

# Enhance business insight and scalability of XML data with new DB2 V9.7 pureXML features

Skill Level: Introductory

[Cynthia M. Saracco \(saracco@us.ibm.com\)](mailto:saracco@us.ibm.com)  
Senior Solution Architect  
IBM

[Matthias Nicola \(mnicola@us.ibm.com\)](mailto:mnicola@us.ibm.com)  
Senior Software Engineer  
IBM

23 Apr 2009

New database design, administration, and development features for pureXML are available in IBM DB2® for Linux®, UNIX®, and Windows®, Version 9.7 (announced April 22, 2009). Find out more about how these technologies can help companies integrate XML data more effectively into business intelligence environments and how companies can cope with growing XML data volumes. This article summarizes the new pureXML capabilities, explains how they can be used, and discusses sample application scenarios.

## Introduction

With many organizations striving to create flexible and reliable IT environments that offer greater insight into critical business operations, core information management systems are under more pressure to adapt to ever-changing business needs. To help companies address this challenge, IBM enhanced its DB2 9.7 features for pureXML in several important ways.

New features in DB2 9.7 enable administrators to leverage new database design options for XML data, including hash partitioning (database partitioning), range partitioning (table partitioning), and multi-dimensional clustering. These options can help companies accommodate large data volumes, exploit parallel processing environments, simplify the addition or removal of time-sensitive data, and speed

performance of many types of queries. Used individually or together, these DB2 design options enable organizations to incorporate XML data into their relational data warehouses; create operational data stores for XML messages, documents, and data feeds; and improve scalability of XML transaction processing workloads.

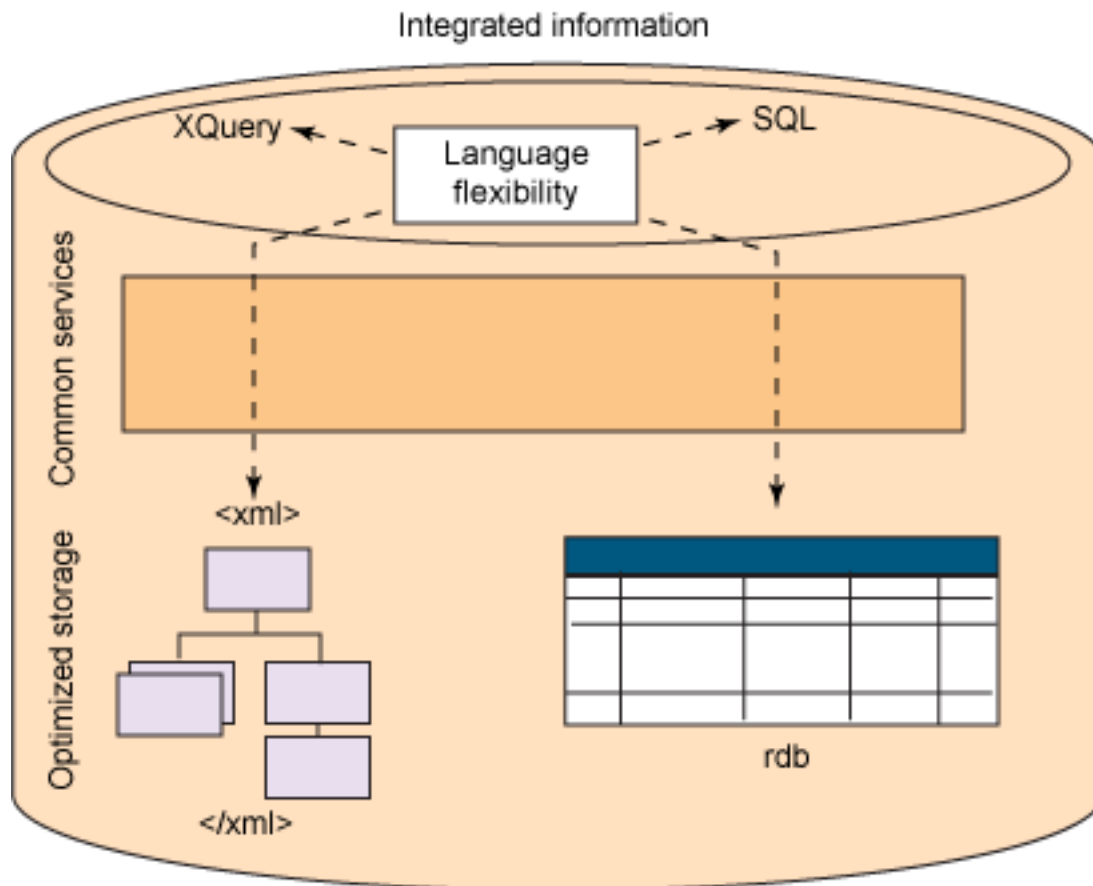
These new database design options are just part of the pureXML enhancements offered in DB2 9.7. This article introduces you to these and other new pureXML capabilities, explaining when they can be most useful and providing tips to help you get started. This article discusses the following:

- [A short overview of DB2 pureXML](#) for those who aren't already familiar with the technology.
- [Sample use cases](#) and application scenarios for DB2 pureXML.
- [Hash-based partitioning](#), which provides significant scalability gains.
- [Range-based partitioning](#), which helps companies *roll in* and *roll out* data over time (a common requirement in data warehouses).
- [Multi-dimensional clustering](#), which often improves performance of analytic queries.
- [Compression of XML data and indexes](#), which improves storage efficiency and runtime performance for certain workloads.
- [User-defined functions](#) that consume, process, and return XML data. Such functions encourage code reuse and simplify the development and maintenance of queries.
- [New stored procedures, system-supplied functions, performance enhancements, and other features](#) that simplify DB2 pureXML administration and application development.

## Reviewing DB2 pureXML

Since 2006, DB2 9 has provided companies with a common application programming interface and database management platform for data modeled in tables and XML hierarchies. This hybrid database management architecture, as shown in Figure 1, enables companies to extend their traditional relational database environments to directly manage XML messages and documents without the need to *shred* or map this data into individual columns of various tables (thereby transforming XML into traditional SQL data types). Instead, XML data can be stored intact in its native hierarchical format alongside relational data. Applications can retrieve relevant portions of the XML data with ease and efficiency. Applications can also easily integrate XML data and relational data.

**Figure 1. DB2 9 architecture with built-in support for relational and XML data**



To leverage DB2 pureXML capabilities, administrators create a table with one or more columns of type XML. XML indexing, query optimization, storage management, and other DB2 services help ensure efficiency and strong runtime performance.

Listing 1 illustrates how simple it is to work with XML data in DB2. The code in Listing 1 does the following:

1. Creates a table with relational and XML columns
2. Indexes a specific portion of the XML column
3. Inserts data into the table
4. Issues queries (in simple SQL, SQL/XML, and XQuery)
5. Updates the value of an XML element stored in an XML document

## Listing 1. Working with DB2 pureXML

```
-- Create a table with an integer and an XML column
CREATE TABLE customer (cid INTEGER, info XML);

-- Create an XML index for customer zip code data
CREATE INDEX idx1 ON customer(info) GENERATE KEYS USING
XMLPATTERN '/customerinfo/addr/zip' AS SQL VARCHAR(5);

-- Populate the table with data using a simple INSERT statement
INSERT INTO customer (cid, info) VALUES (?,?);

-- Retrieve relational data and full XML documents
-- for customers using simple SQL
SELECT cid, info
FROM customer
WHERE cid > 1234;

-- Retrieve names of customers in a specific zip code
-- who have an ID of > 1234 using SQL/XML
SELECT XMLQUERY('$INFO/customer/name')
FROM customer
WHERE cid > 1234 and
XMLEXISTS('$INFO/customer/addr[zip = 95123]');

-- Retrieve an XML element that lists the names of customers
-- in a given zip code using XQuery
xquery for $i in db2-fn:xmlcolumn("CUSTOMER.INFO")/customer
where $i/addr/zip = 95123
return <myresult>{$i/name}</myresult> ;

-- Update XML element value related to zip code
UPDATE customer
SET INFO = XMLQUERY('copy $new := $INFO
                    modify do replace value of $new/customer/addr/zip
                        with 95141
                    return $new')
WHERE ...;
```

## Exploring sample use cases

Service-oriented architectures (SOA), Web-centric applications, and integration projects based on industry-specific standards often rely on XML to define how important business data is represented and exchanged. Furthermore, auditing and compliance initiatives often require that complete records of various business transactions remain accessible for some period of time. Increased use of XML is prompting many companies to evaluate how their databases can be adapted to cope with the complex, highly variable structures inherent in many XML messages and documents. XML data management is impacting both transaction-oriented environments and analytic applications.

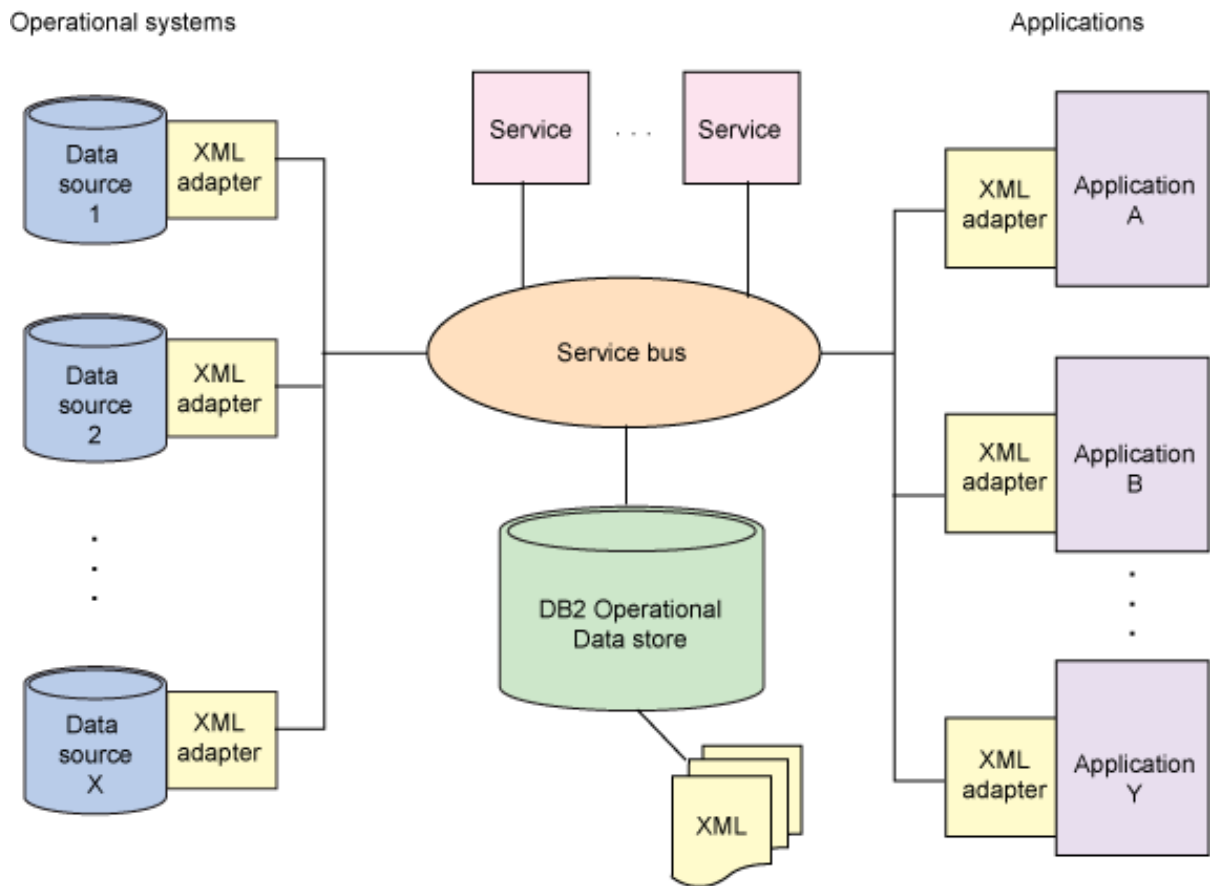
### Integrating operational data

The flexible, self-describing nature of XML makes it well-suited for representing a diverse range of business artifacts. Such artifacts are often stored in existing

relational database management systems or file systems, or the artifacts are generated dynamically by Web services, real-time data feeds, or other software. Processing and analyzing information from these diverse sources presents an obvious challenge. Companies can benefit by maintaining a shared XML-based repository. The variable nature of XML structures accommodates a wide range of business needs. An operational data store (ODS) built on XML can reduce application development costs and can provide an agile infrastructure to accommodate evolving data management demands.

Figure 2 illustrates a sample architecture in which DB2's pureXML technology is used as an operational data store to integrate data from multiple sources on behalf of various business applications. Newer data sources and applications might not even require XML adapters, because they might use XML as their native data exchange format.

**Figure 2. DB2 pureXML as an integrated operational data store**



**Extending a data warehouse**

Since their early adoption in the 1990s, data warehouses have played an increasingly important role in helping companies analyze trends and improve their

business strategies. While relational database management remains the appropriate technology of choice for data warehouses, adding XML data management capabilities can introduce greater flexibility, enabling companies to support evolving business reporting and analysis requirements without incurring significant database schema changes or rewrites of existing application code.

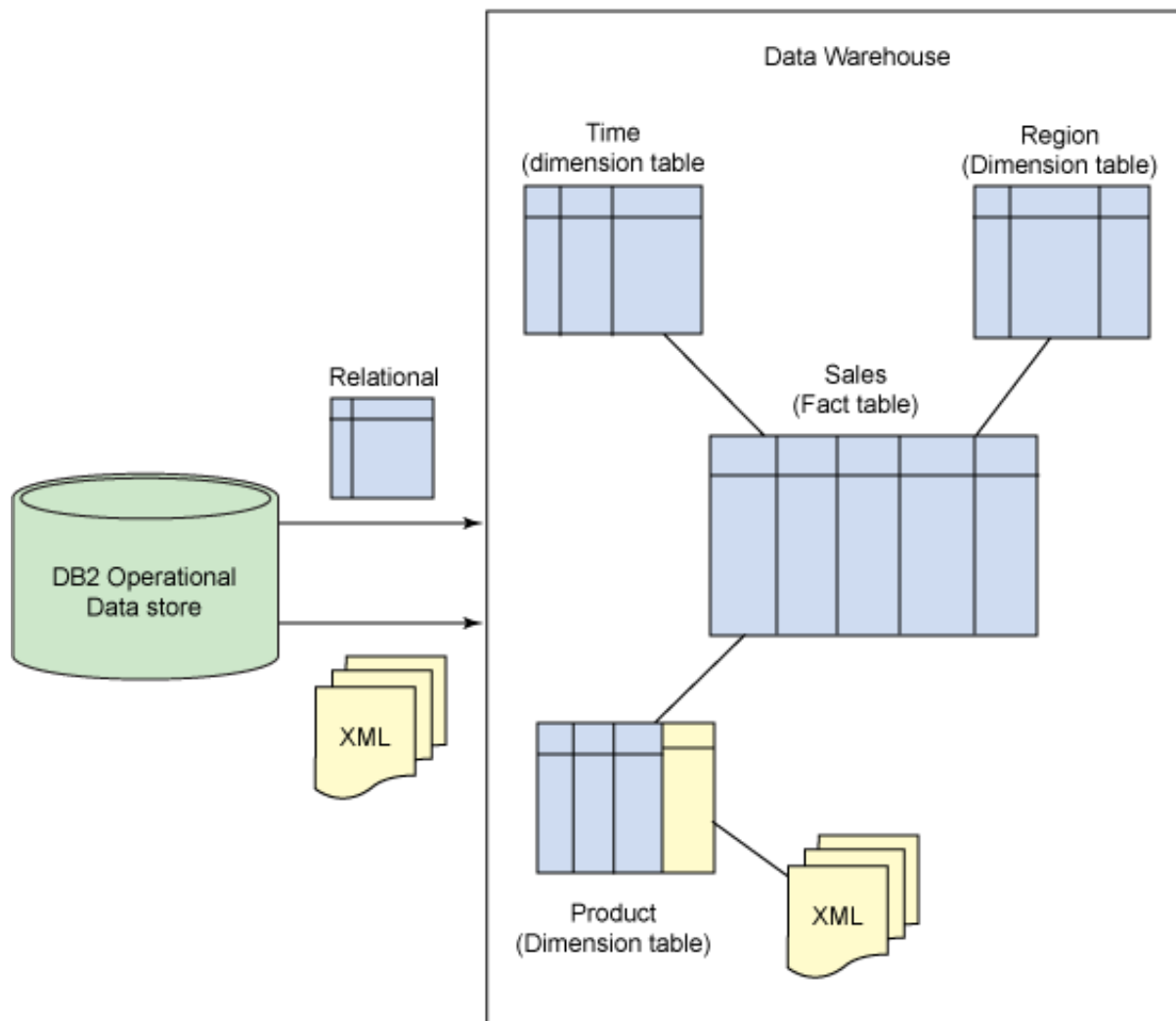
For example, consider a data warehouse that tracks sales information. Using a typical star schema database design, a *fact* table might contain sales data by product, region, and time period. Data in this fact table would typically be joined with data in multiple *dimension* tables to obtain specific details about various products, regions, and so on. Unfortunately, developing a detailed database design for such a warehouse can be challenging.

Consider the challenges involved in designing an appropriate product dimension table if a diverse range of products must be represented. Different types of products have different attributes, making it difficult to pre-determine which attributes of which products should be exposed to business analysts and executives. Quite likely, analysts will want to drill down or slice-and-dice the data about product sales in unanticipated ways, such as wanting to explore sales of women's sweaters by size, color, fabric, neckline, sleeve length, and so on. With a relational-only design, each possible attribute of interest for each product would need to be captured in its own column, leading to a large, unwieldy dimension table. Because attributes vary widely from one type of product to another, such a table would have many rows with sparsely populated columns, which is quite inefficient. And, as new products and new product attributes are introduced over time, the database schema (and any applications that depend on it) would need to be altered, which can be quite costly. Adding a column to a table in a corporate warehouse can also be subject to lengthy review and approval processes in many IT organizations.

A less disruptive way to accommodate such changing business needs is to use one or more XML columns in the data warehouse schema. Commonly used attributes can remain captured in relational columns, while additional details can be maintained in an XML column that readily accommodates variable structures and is easily accessible for query and reporting. Using the previous example, an XML column might be used in the dimension table containing product data. New product attributes that need to be tracked would simply be included as new elements in the XML documents associated with the target products. The database schema would not need to change.

Figure 3 illustrates one way in which a data warehouse might be extended with XML.

**Figure 3. A relational data warehouse extended with XML data management capabilities. The fact table and all dimension tables can include XML columns, although this example shows only a one-dimension table with one XML column.**



In Figure 3, an operational data store periodically feeds the data warehouse new information. Some of this information might be XML, while other information might be relational. XML columns can be added to either dimension or fact tables as needed, although Figure 3 illustrates XML data in only one dimension table (the table for tracking details about various products).

### Supporting XML-centric applications

While DB2 pureXML technology can augment many decision support and analytic databases, it's also well-suited to helping companies manage the increased volumes of XML messages and documents that many service-oriented architectures generate for transaction processing applications. The growing use of electronic forms and Web services are just two factors that contribute to increased volumes of XML data.

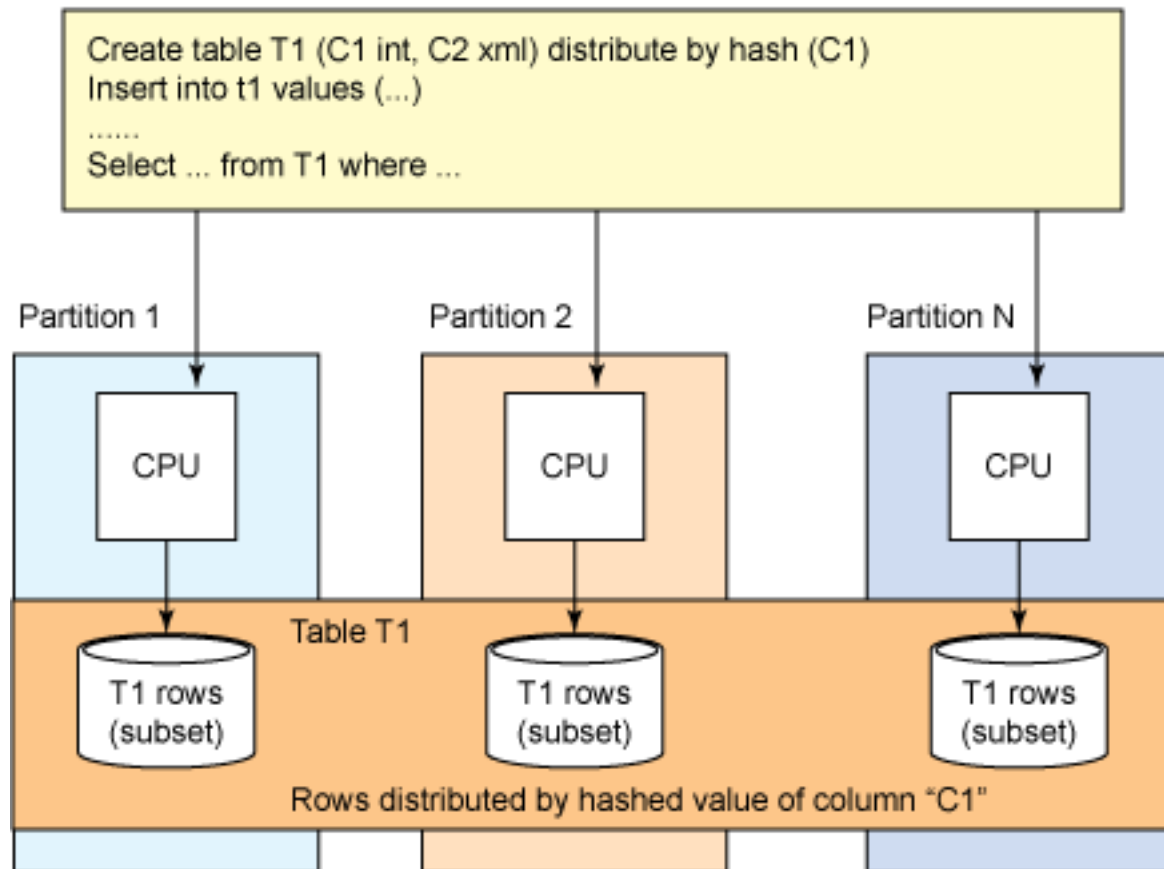
DB2 helps companies cope with increased volumes of XML data by eliminating the need to shred XML into columns of various tables. Storing XML in its native

hierarchical format and making XML immediately accessible through industry-standard XPath, XQuery, and SQL/XML expressions reduces administrative costs and simplifies application development.

## Using hash (database) partitioning for maximum scalability

As data volumes grow, distributing the contents of a database across multiple processors and storage devices can help companies achieve linear scalability. DB2 9.7 extends the DB2 database partitioning feature (DPF) to support both XML and relational data. Previously, DPF supported only relational data.

**Figure 4. Sample DB2 architecture with hash-based partitioning, which allows for parallel processing of queries and other database operations**



As Figure 4 shows, DPF is a physical database design option that uses multiple separate database partitions in a multi-processing environment. Multiple database partitions can be created within a single SMP (symmetric multiprocessing) machine or can be distributed across separate machines in a *shared nothing* environment.

DPF can be useful for read-intensive workloads, including those common to data

warehouse environments. With DPF, each row of a given table is placed in a specific database partition based on the hashed value of the table's distribution key, which is defined when the table is created. When data is read from or written to the database, DB2 automatically directs appropriate work to the relevant partitions. As a result, computing resources associated with multiple partitions can work in parallel to satisfy the original user request. Linear scalability is achieved by adding new partitions as data volumes grow. The DB2 Design Advisor, a built-in administrative tool, can advise administrators about their partitioning design.

To simplify administration of XML data and provide maximum scalability, DB2 9.7 enables XML data to be distributed across multiple database partitions. The data distribution allows many operations to be automatically parallelized, including loading, inserting, querying, updating, deleting, validating, and publishing XML data. In particular, complex and potentially long-running analytical queries can be divided and parallelized, which significantly improves response times.

As with earlier releases of DB2, it is important to select a distribution key that evenly distributes rows across partitions. The distribution key must consist of relational columns, and it cannot reference an XML column. Ideally, this key should contain data with many distinct values to avoid unevenly sized partitions.

Exploiting DPF for XML data in DB2 9.7 is very similar to exploiting DPF for relational data. Specifically, an administrator must define appropriate database objects, such as partition groups, table spaces, buffer pools, and so on. Tables must be created with the `DISTRIBUTE BY HASH` clause of the `CREATE TABLE` statement.

Listing 2 creates a `SALES` table with relational columns for `ORDERID`, `PERSONID`, and `SALESDATE`, as well as an XML column to capture `DETAILS` about the order. Note that the values of the `ORDERID` column dictate how rows will be partitioned for this table.

### Listing 2. Creating a hash-partitioned table that includes an XML column

```
CREATE TABLE sales (  
   orderid      INT NOT NULL,  
    personid   INT,  
    salesdate  DATE,  
    details    XML)  
DISTRIBUTE BY HASH (orderid)
```

Full text search of data of hash-partitioned tables using the Net Search Extender (NSE) is now available.

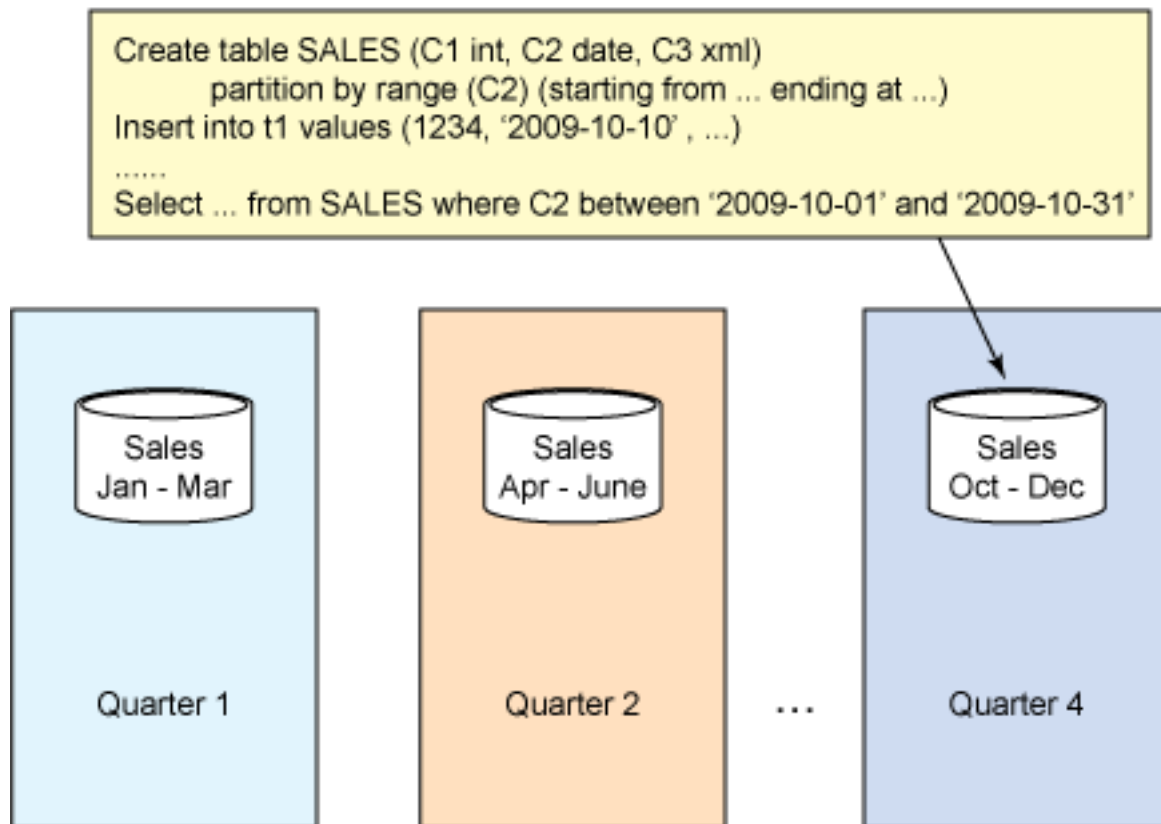
## Using range partitioning to roll in and roll out data over time

A frequent requirement for data warehouses and business intelligence environments involves maintaining a rolling history of select data over a given period of time. For example, a company might want to maintain a rolling five-year sales history so business analysts can assess buying patterns and evaluate emerging trends. In such a scenario, old data might need to be purged or archived on a monthly or quarterly basis (rolled out), and new data might need to be loaded in a similar timeframe (rolled in).

Range partitioning addresses this administrative requirement, and DB2 9.7 extends prior support for this technology to include XML data. Range partitioning—sometimes called table partitioning—segments a table based on the range of values contained in one or more columns. Typically, the partitioning key is time-based, resulting in a design that directs data for a given week, month, or quarter to be stored in a certain partition. Because each partition is treated as a separate database object, administrators can readily roll in (attach) new data or roll out (detach) old data. In addition, many queries enjoy superior runtime performance because DB2 automatically avoids accessing data in partitions that are irrelevant to the user request.

Figure 5 illustrates a sample DB2 environment that partitions data in a sales table by quarter.

**Figure 5. In range-partitioned environments, DB2 targets only the necessary partitions to satisfy a user's request**



Managing a range-partitioned table that includes one or more XML columns isn't much different from managing a range-partitioned table with only relational columns. In particular, the previously supported SQL statements for creating and altering tables for range partitioning as well as attaching and detaching partitions still apply. In addition, the partitioning key must still be based on relational data.

Listing 3 creates a range-partitioned table with relational and XML data, rolls out (detaches) a partition containing old data, and rolls in (attaches) a partition containing recent data.

### Listing 3. Using range partitioning with DB2 pureXML

```
-- Create a range-partitioned table
CREATE TABLE salespart (
  orderid      INT,
  orderdate    DATE,
  ordermonth   INT NOT NULL GENERATED ALWAYS AS
(month(orderdate)),
  orderyear    INT NOT NULL GENERATED ALWAYS AS
(year(orderdate)),
  customerid   INT,
  salesrepid   INT,
  details      XML)
PARTITION BY RANGE (orderyear, ordermonth)
(PART q109 STARTING(2009, 1) ENDING (2009, 3) INCLUSIVE,
```

```
PART q209 ENDING (2009, 6) INCLUSIVE,
PART q309 ENDING (2009, 9) INCLUSIVE,
PART q409 ENDING (2009, 12) INCLUSIVE);

-- Insert or load data for 1Q - 4Q 2009 sales into the table
. . .

-- Create another table to contain new sales data to be
attached
CREATE TABLE currentsales (
   orderid INT,
    orderdate DATE,
    ordermonth INT NOT NULL GENERATED ALWAYS AS
(month(orderdate)),
    orderyear INT NOT NULL GENERATED ALWAYS AS
(year(orderdate)),
    customerid INT,
    salesrepid INT,
    details XML) ;

-- Insert or load new sales data for 1Q 2010 into the
"currentsales" table
. . .

-- Attach a new partition for the 1Q 2010 sales data.
-- Perform an integrity check for index maintenance, range
checking, etc.
ALTER TABLE salespart ATTACH PARTITION q110
STARTING (2010, 1) ENDING (2010, 3) INCLUSIVE
FROM TABLE currentsales ;

SET INTEGRITY FOR salespart IMMEDIATE CHECKED;

-- Detach the partition containing old sales data from 1Q 2009
ALTER TABLE salespart DETACH PARTITION q109 INTO OLDSALES;
```

Full text search of data in range-partitioned tables using the Net Search Extender (NSE) is now available.

## Using multi-dimensional clustering for query performance

Multi-dimensional clustering for tables containing XML data is another new database design option. Prior releases support this capability only for tables that don't have XML columns. Multi-dimensional clustering can be particularly useful for analytical applications, which typically issue queries spanning data contained in multiple columns.

For example, an analytical application might require sales information contained in a large fact table to be retrieved by product, by region, and by date (three dimensions). To support such a query, an administrator can use multi-dimensional clustering to instruct DB2 to physically organize rows in the SALES table by these dimensions. Because the rows related to sales for the same product in the same region and time period would be co-located in one or more data blocks, this design would help reduce I/O and improve runtime performance of multi-dimensional queries. In addition, multi-dimensional clustering can also improve the performance of reorganizing, inserting, and deleting data.

With DB2 9.7, tables containing XML columns can participate in multi-dimensional clustering, provided that the clustering dimensions are defined by relational columns. As in previous releases, administrators use the `ORGANIZE BY DIMENSION (...)` clause of the `CREATE TABLE` statement to specify multi-dimensional clustering. Listing 4 creates a table with multi-dimensional clustering of sales data by product, region, and time.

#### Listing 4. Creating a table for relational and XML data using multi-dimensional clustering

```
CREATE TABLE salesMDC (  
  id          INT,  
  product     VARCHAR(25),  
  region      VARCHAR(25),  
  time        DATE,  
  details     XML)  
ORGANIZE BY (product, region, time)
```

## Exploring XML data and index compression for storage efficiency and performance

Compressing XML data and XML indexes can improve storage efficiency and runtime performance of queries that are I/O bound. DB2 9.7 offers new options in both areas.

Compressing XML data can be done in two ways:

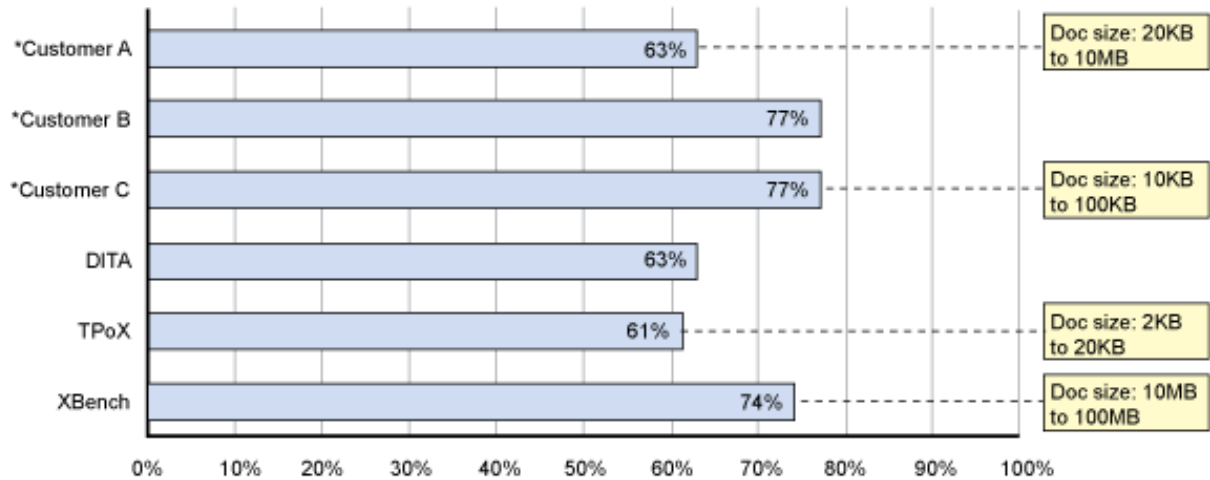
- Small XML documents that occupy less than 32KB of storage space can be *inlined* with any relational data present in the same row, and the entire row can be compressed. This inlining capability, which was introduced in DB2 9.5, remains a useful option.
- Larger XML documents that reside in a data area separate from relational data can also be compressed. By default, DB2 places XML data in a separate location called the XML Data Area (XDA) to handle documents up to 2GB in size. Compression of XML data stored in the XDA is included in the new release of DB2.

In DB2 9.7, the compression capability for XML data is enabled simply through the `COMPRESS YES` clause of the `CREATE TABLE` statement, which compresses both relational and XML columns in the table. To achieve best compression results, DB2 uses two separate compression dictionaries, one for relational columns and one for the XML Data Area of the table. Both dictionaries are automatically generated after the table has been populated with a few megabytes of data. Alternatively, you can

alter a table to enable compression, and then you can reorganize it to compress the data.

Early tests conducted at IBM indicate that compressing XML data stored in the XML Data Area often results in 60 to 80 percent disk space savings, as shown in Figure 6.

**Figure 6. Compressing XML data stored in the XDA can result in 60 to 80 percent disk space savings**



\*Data sets from DB2 customers

Figure 6 shows results from IBM testing six different data sets supplied by customers or available in the public domain. These data sets included documents of various sizes that ranged from 2KB to 100MB.

The degree to which XML data can be compressed depends on the structure and complexity of a given document as well as other factors, so your experience might vary from the test results shown. You can query the administrative view SYSIBMADM.ADMINTABCOMPRESSINFO to determine the degree to which a table has been compressed. Listing 5 shows how querying this view produces information about the percentage of space saved due to the compression of relational data and XML, respectively.

**Listing 5. Exploring compression of relational and XML data**

```
SELECT tabname, object_type, pages_saved_percent
FROM sysibmadm.admintabcompressinfo
WHERE tabname = 'SALES';

TABNAME  OBJECT_TYPE  PAGES_SAVED_PERCENT
-----
SALES    DATA        69
SALES    XML          73

2 record(s) selected.
```

---

Compression reduces disk space needs, which often leads to lower disk I/O and increased buffer pool hit ratios (because compressed pages allow more data to be cached). In many cases, the performance gain due to reduced I/O and better memory utilization outweighs the extra CPU cycles required to compress and decompress data.

New technology in DB2 9.7 also enables relational and XML indexes to be compressed. Indeed, indexes created over compressed tables are themselves compressed by default. Alternatively you can use the `COMPRESS [YES|NO]` clause of the `CREATE INDEX` statement to choose whether an index is compressed independently from the compression of the table. Similar to data compression, compressed indexes reduce physical I/O and increase buffer pool hit ratios, which often leads to a net performance gain. A new table function (`SYSPROC.ADMIN_GET_INDEX_COMPRESS_INFO`) enables administrators to determine page savings for compressed indexes and to estimate potential savings if uncompressed indexes were to be altered for compression.

## Understanding user-defined functions for application flexibility

For some time, DB2 has enabled application developers to create user-defined functions that can be invoked from queries or other SQL statements. Creating such functions encourages code reuse and simplifies query development by incorporating code for commonly needed (and potentially complex) operations into a single module accessible to a variety of developers. Instead of hand-coding these operations into various queries, developers can simply call the function from each query that requires it.

DB2 9.7 supports the XML data type in user-defined functions. Input and output parameters can be of type XML, and user-defined functions that are written in SQL can include XML variables and SQL/XML statements.

Writing user-defined functions that work with XML isn't much different from writing user-defined functions that work with relational data types. The functions can be coded to return a single value (for scalar functions) or multiple values (for table functions). The latter can be particularly useful if you need to extract and return repeating elements from an XML document, such as a sequence of phone numbers or email addresses for a given customer.

Two short examples illustrate how simple it is to create and invoke user-defined functions that work with XML data. Listing 6 creates a scalar function that extracts a given XML element from an input document and shows how this function can be invoked in a simple SQL query to return the name of a given customer. (The "#" symbol is used here to indicate statement terminations.)

## Listing 6. Creating and invoking a scalar user-defined function that works with XML data

```

--- Create the user-defined scalar function called "getname"
CREATE FUNCTION getname(doc XML)
RETURNS VARCHAR(25)
BEGIN ATOMIC
    RETURN XMLCAST(XMLQUERY('$d/customerinfo/name'
                            PASSING doc AS "d")
                   AS VARCHAR(25));
END #

-- Invoke the "getname" scalar function and inspect the result
SELECT getname(info) AS name
FROM customer
WHERE cid = 1002 #

NAME
-----
Jim Noodle

1 record(s) selected.

```

Creating a table function that returns multiple values isn't much more difficult. Listing 7 creates a table function that returns information about the phone numbers for each customer contained in the supplied XML document. This function can be easily invoked from a simple SQL query.

## Listing 7. Creating and invoking a table user-defined function that works with XML data

```

-- Create the user-defined table function called "getname"
CREATE FUNCTION getphone(doc XML)
RETURNS TABLE(type VARCHAR(10), number VARCHAR(20))
BEGIN ATOMIC
    RETURN
        SELECT type, number
        FROM XMLTABLE('$d/customerinfo/phone' PASSING doc AS "d"
                     COLUMNS
                        type VARCHAR(10) PATH '@type',
                        number VARCHAR(20) PATH '.');
END #

--- Invoke the "getphone" table function and inspect the result
SELECT cid, p.type, p.number
FROM customer, TABLE(getphone(info)) p
WHERE cid = 1004 #

CID          TYPE          NUMBER
-----
1004        work          905-555-4789
1004        home          416-555-3376

2 record(s) selected.

```

Of course, DB2 continues to support using the XML data type in stored procedures, which is a capability first offered in an earlier release.

## Discovering more enhancements for administration, application development, and performance

Other pureXML technologies in DB2 9.7 provide more support for administering databases, simplifying application development, and improving runtime performance. This section introduces you to these additional features.

### Analyzing XML inlining

Two new system-supplied functions enable administrators to analyze *inlining* of small XML documents. One such function, `SYSPROC.ADMIN_IS_INLINED`, enables administrators to determine if DB2 was able to inline input XML documents based on the maximum inline length specified when the table was created (or altered). Another function, `SYSPROC.ADMIN_EST_INLINE_LENGTH`, enables administrators to estimate the minimum inline length that would need to be specified to allow DB2 to store a given XML document on the same page as the relational data in the row. Such features can help administrators fine-tune their physical database design.

Listing 8 shows how these new DB2 functions can be used.

#### Listing 8. Analyzing XML data inlining with new DB2 functions

```
-- Create the customer table with a maximum inline
-- length of 1000 bytes for XML data
CREATE TABLE customer(id int, xmlcol XML INLINE LENGTH 1000);

--- Insert or LOAD some data into the table
INSERT INTO customer VALUES (...);
. . .

-- Query the table using two new DB2 functions
-- for analyzing XML inlining
SELECT id, ADMIN_IS_INLINED(info) AS inlined,
        ADMIN_EST_INLINE_LENGTH(info) AS inline_length
FROM customer;

-- Inspect the result set.
-- "1" in the second column indicates that the XML was inlined.
-- "0" in the second column indicates that the XML was not
inlined.
-- "-1" in the third column indicates that the XML is too big
-- to be inlined with the current page size.
```

ID	INLINED	INLINE_LENGTH	
1000	1	770	-- Inlined. Uses approx 770 bytes
1001	0	2345	
1002	1	796	
1003	0	1489	-- Not inlined. Inline size of at least 1489 needed
1004	0	1910	
1005	0	-1	-- Too large to be inlined for the given page size

Listing 8 describes how to create a table with an inlined XML column, where XML documents are stored within the relational row if they don't occupy more than 1000 bytes and stored *outlined* in the XML Data Area otherwise. After inserting some data into the table, invoke the two new functions to determine (a) if the XML document in a row was inlined, and (b) the estimated length of the XML document in the current row that would allow the document to be inlined.

## Creating and reorganizing XML indexes online

In previous versions of DB2, creating or reorganizing an index over XML data prevented insert, update, or delete transactions from modifying data in the table while the index was being built. New enhancements in DB2 lift this restriction, allowing for greater flexibility and avoiding application delay or downtime when an XML index needs to be created or reorganized. As a result, DB2 provides increased data availability to applications.

With DB2 9.7, XML index creation allows concurrent write operations by default, just like the creation of relational indexes. When reorganizing XML indexes, database administrators can use the `ALLOW WRITE ACCESS` clause in the `REORG INDEXES` command to allow concurrent writes on the table.

## Decomposing multiple XML documents

The new version of DB2 extends built-in decomposition (or shredding) facilities to work with multiple XML documents. Previous DB2 releases permitted one XML input document for each decomposition operation. In the new DB2 release, a new system-supplied stored procedure (`XDB_DECOMP_XML_FROM_QUERY`) takes an existing DB2 table as input, effectively enabling administrators to decompose data contained in a given column. For database designs that require shredding some or all of the contents of XML data, invocation of this stored procedure can be particularly useful after a bulk load of XML or BLOB data. As with previous releases, DB2's decomposition facility relies on an annotated XML schema to map XML attributes and elements to specific columns of relational tables.

## Retrieving XML schema validation diagnostics

A system-supplied stored procedure, `XSR_GET_PARSING_DIAGNOSTICS`, provides programmers with detailed diagnostic information about any errors detected when DB2 parses an XML document or validates an XML document against an XML schema. This capability, first introduced in DB2 9.5, Fix Pack 3, can help programmers pinpoint problem areas and correct the XML as needed.

If an XML document is not well-formed or is invalid for a given XML schema, invoke the `XSR_GET_PARSING_DIAGNOSTICS` procedure with the document and optionally use the XML schema as input. The procedure produces detailed error information, including:

- The line and column number of the error position in the textual XML document
- An XPath that points to the error location in the document
- The original error message, reason code, and any applicable error tokens

## Using global temporary tables

New support for including XML columns in declared global temporary tables deserves brief mention, because it can help programmers to improve the runtime performance of their applications. Declared global temporary tables enable programmers to retrieve data they will frequently use in their applications, cache them in a temporary table, and manipulate the data repeatedly during their application sessions. Locking and logging are minimized, and the table's contents are deleted when the session terminates.

## Improving SQL access to XML data

Query optimization technology in DB2 9.7 provides increased efficiency for processing queries against relational views of XML data. DB2 automatically employs this technology when appropriate, so programmers don't need to do anything to enjoy the potential runtime performance benefits.

DB2 9.7 promotes the use of XML indexes to process queries against relational views of the XML data. To understand how this works, step through a brief example. Listing 9 creates a table of employee data with an XML column. It also creates an XML index on employee office numbers that might appear as XML elements in employee records. Finally, it creates a view that extracts and exposes employee IDs, first names, last names, and office numbers as relational columns.

### Listing 9. Creating a table, XML index, and relational view

```
-- Create a table with an XML column.
CREATE TABLE emp(doc XML);

-- Create an XML index on the table.
CREATE INDEX officeIdx ON emp(doc) GENERATE KEYS USING
XMLPATTERN '/dept/employee/office' AS SQL VARCHAR(20);

-- Create a relational view of the XML data managed by the
table.
CREATE VIEW emp_rel_view(id, first_name ,last_name ,office) AS
SELECT X.* FROM emp,
  XMLTABLE ('$d/dept/employee' passing doc as "d"
  COLUMNS
    empID      INTEGER      PATH '@id',
    firstname  VARCHAR(5)   PATH 'name/first',
    lastname   VARCHAR(5)   PATH 'name/last',
    office     INTEGER      PATH 'office') AS X;
```

Now consider the SQL statement in Listing 10 that queries the view to select the ID and the first name of all employees that occupy office R344.

### Listing 10. Querying a view with filtering criteria (a query predicate)

```
SELECT id, first_name
FROM emp_rel_view
WHERE office = 'R344';
```

The predicate in the `WHERE` clause constrains the relational column `office`, which the view maps to the path `/dept/employee/office` in the XML column. New optimization technology in DB2 9.7 enables DB2 to automatically use the XML index `officeIdx` defined on the underlying XML column to process this query. Using available XML indexes as appropriate improves runtime performance.

Applications that do not understand XML data directly, such as many business reporting tools, can benefit from such views and the DB2 9.7 technology, which is sometimes referred to as *predicate pushdown*. This optimization feature enables applications to efficiently query XML data through a traditional relational interface with SQL queries.

## Conclusion

With DB2 9.7, IBM continues to extend DB2 pureXML technology in important ways. Capabilities include new database design options that offer scalability and performance improvements for both analytical and transaction-oriented applications. New compression support for XML data and indexes reduces disk space requirements and can improve runtime performance of I/O-bound operations. Support for XML in user-defined functions and global declared temporary tables offer new application development advantages. Finally, new system-supplied functions and stored procedures simplify database design and maintenance efforts.

## Acknowledgments

Thanks to Henrik Loeser and Susan Malaika for their contributions to this article.

# Resources

## Learn

- Refer to the IBM Redbook® [DB2 9 pureXML: Overview and Fast Start](#) by Cynthia M. Saracco, Don Chamberlin, and Rav Ahuja for an introduction to the fundamentals of DB2 pureXML.
- Read IBM white paper [Best Practices – Managing XML Data](#) by Matthias Nicola and Susanne Englert for help implementing efficient pureXML databases and applications.
- Find the [DB2 pureXML Cookbook](#) (IBM Press) by Matthias Nicola and Pav Kumar Chatterjee for a comprehensive guide to DB2 pureXML technology for all supported platforms.
- See [DB2 partitioning features: An overview for data warehouses](#) from Paul McInerney for a strong introduction to DB2 hash partitioning, range partitioning, and multi-dimensional clustering.
- Check out the [DB2 pureXML wiki](#) for a comprehensive set of links to demos, free downloads, technical papers, and more.
- Learn more about Information Management at the [developerWorks Information Management zone](#). Find technical documentation, how-to articles, education, downloads, product information, and more.
- Stay current with [developerWorks technical events and webcasts](#).

## Get products and technologies

- Listen to the 11-minute podcast [Cindy Saracco on pureXML and next-gen DB2](#) for more information about DB2 9.7.
- Download free [DB2 pureXML quick start software bundles](#) for industry-specific XML formats, such as FIXML, FpML, ACORD, CDISC, UNIFI, and others.
- Get your hands on upcoming DB2 technology through the [DB2 early access program](#).
- Build your next development project with [IBM trial software](#), available for download directly from developerWorks.

## Discuss

- [Participate in the discussion forum for this content](#).
- Visit the [native XML database blog](#).
- Check out the [developerWorks blogs](#) and get involved in the [developerWorks community](#).

## About the authors

Cynthia M. Saracco

Cynthia M. Saracco is a senior solutions architect at IBM's Silicon Valley Laboratory who specializes in emerging technologies and database management topics. She has 23 years of software industry experience, has written 3 books, has written more than 60 technical papers, and holds 7 patents.

---

Matthias Nicola

Matthias Nicola is a Senior Software Engineer for DB2 pureXML at the IBM Silicon Valley Lab. His work focuses on all aspects of XML in DB2, including XQuery, SQL/XML, storage, indexing and performance. Matthias also works closely with customers and business partners, assisting them in the design, implementation, and optimization of XML solutions. Before joining IBM, Matthias worked on data warehousing performance for Informix Software. He received his doctorate in computer science from the Technical University of Aachen in Germany.

## Trademarks

IBM, DB2, developerWorks, and Redbook are trademarks of IBM Corporation in the United States, other countries, or both.

Microsoft, Windows, Windows NT, 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.

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

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