COMPSCI 732

Numbering schemes in Native XML Databases (NXDs)

Tamino XML Server

Software AG pioneered the native XML database approach with a product called Tamino. Their aim was to build a DBMS from the bottom up to easily and efficiently store, retrieve, and query XML structured data.

Why use Tamino XML Server?

"An XML server can store XML data natively, which means without further conversion into other formats, and it can query and transform that data using the Web and XML specifications and interfaces. In addition, it integrates cleanly with Web Servers and application servers in existing infrastructures. Furthermore, Tamino XML Server does not try to replace existing relational databases for the tasks that RDBMS systems are well suited."

from http://www.softwareag.com/tamino/

XML	**RDBMS**
• Data in hierarchical structure	• Data in multiple tables
• Nodes have element and/or attribute values	• Cells have a single value
• Elements can be nested	• Atomic cell values
• Elements are ordered	• Row/column order not defined
• Elements can be recursive	• Schema required
• Schema optional	• Little support for recursive elements
• Direct storage/retrieval of XML documents	• Joins often necessary to retrieve XML documents
• Query with XML standards	• Query with SQL retrofitted for XML

Major mismatches between XML data and RDBMS (for storing XML documents)

Tamino XML Server

This type of native XML server system exposes the data and the processing model via XML standards:

• an XML document is generally the fundamental unit of storage

• XML DTDs or schemas rather than RDBMS schemas are used as the "data definition language" that defines the properties of document collections

• XPath or another XML-specific query language such as XQuery is used to locate documents meeting the search criteria

• and some products allow XML data to be processed with SAX, DOM, XSLT, etc., in the actual server engine as opposed to in an external utility.

The paper goes on to say …

1. XML enabled RDBMS are not optimal if you have document centric XML documents that need to be stored efficiently.
2. The distinction between document centric and data centric is fuzzy.
3. Tamino XML Server presents a "virtual database" upon which developers can build Web and electronic business applications using standard XML development tools, schemas, and query languages. The actual reality behind the scenes consists of a combination of native XML data stores and other data sources mapped onto XML.
4. The papers didn't elaborate on the physical storage ☹

Back to NXD generally

Native XML databases are designed especially to store XML documents, and also support:

- Transactions,
- Security,
- Multi-user access,
- Programmatic APIs,
- Query languages,
- …

Other features of NXD

NXDs are most useful for storing document centric documents because they support things like:

- Collections,
- Document order,
- Processing instructions,
- Comments,
- CDATA sections,
- XML query languages.

e.g. Can ask queries like "Get me all documents in which the second section title contains the word "Background""

Collections

Many NXD support collections.
You may have collections of documents or collections of elements.
This is useful for limiting queries to entities in the collection.

e.g., consider a NXD that stores students records (transcripts).
It is possible to define a collection based on year, so that queries can be limited to transcripts for a particular year.

Query languages

The most popular query languages to date are based on XPath.
Since the W3C has defined XQuery, XPath has become more popular.
Previously, many propriety query languages were supported.

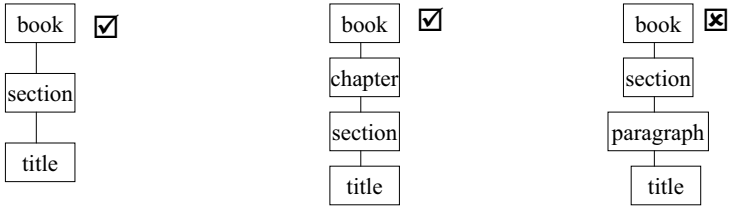
The query language must support different kinds of queries, including paths through the hierarchy, missing levels in the hierarchy, ordering, many documents, collections.

XPath

XPath uses path expressions to navigate through the logical, hierarchical structure of an XML document.
An XPath expression locates nodes within a tree.

book//section/title

Finds all "title" elements that are children of "section" elements which have an ancestor named "book".

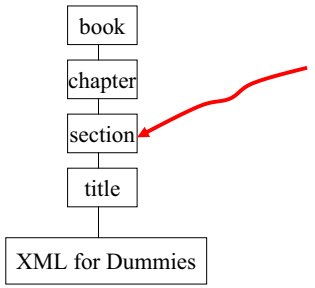


XPath predicate expressions

book//section[contains(title, 'XML')]

Find all sections whose title contains the string "XML".

The result of a path expression is a sequence of distinct nodes in document order.



XPath processing

book//figure/caption

Follow every path beginning at book to check for potential figure descendents, if there is no way to determine the location of "figure" descendents in advance.

Index structures are needed to efficiently perform queries on large document collections.

The indexes must support both structural and value based selections. B+ trees (or similar) work well for value based selections. What about ancestor/descendant and parent/child relationships?

Recapping – so far

- Tamino XML Server is a commercial native XML database. Tamino was the first NXD.
- Because it is a commercial product, we cannot find out much about the structure or organization of data in Tamino.
- NXD support features that most traditional database management systems support.
- Because of the nature of XML, there are some curly questions about how best to support some of these features.
- Querying is important and is more difficult due to the hierarchical nature of the data and the heterogeneity of the structure of the data.
- A query may involve both structure and values.

Numbering schemes for indexes

1. A numbering scheme assigns a unique identifier to each node in a logical document tree.

2. The generated identifiers are used in indexes as a reference to the actual node.

3. A numbering scheme provides mechanisms to quickly determine the structural relationship between a pair of nodes and to identify all occurrences of such a relationship in a single document or a collection of documents.

4. Useful for Xpath queries, such as,
//para[contains(., 'XML')]/parent::node()

Numbering scheme 1

Uses *document id, node position* and *nesting depth* to identify nodes.
Each element is identified by a 3-tuple
(document id, start position:end position, nesting level)

Start and end position might be defined by counting word numbers from the beginning of the document.

Ancestor-descendant relationships can be determined between a pair of nodes as follows:
A node *x* with 3-tuple (D1, S1:E1, L1) is a ancestor of a node *y* with 3-tuple (D2, S2:E2, L2) if and only if D1=D2, S1<S2 and E2<E1.

Example of numbering scheme 1

```
<contact>
  <name>Bill Smith</name>
  <phone>
    <office>3737599</office>
    <home>5993737</home>
  </phone>
</contact>
```

contact: (1, 1:14, 1)
name: (1, 2:5, 2)
phone: (1, 6:13, 2)
office: (1, 7:9, 3)
home: (1, 10:12, 3)

Is office a descendant of contact?
D1=D2, S1<S2, E2<E1
1=1, 1<7, 9<14 ☑
Is office a descendant of name?
D1=D2, S1<S2, E2<E1
1=1, 2<7, 9>5 ☒

Numbering scheme 2 - XISS

Assigns a pair of numbers <order,size> to each node such that

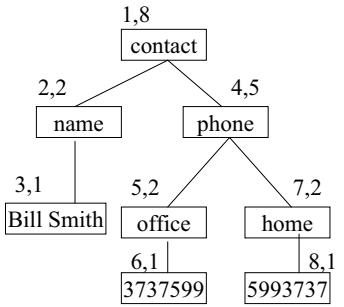
- for each node y and its parent x, order(x) < order(y) and order(y)+size(y) <= order(x)+size(x), and
- for two sibling nodes x and y, if x is the predecessor of y in preorder traversal, order(x)+size(x) <= order(y).

Order is assigned on a preorder traversal of the node tree, size is an arbitrary integer larger than the total number of descendants of the current node.

The ancestor-descendant relationship between two nodes can be determined by the following: x is an ancestor of y if and only if order(x) < order(y) <= order(x)+size(x).

Example of XISS

```
<contact>
  <name>Bill Smith</name>
  <phone>
    <office>3737599</office>
    <home>5993737</home>
  </phone>
</contact>
```



contact: (1, 8)
name: (2, 2)
Bill Smith: (3,1)
phone: (4, 5)
office: (5, 2)
home: (7, 2)
3737599: (6,1)
5993737: (8,1)

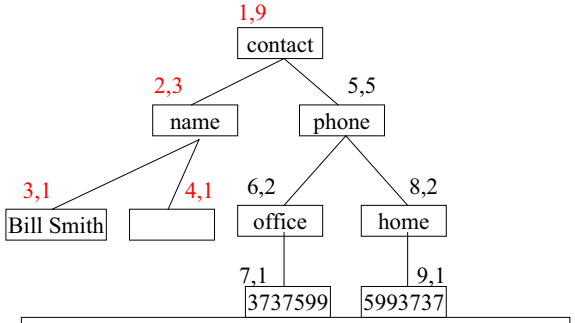
Is office (y) a descendant of contact (x)?
Order(x)<order(y)<=order(x)+size(x)
1<5<=9 ☑
Is office a descendant of name?
Order(x)<order(y)<=order(x)+size(x)
2<5>4 ☒

Advantages of XISS

- Ancestor-descendant relationship can be determined in constant time.

- Supports document updates via node insertions or removals by introducing sparse identifiers between existing nodes, and no reordering of the document tree is necessary unless the range of sparse identifiers is exhausted.

Example of advantages of XISS



Is office (y) a descendant of contact (x)?
Order(x)<order(y)<=order(x)+size(x)
1<6<=10 ☑
Is office a descendant of name?
Order(x)<order(y)<=order(x)+size(x)
2<6>5 ☒

Numbering scheme 3

Models the document tree as a complete k-ary tree, where k is equal to the max number of child nodes of an element in the document.

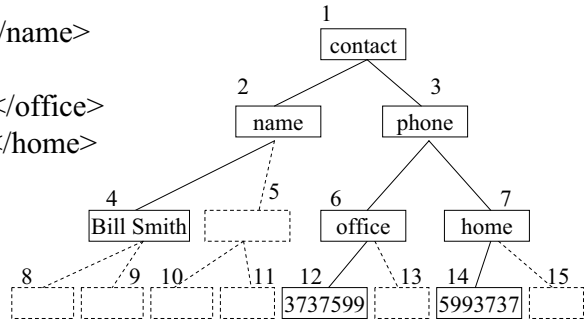
A unique node id is assigned to each node by traversing the tree in level-order.

Because the tree is assumed to be complete, spare ids are assigned at several positions.

This scheme makes it easy to determine the id of parent, sibling and possible child nodes.
e.g. for a *k*-ary tree, $parent_i$ = floor(((*i*-2)/*k*)+1)
for a k-ary tree, the jth child of node i is child(i,j) = k(i-1)+j+1

Example of numbering scheme 3

```
<contact>
  <name>Bill Smith</name>
  <phone>
    <office>3737599</office>
    <home>5993737</home>
  </phone>
</contact>
```



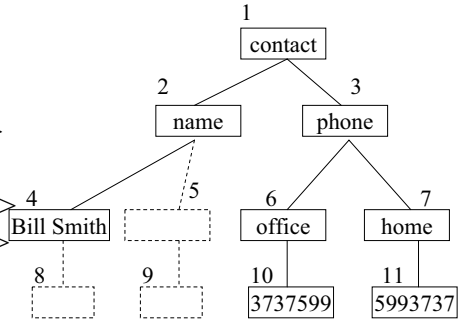
Disadvantage: Completeness constraint imposes major restriction on maximum document size to be indexed, particularly where tree is very unbalanced.
Generally, documents are unbalanced with many nodes at lower levels.

eXist's numbering scheme

Based on numbering scheme 3.
Drops completeness constraint in favor of an alternating scheme.
The number of children a node may have is recomputed for every level of the tree, such that:
for two nodes *x,y* of a tree, size(*x*)=size(*y*) if level(*x*)=level(*y*), where size(*n*) is number of children of node(*n*) and level(*m*) is the length of the path from the root of the tree to *m*.
The information about the number of children a node may have at each level is stored with the document in an array.

Example of eXist's numbering scheme

```
<contact>
  <name>Bill Smith</name>
  <phone>
    <office>3737599</office>
    <home>5993737</home>
  </phone>
</contact>
```



Advantages of eXists numbering scheme

Accounts for the fact that documents are more likely to have more nodes at lower levels in the document tree while there are fewer elements at the top levels of the hierarchy.

Less spare identifiers have to be inserted cf. numbering scheme 3.

Inserting a node at deeper levels in the tree has no effect on the identifiers assigned to nodes at higher levels.

The kinds of queries that are easily answered are those that are asked on the child, attribute and descendant axes, e.g., //para[contains(., 'XML')]/parent::node()

Disadvantages of eXists numbering scheme

eXist does not currently provide an advanced update mechanism.

Documents can be updated as a whole but it isn't possible to manipulate single nodes in eXist.

The authors of the paper are working on this…

Sources

- Tamino, http://www1.softwareag.com/Corporate/products/tamino/, 2004
- Quanzhong Li, Bongki Moon: Indexing and Querying XML Data for Regular Path Expressions. *Proceedings of 27th International Conference on Very Large Data Bases*, 2001.
- Wolfgang Meier, eXist: An Open Source Native XML Database, *Web, Web-Services and Database Systems, NODe 2002 Web and Database-Related Workshops*, LNCS 2593, 2003.