice]'
PASSING XCatalog)

XMLQUERY('/Catalog/Categories/Product[RegPrice > SalePrice/ xs:
double(.)]' PASSING XCatalog)
```

These two XPath expressions use a feature that is not available in XPath 1.0, i.e. to return a sequence of atomic values from the last step of a path expression.

Constructing XML

In DB2 V8, the following XML publishing functions, including XML constructors and other functions, were introduced to construct XML data from relational data: `XMLELEMENT`, `XMLATTRIBUTES`, `XMLNAMESPACES`, `XMLFOREST`, `XMLCONCAT`, and `XMLAGG`. Since there was no external XML data type, the `XML2CLOB` function must be used to get the data out of the DB2 server. These functions provide convenient and high performance alternative to XML construction in applications.

In DB2 9, these functions are extended to handle binary data types using `HEX` or `BASE64` encoding, and take null handling options. New constructors are added to make the constructor set complete; these include: `XMLTEXT`, `XMLPI`, `XMLCOMMENT`, and `XMLDOCUMENT`. Since the functions return the XML data type that is now a first-class SQL type, there is no need to use the `XML2CLOB` function any more.

These functions also take the XML data type as input. You can use these to construct new documents from portions of existing documents extracted by the `XMLQUERY` function. Here is an example of constructing XML from relational data:

```
SELECT XMLDOCUMENT(
        XMLELEMENT(NAME “hr:Department”,
         XMLNAMESPACES(‘http://example.com/hr’ as “hr”),
         XMLATTRIBUTES (e.dept AS “name” ),
         XMLCOMMENT(‘names in alphabetical order’),
         XMLAGG(XMLELEMENT(NAME “hr:emp”, e.lname)
                       ORDER BY e.lname  )
                     ) ) AS “dept_list”
FROM employees e
GROUP BY dept;
```

Notice that you can specify ordering inside the `XMLAGG` function. A sample result may look like the following with formatting spaces inserted for easy reading:

```
<?xml version=“1.0” encoding=“UTF-8”>
<hr:Department xmlns:hr=“http://example.com/hr” name=”Shipping”>
   <!-- names in alphabetical order -->
   <hr:emp>Lee</hr:emp>
   <hr:emp>Martin</hr:emp>
   <hr:emp>Oppenheimer</hr:emp>
</hr:Department>
```

The following is another example to construct a new document, an invoice, from existing data, a purchase order, illustrating how SQL/XML constructors with `XMLQUERY` can be used to achieve some of the XQuery functionality.

```
SELECT XMLDocument(
   XMLElement(NAME “invoice”,
     XMLAttributes( ‘12345’ as “invoiceNo’),
      XMLQuery (‘/purchaseOrder/billTo’ PASSING xmlpo),
      XMLElement(NAME “purchaseOrderNo”,
          PO.ponumber)
      XMLElement(NAME “amount”,
       XMLQuery
        (‘fn:sum(/purchaseOrder/items/item/xs:decimal(USPrice))’
         PASSING xmlpo) )
      )    )
FROM BASICS.PURCHASEORDERS PO,
WHERE PO.ponumber = ‘200600001’;
```

The result may look like this (formatted for ease of reading):

```
<?xml version=“1.0” encoding=“UTF-8”>
<invoice invoiceNo=“12345”>
   <billTo country=”US”>
   <name>Robert Smith</name>
   . . .
  </billTo>
  <purchaseOrderNo>200600001</purchaseOrderNo>
  <amount>188.93</amount>
</invoice>
```

Access methods

DB2 9 introduces several new access methods for XML data. The basic access method is the so-called *DocScan*. It traverses XML data and evaluate XPath expressions using our patent-pending streaming XPath evaluation algorithm, called *QuickXScan*. However, there is no new access type indicator for DocScan in the PLAN_TABLE as it is part of R-Scan if there is an XML column involved.

Three new access type indicators are introduced for XML index-based access. Similar to RID list access, ANDing, and ORing, they include:

- *DocID list access (DX)*.
- *DocID list ANDing (DI for DocID list Intersection)*.
- *DocID list ORing (DU for DocID list Union)*.

As mentioned earlier, XML indexes are only used for the XMLEXISTS predicate evaluation. For example, to evaluate predicate

```
XMLEXISTS('/Catalog/Categories/Product[RegPrice > 100]' PASSING
XCatalog)
```

If you have an XML index on the XCatalog column created with the XML Pattern and type as follows:

```
CREATE INDEX IX1 ON MYTABLE(XCATALOG) GENERATE KEYS USING XMLPATTERN '/
Catalog/Categories/Product/RegPrice' as SQL DECFLOAT
```

DB2 9 will use this index for DocID list access (DX) for the predicate and get unique DocID list from the XML index, then access the base table using the DOCID index and XML table. It will then re-evaluate the document using QuickXScan. Because DB2 9 always re-evaluates XMLEXISTS predicate, the XML pattern of an XML index does not have to exactly match with an XPath expression to apply the index.

Here is another example of using DocID list ANDing (DI) to evaluate a predicate:

```
XMLEXISTS('/Catalog/Categories/Product[RegPrice > 100 and Desc = "Shoe"
]' PASSING XCatalog)
```

Two indexes on the `XCatalog` column with XMLPattern and data types, one is the same as above IX1, and the other is the following IX2:

```
CREATE INDEX IX2 ON MYTABLE(XCATALOG) GENERATE KEYS USING XMLPATTERN '/
Catalog//Desc' as SQL VARCHAR(50);
```

Indexes IX1 and IX2 will be used to get two DocID lists and then DocID list ANDing (DI) will be applied to get a unique DocID list. DB2 9 will then access the base table via the DOCID index and evaluate the predicate through QuickXScan.

XML schema repository

W3C uses a target namespace and optional schema location, both URIs, to identify an XML schema. For example, a target namespace could be "`http://www.ibm.com/software/catalog`" and the schema location could be "`http://www.ibm.com/schemas/software/catalog.xsd`".

However, it is not required to have online schema exist in the specified URIs. In addition, it is not recommended to get a schema online automatically by DB2. Therefore you need to register XML schemas into DB2 XML Schema Repository (XSR) before you can use them in XML schema validation or annotated schema-based decomposition. There is a set of stored procedures for managing an XML schema, and when you register an XML schema, you specify a SQL ID for it. The stored procedures are the following:

- *XSR_REGISTER (rschema, name, schemalocation, xsd, docproperty)*
- *XSR_ADDSCHEMADOC (rschema, name, schemalocation, xsd, docproperty)*
- *XSR_COMPLETE (rschema, name, schemaproperties, isUsedForDecomp)*
- *XSR_REMOVE(rschema, name)*

Assuming you have a schema with schema location '`http://www.n1.com/`**`order.xsd`**' and it also uses two other schema documents '`http://www.n1.com/`**`lineitem.xsd`**' and '`http://www.n1.com/`**`parts.xsd`**' by include or import, and you want to identify this schema using SQL ID `ORDERSCHEMA`, you can use the following stored procedure call sequence, with the root schema document first:

```
XSR_REGISTER('SYSXSR', 'ORDERSCHEMA',
               'http://www.n1.com/order.xsd', :xsd, :docproperty)
XSR_ADDSCHEMADOC('SYSXSR', 'ORDERSCHEMA',
             'http://www.n1.com/lineitem.xsd', :xsd, :docproperty)
XSR_ADDSCHEMADOC('SYSXSR', 'ORDERSCHEMA',
     'http://www.n1.com/parts.xsd', :xsd, :docproperty)
XSR_COMPLETE('SYSXSR', 'ORDERSCHEMA', :schemaproperty, 0)
```

At the `XSR _ COMPLETE` call, DB2 9 will compile the schema into a binary format. When the schema is used, its binary format is loaded to achieve high performance. Any errors will be reported during the compile time also. It invokes Java XML parser so you need Java JDK 1.5 or above installed with DB2 9 server.

Schema validation

To validate an XML document against a registered schema, you invoke the `DSN _ XMLVALIDATE()` UDF. The `DSN _ XMLVALIDATE()` UDF works as the standard `XMLVALIDATE()` SQL function except that it does not retain type annotations. For example, you can validate XML data during INSERT:

```
INSERT INTO BASICS.PURCHASEORDERS VALUES
('2006040001', CURRENT DATE, 'A',
  XMLPARSE(DOCUMENT
    SYSFUN.DSN_XMLVALIDATE(:xmlPo, SYSXSR.ORDERSCHEMA)),
  NULL);
```

DB2 9 invokes a new high performing schema validation parser (XLXP) for validation. However, schema validation is still more costly than parsing only.

Annotated schema-based decomposition

If you want to decompose (shred) an XML document and store the data in regular SQL columns and XML columns of relational tables, you can use a new stored procedure XDBDECOMPXML to achieve this. If you decompose XML data into pure relational data without XML, you are no longer taking the advantages of pureXML technology. You will need to edit schema documents and add annotations to specify how you want the document to be decomposed. The Development Work Bench (DWB) provides a tool to assist the annotation. For details of annotation and the XDBDECOMPXML stored procedure, refer to DB2 9 *XML* Guide [Need URL].

Utilities

All DB2 utilities have been enhanced to handle XML data type and XML related database objects properly or at least recognize the objects. The following is a list of utility features and restrictions:

- *CHECK DATA: checking of base table spaces which contain XML columns.*
- *CHECK INDEX: checking of the DocID, NodeID and XML value indexes.*
- *CHECK LOB: adds error checking to disallow processing of XML table spaces.*
- *COPY INDEX: support taking full image copies and concurrent copies of the DocID, NodeID and XML value indexes.*
- *COPY TABLESPACE: support taking full, incremental image copies and concurrent copies of the XML table spaces.*
- *COPYTOCOPY: support the replication of image copies of XML table spaces, DocID, NodeID and XML value indexes.*
- *EXEC SQL: adds error checking to disallow cross loading of tables with XML columns.*
- *LISTDEF: implements a new XML keyword for constructing lists with and without XML objects.*
- *LOAD: support loading of tables with XML columns.*
- *MERGECOPY: supports merging of image copies of XML table spaces with existing function.*
- *QUIESCE TABLESPACESET: includes XML table spaces and index spaces in the set of quiesced objects.*
- *REAL TIME STATISTICS: gathers existing statistics on the new XML objects.*
- *REBUILD INDEX: supports the rebuilding of the DocID, NodeID and XML value indexes.*
- *RECOVER INDEX: supports the recovery of the DocID, NodeID and XML value indexes and will include XML objects during consistency checking of point-in-time recoveries.*
- *RECOVER TABLESPACE: supports the recovery of the XML table space and will include XML objects during consistency checking of point-in-time recoveries.*
- *REORG INDEX: supports the reorganization of the DocID, NodeID and XML value indexes.*
- *REORG TABLESPACE: supports the reorganization of the XML table space and of base table spaces with XML columns with some restrictions.*

- *REPORT TABLESPACESET: includes XML table spaces, DocID, NodeID and XML value indexes in the set of reported objects.*
- *RUNSTATS INDEX: processes the base table space DocID index normally, collect some statistics for the NodeID index and XML value indexes.*
- *RUNSTATS TABLESPACE: processes the base table space DocID column normally and collect some statistics for all XML table space columns.*
- *UNLOAD: supports the unloading of tables containing XML columns. UNLOAD of XML data FROMCOPY is not supported.*

The database operation and recovery are similar to that of a database with LOB data. The following provides guidelines.

- *To recover base table space, take image copies of all related objects*
  - *Use REPORT TABLESPACESET to obtain a list of related objects*
  - *Use QUIESCE TABLESPACESET to quiesce all objects in the related set*
- *Use SQL SELECT to query the SYSIBM.SYSXMLRELS table for relationships between base table spaces and XML table spaces*
- *COPYTOCOPY may be used to replicate image copies of XML objects.*
- *MERGECOPY may be used to merge incremental copies of XML table spaces.*
- *Point in Time recovery (RECOVER TOCOPY, TORBA, TOLOGPOINT)*
  - *All related objects, including XML objects must be recovered to a consistent point in time*
- *CHECK utilities to validate base table spaces with XML columns, XML indexes and related XML table spaces.*
- *If there is an availability issue with one object in the related set, availability of the others may be impacted.*

Performance monitoring

Since native XML support in DB2 9 is built on top of regular table space structure, there are no special changes in DB2 Performance Expert other than minor things such as new XML locks. XML performance problems can be analyzed through accounting traces and performance traces as usual.

Some configuration information may help you. DB2 9 introduces a new load module `DSNNXML` in the `DBM1` address space for most of native XML processing. Implicit or explicit XMLPARSE invokes z/OS XML System Services within the same address space. XML schema validation invokes a UDF.

Commonalities and differences from DB2 9 for Linux, UNIX and Windows (LUW)

DB2 9 for z/OS XML features are a compatible subset of that of DB2 9 for LUW. The commonalities include:

- *SQL XML data type and DDL, although there are well-known platform-specific options for databases in DDL.*
- *Standard-conforming SQL/XML language with XML query languages (XPath on z/OS, XQuery on LUW).*
- *Indexing: z/OS supports DECFLOAT and VARCHAR(n). LUW also supports VARCHAR(HASHED), DATE and TIMESTAMP.*
- *XML Schema Repository, and schema validation (UDF v.s. BIF).*
- *INSERT/UPDATE/DELETE: versioning in LUW, no versioning in z/OS.*
- *Host language interfaces: PL/I and assembler in z/OS in addition to C/C++, COBOL, Java, and .NET etc.*
- *Annotated schema decomposition*
- *Text search*

The following are z/OS-specific:

- *XPath in SQL/XML only while XQuery is supported in both embedded and top-level in LUW.*
- *XMLTABLE and XMLCAST are not available yet.*
- *XML columns are supported in tables of any encoding DB2 z/OS supports (UTF-8 databases only for LUW).*
- *XML columns are supported in partitioned table spaces and data sharing environment.*
- *Compression is supported for XML table space.*
- *LOAD/UNLOAD, REORG and many utilities are supported for XML objects.*
- *Next-generation parsers are used to provide unprecedented performance.*

Summary

In this whitepaper, we have discussed the business values DB2 9 pureXML brings and some details of the XML features. The flexibility of XML schema and declarative and efficient XML query languages helps eliminate the bottleneck of mapping and schema evolution, improves productivity and quality of application development, and significantly accelerates time-to-market. It can also improve the system performance in processing XML, together with the unparalleled System z reliability, availability and scalability. DB2 9 pureXML marks a new era of database application development, and leads the trend in enterprise XML data management.

For more information

To learn more about IBM DB2 for z/OS visit **ibm.com**/xxxxx/.

Additional resources

[Provide a list of FAQs, etc.?]

TAKE BACK CONTROL WITH **Information Management**

IMW11860-USEN-00