



Indexing XML documents with DB2 9 pureXML™

*By Vitor Rodrigues
Email: vrodrig@NOSPAMus.ibm.com
IBM Software Group*

1. Overview

Just like relational indexes, XML indexes on DB2 9 pureXML are used to improve the performance of queries. The user should always create indexes over frequently accessed data, resulting in significant performance gains for the select statements executed over the indexed data.

Although indexes usually present very positive performance gains for select statements, other statements like insert/update/delete may have their performance slightly decreased, since index keys may need to be created or deleted when these statements are executed.

The number of indexes to create over the user's XML data should be influenced not only by the XML data the user wants to query very frequently, but also by the amount of storage space that the user wants to designate for index keys storage. The more indexes created, the more space will be needed to store index keys.

While traditional relational indexes can contain index keys composed of one or more table columns, XML indexes can only index paths and values in XML documents stored in a single XML column.

This paper provides a short introduction to index creation over XML documents on DB2 9 pureXML. We will cover the basics of the CREATE INDEX statement and explain in detail some of the parameters for this statement. During the paper we will present:

- the basics about the CREATE INDEX statement and the new functionality provided to support XML indexes;
- how to use XPath expressions to select the document nodes you want to index;
- the SQL data types used to create the XML index keys.

2. Populating your database

The first step of our tutorial is the creation of a sample database to store our sample XML documents and indexes. Currently, in DB2 9 pureXML, only Unicode databases provide support for storage of XML documents, together with other traditional SQL data types. An Unicode database can be created running the create db statement, specifying the parameters *codeset* and *territory*. The following statements are used to create the sample database and connect to it.

```
create db sample using codeset utf-8 territory us
connect to sample
```

Next, we need to create the table that will store our XML data. As you can learn from "[Getting off to a fast start with DB2 Viper](#)", XML documents are stored as a new SQL data type, called *XML*.

To our sample, we will create the table "companies", with the columns ID and DOC. The ID column will be of type INTEGER and will store an id to identify the XML documents in our table. The column DOC will be of type XML and will store the XML documents themselves.

```
create table companies (
  id  int primary key not null,
  doc xml)
```

Indexing XML documents with DB2 9 pureXML

Page 3

Our XML indexes will be created over the XML documents stored in the database. To better demonstrate the relationship between the created indexes and the XML documents, we will use two sample documents. Consider the table companies contents as follow:

<i>ID</i>	<i>DOC</i>
1	<pre><company name="Alliance"> <emp id="31664" salary="60000" gender="male"> <name> <first>Luke</first> <last>Moonwalker</last> </name> <dept id="M55">Marketing</dept> <birthdate>1960-07-21</birthdate> </emp> <emp id="42366" salary="50000" gender="female"> <name> <first>Laura</first> <last>Organa</last> </name> <dept id="K55">Sales</dept> <birthdate>1960-07-21</birthdate> </emp> </company></pre>
2	<pre><company name="Empire"> <emp id="31201" salary="60000" gender="male"> <name> <first>Darth</first> <last>Vapor</last> </name> <dept id="M25">Security</dept> <birthdate>1974-08-27</birthdate> </emp> </company></pre>

3. Indexing your XML data

Comparing XML indexes with relational indexes

The main differences found in XML indexes when compared with relational indexes are:

- the index needs to be created on a single XML column (no composite keys are allowed).
- only documents that satisfy the XML pattern are indexed.
- XML index may create zero, one or multiple keys per row.

CREATE INDEX statement

XML indexes can be created using the CREATE INDEX statement from DB2. Indexes on XML columns can be dropped from the database using the DROP INDEX statement, just like we do with relational indexes.

The syntax for creating indexes over XML columns is:

```
CREATE INDEX indexname ON table(column) GENERATE KEY USING XMLPATTERN xmlpattern-clause AS xmltype-clause
```

The CREATE INDEX syntax has been modified to provide XML support. The new clauses supported by CREATE INDEX are listed below.

- **GENERATE KEY USING XMLPATTERN** is used to specify the creation of an XML index.
- ***xmlpattern-clause*** is a pattern expression that identifies the set of nodes in the XML document that will be indexed. This clause supports a subset of XPath expressions.
- ***xmltype-clause*** is the SQL type to which we will cast our XML value. If it's not possible to cast the XML value to the specified SQL type, an error will be thrown.

Creating indexes

Creating your first index

To start, let's create an index on the first names of all employees stored in the XML column `companies.doc`:

```
CREATE INDEX firstnameindex ON companies(doc) GENERATE KEY USING XMLPATTERN  
'/company/emp/name/first' AS SQL VARCHAR(10)
```

This statement will create an index over all XML documents currently stored in column `doc` of table `companies` and over documents inserted in the future. For each document node that matches the specified XPath expression, a new index key will be created.

The following set of nodes will qualify for the previous index:

Node path	Node value	Doc ID
/company/emp/name/first	Laura	1
/company/emp/name/first	Luke	1
/company/emp/name/first	Dart	2

Understanding of XPath expressions

Now let's see how to use XPath expressions to index your XML documents.

"/" Symbol

The "/" is one of the basic axis of XPath expressions, and is used to navigate through the children elements of nodes. Consider the following example:

```
CREATE INDEX firstnameindex ON companies(doc) GENERATE KEY USING XMLPATTERN  
'/company/emp/name/first' AS SQL VARCHAR(10)
```

Indexing XML documents with DB2 9 pureXML

Page 5

The index shown before uses the / axis to navigate through the XML elements, and create a path from the root element (*company*) to the element *first*, that represents the first name of an employee.

“//” Symbol

The other axis commonly used in XML indexes declarations is “//”. Used to navigate through the XML elements and their children, the // axis addresses the current node and all of its descendants.

For example, if you want to create an index over all the *id* attributes in our XML documents, you can use the following example:

```
CREATE INDEX allidindex ON companies(doc) GENERATE KEY USING XMLPATTERN '//@id' AS SQL VARCHAR HASHED
```

“@” Symbol

The symbol “@” provides a user friendly way of defining paths to element attributes. In your XPath expression, the @ symbol should be followed by the attribute name:

```
CREATE INDEX empsalaryindex ON companies(doc) GENERATE KEY USING XMLPATTERN '/company/emp/@salary' AS SQL DOUBLE
```

```
CREATE INDEX deptidindex ON companies(doc) GENERATE KEY USING XMLPATTERN '/company/emp/dept/@id' AS SQL VARCHAR(5)
```

The first of the two previous indexes will create an index key for every employee salary attribute node, while the second index will generate index keys for the attribute *id* of the employee's department.

“*” Symbol

The wild card “*” can be used to specify any document node. This wild card is very useful when we have different child nodes in the path that we want to index. Consider the following XML example:

```
<company name="Empire">
  <emp id="31201" salary="60000" gender="male">
    <name>
      <first>Darth</first>
      <last>Vapor</last>
    </name>
    <dept id="M25">Security</dept>
    <birthdate>1974-08-27</birthdate>
  </emp>
  <asset id="A33214" type="mobile">
    <dept id="K55">Sales</dept>
    <description>Service Laptop</description>
  </emp>
</company>
```

You want to index all dept nodes from all the company assets and employees. The most common way to do it is to create two different indexes using two XPath expressions:

```
CREATE INDEX empidindex ON companies(doc) GENERATE KEY USING XMLPATTERN '/company/emp/dept/@id' AS SQL VARCHAR (10)
```

```
CREATE INDEX assetidindex ON companies(doc) GENERATE KEY USING XMLPATTERN '/company/asset/dept/@id' AS SQL VARCHAR (10)
```

Other possible way of doing is to use the wildcard * to match any of the company's child elements:

```
CREATE INDEX companychildindex ON companies(doc) GENERATE KEY USING XMLPATTERN  
'/company/*/dept/@id' AS SQL VARCHAR (10)
```

The previous index will generate index keys for all dept nodes that are child of a company's child node, be it an *emp* node or an *asset* node.

Although you can save time and decrease the number of indexes by using the symbols “//” and “*”, it is a better practice to define multiple indexes using fully specified paths instead. Using the full path in the index definition allows the user to control exactly which nodes will be indexed. When the symbols “//” or “*” are used, the defined indexes may be creating index keys over unwanted nodes.

Besides the previous XPath axis, DB2 9 pureXML also provides support of the following XPath axis: *child::*, *descendant::*, *self::*, *descendant-or-self::* and *attribute::*.

Index data types

When we create indexes on XML documents, the XML value we want to index needs to be converted to a SQL data type. We can cast XML data types to five different SQL types: DOUBLE, VARCHAR (n), VARCHAR HASHED, DATE and TIMESTAMP.

DOUBLE

The data type DOUBLE should be used to index numeric XML node values. Note that loss of precision may occur if the indexed node value is greater than the precision supported by DB2 for the data type DOUBLE.

```
CREATE INDEX empsalary ON companies(doc) GENERATE KEY USING XMLPATTERN  
'/company/emp/@salary' AS SQL DOUBLE
```

Since we know that the employee's salary is a numeric value, the SQL data type DOUBLE is the most accurate type to use as the type for the index keys.

VARCHAR (n)

This index data type is to be used to create indexes over string node values that have a fixed maximum length size. The index data type VARCHAR (n) should be used every time you want to index nodes with fixed length string value, since it gives you all the index functionality regarding comparison and sorting.

The syntax for using this data type is:

```
CREATE INDEX firstnameindex ON companies(doc) GENERATE KEY USING XMLPATTERN  
'/company/emp/name/first' AS SQL VARCHAR(10)
```

This CREATE INDEX statement will create index keys for all the eligible nodes in the XML column, if all the node values length is less or equal to 10. The minimum value for *n* is 1 and the maximum value it's page size dependent (see DB2 9 pureXML XML guide for relationship between database pagesize and *n* maximum value).

If this form of VARCHAR is specified, DB2 uses *n* as a size constraint. So, if you try to insert a XML document with nodes that are to be indexed and have values that are longer than *n*, the document will not be inserted. By the same reason, if you try to create an index over an XML column and the column

Indexing XML documents with DB2 9 pureXML

Page 7

contains documents with nodes that are to be indexed and have values that are longer than n, the index will not be created.

This constraint guarantees that if the index exists, all the candidate document nodes in the XML column will be correctly indexed and have an associated index key generated.

VARCHAR HASHED

This data type is used to handle indexing of arbitrary length character strings. The length of indexed string have no limits. For the indexed string, DB2 will generate an eight byte hash code over the entire string. Here is an example of an index using VARCHAR HASHED as the index key type:

```
CREATE INDEX empindex ON companies(doc) GENERATE KEY USING XMLPATTERN  
'/company/emp' AS SQL VARCHAR HASHED
```

Since this type of indexes use a hash value for each indexed node value, they are useful for equality comparisons of string values with length longer than the supported by VARCHAR (n) for the used pagesize. For other comparison types, like < and >, you should use VARCHAR (n) instead whenever is possible.

DATE

SQL data type DATE is used to index XML date node values. Example:

```
CREATE INDEX empbirthindex ON companies(doc) GENERATE KEY USING XMLPATTERN  
'/company/emp/birthdate' AS SQL DATE
```

TIMESTAMP

The data type TIMESTAMP is used to index XML DATETIME node values. Suppose that for some reason, we need to have the employee's hire date with a very high precision. In this case, would create a new *hiredate* element for each employee, and use a timestamp to store the value, instead of a single date. Consider this example of a *hiredate* element:

```
<hiredate>2005-02-14T07:45:57.345332Z</hiredate>
```

For this XML value, we need to create an index key of type TIMESTAMP, as the node value cannot be stored using data type DATE.

The syntax to create the index is:

```
CREATE INDEX empbirthindex ON companies(doc) GENERATE KEY USING XMLPATTERN  
'/company/emp/hiredate' AS SQL TIMESTAMP
```

4. Advanced Features

Selecting types of nodes

All the examples shown until now use XML document node names (from both element and attribute nodes) to create the index XPath expression. Although this is the most common way of using pattern expressions to index document nodes, we can also index document nodes based on their types instead of their name.

Indexing XML documents with DB2 9 pureXML

Page 8

The most common node types used are *text()* and *node()*. The *text()* node type identifies the text contents of a node. For example, if you want to index the text contents of the employee's department, you can create the following index:

```
CREATE INDEX empdeptindex ON companies(doc) GENERATE KEY USING XMLPATTERN
'/company/emp/dept/text()' AS SQL VARCHAR (20)
```

This index will create an index key for the text contents value of each dept node. In the case of a leaf node, *text()* returns the node value itself. Please see next section for more details on the use of *text()* on the XPath expression.

The *node()* node type can be used in the XPath expression to match any node, just like the wildcard ***.

Remember the index created in the section where we introduced the wildcard *** :

```
CREATE INDEX idindex ON companies(doc) GENERATE KEY USING XMLPATTERN
'/company/*/dept/@id' AS SQL VARCHAR (10)
```

We can rewrite this index by using the node type *node()* in the XPath expression:

```
CREATE INDEX idindex ON companies(doc) GENERATE KEY USING XMLPATTERN
'/company/node()/dept/@id' AS SQL VARCHAR (10)
```

The complete list of node types supported by DB2 9 pureXML is: *node()*, *text()*, *comment()* and *processing-instruction()*.

Using text() node in the path

The presence of *text()* node in the XPath expression is non-trivial and should be considered carefully, as it affects the index key generation for nonleaf elements.

For example, consider the following XML document:

```
<title>This is a <bold>great</bold> book about XML</title>
```

The path expression */title* will index the value "This is a great book about XML", while the path expression */title/text()* will index the value "This is a book about XML".

When the node name is used without *text()*, the result is the concatenation of its text contents with its children's text contents. If, however, you use the *text()* node in the path expression, you will only get the text contents of the current node and not of its children.

Overall, use of *text()* in the path expression is not recommended:

- It creates a less user friendly path, that requires more typing;
- Queries must also use *text()* for successful index matching.
- XMLSchema validation only applies to elements and not text nodes.

Using the UNIQUE keyword

The UNIQUE keyword can be used in the CREATE INDEX statement to enforce uniqueness across all XML documents stored in a single XML column.

The uniqueness of a node is enforced using the index data type, the XML path to the node and the value of the node after cast to the index data type.

Indexing XML documents with DB2 9 pureXML

Page 9

The syntax for creating unique indexes over XML documents is as follows:

```
CREATE UNIQUE INDEX uniqueempidindex ON companies(doc) GENERATE KEY USING
XMLPATTERN '/company/emp/@id' AS SQL DOUBLE
```

The previous statement will enforce uniqueness over all attributes id nodes that match the XPath expression `/company/emp/@id`

Note that when the UNIQUE keyword is specified, the XPath expression cannot contain:

- descendant::
- descendant-or-self::
- //
- wildcards for XML name test
- node() or processing-instruction() for XML node kind test

There are also cases when unique XML document node values can result in duplicate index keys when converting to the index data type:

- DOUBLE: unbounded decimal types and 64 bit integers may lose precision when stored as DOUBLE.
- VARCHAR HASHED: unique character strings may result in the same hash value.

Note that when you are creating an unique index, the uniqueness of the value is only enforced after the value is casted to the target data type.

Using XML Namespaces

XML Namespaces are commonly used in XML documents, and they carry indexing implications.

Consider the following CREATE INDEX statement:

```
CREATE INDEX firstnameindex ON companies(doc) GENERATE KEY USING XMLPATTERN
'/company/emp/name/first' AS SQL VARCHAR(10)
```

and the following XML document:

```
<company xmlns="http://www.ibm.com" name="Empire">
  <emp id="31201" salary="60000" gender="male">
    <name>
      <first>Darth</first>
      <last>Vapor</last>
    </name>
    <dept id="M25">Security</dept>
    <birthdate>1974-08-27</birthdate>
  </emp>
</company>
```

In this example document, the default namespace is `http://www.ibm.com`, and all the document nodes belong to this namespace. Because of this, the previous index will not match any document nodes, since the nodes identified by the path expression don't belong to the namespace `http://www.ibm.com`.

To create indexes that match the document nodes in the previous example, we need to add the namespace declaration to the CREATE INDEX statement. The most common way of doing it is using the default namespace declaration:

```
CREATE INDEX firstnameindex ON companies(doc) GENERATE KEY USING XMLPATTERN 'declare
default element namespace "http://www.ibm.com"; /company/emp/name/first' AS SQL
VARCHAR(10)
```

Using the default namespace declaration before the XPath expression will cause the previously defined index to create index keys for the document nodes that match the index's XPath expression. Note that the elements in the XPath expression need to belong to the namespace `http://www.ibm.com`.

We can also define a namespace prefix to use on element and attribute names on the XML the path:

```
CREATE INDEX firstnameindex ON companies(doc) GENERATE KEY USING XMLPATTERN 'declare
namespace "a=http://www.ibm.com"; /a:company/a:emp/a:name/a:first' AS SQL VARCHAR(10)
```

In case your XML documents contain more than one namespace, you can use multiple namespace prefixes in the XPath expression in order to be able to index your document nodes.

Using the EXPLAIN statement

IBM DB2 9 pureXML also provides you a way of determining if the XML indexes you created are or not useful. By using the EXPLAIN statement, you can generate access plans to your queries. Examining these access plans, you can determine if your indexes are being used or not for your queries. Please refer to DB2 9 pureXML documentation for more details on the use of the EXPLAIN statement.

5. Related Readings

1. Saracco, Cynthia M.. "Get off to a fast start with DB2 Viper", IBM developerWorks, Mar. 2, 2006.
(<http://www-128.ibm.com/developerworks/db2/library/techarticle/dm-0603saracco/>)
2. Chamberlin, Don and Saracco, Cynthia M.. "Query DB2 XML data with XQuery", IBM developerWorks, Apr. 6, 2006.
(<http://www-128.ibm.com/developerworks/db2/library/techarticle/dm-0604saracco/>)
3. IBM DB2 Viper XML guide, IBM, 2006.
(<ftp://ftp.software.ibm.com/software/data/pubs/papers/db2xmlguide.pdf>)
4. IBM DB2 9 pureXML Release Candidate 1 for Linux, UNIX, and Windows information center
(<http://publib.boulder.ibm.com/infocenter/db2luw/v9/index.jsp>)



© Copyright IBM Corporation, 2006
IBM Canada
8200 Warden Avenue
Markham, ON
L6G 1C7
Canada

All Rights Reserved.

Neither this documentation nor any part of it may be copied or reproduced in any form or by any means or translated into another language, without the prior consent of the IBM Corporation.

DB2, DB2 Universal Database, IBM, and the IBM logo are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries or both.

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

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurement may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

The information contained in this document is subject to change without any notice. IBM reserves the right to make any such changes without obligation to notify any person of such revision or changes. IBM makes no commitment to keep the information contained herein up to date.

References in this publication to IBM products or services do not imply that IBM intends to make them available in all countries in which IBM operates.