

pureXML in DB2 9: Which way to query your XML data?

Skill Level: Intermediate

[Matthias Nicola \(mnicola@us.ibm.com\)](mailto:mnicola@us.ibm.com)
DB2/XML Performance
IBM Silicon Valley Laboratory

[Fatma Ozcan \(fozcan@almaden.ibm.com\)](mailto:fozcan@almaden.ibm.com)
Research Staff Member
IBM Almaden Research Center

08 Jun 2006

Updated 28 Aug 2007

DB2 9 introduces pureXML® support, which means that XML data is stored and queried in its inherent hierarchical format. To query XML data, DB2 offers two languages, SQL/XML and XQuery. You can use XQuery and SQL separately, but you can also use XQuery embedded in SQL and vice versa. This gives you a lot of flexibility and options for querying your XML data. Each of these options is useful under certain circumstances. In this article we describe these options, their respective advantages and disadvantages, and guidelines for choosing the right one for your needs. This article has been updated for DB2 9.5.

Introduction

The pure XML support in DB2 offers efficient and versatile capabilities for managing your XML data. DB2 stores and processes XML in its inherent hierarchical format, avoiding the performance and flexibility limitations that occur when XML is stored as text in CLOBs or mapped to relational tables. Unlike XML-only databases, DB2 V9 also provides seamless integration of relational data and XML data within a single database, even within a single row of a table. This flexibility is reflected in the language support, which allows you to access relational data, XML data, or both at

the same time. You can query your XML in any of the following four ways:

- Plain SQL (no XQuery involved)
- SQL/XML, that is, XQuery embedded in SQL
- XQuery as a stand-alone language (no SQL involved)
- XQuery with embedded SQL

For an introduction to querying XML data with XQuery and SQL/XML please see [earlier articles](#) published on developerWorks, "Query DB2 XML Data with SQL" and "Query DB2 XML data with XQuery." We assume that you are familiar with the concepts introduced in these two articles. Note that XPath is a sublanguage of XQuery, so it is implicitly included wherever we mention XQuery. You may already know XPath if you have used XSLT style sheets or location paths in the DB2 XML Extender. In many situations XPath is sufficient to extract XML values or express XML predicates, so you can get started with XPath even if you are not yet familiar with all the additional features of XQuery.

DB2 enables you to leverage all these options to maximize your productivity and adapt your queries to the needs of your applications. The questions that we address in this article are:

- What are the key characteristics and the advantages and disadvantages of these four options?
- Which one should you use in which case?

Let's start with a high level summary and then dive into each option for more details and specific examples.

Summary and guidelines

You can express many queries in plain XQuery, in SQL/XML, or XQuery with embedded SQL. In certain cases you may find one of the options more intuitive to express your query logic than the other. In general, the "right" approach for querying XML data needs to be chosen on a case-by-case basis, taking the application's requirements and characteristics into account. However, we can summarize the following guidelines.

- **Plain SQL without any XQuery or XPath** is really only useful for full-document retrieval and operations such as insert, delete, and update of whole documents. Selection of documents must be based on non-XML columns in the same table.

- **SQL/XML with XQuery or XPath embedded in SQL statements** provides the broadest functionality and the least restrictions. You can express predicates on XML columns, extract document fragments, pass parameter markers to XQuery expressions, use full-text search, aggregation and grouping at the SQL level, and you can combine and join XML with relational data in a flexible manner. Most applications are well served by this approach. Even if you don't need all of these advantages right away, you may still want to choose this approach to keep your options open for future extensions.
- **XQuery** is a powerful query language, specifically designed for querying XML data. As such, it is a good option if your applications require querying and manipulating XML data only, and do not involve any relational data. This may sometimes be simpler and more intuitive. Also, if you are migrating from an XML-only database to DB2 9 and already have existing XQueries, you may prefer to stick with plain XQuery.
- **XQuery with embedded SQL** can be a good choice if you want to leverage relational predicates and indexes as well as full-text search to pre-filter the documents from an XML column which are then input to an XQuery. SQL embedded in XQuery also allows you to run external functions on the XML columns. But, if you need to perform data analysis queries with grouping and aggregations, you may prefer SQL/XML.

No matter what combination of SQL and XQuery you choose in one statement, DB2 uses a single hybrid compiler to produce and optimize a single execution plan for the entire query -- without incurring a performance penalty for query execution.

The following table summarizes the respective advantages of the four different options for querying XML data.

Table 1. Summary

	Plain SQL	SQL/XML	Plain XQuery	XQuery w/ embedded SQL/XML
XML Predicates	-	++	++	++
Relational Predicates	++	++	-	+
XML and Relational Predicates	-	++	-	++
Joining XML with Relational	-	++	-	++
Joining XML with XML	-	+	++	++

Transforming XML Data	-	o	++	++
Insert, Update, Delete	++	++	-	-
Parameter Markers	+	++	-	-
Full Text Search	+	++	-	++
XML Aggregation and Grouping	-	++	o	o
Function Calls	++	++	-	++

In the table above, "-" indicates that the given language does not support a feature, "+" means that the feature is supported but that a more efficient or convenient way may exist, "++" signifies the given language is very well suited to express a feature, and finally "o" means that although the feature can be expressed it is somewhat awkward or inefficient.

Now, let's define some sample data and tables so that we can look at concrete query examples.

Sample tables and data

For the discussion of the XML query options we use three tables which are shown below. The dept table has two columns named unitID and deptdoc. Each row of the dept table describes one department of a fictitious company. The unitID column identifies the unit containing the department (a unit may contain multiple departments), and the deptdoc column contains an XML document listing the employees in the department. The project table has a single column named projectDoc of type XML. Each row of the projectDoc table contains an XML document describing a particular project. A project may involve more than one department. To illustrate hybrid relational and XML queries as well as joins, we also have a pure relational table unit which lists the name, manager, and so on, for each unit. A unit can have multiple departments.

```
create table dept( unitID char(8), deptdoc xml)
```

unitID	deptdoc
WWPR	<pre><dept deptID="PR27"> <employee id="901"> <name>Jim Qu</name> <phone>408 555 1212</phone> </employee></pre>

	<pre><employee id="902"> <name>Peter Pan</name> <office>216</office> </employee> </dept></pre>
WWPR	<pre><dept deptID="V15"> <employee id="673"> <name>Matt Foreman</name> <phone>416 891 7301</phone> <poffice>216</office> </employee> <description>This dept supports sales world wide</description> </dept></pre>
S-USE	...
...	...

```
create table project(projectDoc xml)
```

projectDOC

```
<project ID="P0001">
  <name>Hello World</name>
  <deptID>PR27</deptID>
  <manager>Peter Pan</manager>
</project>
```

```
<project ID="P0009">
  <name>New Horizon</name>
  <deptID>PR27</deptID>
  <deptID>V15</deptID>
  <manager>Matt Foreman</manager>
  <description>This project is brand new</description>
</project>
```

```
create table unit( unitID char(8) primary key not null, name
char(20), manager varchar(20),...)
```

unitID	name	manager	...
WWPR	Worldwide Marketing	Jim Qu	...
S-USE	Sales US East Coast	Tom Jones	...

... ..

Plain SQL

You can use plain SQL without any XPath or XQuery to read full documents without XML predicates. This is easy and simple whenever the application can identify XML documents for full document retrieval based on relational predicates on the same table. For example, Query 1 retrieves all the department documents for unit "WWPR":

Query 1

```
select deptdoc
from dept
where unitID = 'WWPR';
```

Similarly, Query 2 returns the XML data for all departments which are managed by Jim Qu.

Query 2

```
select deptdoc
from dept d, unit u
where d.unitID = u.unitID and u.manager= 'Jim Qu';
```

The obvious drawback is that you can't express predicates on the XML data itself or retrieve just fragments of an XML document. In our example, plain SQL is not sufficient to select only department PR27 and only return the employee names.

If your queries do not require predicates on the XML column and always return full XML documents, then plain SQL may be sufficient for you. In this case you could store XML just as well in VARCHAR or CLOB columns, which may give you performance advantages for full-document insert and retrieval operations.

Without using any of the SQL/XML or XQuery support in DB2 V9, plain SQL still allows full-text search conditions in your queries. With the [DB2 Net Search Extender](#) you can create full-text indexes over XML columns to support text search ranging from basic keyword search to advanced search with stemming, thesaurus and fuzzy search in 37 languages. You can also restrict your text search to certain document sections identified by path expressions. Based on text-index lookup, Query 3 returns all department documents which contain the string "sales" anywhere in the document under /dept/description:

Query 3

```
select deptdoc
from dept
where CONTAINS(deptdoc, 'SECTION("/dept/description") "sales" ')=1;
```

SQL/XML (XQuery/XPath embedded in SQL)

SQL/XML is part of the SQL language standard and defines a new XML data type together with functions for querying, constructing, validating, and converting XML data. DB2 V8 already featured numerous SQL/XML publishing functions so users can construct XML from relational data, with XMLELEMENT, XMLATTRIBUTE, XMLFOREST, XMLAGG and other functions.

DB2 9 adds new SQL/XML query functions, including XMLQUERY, XMLTABLE and the XMLEXISTS predicate. These functions allow users to embed XQuery or simple XPath expressions in SQL statements.

As illustrated in Query 4, the XMLQUERY function is typically used in the select clause to extract XML fragments from an XML column, while XMLEXISTS is commonly used in the where clause to express predicates over XML data.

Query 4

```
select unitID, XMLQUERY('for $e in $d/dept/employee return
  $e/name/text()')
                        passing d.deptdoc as "d")
from dept d
where unitID LIKE 'WW%' and
      XMLEXISTS('$d/dept[@deptID = "V15"]' passing d.deptdoc as
  "d");
```

This sample query uses XMLEXISTS to select the "WW" department V15 and applies XMLQUERY to return all employee names from that department document. The result is:

WWPR	Matt Foreman
------	-----------------

This query also highlights how you can use SQL/XML to query XML and relational data in an integrated manner. The select clause retrieves data from both relational and XML columns, and the where clause contains both relational and XML predicates. DB2 9.5 can use XML indexes and relational indexes at the same time to evaluate these predicates and maximize query performance.

In DB2 9.5, you can write this query even simpler by omitting the "passing" clause in the XMLEXISTS and XMLQUERY functions. If you only pass the XML column

"deptdoc" into the XQuery then you can simply reference the column as \$DEPTDOC in the XQuery expression, without the "passing" clause. You see this in Query 5.

Query 5

```
select unitID, XMLQUERY('for $e in $DEPTDOC/dept/employee return
    $e/name/text()')
from dept d
where unitID LIKE 'WW%' and
    XMLEXISTS('$DEPTDOC/dept[@deptID = "V15"]');
```

We can express the same query using the XMLTABLE function, shown in Query 6. In this format, we specify conditions to restrict the input data and to extract the output values that we are interested in. In Query 6 the XQuery expression in the XMLTABLE function identifies the employees working in department V15 and the path expression in the COLUMNS clause ("name/text()") return their names. Its output is the same as Query 4 and 5.

Query 6

```
select d.unitID, T.name
from dept d, XMLTABLE('$d/dept[@deptID="V15"]/employee' passing
    d.deptdoc as "d"
                                COLUMNS
                                name varchar(50) path 'name/text()' ) as
T
where unitID LIKE 'WW%';
```

SQL/XML advantages

The SQL/XML approach has the following advantages:

- SQL/XML is good if you have an existing SQL application and you need to add a little XML functionality here and there, step by step.
- SQL/XML is good if you are an SQL fan and if you want to keep SQL as the primary language, because this is what you and your team are most familiar with.
- SQL/XML is good if your queries need to return data from relational columns and from XML columns at the same time.
- SQL/XML is good if your queries require full-text search conditions, as shown in Query 3 above.
- SQL/XML is good if you want results returned as sets and missing XML elements represented with nulls.
- SQL/XML is good if you want to use parameter markers, because DB2 V9

XQuery does not support external parameters. The passing mechanisms in XMLQUERY, XMLTABLE and XMLEXISTS allow you to pass an SQL parameter marker as a variable (\$x) into the embedded XQuery expression:

Query 7

```
select deptdoc
from dept
where XMLEXISTS('$d/dept[@deptID = $x]'
                passing deptdoc as "d", cast(? as varchar(8))
                as "x");
```

- SQL/XML is good for applications that need to integrate relational and XML data. It provides the easiest means to join XML data and relational data. The following example selects the unit IDs of all departments that have an employee who is listed as a manager in the unit table. This query performs a join between a relational value (unit.manager) and an XML value (//employee/name):

Query 8

```
select u.unitID
from dept d, unit u
where XMLEXISTS('$d//employee[name = $m]'
                passing d.deptdoc as "d", u.manager as "m");
```

To accomplish this join, we pass the unit manager into the XMLEXISTS predicate such that the actual join condition is an XQuery predicate. Conversely, we can extract the employee name from the XML documents into the SQL context such that the join condition is an SQL predicate:

Query 9

```
select u.unitID
from dept d, unit u
where u.manager = XMLCAST(XMLQUERY('$d//employee/name '
                                   passing d.deptdoc as "d") as
                           char(20));
```

Typically, Query 8 is preferable over Query 9 because the XMLCAST function expects a single input value. It would fail in our example for departments with more than one employee. The latter option with XMLCAST can be useful for joins on XML values which occur only once per document, because it allows the usage of a relational index on unit.manager. That index cannot be used in Query 8 because the join condition is not a relational predicate but an XQuery predicate.

- SQL/XML is good for grouping and aggregation of XML. The XQuery language does not provide an explicit group-by construct. Although grouping and aggregation can be expressed in XQuery using self-joins, it is quite awkward. As an example, let's count employees grouped by office, in other words, the number of employees in each office. Query 10 shows how this can be done in plain XQuery. The db2-fn:xmlcolumn function in Query 10 provides access to XML data in DB2. It takes the name of an XML column in a as an argument and returns the sequence of XML values stored in that column. It is easier to use SQL/XML functions, such as XMLTABLE or XMLQUERY to extract the data items from XML columns and then use familiar SQL concepts to express grouping and aggregation on top of that. Query 11 returns the same logical result as Query 10 but uses XMLTABLE and the SQL group by clause.

Query 10

```
XQUERY
for $o in
distinct-values(db2-fn:xmlcolumn("DEPT.DEPTDOC")/dept/employee/office)
let $emps :=
db2-fn:xmlcolumn("DEPT.DEPTDOC")/dept/employee[office/text()=$o]
return
<result><office>{$o}</office><cnt>{count($emps)}</cnt></result>;

Result:
<result><office>216</office><cnt>2</cnt></result>
```

Query 11

```
select X.office, count(X.emp)
from dept, XMLTABLE ('$d/dept/employee' passing deptdoc as "d"
  COLUMNS
    emp          VARCHAR(30)      PATH 'name',
    office       INTEGER          PATH 'office ') as X
GROUP BY X.office;

Result:
216    2
-      1
```

In Query 11 the XMLTABLE function extracts /dept/employee/name and /dept/employee/office from every document in form of a table with two columns named "emp" and "office". Using SQL group by and aggregation functions on that table is usually more efficient than producing the same result in plain XQuery.

Note that we get an extra row from Query 10, because SQL's group by also produces a group for NULL values, and there is one employee in our sample table with no office information. Query 9 doesn't produce a row for that employee because the for loop iterates over the distinct office values which do not include missing office information.

SQL/XML disadvantages

- SQL/XML is not always the best choice to transform an XML document into another XML document. Using standalone XQuery can be more appropriate for that, and often more intuitive when dealing with XML data only. But, DB2 9.5 also offers an XSLTRANSFORM function which you can use in SQL statements to transform XML documents with XSLT stylesheets. In particular, the XMLQUERY function can be an argument to the XSLTRANSFORM function.
- You may find that expressing joins between two XML columns, or more generally between two XML values, can be more intuitive in plain XQuery than SQL/XML. For example, Query 12 joins the XML columns of the dept and project tables to return the employees who work on any of the projects.

Query 12

```
select XMLQUERY('$d/dept/employee' passing d.deptdoc as "d")
from dept d, project p
where XMLEXISTS('$d/dept[@deptID=$p/project/deptID]'
                passing d.deptdoc as "d", p.projectDoc as "p");
```

In Query 12, we pass every dept and project document into the XQuery inside XMLEXISTS and express the join condition there. You may find it easier to write such a join in plain XQuery (Query 14).

XQuery as a stand-alone language

Let's take step back for a moment and ask: What is XQuery, and why do we need it? Just like SQL is a query language designed for the relational data model, XQuery is a language designed specifically to query XML data. Since XML data can be very different from relational data, we need a dedicated language to handle XML data efficiently. Relations are flat, highly structured, strongly typed, and unordered while XML data is ordered, nested, hierarchical, optionally typed and often irregular and semi-structured. SQL cannot handle that, but XQuery is designed for it. Specifically, XQuery is designed to navigate through XML document trees and extract XML fragments, but also includes expressions to create, manipulate and iterate over sequences of XML items and construct new XML data.

IBM has extended all of DB2's major application programming interfaces (APIs) to support XQuery as a first-class language, just like SQL. This includes CLI/ODBC, embedded SQL, JDBC, and .NET. Consequently, the DB2 command line processor also supports XQuery. You can submit XQueries as-is but need to start them with the XQUERY keyword to inform DB2 to use the XQuery parser, as in the following

example:

Query 13

```
XQUERY
for $dept in db2-fn:xmlcolumn("DEPT.DEPTDOC")/dept
where $dept/@deptID="PR27"
return $dept/employee/name;
```

Query 13 iterates over each "dept" element of each department document and returns the names of the employees who work in department PR27.

XQuery advantages

- XQuery is good for XML-only applications that do not need (or do not wish) to use SQL or relational structures.
- XQuery is good for migration from an XML-only database to DB2 V9. Existing XQueries can often run with only minor changes in DB2. For example, the input data for an XQuery comes from the db2-fn:xmlcolumn() function in DB2, while other databases may call it collection(). In this case only a simple rename is needed.
- XQuery is good if the query results need to be embedded in (and returned in) newly constructed XML documents that differ from those in the database.
- XQuery is very good to express joins between two XML documents, as well as to union XML values. For example, you can express the join in Query 12 in a more straight-forward and efficient manner using XQuery as shown in Query 14.

Query 14

```
XQUERY
for $dept in db2-fn:xmlcolumn("DEPT.DEPTDOC")/dept
  for $proj in db2-fn:xmlcolumn("PROJECT.PROJECTDOC")/project
where $dept/@deptID = $proj/deptID
return $dept/employee;
```

XQuery disadvantages

- With plain XQuery you can not exploit full-text search capabilities provided by the DB2 Net Search Extender (NSE). You need to involve SQL for full-text search.
- Plain XQuery does not allow you to call SQL user defined functions (UDFs) or external UDFs written in C or Java.

- Currently, DB2 does not provide a way to invoke a stand-alone XQuery with parameter markers. In order to pass parameters to XQuery, you must use SQL/XML to convert a parameter-marker ("?") into a named variable. For example, in Query 13 you might be tempted use a question mark (?) as a SQL-style parameter marker instead of the literal value "PR27". But that would be an invalid query.
In Query 7 you saw that SQL/XML allows you to pass an SQL parameter marker as a variable into the embedded XQuery expression. SQL/XML queries often have an XMLQUERY function in the select clause to extract parts of the XML documents, and an XMLEXISTS predicate in the where clause to filter the qualifying documents. If you prefer to express the entire query logic in a single FLOWR expression, instead of having two separate XQuery calls (one in XMLQUERY and one in XMLEXISTS) and still use parameter markers, consider rewriting Query 13 into Query 15:

Query 15

```
values( XMLQUERY(  
    ' for $dept in db2-fn:xmlcolumn("DEPT.DEPTDOC")/dept  
      where $dept/@deptID = $z  
      return $dept/employee/name'  
    passing cast(? as varchar(8)) as "z" ) );
```

Query 15 is an SQL/XML statement because it is simply the values clause of an SQL fullselect. A values clause returns a table of values by specifying an expression for each column in the result table. In Query 15 the result table has a single row and a single column of type XML, and the XMLQUERY function produces the value for the output column. All employee names will be returned in a single XML value to the client. The FLOWR expression in Query 15 is almost the same as the one in Query 13, except Query 15 contains an external variable (\$z), which is passed into the XMLQUERY function as a parameter.

XQuery with embedded SQL

XQuery alone allows you to access XML data, and only XML data. This is perfectly fine if you are dealing with XML data only, but insufficient if your applications require combined access to XML and relational data, leveraging the full power of both languages and data models. This is possible with SQL/XML discussed earlier in this paper, embedding XQuery in SQL. Conversely, embedding SQL in XQuery opens up additional possibilities. In addition to the list of advantages and disadvantages we discussed in the previous section, the following aspects are important:

Advantages of XQuery with embedded SQL

- XQuery with embedded SQL is good if you would like to process only a subset of the XML documents based on conditions on relational columns. It can be desirable to apply an XQuery only to a subset of the XML documents in an XML column. In particular, you can use relational predicates to restrict the input to a particular XQuery. For this purpose, DB2 provides another input function, `db2-fn:sqlquery`, to invoke an SQL query from within XQuery. This function takes in an SQL SELECT statement and returns an XML column as output. For example, Query 16 does not consider all documents in the XML column `deptdoc`, but has an embedded SQL statement which pre-filters the XML documents by joining to the `unit` table where a predicate is applied:

Query 16

```
XQUERY
for $emp in db2-fn:sqlquery("select deptdoc
                           from dept d, unit u
                           where d.unitID=u.unitID and
                           u.manager = 'Jim Qu'")/dept/employee
where $emp/office = 216
return $emp/name;
```

Regular relational indexes on the `unitID` columns of both tables as well as on the `manager` column in the `unit` table can help speed up the embedded SQL query. DB2 9.5 can even use XML and relational indexes at the same time, such as relational indexes for the embedded SQL statement plus an XML index for the XML predicate `$emp/office = 216`.

- XQuery with embedded SQL allows you to exploit full text search, because you can use the text search function "contains" in the where clause of the embedded SQL statement. In Query 17, the SQL statement inside the XQuery selects documents from the `dept` table where the word "sales" occurs in anywhere in `/dept/description`. The full-text index finds these XML documents quickly which are then input to the `FLWOR` expression that extracts all employee names from these documents. For our sample tables, Query 18 returns the same result but is expressed in SQL/XML notation.

Query 17

```
XQUERY
for $emp in db2-fn:sqlquery("
select deptdoc from dept
where CONTAINS(deptdoc,
'SECTION("/dept/description") "sales" ')=1
")/employee
return $emp/name;
```

Query 18

```
select XMLQUERY('$d//employee/name' passing deptdoc as "d")
from dept
where CONTAINS(deptdoc, 'SECTION("/dept/description") "sales" ')=1;
```

- XQuery with embedded SQL can be useful for applications that need to integrate relational and XML data. You can query XML data and relational data in a combined manner. This applies to SQL/XML as well. But, you may find that joins between XML and relational values are easier in SQL/XML with the XMLEXISTS function. Compare Query 19 and Query 20. Query 19 returns the deptID of those departments which have a unit manager among their employees. The embedded SQL statement casts the manager names in the unit table into an XML type (in this case into an XML string), and feeds them into the XQuery. The XQuery where clause contains the join condition, comparing the manager names with the employee name elements. Query 20 expresses the same join in SQL/XML notation.

Query 19

```
XQUERY
for $m in db2-fn:sqlquery('select XMLCAST(u.manager as XML) from
unit u')
for $d in db2-fn:xmlcolumn("DEPT.DEPTDOC")/dept
where $d/employee/name = $m
return $d/data(@deptID);
```

Query 20

```
select XMLQUERY('$d/dept/data(@deptID)' passing d.deptdoc as "d")
from dept d, unit u
where XMLEXISTS('$d/dept/employee[name = $m]'
                passing d.deptdoc as "d", u.manager as "m");
```

As an aside, think about how you would modify Query 19 and 20 if you wanted to return the matching employee name instead of the deptID. This is very easy in Query 19. You would simply change the return clause to: `return $m`. In Query 20, the intuitive idea is to change the path in the XMLQUERY function to: `XMLQUERY('$d/dept/employee/name' passing d.deptdoc as "d")`. But, this would return the names of *all* the employees in the given department and not just the manager's name. To extract only the manager's name from a matching department document, the XMLQUERY function needs to include a predicate on name similar to the XMLEXISTS in Query 20, i.e.

`XMLQUERY('$d/dept/employee/name[. = $m]' passing d.deptdoc as "d", u.manager as "m")`. The difference is that XMLEXISTS applies the predicate to qualify entire documents, but the

XMLQUERY function applies the predicate to find a specific name *within* a given document that XMLEXISTS has qualified.

- DB2 9.5 allows you to pass parameters from XQuery to the SQL statement that is embedded in the db2-fn:sqlquery function. This can be useful in a variety of situations. One Example is Query 21 which expresses the same join as Queries 19 and 20. The "for" clause iterates over the employee names, and the "where" clause checks whether a manager by the same name exists in the unit table. The db2-fn:sqlquery function takes \$n as an additional argument. When evaluating the select statement the function "parameter(1)" assumes the value of \$n. Multiple such parameters can be used. The XQuery "where" clause evaluates to false if the db2-fn:sqlquery function returns an empty sequence, and true if it returns a non-empty sequence. This is due to the existential semantics of XQuery. Hence, the select clause could just as well be `select XMLCAST('yes' as XML)` because only the existence of a value matters here, not the value itself.

Query 21

```
XQUERY
for $n in db2-fn:xmlcolumn("DEPT.DEPTDOC")/dept/employee/name
where db2-fn:sqlquery('select XMLCAST(manager as XML) from unit
                    where manager = parameter(1)', $n)
return $n/../../../../data(@deptID);
```

Why would you care to express the join this way when you already have the options in Query 19 and 20? In Query 21 the join predicate is at the SQL level which allows DB2 to use a relational index on unit.manager. Query 19 and 20 express the join condition at the XML level so that an XML index on /dept/employee/name can be used. This is similar as for Query 8 vs. 9 above.

- As indicated in Query 19, XQuery with embedded SQL can be useful to feed relational data into an XQuery. This allows you to combine and merge XML and relational data. Query 22 constructs a result document that contains unit and department information. The department information is an XML document retrieved from the XML column deptdoc. The unit information comes from the relational table unit. The embedded SQL statement uses SQL/XML publishing functions to construct an XML element "Unit" with three child elements whose values are taken from the relational columns of the unit table, i.e. the columns unitID, name, and manager.

Query 22

```
XQUERY
```

```

let $d := db2-fn:sqlquery("select deptdoc from dept where unitID
= 'WWPR' ")
let $u := db2-fn:sqlquery("select XMLELEMENT(NAME "Unit",
XMLFOREST(unitID, name, manager))
from unit where unitID = 'WWPR' ")
return <result>
    <units>{$u}</units>
    <department>{$d}</department>
</result>;

```

The output of Query 22 will look like this:

```

<result>
  <units><unit>
    <UNITID>WWPR</UNITID>
    <NAME>World Wide Markeing</NAME>
    <MANAGER>Jim Qu</MANAGER>
  </unit>
  <unit>
    <UNITID>WWPR</UNITID>
    <NAME> ... </NAME>
    <MANAGER> ... </MANAGER>
  </unit>
  ....
</units>
<department>
  <dept deptID="PR27">
    <employee id="901">
      <name>Jim Qu</name>
      <phone>408 555 1212</phone>
    </employee>
    <employee id="902">
      <name>Peter Pan</name>
      <office>216</office>
    </employee>
  </dept>
</department>
</result>

```

- XQuery with embedded SQL is good because the embedded SQL statement may contain calls to user-defined functions (UDFs). UDFs are widely used in many IT organizations to encapsulate critical business logic and simplify application development requirements. This is significant, because a plain XQuery cannot call UDFs.

Disadvantages of XQuery with embedded SQL

- Although db2-fn:sqlquery allows embedding an SQL statement inside XQuery, currently it is not possible to have regular SQL-style parameter markers in the embedded SQL statement.
- In DB2 9, db2-fn:sqlquery does not allow passing values from XQuery into SQL. But, this limitation has been lifted in DB2 9.5, as shown in Query 21.

XML query results

One thing you need to be aware of is that depending on how you write a specific query, DB2 may deliver the query results in different formats. For example, plain XQuery returns items (such as elements or document fragments) in the result set as one item per row, even if multiple items come from the same document (row) in the database. On the other hand, SQL/XML may return multiple items in a single row, as opposed to separate rows, when the XMLQUERY function is used. You may find this desirable in some cases, but not in others, depending on your particular application. Let's look at examples.

Query 23 and Query 24 are both asking for the employee names in the dept documents. Query 23 is expressed in SQL/XML whereas Query 24 is written in XQuery.

Query 23

```
select XMLQUERY('$d/dept/employee/name' passing deptdoc as "d")
from dept;
```

Query 24

```
XQUERY db2-fn:xmlcolumn("DEPT.DEPTDOC")/dept/employee/name;
```

Query 23 returns one row for each dept document, and each row contains the names of employees working in that department:

```
<name>Jim Qu</name> <name>Peter Pan </name>
<name>Matt Foreman</name>
```

Query 24, on the other hand, returns each employee name as a separate row:

```
<name>Jim Qu</name>
<name>Peter Pan</name>
<name>Matt Foreman</name>
```

The result of Query 24 is usually easier to consume by an application, i.e. one XML value at a time. However, in this case you don't know which name elements come from the same department document. The output of Query 23 preserves this information, but the result may be harder to consume by the application because it may require splitting up these results rows to access the names individually. If the application uses an XML parser to ingest each XML result row from DB2, the first

result row from Query 21 would be rejected by the parser because it is not a well-formed document (it lacks a single root element). To solve this, you can add a single root element by adding an XMLELEMENT constructor to Query 23, as shown in Query 25:

Query 25

```
Select XMLELEMENT(name "employees",
      XMLQUERY('$d/dept/employee/name' passing d.deptdoc as "d"))
from dept d;
```

This changes the query result such that each result row is a well-formed XML document:

```
<employees><name>Jim Qu </name><name>Peter Pan
</name></employees>
<employees><name>Matt Foreman</name></employees>
....
```

Recall that Query 15 uses an SQL values clause and the XMLQUERY function to allow parameter passing into XQuery. But, the output of Query 15 is a single row, which contains all the employee names. If you prefer to get each employee name in a separate row, and also need to use parameter markers, this can be achieved with the XMLTABLE function shown in Query 26:

Query 26

```
select X.*
from dept d, XMLTABLE('for $dept in $d/dept
                      where $dept/@deptID = $z
                      return $dept/employee/name'
                      passing d.deptdoc as "d", cast(? as
varchar(10)) as "z"
                      COLUMNS
                      "name" XML PATH '.' ) as X ;
```

Summary

DB2 pureXML offers a rich set of options for querying XML data. You will choose the option that is right for you, based on your application's requirements and characteristics. If you need to combine relational and XML data, then SQL/XML is the best choice in most cases. In particular, SQL/XML is the option which allows parameter markers against XML data. If you have XML-only applications, stand-alone XQuery is a powerful choice, and can be augmented with embedded SQL to allow full-text search and invocation of UDFs. The examples discussed in

this article help you make an informed decision. You can use our query patterns in this paper as a starting point for writing your own queries against your XML data.

A general guideline is to introduce only as much complexity in your queries as you really need. For example, it is certainly possible to have an XQuery with an embedded SQL statement which has an embedded XQuery, and so on. But, our experience shows that nesting the two languages more than one level deep is usually not needed to express the desired query logic. Therefore, we recommend using only one level of embedding XQuery into SQL or vice versa.

Acknowledgements

Thanks to Don Chamberlin, Bert van der Linden, Cindy Saracco, Jan-Eike Michels, and Hardeep Singh for their reviews of this article.

Downloads

Description	Name	Size	Download method
Script for queries in this article	DDLandQueries.txt	10KB	HTTP

[Information about download methods](#)

Resources

Learn

- The article "[What's new in DB2 Viper: XML to the core](#)" (Cynthia M. Saracco, developerWorks, February 2006) describes the support for XML in DB2 9.
- The article "[Query DB2 XML Data with SQL](#)" (Cynthia M. Saracco, developerWorks, March 2006) shows you how to query data stored in XML columns using SQL and SQL/XML.
- The article "[Query DB2 XML data with XQuery](#)" (Don Chamberlin and Cynthia M. Saracco, developerWorks, April 2006) shows you how to query data stored in XML columns using XQuery.
- The article "[XMLTABLE by example, Part 1: Retrieving XML data in relational format](#)" (Vitor Rodrigues and Matthias Nicola, developerWorks, August 2007) describes various ways to use the XMLTABLE function.
- The [Net Search Extender](#) gives you full-text search capabilities for DB2 V8.
- Read a paper entitled "[Native XML Support in DB2 Universal Database](#)" by Matthias Nicola and Bert van der Linden.
- Learn more on this topic in the article "[Integration of SQL and XQuery in IBM DB2](#)" (F. Ozcan et al., IBM Systems Journal, April, 2006).
- Stay current with [developerWorks technical events and Webcasts](#).
- Learn about [DB2 Express-C](#), the no-charge version of DB2 Express Edition for the community.

Get products and technologies

- Download the DB2 9 [test drive](#) to try out the query techniques described in this article.
- Now you can use DB2 for free. Download [DB2 Express-C](#), a no-charge version of DB2 Express Edition for the community that offers the same core data features as DB2 Express Edition and provides a solid base to build and deploy applications.

Discuss

- [Participate in the discussion forum for this content](#).
- Participate in [developerWorks blogs](#) and get involved in the developerWorks community.

About the authors

Matthias Nicola

Dr. Nicola is the technical lead for XML database performance at IBM's Silicon Valley Lab. His work focuses on all aspects of XML performance in DB2, including XQuery, SQL/XML, and all native XML features in DB2. Dr. Nicola works closely with the DB2 XML development teams as well as with customers and business partners who are using XML, assisting them in the design, implementation, and optimization of XML solutions. Prior to joining IBM, Dr. Nicola worked on data warehousing performance for Informix Software. He also worked for four years in research and industry projects on distributed and replicated databases. He received his doctorate in computer science in 1999 from the Technical University of Aachen, Germany.

Fatma Ozcan

Dr. Ozcan has been a research staff member at IBM's Almaden Research Center since 2001. She is a member of the DB2 XML compiler team and a technical leader in the area of XML query languages, XQuery semantics, and rewrite optimization. She received a Ph.D. degree in computer science from the University of Maryland at College Park in 2001. Her research interests include XML query languages and query optimization, integration of heterogeneous information systems, and software agents. She is a member of ACM SIGMOD.