IBM’s new hybrid DB2 puts the full power of a relational engine to work on a truly native XML store that sits side by side with DB2’s relational data repository.
Relational databases drive most businesses of any size today. Popular and important as these databases are, they’re simply not a great match for semi-structured (and hierarchical) content represented in XML. Because enterprises have, in aggregate, trillions of dollars invested in relational data and relational database management systems (RDBMSs), simply replacing RDBMSs with a pure XML store isn’t an option. Adding an XML-only database into the infrastructure adds yet another integration and complexity challenge.

IBM is about to introduce true-native support for both XML and relational data. This evolutionary technology, now in beta tests with a small group of IBM customers, provides hybrid relational/XML storage from the ground up. That means DB2 will no longer need the XML Extender (just as it doesn’t need an SQL Extender). DB2 will simply handle XML natively. (There are varying definitions of “native” XML support. To clear up the confusion about what’s typically called “native” today, see the sidebar on page 45.)

In the hybrid version, XML is handled as a new data type. Nearly every DB2 component, tool, and utility has been enhanced to recognize and handle this new data type. The new storage paradigm retains XML in a parsed, annotated tree form—similar to the XML Document Object Model (DOM)—that’s separate from the relational data store (see Figure 1, page 44).

On top of both data stores (relational and XML) sits one hybrid database engine. That single engine can process XQuery, XPath, SQL, and SQL/XML. The engine features a bilingual query compiler with parsers for both SQL and XQuery. So developers can access information using either language (or both together) according to what makes the most sense in specific situations. A hybrid DB2 provides the flexibility to shift (between XML and SQL) paradigms as information management needs change.

Storing relational and XML data in a database management system that understands and supports both models at every level (from the client, through the engine, down to the disk) provides flexibility and consistently fast performance. The XML data inherits the same backup and recovery, optimization, scalability, and high availability DB2 offers for relational data. Ultimately, a unified XML/relational database keeps things simple by avoiding the need to integrate XML and relational data from separate stores.

**NATIVE BENEFITS**
The first generation of XML support in relational databases was based on either shredding (or decomposing) documents to fit into relational tables or storing documents intact as character or binary large objects (CLOBs or BLOBs). Each of these choices attempts to force XML into a relational model. However, these approaches have serious limitations in capability and performance. The hybrid model stores XML in a model similar to the DOM. The XML data is formatted to buffered data pages for faster navigation and query execution as well as simpler indexing.

When DB2’s true-native XML support debuts with the next major release, existing support for storing XML documents shredded in relational tables or intact as CLOBs and BLOBs will continue. Support for shredding is important because XML can be used to feed existing relational schemas. However, true-native storage offers significant advantages in these areas:

**Storage.** DB2’s native XML technology will store XML with node-level granularity instead of document-level. While interacting with IBM’s native XML support, the abstraction shown is a column of type XML in a relational table. This column has no maximum length and no mandatory constraining XML schema. Any well-formed XML statement can be inserted into that column. Therefore, the following statement is a valid table definition:

Create table dept (deptID int, deptdoc xml)

A table isn’t limited to a single column of any given type, so the following statement is equally valid:

Create table dept2 (deptID int, deptinfo xml, orgchart xml, employees xml)

In the physical storage layer, the primary storage unit in the IBM implementation is a node. A node exists on a page along with other nodes from the same or different documents. Each node is linked not only to its parent, but also to its children. As a result, navigating to a node’s
Thus there may be zero, one, or multiple index entries for a single row in a table (which is significantly different from indexes on relational columns).

You can create indexes on multiple path expressions on any given column of type XML. Therefore, the following statements are also valid:

create index idx1 on dept(deptdoc) generate key using xmlpattern '/dept/employee/name' as sql varchar(35);
create index idx2 on dept(deptdoc) generate key using xmlpattern '/dept/employee/@id as sql int;

Furthermore, path expressions can include both wildcards and descendant-or-self axis traversal, so the following statements are also valid:

Create Index IX3 on dept(deptdoc) generate keys using xmlpattern '/dept/*/name' as sql varchar(20)
Create Index IX4 on dept(deptdoc) generate keys using xmlpattern '//office' as sql double
Create Index IX5 on dept(deptdoc) generate keys using xmlpattern '/dept/employee/*' as sql varchar(20)

Query. XQuery, the new language for querying XML data, is designed to handle diverse schemas, including constructs such as sequences (instead of sets, as in SQL), multiple nested sequences, and sparse attributes. XQuery can also support heterogeneous schemas and dynamic schema changes.

The IBM implementation has no stand-alone XQuery or XPath processor. The basic XQuery and XPath primitives are built directly into the query engine. The query compiler itself is bilingual, having two interoperating query language parsers—one for SQL and the other for XQuery—to generate a new variation of the Query Graph Model designed to process relational and XML data. Because the intermediate query representation is language-neutral, XQuery, SQL and combinations of XQuery and SQL compile into the same...
WHAT IS TRUE NATIVE?

Each of the currently available (non-native) methods for managing XML in relational databases attempts to make XML conform to the relational model in some way. These approaches include:

Shredding. Most major RDBMSs (including DB2) support shredding. Shredding involves defining a relational schema that corresponds to the XML (for example, representing parent/child relationships in the XML as one or more child tables in a referential integrity constraint with its parent) and defining a mapping from the XML data to the relational schema.

Shredding is a good fit in existing relational environments. However, mapping can be complex and fragile, and you must define a mapping for each XML document you want to store. If the XML schema changes, the mapping may no longer be valid or may require a complex change process. Once decomposed, the data ceases to be XML, loses any digital signature, and becomes difficult and expensive to reconstruct (often requiring many joins).

Storing XML as a CLOB. All major vendors support storing entire XML documents in a variable length character type (VARCHAR) or as CLOBs. If XML documents are inserted into CLOB or VARCHAR columns, they are typically inserted as unparsed text objects. CLOBs preserve the original document and provide uniform handling of any XML, including highly volatile schemas. Avoiding XML parsing at insert time guarantees high insert performance. However, without XML parsing, XML document structure is entirely ignored. This precludes the database from doing intelligent and efficient search and sub-document level extract operations on the stored text objects. The only remedy is to invoke the XML parser at query execution time to “look into” the XML documents so that search conditions can be evaluated. The high insert performance comes at the cost of low search and extract performance.

BLOB (pseudo native). BLOB-based storage is conceptually similar to CLOB storage; however, instead of storing the XML data as a parsed string, BLOBs store it in a proprietary post-parse binary representation. This approach is sometimes called pseudo native, because the data representation remains in XML within the BLOB.

However, the underlying storage for a document is virtualized as a single contiguous byte range, which can cause performance problems. Updating can require the entire document to be rewritten and (looked). Access to portions of the document might require the entire document to be read from disk.

True native. True native storage holds the post-parsed data on disk, enabling individual nodes of the data model to be stored independently—that is, not as a stream—and then interconnected. True native storage provides the advantages of BLOB and CLOB, but resolves the remaining performance issues because the document storage isn’t virtualized as a single contiguous byte range. The storage for the entire set of documents is virtualized as a contiguous byte range; however, individual nodes can be relocated in this range with minimal impact on other nodes and indexing.

Native XML Storage improves on the BLOB approach because it provides more consistent behavior as the size of documents increases or when the amount of data to access is a small percentage of the total document size.

Right Model, Right Task

A true-native XML data store does more than expose XML to its clients—it represents the XML as XML throughout the entire data engine stack (from client to disk and back out again).

Hybrid systems don’t mandate that all data be represented as relational data, nor do they require that all data be in XML; instead, they provide the choice of the right model for the right task.

Deeper levels of nesting (SQL within XQuery in which SQL itself contains nested XQuery) is supported.

Integration and performance. With IBM’s hybrid approach, there’s no need to predefine XML schema, limit documents to a given schema, or provide any mapping between XML and relational models.

The hybrid approach offers an important advantage over shredding: It eliminates the cost of joins and other processing necessary to reconstruct XML documents. In the case of complex documents, these costs can be very significant.

When compared to CLOB approaches, truly native storage eliminates the need to parse XML documents at query time. Given XML parsing costs, CLOB-based approaches are impractical if any form of search into the document—that is, parsing—is necessary. CLOB should be considered only when the usage models are expected to be full document insertion, search by purely relational attributes, and full document retrieval.

Native storage improves on the BLOB approach because it provides more consistent behavior as the size of documents increases or when the amount of data to access is a small percentage of the total document size.

[whwysquare]

WHERE $e/office = 344
Return $e/name

Deeper levels of nesting (SQL within XQuery in which SQL itself contains nested XQuery) is supported.

Integration and performance. With IBM’s hybrid approach, there’s no need to predefine XML schema, limit documents to a given schema, or provide any mapping between XML and relational models.

The hybrid approach offers an important advantage over shredding: It eliminates the cost of joins and other processing necessary to reconstruct XML documents. In the case of complex documents, these costs can be very significant.

When compared to CLOB approaches, truly native storage eliminates the need to parse XML documents at query time. Given XML parsing costs, CLOB-based approaches are impractical if any form of search into the document—that is, parsing—is necessary. CLOB should be considered only when the usage models are expected to be full document insertion, search by purely relational attributes, and full document retrieval.

Native storage improves on the BLOB approach because it provides more consistent behavior as the size of documents increases or when the amount of data to access is a small percentage of the total document size.

RIGHT MODEL, RIGHT TASK

A true-native XML data store does more than expose XML to its clients—it represents the XML as XML throughout the entire data engine stack (from client to disk and back out again).

Hybrid systems don’t mandate that all data be represented as relational data, nor do they require that all data be in XML; instead, they provide the choice of the right model for the right task.

Anjul Bhambhri
[bhambhri@us.ibm.com] is the senior development manager for XML support in DB2 and heads the XML effort across DB2 UDB.