Indexes on xml Data Type Columns

Source	http://msdn2.microsoft.com/en-us/library/ms191497.aspx

Indexes on xml Data Type Columns

XML instances are stored in xml type columns as large binary objects (BLOBs). These XML instances can be large, and the stored binary representation of xml data type instances can be up to 2 GB. Without an index, these binary large objects are shredded at run time to evaluate a query. This shredding can be time-consuming. For example, consider the following query:

```
WITH XMLNAMESPACES
('http://schemas.microsoft.com/sqlserver/2004/07/adventure-
works/ProductModelDescription' AS "PD")
SELECT CatalogDescription.query('
  /PD:ProductDescription/PD:Summary
') as Result
FROM Production.ProductModel
WHERE CatalogDescription.exist
('/PD:ProductDescription/@ProductModelID[.="19"]') = 1
```

To select the XML instances that satisfy the condition in the WHERE clause, the XML binary large object (BLOB) in each row of table Production.ProductModel is shredded at run time. Then, the expression (/PD:ProductDescription/@ProductModelID[.="19"]) in the exist() method is evaluated. This run-time shredding can be costly, depending on the size and number of instances stored in the column.

If querying XML binary large objects (BLOBs) is common in your application environment, it helps to index the xml type columns. However, there is a cost associated with maintaining the index during data modification.

XML indexes fall into the following categories:

- **Primary XML index**
- **Secondary XML index**

The first index on the xml type column must be the primary XML index. Using the primary XML index, the following types of secondary indexes are supported: PATH, VALUE, and PROPERTY. Depending on the type of queries, these secondary indexes might help improve query performance.

Primary XML Index

The primary XML index is a shredded and persisted representation of the XML BLOBs in the xml data type column. For each XML binary large object (BLOB) in the column, the index creates several rows of data. The number of rows in the index is approximately equal to the number of nodes in the XML binary large object.

Each row stores the following node information:

- **Tag name such as an element or attribute name.**
- **Node value.**
- **Node type such as an element node, attribute node, or text node.**
- **Document order information, represented by an internal node identifier.**
- **Path from each node to the root of the XML tree. This column is searched for path expressions in the query.**
- **Primary key of the base table. The primary key of the base table is duplicated in the primary XML index for a back join with the base table, and the maximum number of columns in the primary key of the base table is limited to 15.**

This node information is used to evaluate and construct XML results for a specified query. For optimization purposes, the tag name and the node type information are encoded as integer values, and the Path column uses the same encoding. Also, paths are stored in reverse order to allow matching paths when only the path suffix is known. For example:

- **//ContactRecord/PhoneNumber where only the last two steps are known**

  OR
- **/Book/*/Title where the wildcard character (*) is specified in the middle of the expression.**

The query processor uses the primary XML index for queries that involve xml data type methods and returns either scalar values or the XML subtrees from the primary index itself. (This index stores all the necessary information to reconstruct the XML instance.)

For example, the following query returns summary information stored in the CatalogDescription xml type column in the ProductModel table. The query returns <Summary> information only for product models whose catalog description also stores the <Features> description.

```
WITH XMLNAMESPACES
('http://schemas.microsoft.com/sqlserver/2004/07/adventure-
works/ProductModelDescription' AS "PD")
SELECT CatalogDescription.query('
  /PD:ProductDescription/PD:Summary
') as Result
FROM Production.ProductModel
WHERE CatalogDescription.exist ('/PD:ProductDescription/PD:Features')
= 1
```

With regard to the primary XML index, instead of shredding each XML binary large object instance in the base table, the rows in the index that correspond to each XML binary large object are searched sequentially for the expression specified in the exist() method. If the path is found in the Path column in the index, the <Summary> element together with its subtrees is retrieved from the primary XML index and converted into an XML binary large object as the result of the query() method.

Note that the primary XML index is not used when retrieving a full XML instance. For example, the following query retrieves from the table the whole XML instance that describes the manufacturing instructions for a specific product model.

```
USE AdventureWorks;
SELECT Instructions
FROM Production.ProductModel
WHERE ProductModelID=7;
```

Secondary XML Indexes

To enhance search performance, you can create secondary XML indexes. A primary XML index must first exist before you can create secondary indexes. These are the types:

- **PATH secondary XML index**
- **VALUE secondary XML index**
- **PROPERTY secondary XML index**

PATH Secondary XML Index

If your queries generally specify path expressions on xml type columns, a PATH secondary index may be able to speed up the search. As described earlier in this topic, the primary index is helpful when you have queries that specify exist() method in the WHERE clause. If you add a PATH secondary index, you may also improve the search performance in such queries.

Although a primary XML index avoids having to shred the XML binary large objects at run time, it may not provide the best performance for queries based on path expressions. Because all rows in the primary XML index corresponding to an XML binary large object are searched sequentially for large XML instances, the sequential search may be slow. In this case, having a secondary index built on the path values and node values in the primary index can significantly speed up the index search. In the PATH secondary index, the path and node values are key columns that allow for more efficient seeks when searching for paths. The query optimizer may use the PATH index for expressions such as those shown in the following:

- **/root/Location which specify only a path**

  OR

- **/root/Location/@LocationID[.="10"] where both the path and the node value are specified.**

The following query shows where the PATH index is helpful:

```
WITH XMLNAMESPACES
('http://schemas.microsoft.com/sqlserver/2004/07/adventure-
works/ProductModelDescription' AS "PD")
SELECT CatalogDescription.query('
  /PD:ProductDescription/PD:Summary
') AS Result
FROM Production.ProductModel
WHERE CatalogDescription.exist
('/PD:ProductDescription/@ProductModelID[.="19"]') = 1
```

In the query, the path expression /PD:ProductDescription/@ProductModelID and value "19" in the exist() method correspond to the key fields of the PATH index. This allows for direct seek in the PATH index and provides better search performance than the sequential search for path values in the primary index.

VALUE Secondary XML Index

If queries are value based, for example, /Root/ProductDescription/@*[. = "Mountain Bike"] or //ProductDescription[@Name = "Mountain Bike"], and the path is not fully specified or it includes a wildcard, you might obtain faster results by building a secondary XML index that is built on node values in the primary XML index.

The key columns of the VALUE index are (node value and path) of the primary XML index. If your workload involves querying for values from XML instances without knowing the element or attribute names that contain the values, a VALUE index may be useful. For example, the following expression will benefit from having a VALUE index:

- **//author[LastName="someName"] where you know the value of the <LastName> element, but the <author> parent can occur anywhere.**
- **/book[@* = "someValue"] where the query looks for the <book> element that has some attribute having the value "someValue".**

The following query returns ContactID from the Contact table. The WHERE clause specifies a filter that looks for values in the AdditionalContactInfo xml type column. The contact IDs are returned only if the corresponding additional contact information XML binary large object includes a specific telephone number. Because the <telephoneNumber> element may appear anywhere in the XML, the path expression specifies the descendent-or-self axis.

```
WITH XMLNAMESPACES (
  'http://schemas.microsoft.com/sqlserver/2004/07/adventure-
works/ContactInfo' AS CI,
  'http://schemas.microsoft.com/sqlserver/2004/07/adventure-
works/ContactTypes' AS ACT)
SELECT ContactID
FROM   Person.Contact
WHERE
AdditionalContactInfo.exist('//ACT:telephoneNumber/ACT:number[.="111-
111-1111"]') = 1
```

In this situation, the search value for <number> is known, but it can appear anywhere in the XML instance as a child of the <telephoneNumber> element. This kind of query might benefit from an index lookup based on a specific value.

PROPERTY Secondary Index

Queries that retrieve one or more values from individual XML instances might benefit from a PROPERTY index. This scenario occurs when you retrieve object properties by using the value() method of the xml type and when the primary key value of the object is known. The PROPERTY index is built on columns (PK, Path and node value) of the primary XML index where PK is the primary key of the base table.

For example, for product model 19, the following query retrieves the ProductModelID and ProductModelName attribute values using the value() method. Instead of using the primary XML index or the other secondary XML indexes, the PROPERTY index may provide faster execution.

```
WITH XMLNAMESPACES
('http://schemas.microsoft.com/sqlserver/2004/07/adventure-
works/ProductModelDescription' AS "PD")
SELECT
CatalogDescription.value('(/PD:ProductDescription/@ProductModelID)[1]'
, 'int') as ModelID,
CatalogDescription.value('(/PD:ProductDescription/@ProductModelName)[1
]', 'varchar(30)') as ModelName
FROM Production.ProductModel
WHERE ProductModelID = 19
```

~~~ End of Article ~~~